A stacked deep convolutional neural network to predict the remaining useful life of a turbofan engine

David Solís-Martín [1,2] (iD), Juan Galán-Páez[1,2] (iD), and Joaquín Borrego-Díaz[2] (iD)

[1] *Datrik Ingellicence S.A., Seville, Spain*
david.solis@datrik.com, juan.galan@datrik.com

[2] *Department of Computer Science and Artificial Intelligence, Seville University, Spain*
dsolis@us.es, juangalan@us.es, jborrego@us.es

ABSTRACT

This paper presents the data-driven techniques and methodologies used to predict the remaining useful life (RUL) of a fleet of aircraft engines that can suffer failures of diverse nature. The solution presented is based on two Deep Convolutional Neural Networks (DCNN) stacked in two levels. The first DCNN is used to extract a low-dimensional feature vector using the normalized raw data as input. The second DCNN ingests a list of vectors taken from the former DCNN and estimates the RUL. Model selection was carried out by means of Bayesian optimization using a repeated random subsampling validation approach. The proposed methodology was ranked in the third place of the 2021 PHM Conference Data Challenge.

1. INTRODUCTION

Remaining useful life (RUL) is a cross-disciplinary field that belongs to the wider field of Prognostics and Health Management (PHM). This discipline studies the system behavior during its lifetime, that is, from the last check or maintenance performed on it until the system fails or the degradation of the system performance exceeds a certain threshold. Therefore, RUL enables to estimate the future reliability and scans the degradation of the system along the time (Peng, Wang, & Zi, 2018). The final goal is to estimate the remaining time until such failure occurs or a degradation level is reached, given the system working condition at any point of the system's lifetime.

Two main methodologies for RUL estimation can be found in the literature: model-based methods and data-driven methods. Model-based methods establish a degradation model based on physics and statistics to predict the degradation

trend of the system. Model-based methods require a strong knowledge on the physical behavior of each specific component and failure type, thus it is difficult to develop models for the current increasingly complex systems (Z. Zhao, Liang, Wang, & Lu, 2017). Moreover, some assumptions need to be taken on the models, thus those can be biased. For all these reasons, model-based methods have a limited prediction performance and have to be designed ad hoc for each different type of machinery.

On the other hand, data-driven methods use machine learning algorithms to learn the degradation trend of the system using historical condition monitoring data. In contrast to model-based methods, prior expertise is not required. Their application can be straightforward when enough data of quality are available.

Among the existing data-driven methods, Deep Learning (DL) is one of the most popular and promising fields of study. During the last decade, the use of DL techniques has surged significantly. Especially in complex tasks with high-dimensional nonlinear data. DL has had an enormous success in image processing, natural language processing, and signal processing. For this reason, it is not unexpected that DL based approaches are also widespread within the context of Prognostics and Health Management (PHM) research. One type of network that has been used to deal with sequence data is the deep convolutional neural network (DCNN) (LeCun, Haffner, Bottou, & Bengio, 1999). For example, in (B. Zhao, Lu, Chen, Liu, & Wu, 2017) the authors applied DCNN to time series classification over a variety of datasets. Within of the context of PHM, (Babu, Zhao, & Li, 2016) and (X. Li, Ding, & Sun, 2018) authors have applied DCNN to RUL prediction of aircraft turbofan engines.

2. DATA AND PROBLEM DESCRIPTION

The Commercial Modular Aero-Propulsion System Simulation (CMAPSS) is a modeling software developed at NASA.
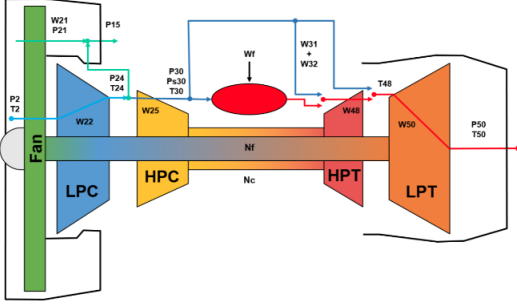
Figure 1. Schematic of the model used in CMAPSS (Arias Chao et al., 2021).

It was used to create the previous CMAPSS dataset (Saxena, Goebel, Simon, & Eklund, 2008). That first dataset was built by only taking samples from a system that is already degraded. "*Therefore, the onset of the fault cannot be predicted; only the evolution of the fault can*" (Arias Chao, Kulkarni, Goebel, & Fink, 2021).

