Self-supervised learning for efficient remaining useful life prediction

Wilhelm Söderkvist Vermelin[1], Andreas Lövberg[2], and Konstantinos Kyprianidis[3]

[1,2] *RISE Research Institutes of Sweden, Mölndal, Västra Götaland, 431 53, Sweden*
wilhelm.soderkvist.vermelin@ri.se
andreas.lovberg@ri.se

[1,3] *Mälardalen University, Västerås, Västmanland, 722 20, Sweden*
konstantinos.kyprianidis@mdu.se

ABSTRACT

Canonical deep learning-based remaining useful life prediction relies on supervised learning methods, which in turn requires large data sets of run-to-failure data to ensure model performance. In a considerable class of cases, run-to-failure data is difficult to collect in practice as it may be expensive and unsafe to operate assets until failure. As such, there is a need to leverage data that are not run-to-failure but may still contain some measurable, and thus learnable, degradation signal. In this paper, we propose utilizing self-supervised learning as a pretraining step to learn representations of data which will enable efficient training on the downstream task of remaining useful life prediction. The self-supervised learning task chosen is time series sequence ordering, a task that involves constructing tuples each consisting of n sequences sampled from the time series and reordered with some probability p. Subsequently, a classifier is trained on the resulting binary classification task; distinguishing between correctly ordered and shuffled tuples. The classifier's weights are then transferred to the remaining useful life prediction model and fine-tuned using run-to-failure data. To conduct our experiments, we use a data set of simulated run-to-failure turbofan jet engines. We show that the proposed self-supervised learning scheme can retain performance when training on a fraction of the full data set. In addition, we show indications that self-supervised learning as a pretraining step can enhance the performance of the model even when training on the full run-to-failure data set.

1. INTRODUCTION

Access to labeled data is a frequent issue in real-world applications and Prognostics and Health Management (PHM) related use cases such as Remaining Useful Life (RUL) prediction, is no exception. This issue limits the applicability of supervised learning techniques, the predominant machine learning paradigm. RUL prediction of some asset is defined as the task of mapping input data X to a target variable y which represent the time or cycles left until (critical) failure causes the asset to lose desired function:

$$f : X \mapsto y, \tag{1}$$

Approaching this problem with machine learning, the mapping f is learned using data of the form $\mathcal{D} = \{\mathbf{X}_t^n, y_t^n\}_{t=0}^{T_n}$ for $n = 1, \ldots, N$, where $\mathbf{X}_t^n$ is data collected until time t for the n:th asset and y_t^n is the corresponding remaining useful life at time t. In addition, $y_{T_n}^n = 0$, where T_n is the time or number of cycles at failure for the n:th asset. It is often convenient to scale RUL such that it is confined to the unit interval, i.e. $y_{T_n}^n \in [0, 1],\ t = 1, \ldots, T_n,\ n = 1, \ldots, N$. In this case, $y_{T_n}^n$ is interpreted as fractional remaining useful life. Supervised learning works well for this problem provided that the data set $\mathcal{D}$ is sufficiently large and diverse, meaning it contains a collection of assets monitored for some time.

What RUL and end-of-life (EOL) means in practice varies across domains and needs to be defined for the system at hand. In the case of aircraft turbofan jet engines, performance degradation and mechanical failure are often two different physical mechanisms. Aircraft turbofan jet engines typically fail due to low-cycle thermal fatigue, creep or oxidation. On the other hand, performance degradation occurs mainly due to fouling, erosion as well as seal wear and blade tip rubbing. RUL in a jet engine is often set through a limiting parameter, like measured turbine temperature. Once that temperature limit is reached – and temperature in the turbine is increasing due to performance degradation of the different components – then the engine is considered to have reached its life limit and must go for overhaul. From an airworthiness perspective this is backed by a series of cyclic mechanical tests that have been carried to that prescribed temperature level. This

is how RUL and performance degradation are linked and why in some cases there is a focus on forecasting the evolution of such temperatures (Kefalas et al., 2022). In this paper, such discussions are not in scope and we will use the provided RUL signal in the data set as given, however, such considerations should be kept in mind.

