Ensemble Learning Based Convolutional Neural Networks for Remaining Useful Life Prediction of Aircraft Engines

Thambirajah Ravichandran[1], Bolun Cui[1], Yuan Liu[2], Sri Namachchivaya[1], Amar Kumar[2], and Alka Srivatsava[2]

[1] *University of Waterloo, Waterloo, ON, N2L 3G1, Canada*
travicha@uwaterloo.ca

[2] *Tecsis Corporation, 201-203 Colonnade Road, Ottawa, ON, K2E 7K3, Canada*
yliu@tecsis.edu

Abstract

Remaining useful life (RUL) prediction is an essential task of Prognostics and Health Management (PHM) of aircraft engines performed utilizing the data collected from multiple sensors to ensure their safety. While many studies have been reported on RUL prediction for aircraft engines, only a few of them focus on ensemble learning based convolution neural network (CNN) models for RUL prediction. This paper proposes a new data-driven approach based on a multistage ensemble learning strategy for developing CNN models for RUL prediction of aircraft engines. The proposed approach places a major emphasis on generating diverse CNN models by exploring 2D CNN models and 1D CNN models with multiple channels and developing a multistage ensemble approach employing sparsity promoting model selection and weight learning methods to utilize only a subset of available models thus improving the RUL prediction performance. The effectiveness of the proposed approach is validated using the NASA C-MAPSS dataset for aircraft engines.

1. Introduction

In Prognostics, the goal is to estimate the Remaining Useful Life (RUL) of a component or subsystem, indicating the time remaining before failure (Kalgren et al., 2006). Accurate RUL prediction is essential for ensuring reliability, safety, and cost-effective maintenance scheduling in areas like aircraft engines. Recent literature categorizes RUL prediction approaches into three types: model-based, data-driven, and hybrid approaches (Pecht & Jie Gu, 2009; Heimes, 2008; Heng et al., 2009).

Model-based approaches construct degradation process models based on physical principles of the target system, such as particle filters (Jouin et al., 2015) and Weibull distribution (Ali et al., 2015). While these approaches can achieve high accuracy if the system's degradation process is precisely modeled, they often require unrealistic prior knowledge of the target system.

Data-driven approaches rely on large amounts of sensor data and include machine learning (ML) techniques like neural networks (Gebraeel et al., 2004) support vector regression (SVR) (Khelif et al., 2017), and Bayes networks (Mosallam et al., 2016). Although these models are easier and less costly to develop, they may not consistently meet real-world performance requirements. Hybrid approaches combine model-based and data-driven techniques, leveraging physical models and updating parameters using data-driven methods.

Deep learning (DL) methods have emerged as effective tools for pattern recognition and predictive modeling tasks (Krizhevsky et al., 2017; Hinton et al., 2012). DL excels at capturing representative information from raw input data by utilizing complex deep structures and extracting high-level abstractions efficiently compared to shallow networks. Convolutional neural networks (CNNs), in particular, have achieved significant success in image and speech processing applications due to their ability to handle scale, shift, and distortion invariance, and their use of local receptive fields, shared weights, and spatial sub-sampling. During the recent past, CNNs have also been successfully introduced to the field of PHM for mechanical systems.

Recently, various studies have successfully demonstrated the application of CNNs for RUL estimation mainly taking advantage of their excellent automatic feature extracting capabilities for building regression models. Babu et al. (2016) introduced CNNs for RUL prediction by incorporating automated feature learning from the raw sensor data in a systematic way. Li et al. (2018) proposed a data-driven approach for prognostics using deep CNNs with a time window approach for sample preparation, allowing better feature extraction by CNNs. Yang et al. (2019) introduced further enhancements to the feature extraction ability of CNNs by incorporating a new kernel module for RUL prediction where the kernels are selected automatically. Lately, there has been a surge in research activities focusing on RUL prediction for turbofan engines using Long Short-Term Memory (LSTM) networks (Kong et al., 2019), modified CNNs incorporating temporal aspects (Yu et al., 2021), and hybrid formulations of LSTMs and CNNs (Li et al., 2019b; Mo et al., 2020; Peng et al., 2021). Vollert and Theissler (2021) provides an excellent review of the state-of-the-art ML and DL techniques including LSTMs and CNNs for RUL prediction using C-MAPSS datasets (Saxena & Goebel, 2008).

There have been increased interest in applying ensemble learning techniques for data-driven PHM applications in general, and RUL prediction in particular (Hu et al., 2012; Li et al., 2019a; Zeng & Cheng, 2020). The performance of a data-driven RUL prediction method could be influenced by various factors such as varying operational conditions and environmental uncertainties, varying linear or nonlinear degradation patterns of different system units, and the variability in the available number of sensors and the number of data samples. To effectively handle these varied scenarios, ensemble learning approaches have been investigated demonstrating superior generalization performance over single-model-based methods(Zhang et al., 2017; Shi et al., 2021). Wen et al. (2019) proposed a new residual CNN (ResCNN) in combination with a simple k-fold ensemble learning approach for RUL prediction turbofan engines. An ensemble learning finds or learns an appropriate combination multiple models (aka *base models or base learners*) by taking advantage of each base model so as to improve the generalization performance of the final ensemble model. The success of ensemble learning, developed using either same or different ML or DL algorithms, critically depends on the diversity among the base models generated. Increasing diversity among base models and finding an appropriate combination of diversified base models are the two key tasks in ensemble learning.

While many of the above studies resulted in very promising results for RUL prediction for various mechanical systems, only a few of them focus on ensemble learning of CNN models for RUL prediction for aircraft engines. Additionally, most of these studies work with a limited number of pre-selected diverse base models. In a typical ML or DL based RUL prediction model development process, many model candidates are explored through the use of K-fold or repeated holdout cross-validation combined with hyperparameter optimization of ML/DL models. These resulting multiple models provide an ample opportunity to construct a robust and high-performance ensemble model for RUL prediction. Hence, the major objectives of this study are: (i) to generate diverse RUL prediction models by exploring two different CNN model architectures, namely 2D CNN and 1D CNN with multiple channels and using time window approach for handling time-series data for better feature extraction by CNNs, (ii) to develop a multistage ensemble learning of CNN models for RUL prediction employing *sparsity promoting* model selection and weight learning methods in sequential and/or simultaneous manner to systematically and effectively utilize many base models available for ensemble model formation, and (iii) to demonstrate the proposed multistage ensemble learning of CNN models for RUL prediction of aircraft engines using the NASA C-MAPSS dataset.

The remainder of this paper is outlined as follows. Section 2 of this paper presents a brief introduction to convolutional neural networks (CNN) followed by two CNN model architectures combined with a time-window approach. Section 3 describes a multistage ensemble learning-based approach for CNN model development for RUL prediction. The effectiveness of the proposed method is demonstrated using the C-MAPSS dataset in Section 4. Section 5 provides the conclusion of this study.

2. CNN BASED RUL PREDICTION

In this study, our focus is on the investigation of ensemble learning techniques combined with convolutional neural networks by exploiting their individual strengths for the development of better performing RUL prediction models for aircraft engines. Before presenting the details on the proposed ensemble learning based CNN models for RUL prediction, the CNN based RUL prediction method is outlined in this section by describing first a brief background in CNN regression followed by two CNN model architectures along with the time-window approach adopted to handle time-series input data.