A new dataset, named N-CMAPSS, has been created providing the full history of the trajectories starting with a healthy condition until the failure occurs. A schematic of the turbofan model used in the simulations is shown in Figure 1. All rotation components of the engine (fan, LPC, HPC, LPT, and HPT) can be affected by the degradation process. This model is defined as a nonlinear system denoted by:

$$\left[x_s^{(t)}, x_v^{(t)} \right] = F \left(w^{(t)}, \theta^{(t)} \right) \tag{1}$$

where x_s are the physical properties, x_v are the unobserved properties, w is the scenario-descriptor operating conditions and θ are the health model parameters.

2.1. Data

The dataset provided for the challenge consists of data from 90 simulated flights (extracted from the N-CMAPSS[1] (Arias Chao et al., 2021)). Seven different failure modes, related to flow degradation or subcomponent efficiency that can be present in each flight have been defined. The flights are divided into three classes depending on the length of the flight. In class 1, fall flights with a duration from 1 to 3 hours, class 2 is composed of flights between 3 and 5 hours, and class 3 contains flights that take more than 5 hours. Each flight is divided into cycles, covering climb, cruise, and descend operations.

2.2. Problem definition

The problem revolves around the development of a model $\mathcal{G}$ capable of predicting the remaining useful life Y of the sys-

[1]The N-CMAPSS dataset can be downloaded from the NASA data repository

Table 1. Variable description.

Symbol	Set	Description	Units
alt	W	Altitude	ft
Mach	W	Flight Mach number	-
TRA	W	Throttle-resolver angle	%
T2	W	Total temperature at fan inlet	°R
Wf	X_s	Fuel flow	pps
Nf	X_s	Physical fan speed	rpm
Nc	X_s	Physical core speed	rpm
T24	X_s	Total temperature at LPC outlet	°R
T30	X_s	Total temperature at HPC outlet	°R
T48	X_s	Total temperature at HPT outlet	°R
T50	X_s	Total temperature at LPT outlet	°R
P15	X_s	Total pressure in bypass-duct	psia
P2	X_s	Total pressure at fan inlet	psia
P21	X_s	Total pressure at fan outlet	psia
P24	X_s	Total pressure at LPC outlet	psia
Ps30	X_s	Static pressure at HPC outlet	psia
P40	X_s	Total pressure at burner outlet	psia
P50	X_s	Total pressure at LPT outlet	psia
Fc	A	Flight class	-
h_s	A	Health state	-

tem, using the sensor outputs X_s, the scenario descriptors W and auxiliary data A. The different variables available to estimate the RUL of the system are described in table 1. The former is an optimization problem that can be denoted as:

$$argmin \sum \mathcal{S} \left(y - \hat{y} \right) \tag{2}$$

where y and $\hat{y} = \mathcal{G}(x_s, w, a)$ are, respectively, the expected and estimated RUL. $\mathcal{S}$ is a scoring function defined as the average of the root-mean-square error (RMSE) and the NASA's scoring function (N_s) (Saxena et al., 2008):

$$\mathcal{S} = 0.5 \cdot RMSE + O.5 \cdot N_s \tag{3}$$

$$N_s = \frac{1}{M} \sum exp(\alpha|y - \hat{y}|) - 1 \tag{4}$$

being M the number of samples and α equal to $\frac{1}{13}$ in case $\hat{Y} < Y$ and $\frac{1}{10}$ otherwise.

3. PROPOSED METHODOLOGY

This section introduces the different phases or processes in which the proposed methodology can be decomposed.

3.1. Data normalization

The 20 variables used to develop the different models (see table 1) have different scales. For that reason, directly feeding the proposed networks with such data will slow down the learning and convergence of the models. Hence, a data normalization step, before the training stage, is required to ho-
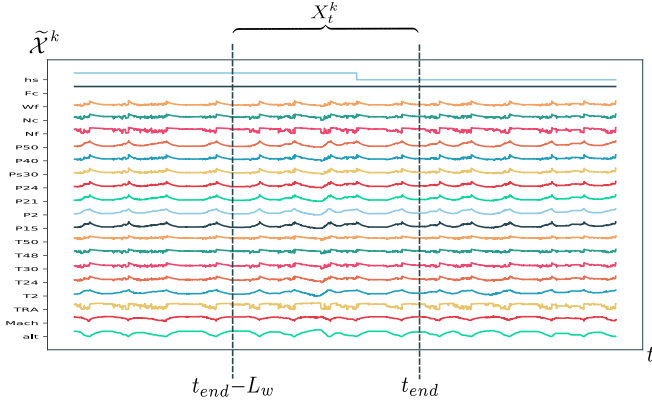
Figure 2. Sliding window.