In practice, collecting run-to-failure data is often associated with high risk; letting assets run until end-of-life (EOL) can have substantial monetary consequences and can be unsafe. In fact, in the case of jet engines, the basis of airworthiness is to show through cyclic mechanical tests that the engine will not fail as long as it does not reach or exceed certain operating conditions. In this case, RUL becomes more of a remaining time to certification limit, and the task is primarily about predicting when performance degradation will cause certain measure parameters to reach their upper certification limit, requiring maintenance action (overhaul). In jet engines, one often speak of Time Between Overhaul (TBO) rather than RUL. Consequently, for several classes of assets, true run-to-failure data are rare or can only be obtained by means of simulation. Low access to run-to-failure data means that the supervised machine learning approach is less likely to yield fruitful results. It is, however, relatively cheap and safe to collect data during asset operation which are not run-to-failure, meaning that some Maintenance, Overhaul or Repair (MRO) action has been taken before EOL. Hence, there is a need for leveraging data that are not run-to-failure in RUL prediction.

To address this issue, we look for strategies used in other applications where unlabeled data are prevalent. The use of weakly labeled or unlabeled data has previously been investigated within PHM research where the focus has been on semi-supervised learning (Listou Ellefsen, Bjørlykhaug, Æsøy, Ushakov, & Zhang, 2019; Yoon et al., 2017). Broadly speaking, in semi-supervised learning one seeks to utilize unsupervised learning techniques to enhance predictive performance of supervised learning algorithms. Another promising approach is self-supervised learning (SSL) which will be the focus of this paper. SSL is a new paradigm in machine learning, in particular deep learning, which belongs to neither of the traditional machine learning branches; supervised learning and unsupervised learning. In essence, SSL works by constructing an artificial supervision signal from the training data and pretraining the deep neural network parameters to this artificial task. The network parameters are then fine-tuned to the target task using labeled data. SSL has seen good success in areas such as Natural Language Processing (NLP) (Zhou, Li, & Xie, 2021; Devlin, Chang, Lee, & Toutanova, 2018) and is expanding into other areas such as computer vision (Jing & Tian, 2021; Caron et al., 2021).

There has not been much research on using self-supervised learning in predictive maintenance or prognostics and health management. We have identified one prior work on SSL within prognostics and health management (Krokotsch, Knaak, & Gühmann, 2022). This work is addressing the same issue identified in this paper; run-to-failure data are scarce and there is a need to leverage data that are not full run-to-failure trajectories. In our work the SSL scheme is constructed differently from (Krokotsch et al., 2022) such that our proposed SSL pretraining involves a classification task whereas their SSL pretraining is a regression task.

The research questions addressed in this paper are the following:

1. How can SSL pretraining be used to improve remaining useful life predictions, in particular when full run-to-failure trajectories constitute a small portion of the full data set?
2. Can representations of data be learned from the proposed sequence ordering self-supervision task that are useful for remaining useful life prediction?

To conduct our experiments we use the "Commercial Modular Aero-Propulsion System Simulation" (CMAPSS) data set (Saxena, Goebel, Simon, & Eklund, 2008). The CMAPSS data set is a widely used data set for benchmarking and developing prognostics algorithms. It consists of simulated run-to-failure trajectories of turbofan engines. The trajectories comprise multivariate time series where 24 sensors and 3 operational settings are given for each flight cycle until failure. The objective is to use the trajectories a training set to create a model that can predict the remaining useful life of a number of truncated trajectories in a test set. In other words, the trajectories in the test set are incomplete. Details of the data set can be found in (Saxena et al., 2008).

Since its release, a wide range of different neural network architectures has been trained on the RUL estimation task using this data set. While these deep learning techniques typically outperform more traditional prognostics methods, they still require large amounts of labeled data (i.e. run-to-failure data). Very few studies has explored the use of semi- or unsupervised learning methods to overcome this limitation.