2.1. Convolutional Neural Network Regression Models

Convolutional neural networks belong to feed-forward types of neural networks because the information flows forward directly through the layers of the model (Schmidhuber, 2015) and there are no feedback connections involved in this type of models. The perceptual field of a convolutional unit with a given weight vector (filter) is moved step by step over a two-dimensional array of input values, such as pixels of an image (usually with several such filters). The resulting two-dimensional array of subsequent activation events for this unit can provide input to higher-level units, and so on. Among the many deep neural network models, CNNs are commonly applied to analyze imagery data and have achieved great success in computer vision applications (Wu et al., 2019).

The typical CNN architecture shown in Figure 1 contains a set of elementary consecutive blocks, namely, one input layer, multiple convolutional and pooling layers, several fully connected layers and one output layer. The input layer defines the data structure of the input used. A convolutional layer follows the input layer and performs the convolution operation over the input data. The size of the filters (also known as kernels) depends on the input data structure. Two-dimensional filters are used for grid-like inputs such as imagery data, whereas one-dimensional filters are used for time-series data. The size of each filter defines its receptive field. As part of the convolutional layer, a point-wise nonlinear activation function (such as sigmoid or ReLU) is applied. The convolutional layer is then followed by a so-called pooling layer, whose role is to reduce the number of parameters by sub-sampling the filtered signals. One common strategy to perform this operation is called max-pooling and consists of extracting only the maximum value of a fixed-sized batch of consecutive inputs. The CNN architecture is formed by stacking several instances of convolutional and pooling layers alternately through the network. The final filtered signals are then flattened and fed into a sequence of fully connected (FC) layers that map them into the output layer.

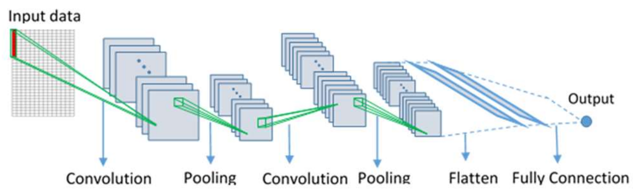


Figure 1. Typical convolutional neural network architecture.

In this study, a direct approach for RUL prediction is investigated by exploring CNN regression models to map the relationship between a set of input features and the associated target RUL value extracted from the run-to-failure trajectories of aircraft engines. For on-line prediction, RUL is estimated using the trained CNN model and the features extracted online. This direct approach for RUL prediction has the advantage of avoiding the setting of a failure threshold.

CNNs are able to extract representation information from raw input signals through multiple nonlinear transformations and approximate complex nonlinear functions and are used as the main architecture in this study. For RUL prediction, the adapted CNN regression models consist of multiple convolution and pooling layers followed by a fully connected layer. After identifying relevant input features, we employ the following two distinct CNN model architectures for RUL prediction: 2D CNN and 1D CNN with multiple channels as described below.

2.2. 2D CNN Models for RUL Prediction

For 2D CNN models, the time-series input data sample is prepared in 2D format, that facilitates the application of convolution operation. The dimension of the 2D input is L_{tw} x n_f where L_{tw} denotes the length of time sequence window and n_f is the number of selected features. This input data preparation involves grouping of sensor measurements from time $t\text{-}L_{tw}+1$ to t to create L_{tw} x n_f dimensional matrices as inputs for training. The raw features are usually obtained from multiple sensor measurements. The target value for each matrix was determined by the remaining useful life at the last time instance in the window. More details of this 2D input data preparation for 2D CNN models will be discussed in Section 4.

Various 2D CNN model architectures for RUL prediction can be experimented using the flexibility provided by the hyperparameters such as type and number of layers, neurons, and activation functions, etc. as described above in Section 2.1.

For these 2D CNN models, a 1D array is used as the kernel or filter. This choice is made to avoid filtering along the dimension that contains various features in the input matrices. CNNs are particularly effective in processing time-series data. Left plot of Figure 2 (left) illustrates the shape of the inputs and the kernels applied to the 2D CNN models.

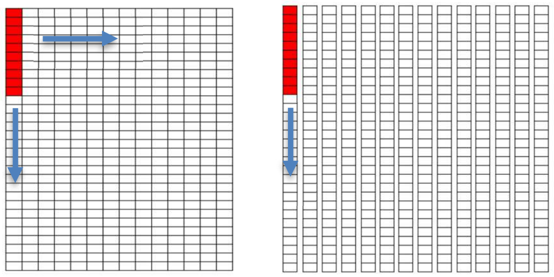


Figure 2. 2D input with 1D kernel applied to 2D CNN model (left) and 1D input with 1D kernel applied to 1D CNN model (right).

2.3. 1D CNN Models for RUL Prediction

An issue with the 2D CNN model is its use of the same kernel for all the features, disregarding their unique patterns. To address this, we propose a 1D CNN model with multiple channels for RUL prediction. This architecture treats different features independently by assigning them to separate channels available with CNN.

In the 1D CNN model with multiple channels, the input consists of n_f number of different L_{tf} x 1 dimensional vectors, each representing the time window of a feature and assigned to an individual channel. Right plot of Figure 2 (right) depicts the shape of the 1D input data and their assignment to separate channels along with the application of a 1D kernel in the 1D CNN model.

2.4. Performance Metrics and Model Validation

After building the CNN models for RUL prediction, it is important to evaluate their prediction performance on the test dataset. In this study, two performance metrics, namely RMSE and score function are used to evaluate the performance of the CNN models. Root mean square error (RMSE) is a very common metric widely used to evaluate the performance of regression models. The expression for computing RMSE of RUL models can be given in (1), where N is the total number of engines in the test dataset.

$$h_i = (Predicted\ RUL_i - True\ RUL_i)$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} h_i^2} \tag{1}$$

Score function is a metric that is particularly suitable for evaluating RUL models in PHM context as suggested in (Saxena & Goebel, 2008). This score function (S) can be computed using (2), where N is the total number of engines in the test dataset and S is the score.

$$h_i = (Estimated\ RUL_i - True\ RUL_i)$$

$$S\ or\ score = \begin{cases} \sum_{i=1}^{N}\left(e^{-\frac{h_i}{13}} - 1\right)\ for\ h_i < 0 \\ \sum_{i=1}^{N}\left(e^{\frac{h_i}{10}} - 1\right)\ for\ h_i > 0 \end{cases} \tag{2}$$

The desirable characteristic of this score metric is that it penalizes late predictions harder than the early prediction. This aligns with the risk adverse mindset because late predictions could possibly result in catastrophes while early predictions can only lead to the wastage of resources.

For validating CNN based RUL prediction models, data can be split three ways as a training set, a validation set, and a test set. In this study, a bootstrapping-like approach is adopted for holdout cross-validation (CV) by selecting randomly a small percentage of the overall training dataset as validation data samples *without replacement*. The remaining data samples will be used as the training dataset. This provides us with a non-overlapping division of the original training dataset into training and validation datasets. This procedure is also known as *a holdout* CV. To reduce the influence of randomness introduced by the data split this bootstrapping procedure is repeated *q* times. Over these *q* times repeated runs, model performance measures are computed using the validation dataset and the average of these performance measures are outputted as the model performance measures based on the holdout CV.