mogenize the variables into a common scale. More precisely, the standard normalization is used in this paper:

$$x'_f = \frac{x_f - \mu_f}{\sigma_f} \quad (5)$$

where x_f is the data of a feature f, and μ_f and σ_f are its mean and standard deviation, respectively. Note that the mean and variance are computed for each training cross-validation set and are also used to normalize the validation sets.

3.2. Time window processing

After the data normalization, the inputs of the network are generated using a sliding time window through the normalized data. The size of the window, denoted as L_w, is a parameter to be selected during the model selection stage. The inputs generated can be expressed as $X_t^k = [\widetilde{\mathcal{X}}_{t_{end}-L_w}^k, ..., \widetilde{\mathcal{X}}_{t_{end}}^k]$ (see Figure 2) and the paired ground-RUL label is Y_t. With this approach, for each unit, $T^k - L_w$ samples will be considered. Where T^k is the total run time in seconds of each unit.

It is common to set a constant RUL at the beginning of the experiment (H. Li, Zhao, Zhang, & Zio, 2020), considering a healthy condition of the system during that early stage. In this work, this approach has not been followed since this kind of assumptions can introduce a bias in the model. Instead, the ground-RUL label has been set as a linear function of cycles from the RUL of each unit $Y_t^k = TUL^k - C_t^k$, where TUL^k is the total useful life of the unit k in cycles and C_t^k is the number of past cycles from the beginning of the experiment at time t.

3.3. Level 1 and level 2 model: Convolutional Neural Networks

The modeling phase consists of two levels (see Figure 3). In the first level, the goal is to find a good model to produce a time-windowed encoding of the raw input signals. This en-

LEVEL 1: ENCODING
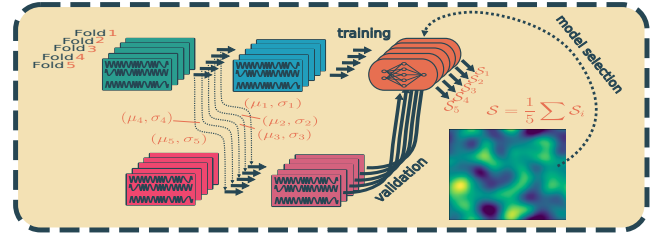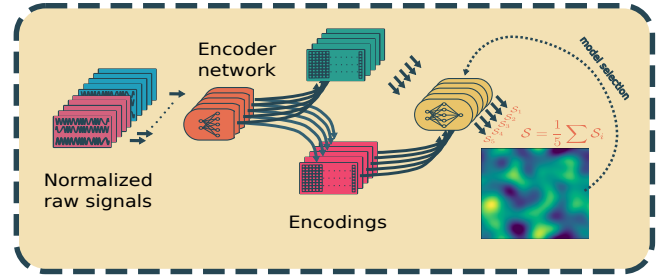

LEVEL 2: TRENDING

Figure 3. Methodology

coding is needed due to the high dimension of the input data. Another goal of this encoding step, besides the dimension reduction of the input, is to remove as much noise as possible. Such an encoding is used as the level 2 model input. The goal of this second level model is to provide an estimation of the RUL.

For both level models, Deep Convolutional Neural Networks (DCNN) have been selected. DCNN is a specialization of fully connected networks (FCN) that have been widely applied in image processing, natural language processing, and speech recognition with great success. The CNN uses parameter sharing and subsampling to extract feature maps with the most significant local features. The main operations in a DCNN are convolution and polling . The convolution operation implements parameter sharing and local receptive fields. The equation of the convolution is as follows:

$$S[i,j] = \sum_m \sum_n I[i + d*n, j + d*m] K[n,m] \quad (6)$$

where I is an input matrix, K is the kernel matrix (or convolution's parameters) of size n x m, d is the dilation rate, and S is the result of the convolution, called feature map. The kernel matrix is slipped across the input matrix looking for a pattern present in any place of it. Thereby, comparing with a FCN, the number of weights is reduced and, additionally, the overfitting chance. The pooling operation performs a downscaling by applying a statistic operation to each region of the input, which was previously divided into rectangular pool-
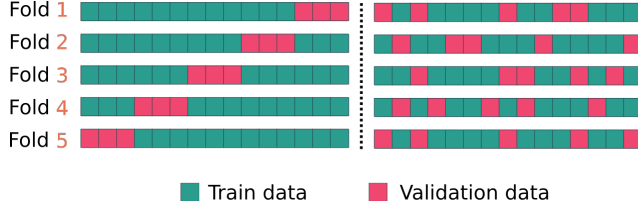
3

Figure 4. k-fold cross validation strategy (left), and k-fold random sampling strategy (right).

Figure 5. Level 1 networks architectures tested.

ing regions. The pooling operation serves a few porpouses: reduces the computational requirements for the upper layer since the feature maps are downscaled, reduces the number of connections (parameters) for the upper fully connected layers, provides a form of spatial transformation invariance and helps mitigating the overfitting risk.

3.4. Cross validation

Neural networks have a high number of parameters to be adjusted. Additionally, the size of the sliding window has to be optimized too. To deal with the overfitting problem during the model selection phase, a bunch of validation techniques can be found in the literature. The classical single hold-out strategy is discarded since it can easily overfit the hold-out set and lead to poor generalization results. The k-fold cross-validation has the disadvantage that the size of the validation set decreases as k (the number of folds) is increased. There exist alternatives as the k-fold repeated random subsampling validation that overcome this issue (see Figure 4). The later strategy has been selected in this work, with $k = 5$ and a validation fold size of 30% of the training set, that is, 27 random units from a total of 90.

3.5. Model Selection

The goal of the model selection phase is to obtain an optimal parameter configuration for each model. To this aim, Bayesian optimization has been selected as the optimization strategy. The models were trained using the RMSE as the loss function, since the NASA score is not differentiable. However, the loss function used in the Bayesian optimization to decide the next set of model parameters to be tested, is the score $\mathcal{S}$, defined in equation (3).

The well-known *early stopping* method, with a patience factor of 8 epochs, is used as a signal to finish the model training process. Hence, the training of the model will be stopped if the training loss on the validation fold has not been improved during the last 8 training epochs. Additionally, the learning rate is decreased by a factor of 0.1 when no improvement is seen on the validation loss in the last 3 training epochs.
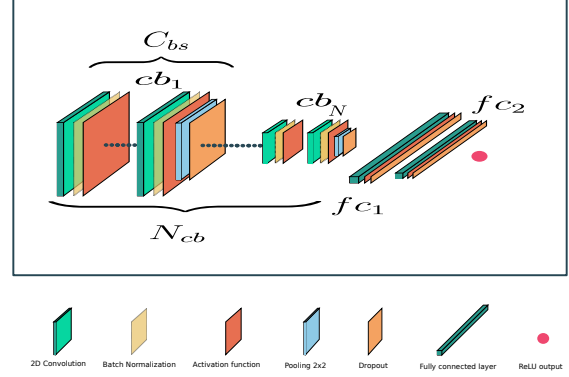
4. EXPERIMENTS AND RESULTS

This section describes the settings and parameters used to find the best models. Then the results of each level are compared and the final solution is described. Finally, this approach is applied to the hidden test data.

4.1. Level 1: encoding

The encoding model at level 1 of the stacking aims to carry out a dimension and noise reduction on the raw signals. To achieve this task, DCNN architecture was used. The classical DCNN architecture, which is shown in Figure 5, can be divided into two parts. The first one is a stacking of N_b blocks composed of convolution and pooling layers. The goal of this first part is to extract characteristics potentially useful for the task. The second part consists of fully connected layers and in this case will perform the regression of the RUL.

To obtain an optimal hyperparameter configuration, Bayesian optimization was executed during 100 iterations. The first 10 iterations, consist in models with randomly selected hyperparameters. These first 10 random points of the model hyperparameter search space are used by the Bayesian process to create the initial estimation of the error surface. After these 10 iterations, the optimization process applies Bayesian rules to select the most promising point (model hyperparameter) to be tested. Table 2 summarizes the input parameter ranges and the best parameter configuration found for the L1 model.