2. METHODOLOGY

The main motivation for using SSL is that in typical condition monitoring data, only a small portion of data (if any) are full run-to-failure trajectories. Since the CMAPSS data set training data are full run-to-failure trajectories, we truncate these time series for the self-supervised training to emulate a lack of complete run-to-failure trajectories. More precisely, the last 10 % of the full run-to-failure trajectory is left out in SSL pretraining, meaning the age of the engine used in the SSL training step at most 90 % of EOL. The self-supervised learning scheme we consider is a sequence ordering task where sequences of the CMAPSS data are sampled and put into sets

of sequences, which we will refer to as k-tuples (k being the number of sequences in the set).

This idea is inspired by a learning scheme called "shuffle and learn" (Misra, Zitnick, & Hebert, 2016). In this paper the authors are pretraining a deep neural network (DNN) to recognize whether sequences of video frames are in order or shuffled. They show that this pretraining step helps the DNN to learn useful representations for the downstream task of action recognition, so much so that they were able to exceed the performance of the current state-of-the-art at the time.

Our self-supervision task is similar where we instead of frames will use short slices of the engine time series data and construct k-tuples which are either ordered or shuffled, see Figure 1. This SSL pretraining task is useful for the downstream task of RUL estimation since the network will learn to order sequences in terms of time until end-of-life. Since sequences closer to end-of-life have a higher degree of degradation, the network will be able will implicitly learn representations of degradation level, which in turn is a proxy for RUL.

SSL is used to train a feature extractor, referred to as g. The feature extractor is a DNN with one-dimensional convolutional layers and a fully connected layer which outputs embedded input data (also referred to as encodings). In Figure 2 the architecture of the feature extractor is shown.

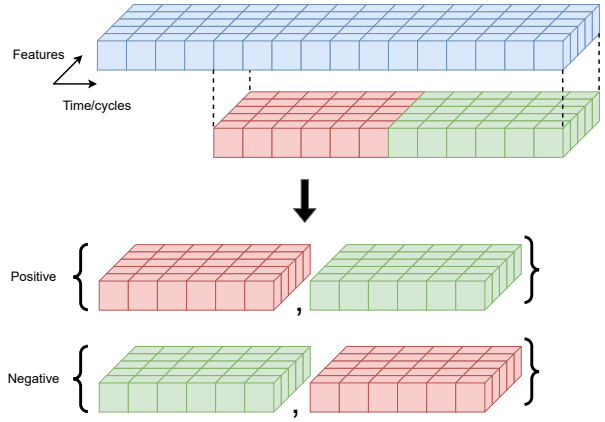

Figure 1. An overview of the sequence ordering SSL classification. A slice of the engine time series data is sampled and split up into k parts (in the above image, $k = 2$), forming a k-tuple of sequences of data from the original time series. The tuple is then shuffled with some probability or the same order is kept. If a tuple is shuffled or not it is referred to as a negative or a positive tuple, respectively.

The full architecture of the SSL scheme is shown in Figure 3. The self-supervised training is performed by feeding sequences from the k-tuple into k copies of the feature extractor g, also known as siamese networks (Chicco, 2021), to obtain k encodings $e_1, \ldots, e_k$ where $e_i \in \mathbb{R}^{d_e}, i = 1, \ldots, k$. d_e is

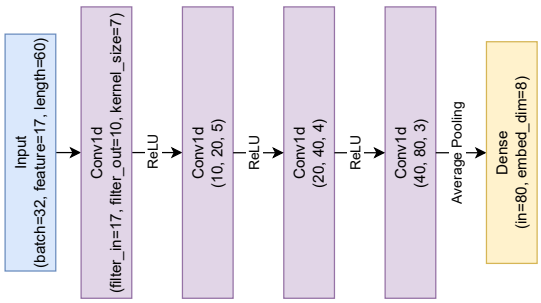

Figure 2. Overview of the feature extractor model architecture. The model is a deep neural network with four one-dimensional convolutional layers and a fully connected (dense) output layer.

referred to as the encoding dimension, which is a hyperparameter to be chosen as such. Concatenation of the encoded slices is performed and we obtain a $k \times d_e$-dimensional vector: $e_1 \oplus \cdots \oplus e_k \in \mathbb{R}^{k \times d_e}$ which is finally fed into a classification head which is fully connected multilayer perceptron (MLP). Since the SSL task has two classes, either the k-tuple is correctly ordered or not, the classification error is measured using binary cross entropy loss, defined as follows:

$$\ell_{\text{BCE}} = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})] \tag{2}$$

where y is the true class label and $\hat{y}$ is the estimated probability of the class.