3. Ensemble Learning Based RUL Prediction

Ensemble learning is a process of combining multiple models using some model combination strategies to form a final ensemble model. Typical ML/DL approaches try to generate a single best model from given training data, whereas ensemble learning methods try to generate multiple models to solve the same problem. Ensemble learning generally provides ensemble models with improved accuracy and/or robustness in most applications due to the availability of accurate and diverse multiple models for combining them into a single ensemble model. Well known ensemble learning algorithms studied in the literature include stacking (Wolpert, 1992; Breiman, 1996a), bagging (Breiman, 1996b), and boosting (Freund & Schapire, 1996) algorithms.

3.1. Parallel Ensemble Learning Methods

In parallel ensemble learning methods, the base models are generated in parallel. The basic motivation of parallel ensemble methods is to exploit the independence between the base models since the error can be reduced dramatically by combining independent base models. After generating a set of base models, rather than trying to find the best single best model, ensemble learning resorts to model combination to achieve a better generalization ability, where the combination method plays a crucial role.

The parallel ensemble learning process studied in this paper for developing ensemble CNN models can be implemented in three phases as shown in Figure 3: 1) generation of base CNN models, 2) selection of base CNN models, and 3) aggregation of the selected base CNN models using some combination methods. In the first phase, a pool of base CNN models is generated. In the second phase, a subset of base CNN models is selected. Finally, an ensemble CNN model is formed by combining the selected base CNN models using a model combination method. To get a final ensemble model with improved generalization, it is essential that the base CNN models should be as accurate as possible, and as diverse as possible.

It is worth noting here that generally the computational cost of developing an ensemble of models is not much larger than generating a single model. This is because typically one need to generate multiple models when developing a single model using ML/DL techniques due to the requirement of multiple cross-validation and hyperparameter optimization, and this is comparable to generating base models in ensemble learning, while the computational cost for combining base models is often small.

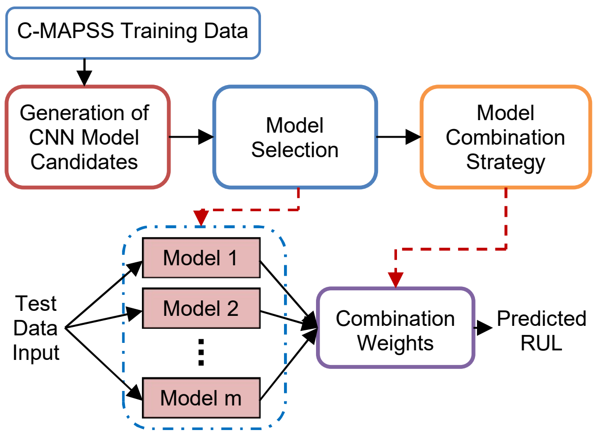


Figure 3. Proposed ensemble learning of CNN models.

3.2. Generation of Diverse Base Models

As stated before, diversity among base models is a very important factor contributing to the success of ensemble learning in general and the multistage ensemble formulation adopted here in particular. Generating diverse base models is not easy due to the fact that the individual models are trained for the same task from the same training data, and thus they are usually highly correlated.

There is no generally accepted formal definitions and measures for ensemble diversity, however, there are few effective heuristic mechanisms one can use to generate base model candidates with diversity for multistage ensemble construction. Most widely used diversity generation mechanism include: (i) data sample manipulation, (ii) input feature manipulation, and (iii) learning parameter manipulation among other things. In this study, to generate diverse CNN models in Section 2, all of these three diversity generation mechanisms are used together. Data sample manipulation is used in repeated holdout CV. Input feature manipulation is utilized in devising 2D and 1D input layers, and learning parameter manipulation is used in optimizing both 2D and 1D CNN model architectures.

3.3. Ensemble Model Combination

In this section, three different model combination methods, namely simple average, weighted average, and the stacking scheme implemented using the nonnegative least squares (NNLS) method, are described.

3.3.1. Simple Average (SA)

Due to its simplicity and effectiveness, simple average (SA) method is the most popular model combination method for regression problems. SA obtains the combined output by averaging the outputs of individual learners directly. Suppose we are given a set of m individual learners $\{h_1, \dots, h_M\}$ and the output of individual base model h_i for the data instance $\boldsymbol{x}$ (of CV data) is $h_i(\boldsymbol{x}) \in$ R, then the final prediction is given as

$$h_{en}(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m} h_i(\boldsymbol{x}) \tag{3}$$

3.3.2. Weighted Average (WA)

Weighted average (WA) method obtains the combined output by averaging the outputs of individual models with different weights implying different importance. Specifically, WA gives the combined output $h_{en}(\boldsymbol{x})$ as

$$h_{en}(\boldsymbol{x}) = \sum_{i=1}^{m} w_i h_i(\boldsymbol{x}) \tag{4}$$

where $w_i \geq 0$ is the non-negative weight for h_i, and these weights are usually assumed to be constrained by

$$\sum_{i=1}^{m} w_i = 1 \tag{5}$$

In this study, we adopted the following accuracy-based weighting (AW) method as the weighted average (WA) method. In this accuracy-based weighting method, the weight w_i of i^{th} base model in (4) can be defined as the normalization of the corresponding inverse of cross-validation RMSE:

$$w_i = \frac{(RMSE_i^{CV})^{-1}}{\sum_{i=1}^{m}(RMSE_i^{CV})^{-1}} \tag{6}$$

It can be noted from (6) that a base model with better accuracy has more influence on the predicted output of the ensemble model.

3.3.3. Ensemble Model Combination by Learning: Stacking

The parallel ensemble learning process shown in Figure 3 can use the *stacked regression* method (Breiman, 1996a) for determining model combination weights. Stacked regression is a model combination strategy using linear combinations of base models (also known as level-1 models) to give improved prediction accuracy. Here, the model combiner is called a level-2 model. In other words, stacked regression performs a weighted average (WA) type model combination as given (4) for parallel ensemble learning-based regression applications. In stacked regression, the basic idea is to train the base (level-1) models using the original training data, and then use CV data and any typical regression method to determine the coefficients for the level-2 model. However, for training the level-2 model, generally least squares methods under non-negativity constraints are used. The non-negativity constraint is needed to guarantee that the generalization performance of the stacked ensemble will be better than selecting the single best model (Breiman, 1996a). It can be noted here that the SA strategy for model combination can be considered as a special case of the stacked regression which is a WA strategy with specific constraints on the weights.

3.3.4. Nonnegative Least Squares (NNLS) Method for Model Combination Learning

To efficiently implement the training of the level-2 model with the least squares algorithm under non-negativity constraints, there is one dedicated algorithm available. This algorithm is called the nonnegative least squares (NNLS) method (Lawson & Hanson, 1974) which is briefly described here as a technique for model combination learning.