Figure 7 (top) shows the predictions for 4 units made by the DCNN model. The confidence intervals have been computed as $[\mu_p - 3\sigma_p, \mu_p + 3\sigma_p]$ where μ_p and σ_p are the mean and variance, respectively, of the cross-validation predictions. As it can be observed in the charts, the models are more reliable when the failure occurs earlier. In the first cycles, the prediction seems to be almost constant, until the model is able to catch the descending trend.

Table 2. Parameter ranges used in the Bayesian optimization and values of the best models for level 1 (L1) and level (L2)

Patameter	Range or value	L1	L2
L_w	[100, 500]	161	-
B_s	32,64	116	31
C_{bs}	[2,4]	4	4
N_{cb}	[2,4]	4	4
l_1	[0, 1e-3]	7.23e-4	6.96e-4
l_2	[0, 1e-3]	0	1.73e-5
l_r	[1e-5, 1e-3]	0.001	5.53e-4
dropout	[0, 0.9]	0.13	0.21
fc_1	[64, 256]	100	247
σ_{conv}	{tanh, ReLU, leakyReLU}	tanh	tanh
d_{rate}	[1, 10]	2	2
K_s	{(3,3), (10,1), (10,3)}	(10,1)	(10,1)
σ_{fc}	{tanh, ReLU, leakyReLU}	leakyReLU	leakyReLU
σ_{output}	ReLU	ReLU	ReLU
fc_2	[100, 1000]	-	105
$channels$	[1, 3]	-	3
$step$	[64, 256]	-	989
#Net params		1,514,016	2,575,449
RMSE		10.46	6.24
MAE		7.689	4.27
NASA score		2.13	0.64
CV $\mathcal{S}$ score		6.30	3.44
std($\mathcal{S}$)		0.37	0.63
Ensemble $\mathcal{S}$ score		-	2.95

4.2. Level 2: RUL estimation

The aim of the level 2 model of the stacking is to perform the final RUL predictions. The input of this model is the encoding of the raw signal generated by some of the level 1 models. Therefore, a first step is needed to generate the encoding for the train and test sets per fold. The encodings are extracted from the second full connected layer of 100 neurons.

Each sample input for each candidate level 2 model will be generated using a sliding window over the encodings. Since the encodings are a dimension reduced version of the raw signal, the level 2 model can expand the receptive field of the model and learn from the trend along the time. The parameter *step* defines the gap in seconds between encodings.

Figure 6 shows how the inputs for the models are obtained. The concept of image channel is used to compose the input. Each channel has 100 vector encodings. Therefore, the total number of encodings in the input is $100 \cdot chanells$.

The level 2 model is also a DCNN. The cross-validation schema and parameter ranges to be optimized are the same considered in the level 1 model. Addicionaly, three more pa-
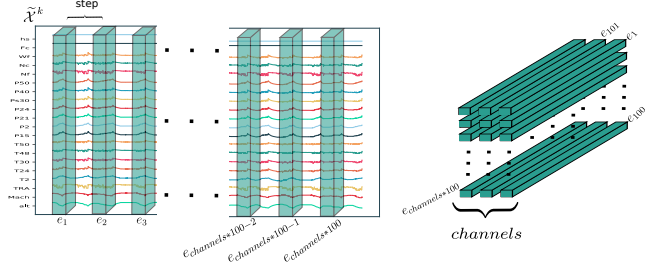

Figure 6. Sliding window for l2 models. Left: window over the raw signals to generate the encodings e_i. Right: encoding input shape for the DCNN.
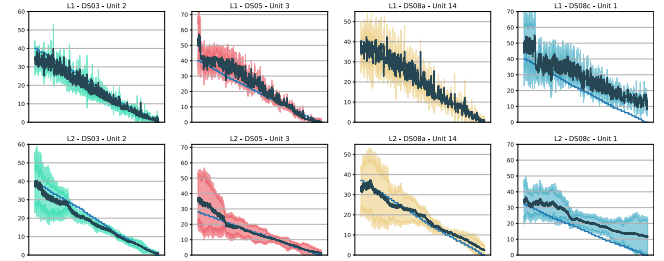

Figure 7. Prediction and confidence interval for 4 units the level 1 model (top) and the level 2 model (bottom).

rameters need to be considered in this L2 model optimization, namely fc_2, $step$, and $channels$ (see Table 2).