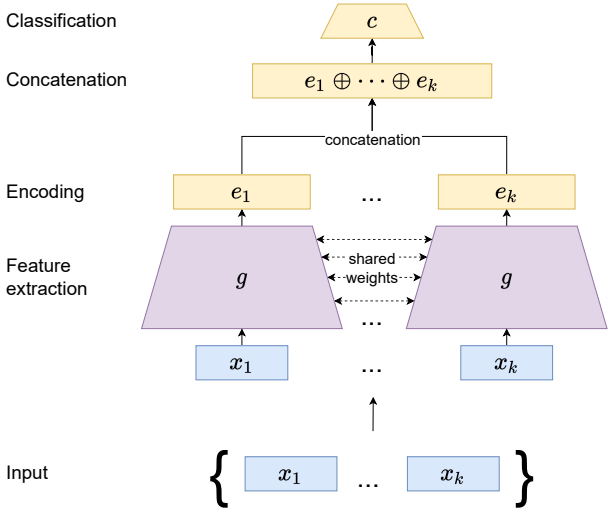

Figure 3. An overview of the SSL pretraining scheme. A k-tuple consisting of k slices of engine time series data is fed into the feature extractor g to obtain encodings $e_1, \ldots, e_k$. The encodings are concatenated and subsequently fed into the classification head which is trained to learn if the original tuple was correctly or incorrectly ordered.

The final RUL estimation model is shown in Figure 4. This model consists of the feature extractor g which produces the

embedded time series sequences (encodings) discussed previously. To obtain RUL estimates the encodings are passed through a RUL prediction head. The RUL prediction head is a feed-forward neural network with one hidden layer and ReLU activations. The final layer in the RUL head maps to a single real number which is interpreted as the estimated RUL, $\hat{y}_{\text{RUL}}$. Since the RUL is scaled such that $y_{\text{RUL}} \in [0, 1]$ the final output is transformed by a hard sigmoid activation function, defined as

$$\text{Hardsigmoid}(x) = \begin{cases} 0 & \text{for } x < -3 \\ 1 & \text{for } x \geq 3 \\ \frac{x}{6} + \frac{1}{2} & \text{otherwise} \end{cases} \tag{3}$$

The RUL model is trained using mean-square-error (MSE) loss defined as follows:

$$\ell_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} (y_{\text{RUL}} - \hat{y}_{\text{RUL}})^2, \tag{4}$$

where y_{RUL} is the true RUL, $\hat{y}_{\text{RUL}}$ is the estimated RUL and N is the number of devices under test (engines). For the baseline model training, the entire model (g and the RUL head) is trained with randomly initialized weights with ordinary supervised learning and MSE-loss.

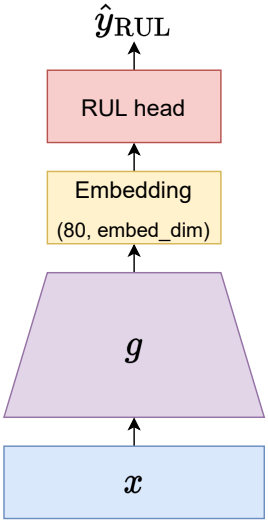

Figure 4. RUL estimation model. The model consists of the feature extraction network g with a fully connected layer which maps encodings to a single real number which is interpreted as the estimated RUL $\hat{y}_{\text{RUL}}$.

In our implementation of the above models, we have chosen the size of the k-tuples as two (meaning $k = 2$). We set the embedding dimension $d_e = 8$. These values were obtained through experimentation and chosen due to better performance on the downstream task of RUL prediction. Intuitively, choosing a low value of k ensures that longer sequences of the time series is seen during pretraining which is closer to the downstream task. The embedding dimension is chosen such that it is large enough to capture variations in the input sequences but small enough to compress the input such that only relevant features are preserved. We use a batch size of 32 and a learning rate of 0.001. The model weights are updated through stochastic optimization using the Adam-optimizer (Kingma & Ba, 2015) with weight decay (Loshchilov & Hutter, 2019). The models were implemented in Python (Van Rossum & Drake Jr, 1995) using the PyTorch deep learning library (Paszke et al., 2019).