The problem of nonnegative least squares (NNLS) can be formulated as a constrained least squares problem where the coefficients are not allowed to become negative. That is, given a data matrix $\boldsymbol{X}$ and a target (output) $\boldsymbol{y}$ vector, the goal is to solve the following nonnegative least squares problem to determine the nonnegative parameter or weight vector $\boldsymbol{b}$:

Nonnegative Least Squares (NNLS)

$$\text{Minimize } \| \boldsymbol{Xb} - \boldsymbol{y} \|_2 \qquad \text{subject to } \boldsymbol{b} \geq 0$$

Here $\boldsymbol{b} \geq 0$ means that each component of the parameter vector $\boldsymbol{b}$ should be non-negative, and $\|.\|_2$ denotes the Euclidean norm of a real vector. In the context of the multistage ensemble learning, the data matrix $\boldsymbol{X}$ refers to the CV data generated by base models/learners, and the target $\boldsymbol{y}$ vector is approximated by the final output of the ensemble model which is the combined output of all the base models/learners. This algorithm can be readily implemented using the Matlab function *lsqnonneg*.

3.4. Ensemble Model Selection

Given a pool of base models, rather than combining all of them, ensemble model selection tries to select a subset of these base models to form the final ensemble model. Hence, this ensemble model selection process will promote sparsity of base models contributing to the final ensemble model thus improving the overall generalization performance. In this section, three different ensemble model selection methods, namely all possible subsets of combinations (APS), NNLS, and ordering-based selection (OBS), are described.

3.4.1. Ensemble Model Selection using All Possible Subsets of Combinations (APS)

For this ensemble model selection approach, a method similar to the *all possible subsets regression* technique is adopted by exhaustively exploring all the possible combinations of subsets of m base models (Miller, 2002). It is well known that for the case when m is more than 40 or 50, it is computationally very challenging to apply this approach without making special arrangements in terms of algorithm modification and parallel implementation. Accordingly, we limit the model subset search to generate model combinations each having at the most $p < m$ diverse models given the total number of diverse base models (m).

3.4.2. Ensemble Model Selection using Nonnegative Least Squares (NNLS)

For the purpose of model selection, this method uses the above-described nonnegative least squares method by exploiting its biggest strength in choosing zero coefficients for the model terms not required for ensemble formulation. This is possible due to the fact that the NNLS method is known to have a sparsity promoting attribute (Foucart & Koslicki, 2014), and hence it can be used simultaneously for model selection and weight learning tasks of the proposed multistage ensemble learning of CNN models resulting in the selection of fewer models as the base models when forming the final ensemble CNN model. This is demonstrated later in Section 4.3.

3.4.3. Ordering-Based Selection (OBS)

In this base model selection method, the base models are ordered according to some performance criterion, and only the base models in the front part are selected for the ensemble formation. In this study, we choose RMSE as the criterion to order the base models and select the top 50% of these models to form the final ensemble.

3.5. Multistage Ensemble Learning Procedure

Using the above major components of the ensemble learning process, we outline below the proposed multistage ensemble learning procedure required for developing the ensemble of CNN based RUL prediction models. Here, to effectively form the ensemble of many CNN models generated during the process of deep learning based RUL prediction model development, a multistage ensemble learning as described below is utilized. In STAGE 1, CNN models with different architectures but trained using the same division of training and validation data, are combined using various weight learning and model selection techniques described above. In STAGE 2, ensemble models formed during STAGE 1 are combined using SA and WA weight learning methods (without model selection), and SA with ordering-based selection. The workflow of this multistage ensemble learning process is given in Figure 4.

STAGE 0:

Step 1: Generate diverse CNN models using both 2D and 1D model architectures employing offline training and holdout CV procedures as described in Section 2. If we train M different CNN model architectures with K times repeat of holdout CV, we will have M x K diverse CNN base models from this step.

STAGE 1:

Step 2: From each holdout CV (for $k = 1{:}K$), collect M CNN base models and perform the following steps:

Step 2a: Determine STAGE 1 ensemble model weights using (i) SA and (ii) WA weight learning methods but without applying any ensemble model selection.

Step 2b: Perform ensemble model selection and determine STAGE 1 ensemble model weights using (iii) all possible subsets selection (APS) with SA weight learning, (iv) ordering-based selection (OBS) with SA weight learning, and (v) NNLS based ensemble model selection and weight learning.

Step 2c: Collect the selected models and their ensemble weights from each holdout CV.

STAGE 2:
Step 3: Form K ensemble outputs using selected models and their ensemble weights from STAGE 1 (Step 2c) and perform the following steps:
Step 3a: Determine STAGE 2 ensemble model weights using SA and WA weight learning methods but without using any ensemble model selection.
Step 3b: Perform ensemble model selection and determine STAGE 2 ensemble model weights using ordering-based selection (OBS) with SA weight learning.
Step 3c: Form the final multistage ensemble output on the test data using selected models and their ensemble weights from STAGE 2 (Step 3a and Step 3b).

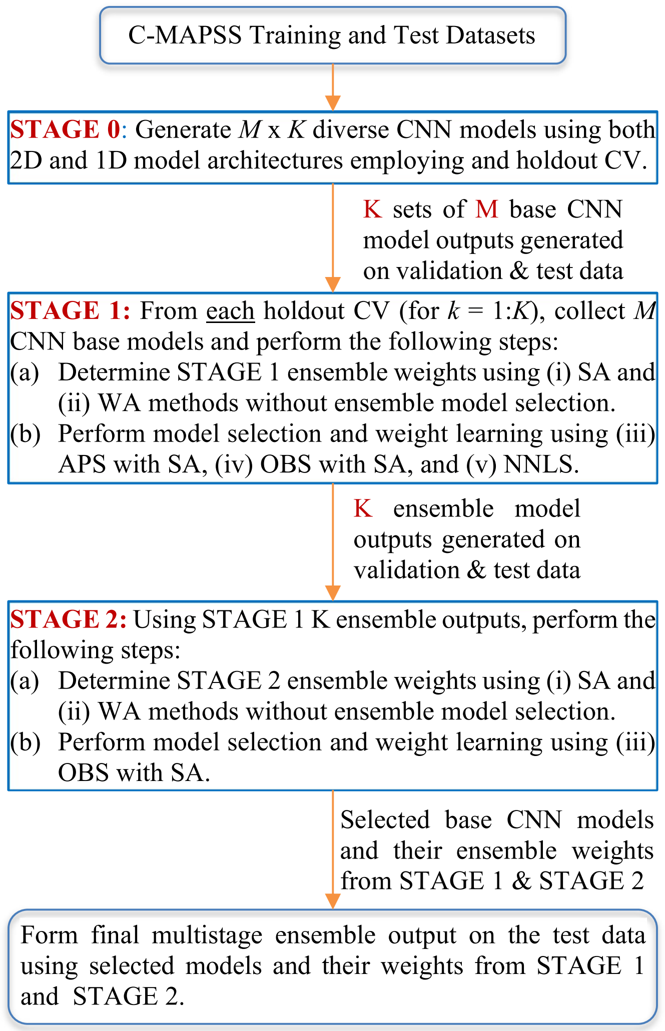

C-MAPSS Training and Test Datasets
STAGE 0: Generate M x K diverse CNN models using both 2D and 1D model architectures employing and holdout CV.
K sets of M base CNN model outputs generated on validation & test data
STAGE 1: From each holdout CV (for k = 1:K), collect M CNN base models and perform the following steps:
(a) Determine STAGE 1 ensemble weights using (i) SA and (ii) WA methods without ensemble model selection.
(b) Perform model selection and weight learning using (iii) APS with SA, (iv) OBS with SA, and (v) NNLS.
K ensemble model outputs generated on validation & test data
STAGE 2: Using STAGE 1 K ensemble outputs, perform the following steps:
(a) Determine STAGE 2 ensemble weights using (i) SA and (ii) WA methods without ensemble model selection.
(b) Perform model selection and weight learning using (iii) OBS with SA.
Selected base CNN models and their ensemble weights from STAGE 1 & STAGE 2
Form final multistage ensemble output on the test data using selected models and their weights from STAGE 1 and STAGE 2.