Table 2 summarizes the best parameter configuration found for the level 2 model, which is quite similar to that of level 1. This could be due to that the level 1 hyperparameters are provided to the optimization process as seed. There exist differences only in the training and regularization parameters. The rest of the architecture is identical. Regarding the additional parameters, it is interesting to note that the value selected for the parameter *step* is 989. Since the level 1 model provides a fairly good estimation of the RUL, the level 2 model only needs a set of sparse encodings to capture the trend and improve the RUL. The cross-validation score of this model is 3.44, while the ensemble score (that is, taking the average of predictions before computing the score), is 2.95. This gap between the cross-validation scores and ensemble predictions means that the predictions of the models have a good level of uncorrelation.

Figure 7 (bottom) shows the predictions and confidence intervals of the level 2 models. Comparing these with the level 1 models (Figure 7 top), one can see that the predictions of the stacked models have been smoothed and improved. Figure 8 presents the relation between the ground truth RUL and the score obtained by each stacked model. Finally, figure 9 shows the performance of each model in each class. Note that the shorter are the flights, the better the model performs. In the same way, the improvement in the level 2 model performance is higher for shorter flights.
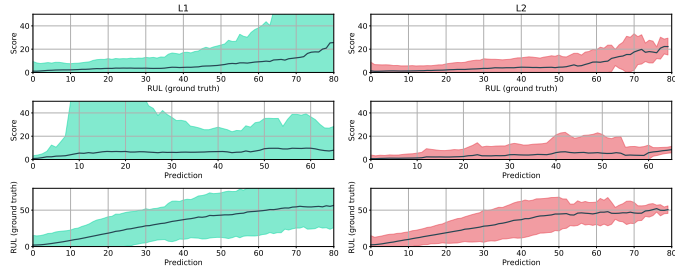
Figure 8. The chart shows the relation beetween RUL and the score (top), the prediction and the score (middle) and the prediction and RUL (bottom).

Figure 9. Performance of the models per class.

4.3. RUL prediction on test data

A common approach to obtain the final model consists in training it using all available data. In this approach, a number of training epochs must be selected, thus the validation data used during the model selection would also be part of the final training set. To select the number of epochs to train, the mean value among the best epoch obtained for each fold is considered. Usually, it would be necessary to save part of the training set as the final test set to validate the performance of the model. Hence, the number of samples in the training/validation set is reduced. A different approach has been used in the proposed methodology, which consists in using all models trained per fold as an ensemble. This approach has two main advantages. The first one is that it is possible to obtain a confidence metric of our final ensemble model performance by means of cross validation. The second one, related to the first, is that the predictions per sample could be used to create a confidence interval of the final mean prediction as it is shown in the Figure 7 and 8.

The hidden validation set to be scored by the competition committed was composed of 38 units. The predictions generated by the final solution are: 22, 21, 19, 24, 13, 18, 12, 22, 11, 6, 22, 24, 19, 10, 19, 21, 20, 17, 20, 26, 19, 12, 11, 13, 9, 37, 25, 4, 18, 25, 18, 14, 11, 12, 22, 10, 19, 19, which scored 3.651. This score is very close to the cross-validation score. An estimation of the score can be obtained by using the prediction/score distribution (Figure 8 bottom). However, the ensemble score is 2.95, which is far from the real score obtained. This means that there is room for improvement by reducing this overfit.

5. CONCLUSIONS

This work has exposed a robust methodology to estimate RUL without the need of expert knowledge on the system studied. The success of this methodology has been demonstrated with the results obtained in the 2021 PHM Conference Data Challenge (ranked in the third place). The goal of the challenge was to predict RUL in turbogan aircraft engines, using data generated with a CMAPSS simulator.
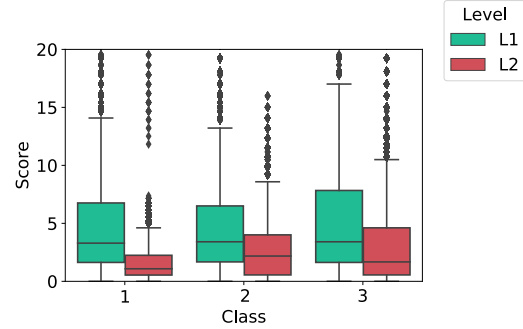
In this work, the process of obtaining the final solution is divided into two learning stages. In the first one, an encoding of the raw data is learned and used as input of the second learning stage to obtain the final model capable of estimating RUL. The finals results show that using one of the most basic architectures found in the literature is enough to achieve an excellent outcome.