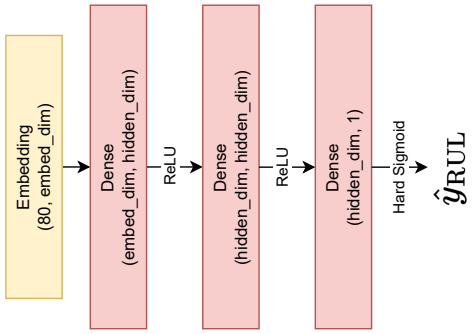

Figure 5. Remaining useful life prediction head. The embedded sequences produced by the feature extractor is passed through the RUL prediction head to obtain RUL estimates.

In the next section, we will compare the performance of the RUL model with and without SSL pretraining.

3. RESULTS

To measure the effects on applying SSL as a pretraining step we will use two performance metrics, root-mean-squared error (RSME) and a scoring function, as proposed in (Saxena et al., 2008), defined as follows

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} d^2}, \tag{5}$$

$$s = \begin{cases} \sum_{i=1}^{N} \exp\left(-\frac{d}{a_1}\right) - 1 & \text{for } d < 0 \\ \sum_{i=1}^{N} \exp\left(\frac{d}{a_2}\right) - 1 & \text{for } d \geq 0 \end{cases} \tag{6}$$

where s is the resulting score, $d = \hat{y} - y$ the difference between the estimated RUL and the true RUL, N is the number of devices under test (engines), and $a_1 = 10$, $a_2 = 13$. The scoring function is defined such that overestimation of RUL is more penalized than underestimation.

In Figure 6 we see a UMAP dimension reduction plot (McInnes, Healy, Saul, & Großberger, 2018) on the embedded time series sequences obtained from the feature extrac-

tor g (cf. Figure 3). The projection maps the encodings (8-dimensional vectors) to 2-dimensional space. The coloring indicates RUL (darker means lower RUL) which in turn is an indicator of degradation level. We observe that the feature extractor has learned representations of data such that slices of the engine time series data are separated into regions of high and low degradation (low and high RUL, respectively). This indicates that the SSL task constructed is useful for learning representations of data which are useful for the downstream task of RUL prediction.

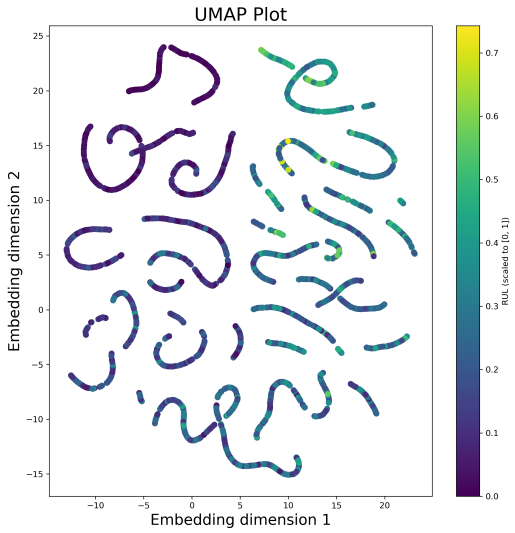

Figure 6. UMAP dimension reduction plot. We observe that the feature extractor obtained from the SSL sequence ordering task has learned to separate sequences of data according to level of degradation (which is proportional to RUL).

In Figures 7 and 8 we see a comparison between the baseline model and the model that has been pretrained using our SSL scheme in terms of root-mean-square error (cf. Equation (5)) for each engine in the test data set. In the first figure we see the results from training on 100 % labeled data (meaning 100 % full run-to-failure trajectories) and in the second Figure only 10 % of training data is labeled. We observe an improvement in RMSE when using SSL pretraining compared to the baseline model. This means that we should expect the pretrained model to be more accurate than the baseline model.