Figure 4. Multistage ensemble learning workflow

4. CASE STUDY AND EXPERIMENTAL RESULTS

4.1. NASA C-MAPSS Dataset and Data Preprocessing

4.1.1. NASA C-MAPSS Dataset

In this study, a simulated dataset of run-to-failure trajectories for a small fleet of aircraft engines under realistic flight conditions issued by the NASA Ames Prognostics Center of Excellence (PCoE) (Saxena & Goebel, 2008) is considered for developing RUL prediction models. This dataset was generated using the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dynamic model (Saxena et al., 2008) and is referred to as the NASA C-MAPSS dataset which is widely used for studying prognostics problems in aircraft engines.

The NASA C-MAPSS dataset consists of four sub-datasets, and for demonstrating the proposed ensemble learning based CNN models for RUL prediction aircraft engines, we consider the first sub-dataset, namely FD001 which consists of 26 columns, including trajectory number (or engine #), time (number of cycles already completed), 3 operational conditions, and 21 sensor measurements. However, some sensor readings have constant values during the lifetime of the engine, and they do not provide valuable information for RUL prediction. Therefore, in this study, only 14 sensor measurements with indices 2, 3, 4, 7, 8, 9, 11, 12, 13, 14, 15, 17, 20 and 21 are used as the raw input features as suggested in the literature (Zhang et al., 2017).

4.1.2. Data Preprocessing

Data preprocessing involves preparing raw data to make it suitable for a machine-learning model. Before we can feed the NASA C-MAPSS dataset into our deep learning models, it is crucial to clean up and transform the data into an appropriate format employing two essential aspects of data preprocessing: smoothing and normalization.

Normalization is the procedure of adjusting feature columns to have a similar scale when different features vary significantly in their scales. There are two commonly used normalization techniques in machine learning: Min-Max normalization and Standardization (Z-score) normalization. For this research, the Min-Max scaling method has been utilized.

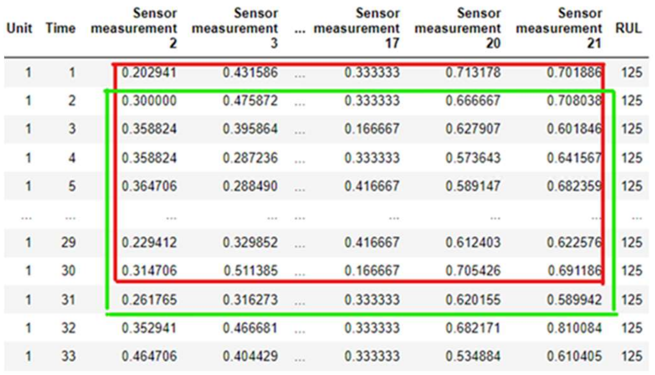
Smoothing plays a crucial role in reducing the noise present in the dataset, thereby enhancing the potential accuracy of RUL predictions. The smoothing technique employed in this research is the Savitzky-Golay filter (Schafer, 2011). By considering 2M + 1 inputs (M previous adjacent data points, the current data point, and M future adjacent data points), the filter calculates the smoothed data by fitting an N-degree polynomial. For this research, M has been set to 10, and N has been set to 3 to obtain the desired smoothing results.

4.1.3. Target RUL Calculation

Since the NASA C-MAPSS dataset doesn't provide target RUL values for the training data, we have to calculate the target RUL for each row to obtain complete training and testing datasets for developing RUL prediction models using the supervised learning procedure. The piecewise linear degradation model approach is adopted in this research to calculate the target RUL. In this method, the RUL for the early phase is set to a constant value 125 (all RUL values larger than 125 is set to 125 in this method). Compared to the method based on the linear degradation model, this method is more likely to generate more realistic RUL target values for developing RUL prediction models using supervised learning.

4.2. CNN Based RUL Prediction Results

The NASA C-MAPSS dataset was prepared for 2D CNN by applying the sliding time-windowing approach as discussed in Section 2.2. This involved grouping sensor measurements from time t-L_{tw}+1 to t (where $t >= L_{tw} = 30$) to create 30x14 dimensional matrices as inputs for training. The target value for each matrix was determined by the remaining useful life at the last time instance in the window. This 2D input data preparation yielded 17,731 input matrices from the training FD001 file, consisting of 20,631 rows and 100 engines. Figure 5 illustrates two 2D input matrices, with the red box representing the first matrix and its target value at $t = 30$, and the green box representing the second matrix with its target value at $t = 31$.



Unit	Time	Sensor measurement 2	Sensor measurement 3	...	Sensor measurement 17	Sensor measurement 20	Sensor measurement 21	RUL
1	1	0.202941	0.431586	...	0.333333	0.713178	0.701886	125
1	2	0.300000	0.475872	...	0.333333	0.666667	0.708038	125
1	3	0.358824	0.395864	...	0.166667	0.627907	0.601846	125
1	4	0.358824	0.287236	...	0.333333	0.573643	0.641567	125
1	5	0.364706	0.288490	...	0.416667	0.589147	0.682359	125
...	...	...	...	...	...	...	...	...
1	29	0.229412	0.329852	...	0.416667	0.612403	0.622576	125
1	30	0.314706	0.511385	...	0.166667	0.705426	0.691186	125
1	31	0.261765	0.316273	...	0.333333	0.620155	0.589942	125
1	32	0.352941	0.466681	...	0.333333	0.682171	0.810084	125
1	33	0.464706	0.404429	...	0.333333	0.534884	0.610405	125

Figure 5. Sliding time-windowing approach

For training CNN models for RUL prediction, the training file was divided into a training dataset of 80 randomly selected engines and a validation dataset of the remaining 20 engines. The maximum number of epochs was set to 300, and the batch size was chosen as 32.

4.2.1. 2D CNN Model Experiments

Given the numerous hyperparameters available for tuning CNN models and the difficulty of pre-identifying the optimal model, we conducted various experiments on 2D CNN models, and the following four 2D CNN architectures yielded better performance among those multiple experiments. To reduce the influence of randomness introduced by the train-validation data split, we ran each 2D CNN model architecture10 times using different random seeds.

For the description of 2D CNN model architectures in this section, we use the following notation to give the parameters associated with each type of layers: the parameters for the convolutional layer are given by (# of filters, filter size, activation function), the max pooling layer parameter is the pool size, and the parameters for the dense layers are given by (# of nodes, activation function).