A number of possible improvements of the current solution will be the target of future research: 1) Smoothing the RUL among cycles could smooth the error space, thus helping in the learning process. 2) In the proposed methodology, the inputs having a number of cycles lower than that required by the network, have been excluded from the training process. In the case of the validation set, these inputs were filled with the earliest encoding. It is expected that training the networks applying the same filling will help reducing the overfitting. 3) Another possible improvement could be to balance the training folds so each failure type is properly represented in each validation set. 4) Train the level 2 model with a random gap between encodings. This allow to expand the training set, have an additional mechanism to compute the confidence intervals, and increase the number of predictions to calculate the final averaged RUL. 5) Finally, it would be interesting to study how predictive are the class and the health state features.

This work has been developed with a focus on reproducible research. To this aim, the parameters obtained and the parameter ranges used for model selection have been well described. In addition, the source code to train the models and validate the results can be found in the source code repository https://github.com/DatrikIntelligence/Stacked-DCNN-RUL-PHM21.

(DIN2019)", of State Programme of Science Research and Innovations 2017-2020.

NOMENCLATURE

$\mathcal{X}$	Raw signals dataset
$\widetilde{\mathcal{X}}$	Normalized signals dataset
$\mathcal{Y}_t^k$	RUL of a unit k at time t
$\widetilde{\mathcal{X}}_t^k$	Normalized signals at time t of a unit k
L_w	Length of the sliding window
$\widetilde{X}_t^k$	Normalized signals of a unit k between $t - L_w$ and t
TUL^k	Total useful life of the unit k
C_t^k	Cycle number of the unit k at time t
$C_b s$	Convolution block size
$N_c b$	Number of convolution blocks
d_{rate}	Dilation rate of a convolution
fc_x	Neurons in the x-th fully connected layer
$\sigma_{conv}, \sigma_{fc}, \sigma_{out}$	Activation function of a convolution layer, fully connected layer and the output layer of a network.
K_s	Kernel size of a convolution
l_1, l_2	Weights for L_1 and L_2 regularization
l_r	Learning rate
B_s	Batch size

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... others (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)* (pp. 265–283).

Arias Chao, M., Kulkarni, C., Goebel, K., & Fink, O. (2021). Aircraft engine run-to-failure dataset under real flight conditions for prognostics and diagnostics. *Data, 6*(1), 5.

Babu, G. S., Zhao, P., & Li, X.-L. (2016). Deep convolutional neural network based regression approach for estimation of remaining useful life. In *International conference on database systems for advanced applications* (pp. 214–228).

Gulli, A., & Pal, S. (2017). *Deep learning with keras*. Packt Publishing Ltd.

LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision* (pp. 319–345). Springer.

Li, H., Zhao, W., Zhang, Y., & Zio, E. (2020). Remaining useful life prediction using multi-scale deep convolutional neural network. *Applied Soft Computing, 89*, 106113.

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

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., & Stoica, I. (2018). Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*.

Peng, Y., Wang, Y., & Zi, Y. (2018). Switching state-space degradation model with recursive filter/smoother for prognostics of remaining useful life. *IEEE Transactions on Industrial Informatics, 15*(2), 822–832.

Saxena, A., Goebel, K., Simon, D., & Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 international conference on prognostics and health management* (pp. 1–9).

Zhao, B., Lu, H., Chen, S., Liu, J., & Wu, D. (2017). Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics, 28*(1), 162–169.

Zhao, Z., Liang, B., Wang, X., & Lu, W. (2017). Remaining useful life prediction of aircraft engine based on degradation pattern learning. *Reliability Engineering & System Safety, 164*, 74–83.

APPENDIX

Hardware and software

The framework used in this work has been developed with Python 3.7. For network design and training, Tensorflow 2.0 (Abadi et al., 2016) and Keras 2.3 (Gulli & Pal, 2017) have been used. As model optimization framework, Tune 1.6.0 (Liaw et al., 2018) was selected.

The hardware used was a server with two GTX 1080Ti GPUs, 32GB of RAM memory, and 4 threads. The two GPUs were used to distribute different network training.