In Figures 9 and 10 we compare scores, as defined in Equation (6), between the baseline model and the model that has been pretrained using SSL, for each engine in the test data set. As before, in the first figure we see the results from training on 100 % labeled data and in the second Figure only 10 % of training data is labeled. We observe that there is a distinct improvement in score when using SSL pretraining compared to the baseline model. The scoring function (6) is constructed such that overestimation of RUL is penalized to a higher degree than underestimation. From this we conclude that the SSL pretrained model is overestimating RUL to a lesser extent than the baseline model, while also being more accurate (the SSL pretrained model has lower RMSE than the baseline model). We can observe this in Figures 11 and 12. In Figure 11 the RUL predictions of both models are shown with the true RUL of each engine drawn in solid black. The vertical axis represent each engine in the test set, sorted from high to low RUL. We observe that the SSL pretrained model is more accurate, especially in the high degradation regime (low RUL) and also overestimates RUL to a lesser extent, as compared to the baseline model. The corresponding results at

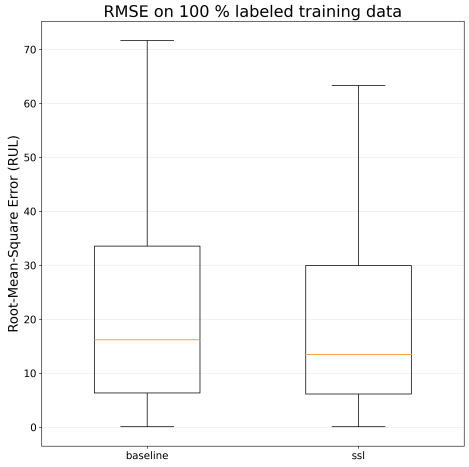

Figure 7. Root-mean-square errors for the baseline model and the model with SSL pretraining when 100 % labeling is used in training data (training data are 100 % full run-to-failure trajectories). The model architecture is kept constant, the only difference is whether SSL pretraining has been applied or not. We observe an improvement in RMSE if SSL pretraining is applied.

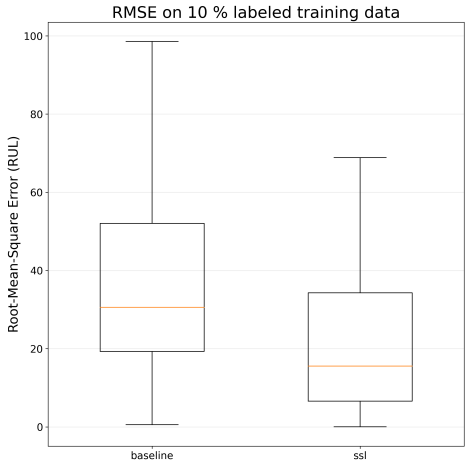

Figure 8. Root-mean-square errors for the baseline model and the model with SSL pretraining when 10 % labeling is used in training data (training data are 10 % full run-to-failure trajectories). We observe an improvement in RMSE if SSL pretraining is applied.

10 % labeling is shown in Figure 12. We again observe similar results, the SSL pretrained model is more accurate and does less overestimation of RUL. We note that the baseline model is not accurate in this scenario due to the low labeling proportion while the SSL pretrained model manages to retain adequate performance.

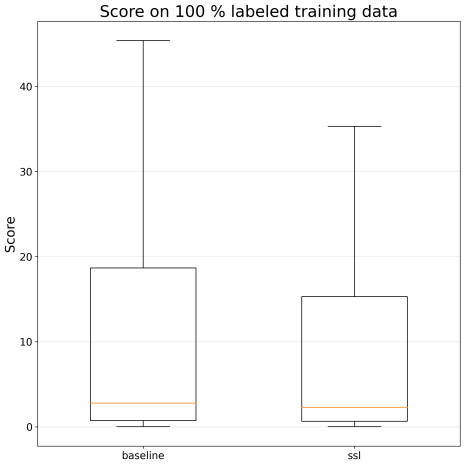

Figure 9. Scores (cf. Equation (6)) for the baseline model and the model with SSL pretraining when 100 % labeling is used in training data. We observe an improvement in score if SSL pretraining is applied.