***2D-CNN-Architecture 1*:** In the initial 2D CNN architecture using the 2D input data and 1D kernel, we used three Conv2D layers with parameters (64, (10, 1), relu), (32, (8, 1), relu), and (16, (5, 1), relu), respectively, and two MaxPooling2D layers with a pool size of 2 for down-sampling, followed by a flatten layer and two dense layers with parameters (32, relu) and (1, linear), respectively.

***2D-CNN-Architecture 2*:** In this 2D CNN architecture, we utilized a higher number of kernels in the convolutional layers compared to the 2D CNN architecture 1. Specifically, we used three Conv2D layers with parameters (128, (10, 1), relu), (64, (8, 1), relu), and (32, (5, 1), relu), respectively, and two MaxPooling2D layers with a pool size of 2 for down-sampling, followed by a flatten layer and two dense layers with parameters (32, relu) and (1, linear), respectively.

***2D-CNN-Architecture 3*:** This 2D CNN model architecture is similar to the 2D CNN model architecture 1 except it includes an additional convolutional layer just after the third convolutional layer. It consists of four Conv2D layers with parameters (64, (10, 1), relu), (32, (8, 1), relu), (16, (5, 1), relu), and (8, (5, 1), relu), respectively, and two MaxPooling2D layers with a pool size of 2 for down-sampling, followed by a flatten layer and two dense layers with parameters (16, relu) and (1, linear), respectively.

***2D-CNN-Architecture 4*:** In this 2D CNN architecture, an additional convolutional layer was added after the third convolutional layer, making it similar to the 2D CNN model architecture 2. It consists of four Conv2D layers with parameters (128, (10, 1), relu), (64, (8, 1), relu), (32, (5, 1), relu), and (16, (5, 1), relu), respectively, and two MaxPooling2D layers with a pool size of 2 for down-sampling, followed by a flatten layer and two dense layers with parameters (16, relu) and (1, linear), respectively.

4.2.2. 1D CNN Model Experiments

Similarly in the case of 2D CNN models, we conducted various experiments on 1D CNN models, and the following four 1D CNN model architectures yielded better performance among those multiple experiments.

***1D-CNN-Architecture 1*:** In the initial 1D CNN architecture using the 1D input data with multiple channels and 1D kernel, we used three Conv1D layers with parameters (64, (10, 1), relu), (32, (5, 1), relu), and (16, (5, 1), relu), respectively, and two MaxPooling1D layers with a pool size of 2 for down-sampling, followed by a flatten layer and two dense layers with parameters (32, relu) and (1, linear), respectively.

***1D-CNN-Architecture 2*:** This 1D CNN model architecture is similar to the 1D CNN architecture 1, but with additional kernels in the convolutional layers and more nodes in the dense layers. We used three Conv1D layers with parameters (256, (10, 1), relu), (64, (5, 1), relu), and (16, (5, 1), relu) respectively, and two MaxPooling1D layers with a pool size of 2 for down-sampling, followed by a flatten layer and two dense layers with parameters (64, relu) and (1, linear), respectively.

***1D-CNN-Architecture 3*:** This 1D CNN model architecture is similar to the 1D CNN model architecture 1, but with two additional convolutional layers just after the third convolutional layer. It consists of five Conv2D layers with parameters (64, (10, 1), relu), (32, (8, 1), relu), (16, (5, 1), relu), (8, (5, 1), relu), and (4, (5, 1), relu), respectively, and two MaxPooling2D layers with a pool size of 2 for down-sampling, followed by a flatten layer and two dense layers with parameters (32, relu) and (1, linear), respectively.

***1D-CNN-Architecture 4*:** This 1D CNN model architecture 4 is comparable to the 1D CNN model architecture 2, as it incorporates an additional convolutional layer after the third convolutional layer. It includes four Conv2D layers with parameters (128, (10, 1), relu), (64, (8, 1), relu), (32, (5, 1), relu), and (16, (5, 1), relu), respectively, and two MaxPooling2D layers with a pool size of 2 for down-sampling, followed by a flatten layer and two dense layers with parameters (16, relu) and (1, linear), respectively.

4.2.3. Results of CNN Models

Table 1 presents the performance comparison of CNN model architectures with varying numbers of kernels and layers on test dataset using the average values from 10 CV runs. Figure 6 shows the box-plot performance comparison of the same. These results indicate that increasing the number of kernels in the CNN model architecture enhances performance for both 2D and 1D CNN models. However, when it comes to adding more layers, it only improves the results for 2D CNN models, not for 1D CNN models. These findings suggest that the number of kernels plays a crucial role in improving performance across both model types, while the impact of additional layers is limited to 2D CNN models.

Table 1. Performance comparison of CNN models with different architectures.

Models	**RMSE**	**Score**
2D-CNN-Architecture 1	12.76	285.2
2D-CNN-Architecture 2	12.61	269.4
2D-CNN-Architecture 3	12.63	272.6
2D-CNN-Architecture 4	12.51	254.3
1D-CNN-Architecture 1	12.91	287.3
1D-CNN-Architecture 2	12.78	274.0
1D-CNN-Architecture 3	13.23	314.4
1D-CNN-Architecture 4	12.95	277.9

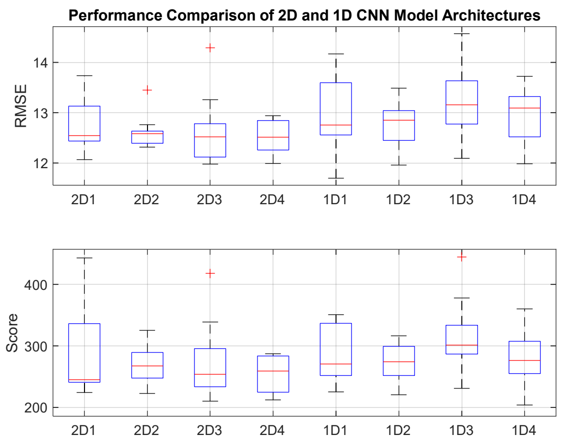


Figure 6. Performance comparison of CNN models.

4.3. Ensemble Learning Based RUL Prediction

Based on 80 diverse CNN base models generated using 8 different CNN model architectures with 10 times repeat for cross-validation in Section 4.2 (see Table 1 and Figure 6), we conducted various ensemble learning experiments using the multistage ensemble learning procedure given in Section 3.5. In STAGE 1, using validation data from 20 engines randomly selected during each run, ensemble learning was performed using five different combinations of weight learning (WL1) and model selection (MS1) methods as outlined in Section 3.5. Each of these WL1 and MS1 combination yields 10 ensemble models (one from each CV run) and their RMSE and score performance values obtained using validation data. In STAGE 2, these 10 ensemble models are further combined using three different combinations of weight learning (WL2) and model selection (MS2) methods as outlined in Section 3.5. It should be noted here that no data are used during STAGE 2 weight learning and model selection.

The test data RMSE and score results from these multistage ensemble learning experiments on CNN models are given in Table 2 utilizing the abbreviations used for weight learning

(WL1 & WL2) and model selection (MS1 & MS2) methods in STAGE 1 and STAGE 2 as described in Section 3.5. The last column of Table 2 indicates the overall number (#) of base models that took part in each of the multistage ensemble model learning experiments. Figure 7 shows the percentage improvement in RMSE and score values of the best ensemble CNN model (obtained with WL1-SA, MS1-APS, WL2-SA and MS2-OBS yielding RMSE and score values of 11.28 and 207.6, respectively) compared to the base models used for multistage ensemble learning illustrating an average improvement of 11.7% and 23.6% for RMSE and score values, respectively. It can be noted further from Table 2 that when model selection is used in STAGE 1 and/or STAGE 2, the resulting final ensemble model uses fewer base models (sparse model selection) and at the same time improving the generalization performance of the final ensemble CNN model for RUL prediction. This finding is also in line with the well-known observation in the ensemble learning literature that *ensemble pruning* to avoid overfitting improves generalization performance (Zhou et al., 2002) of ensemble models. Figure 8 shows the relationship between the number of base models used and the resulting accuracy of the ensemble CNN models by displaying three clusters of models. It is interesting to observe that in Figure 8 the best performing Cluster 3 is obtained when both MS1 and MS2 model selections are used. Using model selection only in either STAGE 1 or STAGE 2 results in moderately performing Cluster 2 and not using any model selection in both stages results in the least performing Cluster 1.

Table 2. Performance comparison of ensemble learning based CNN models for RUL prediction.

WL1	MS1	WL2	MS2	RMSE	Score	#
SA	-	SA	-	11.85	230.9	80
WA	-	SA	-	11.85	230.7	80
SA	OBS	SA	-	11.58	215.2	40
SA	APS	SA	-	11.55	218.7	35
NNLS	NNLS	SA	-	11.58	221.1	48
SA	-	WA	-	11.85	230.7	80
WA	-	WA	-	11.84	230.4	80
SA	OBS	WA	-	11.57	215.1	40
SA	APS	WA	-	11.54	218.4	35
NNLS	NNLS	WA	-	11.57	220.9	48
SA	-	SA	OBS	11.72	226.3	40
WA	-	SA	OBS	11.71	225.9	40
SA	OBS	SA	OBS	11.37	208.6	20
SA	APS	SA	OBS	11.28	207.6	15
NNLS	NNLS	SA	OBS	11.31	209.8	23

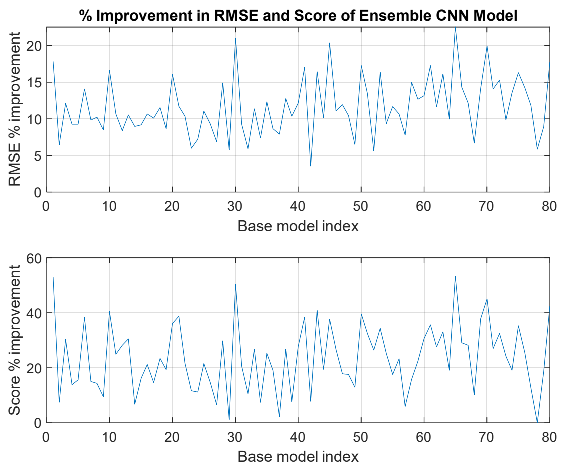


Figure 7. Performance improvement of the best ensemble CNN model compared to 80 base CNN models.

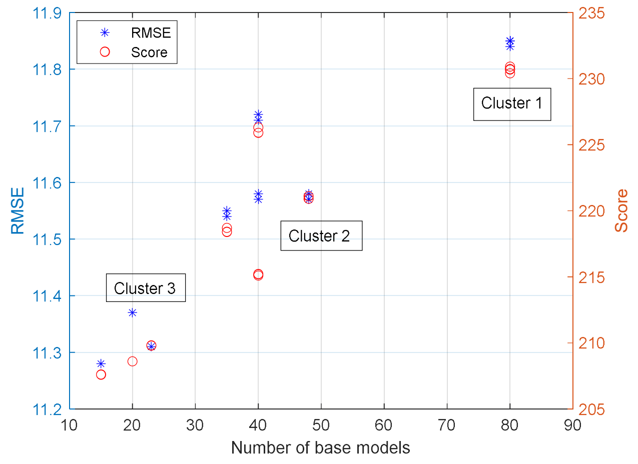


Figure 8. Ensemble CNN model RMSE (left) and score (right) performance versus number of base models.

Finally, the comparison of our proposed ensemble CNN with other popular deep learning methods published in the last few years is given in Table 3 which shows the RMSE and score function values generated on the C-MAPSS FD001 test sub-dataset. The proposed multistage ensemble learning based CNN has achieved promising performance comparable to the state-of-the-art results. Further improvements to the proposed method can be made by making enhancements to diverse base CNN models and also considering other deep learning models as base models along with CNN models for RUL prediction. Future work will consider these enhancements and the evaluation of the proposed method on the other C-MAPSS sub-datasets.

Table 3. Comparison of ensemble CNN results with other deep learning methods for RUL prediction.

Method	RMSE	Score
First CNN for RUL (Babu, 2016)	18.44	1290
DCNN (Li et al, 2018)	12.61	273.7
CNN + LSTM (Kong et al., 2019)	16.16	303
CNN (Yang et al., 2019)	12.18	224.16
DAG (Li et al., 2019)	11.96	229
Ensemble ResCNN (Wen et al., 2019)	12.16	212.48
CNN + LSTM (Mo et al., 2020)	12.19	259
LSTM + FCLCNN (Peng et al., 2021)	11.17	204
BLS + TCN (Yu et al., 2021)	12.08	243
Proposed Method on Ensemble CNN	11.28	207.6

5. CONCLUSION AND FUTURE WORK

In this paper, a new data-driven approach for RUL prediction of aircraft engines has been investigated using multistage ensemble learning based convolutional neural networks. To generate diverse base models, two CNN model architectures, namely 2D CNN and 1D CNN with multiple channels, were explored. Various CNN model experiments were performed to optimize their model architectures and hyperparameters, and using these resulting optimal CNN models, a multistage ensemble approach was investigated employing sparsity promoting model selection and weight learning methods to utilize only a subset of available models. The key findings of this work along with future directions can be summarized as follows:

- The effectiveness of the proposed approach was validated using the NASA C-MAPSS dataset for aircraft engines. The results showed that the average percentage improvement of 11.7% and 23.6% for RMSE and score values, respectively, of the best ensemble CNN model compared to the base models used for ensemble learning. Furthermore, the proposed multistage ensemble learning based CNN has achieved promising performance comparable to the state-of-the-art results. These results demonstrate the effectiveness of the proposed ensemble learning based CNN models in accurately predicting RUL based on sensor data.
- It can also be observed from this study that when model selection is used in both stages of the proposed multistage ensemble learning process, the resulting final ensemble CNN model uses fewer base models (sparse model selection) and at the same time improving the generalization performance of the ensemble CNN model for RUL prediction.
- For future work, we will consider the evaluation of the proposed method on the other C-MAPSS sub-datasets and other PHM datasets. More weight learning and model selection methods will be investigated to further improve the proposed multistage ensemble learning approach Also, further improvements to the proposed method can be made by making enhancements to diverse base CNN models and considering other deep learning models as base models along with CNN models for RUL prediction.

REFERENCES

Ali, J., Chebel-Morello, B., Saidi, L., Malinowski, S., & Fnaiech, F. (2015). Accurate bearing remaining useful life prediction based on Weibull distribution and artificial neural network. *Mechanical Systems and Signal Processing*, *56–57*, 150–172.

Babu, G.S.; Zhao, P.; Li, X.L. Deep CNN Based Regression Approach for Estimation of Remaining Useful Life. In *Proceedings of the International Conference on Database Systems for Advanced Applications*, Dallas, TX, USA, 16–19 April 2016.

Breiman, L. (1996a). Bagging predictions. *Machine Learning*, *24*(2), 123–140.

Breiman, L. (1996b). Stacked regressions. *Machine Learning*, *24*(1), 49–64.

Freund, Y., & Schapire, R. E. (1996). Experiments with a New Boosting Algorithm. *Proceedings of the 13th International Conference on Machine Learning*, 148–156.

Foucart, S., & Koslicki, D. (2014). Sparse recovery by means of nonnegative least squares. *IEEE Signal Processing Letters*, *21*(4), 498–502.

Gebraeel, N. Z., Lawley, M. A., Liu, R., & Parmeshwaran, V. (2004). Residual life predictions from vibration-based degradation signals: a neural network approach. *IEEE Transactions on Industrial Electronics*, *51*, 694–700.

Heimes, F. O. (2008). Recurrent neural networks for remaining useful life estimation. *2008 International Conference on Prognostics and Health Management*, 1–6.

Heng, A., Zhan, S., Tan, A., & Mathew, J. (2009). Rotating machinery prognostics: State of the art, challenges and opportunities. *Mechanical Systems and Signal Processing*, *23*, 724–739.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Vanhoucke, V., Nguyen, P., Sainath, T., & Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition. *Ieee Signal Processing Magazine*, *2*(november), 1–27.

Hu, C., Youn, B. D., Wang, P., & Taek Yoon, J. (2012). Ensemble of data-driven prognostic algorithms for robust prediction of remaining useful life. *Reliability Engineering and System Safety*, *103*, 120–135.

Jouin, M., Gouriveau, R., Hissel, D., & Zerhouni, N. (2015). *Particle filter-based prognostics : review , discussion and perspectives Particle filters - Theory and generalities*.

Kalgren, P. W., Byington, C. S., Roemer, M. J., & Watson, M. J. (2006). Defining PHM, A Lexical Evolution of Maintenance and Logistics. *2006 IEEE Autotestcon*, 353–358.

Khelif, R., Chebel-Morello, B., Malinowski, S., Laajili, E., Fnaiech, F., & Zerhouni, N. (2017). Direct Remaining Useful Life Estimation Based on Support Vector Regression. *IEEE Transactions on Industrial Electronics*, *64*(3), 2276–2285.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90.

Kong, Z., Cui, Y., Xia, Z., & He, L. (2019). Convolution and long short-term memory hybrid deep neural networks for remaining useful life prognostics. *Applied Sciences (Switzerland)*, 9(19).

Lawson, C. L., & Hanson, R. J. (1974). *Solving Least Squares Problems.* Prentice-Hall, New York.

Li, X., Ding, Q., & Sun, J. Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering and System Safety*, *172*(December 2017), 1–11.

Li, Z., Goebel, K., & Wu, D. (2019a). Degradation Modeling and Remaining Useful Life Prediction of Aircraft Engines Using Ensemble Learning. *Journal of Engineering for Gas Turbines and Power*, *141*(4), 1–10.

Li, J., Li, X. and He, D. (2019b). A Directed Acyclic Graph Network Combined With CNN and LSTM for Remaining Useful Life Prediction. *IEEE Access*, 7, pp. 75464–75475.

Miller, A. (2002) Subset Selection in Regression. Second. Boca Raton: Chapman & Hall.

Mo, H., Lucca, F., Malacarne, J., & Iacca, G. (2020). Multi-Head CNN-LSTM with Prediction Error Analysis for Remaining Useful Life Prediction. *2020 27th Conference of Open Innovations Association (FRUCT)*, pp. 164–171.

Mosallam, A., Medjaher, K., & Zerhouni, N. (2016). Data-driven prognostic method based on Bayesian approaches for direct remaining useful life prediction. *Journal of Intelligent Manufacturing*, *27*(5), 1037–1048.

Pecht, M., & Jie Gu. (2009). Physics-of-failure-based prognostics for electronic products. *Transactions of the Institute of Meas. and Control*, *31*(3–4), 309–322.

Peng, C., Chen, Y., Chen, Q., Tang, Z., Li, L., Gui, W., (2021). A remaining useful life prognosis of turbofan engine using temporal and spatial feature fusion. *Sensors (Switzerland),* 21(2), pp. 1–21.

Saxena, A., & Goebel, K. (2008). Turbofan Engine Degradation Simulation Data Set. *NASA Ames Prognostics Data Repository*.

Saxena, Aakanksha, Goebel, K., Simon, D., & Eklund, N. H. W. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. *2008 International Conference on Prognostics and Health Management*, 1–9.

Shi, J., Yu, T., Goebel, K., & Wu, D. (2021). Remaining Useful Life Prediction of Bearings Using Ensemble Learning: The Impact of Diversity in Base Learners and Features. *Journal of Computing and Information Science in Engineering*, *21*(2), 1–12.

Schafer, R. (2011). What Is a Savitzky-Golay Filter? [Lecture Notes]. *IEEE Signal Processing Magazine - IEEE SIGNAL PROCESS MAG*, *28*, 111–117.

Schmidhuber, J. (2015). Deep learning in neural networks : An overview. *Neural Networks*, *61*, 85–117.

Vollert, S., & Theissler, A. (2021). Challenges of machine learning-based RUL prognosis: A review on NASA's C-MAPSS data set. *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–8.

Wen, L., Dong, Y., & Gao, L. (2019). A new ensemble residual convolutional neural network for remaining useful life estimation. *Mathematical Biosciences and Engineering*, 16(2), 862–880.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, *5*(2), 241–259.

Wu, H., Fang, W. Z., Kang, Q., Tao, W. Q., & Qiao, R. (2019). Predicting Effective Diffusivity of Porous Media from Images by Deep Learning. *Scientific Reports*, *9*(1), 1–12.

Yang, H., Zhao, F., Jiang, G., Sun, Z., & Mei, X. (2019). A novel deep learning approach for machinery prognostics based on time windows. *Applied Sciences (Switzerland),* 9(22).

Yu, K., Wang, D. and Li, H. (2021). A prediction model for remaining useful life of turbofan engines by fusing broad learning system and temporal convolutional network. *8th Int. Conf. Inf., Cybern., Comput. Social Syst. (ICCSS)*, pp. 134–142.

Zeng, J., & Cheng, Y. (2020). An ensemble learning-based remaining useful life prediction method for aircraft turbine engine. *IFAC-PapersOnLine*, *53*(3), 48–5.

Zhang, C., Lim, P., Qin, A. K., & Tan, K. C. (2017). Multiobjective Deep Belief Networks Ensemble for Remaining Useful Life Estimation in Prognostics. *IEEE Transactions on Neural Networks and Learning* Systems, 28(10), 2306–2318.

Zhou, Z. H., Wu, J. and Tang, W. (2002). Ensembling neural networks: Many could be better than all. *Artificial Intelligence,* 137(1–2), pp. 239–263