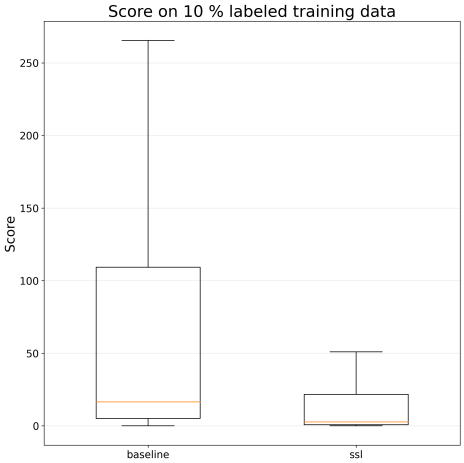

Figure 10. Scores for the baseline model and the model with SSL pretraining when 10 % labeling is used in training data. We observe an improvement in score if SSL pretraining is applied.

4. CONCLUSIONS

In this work we address the challenge that most applications and assets that would benefit from prognostics models lack sufficient run-to-failure trajectories, which in turn limits the applicability of supervised learning, the dominant machine learning approach in prognostics and health management. We propose using self-supervised learning to reduce the need of large volumes of run-to-failure data. Self-supervised learning is used as a pretraining step when training a deep neural network which is subsequently fine-tuned to the target task of remaining useful life estimation. The self-supervision task we have constructed is a sequence ordering scheme where the neural network is pretrained on the task of identifying if collections of the original data set is in correct order or not, a binary classification task. To conduct our experiments we use the CMAPSS data set which consists of simulated sensor measurements of run-to-failure aircraft engines.

We show that the feature extractor trained using self-supervised learning has learned representations of data that are useful for the downstream task of RUL prediction.

We indicate that self-supervised pretraining yields improved results over the baseline model which is trained using regular supervised learning. We see that the there is an improvement in accuracy in terms of root-mean-square-error and the improvement in scoring function, which penalizes remaining useful life overestimation, is distinct.

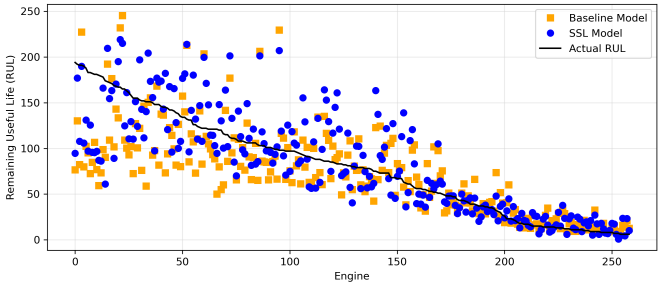

Figure 11. RUL predictions of the baseline model and the SSL pretrained model, when trained on 100 % labeled data. The true RUL is shown in black. The engines are sorted on the vertical axis, from high to low RUL. We observe that the pretrained model is more accurate in the high degradation regime and overestimates RUL less than the baseline model.

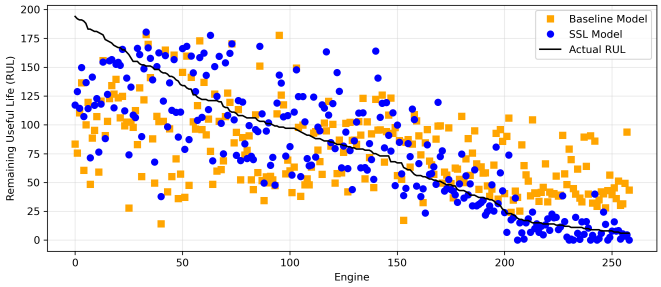

Figure 12. RUL predictions of the baseline model and the SSL pretrained model, when trained on 10 % labeled data. The true RUL is shown in black. We observe that the pretrained model is more accurate in the high degradation regime and overestimates RUL less than the baseline model.

We thus conclude that we have showed that self-supervised learning can be used to improve RUL predictions, in particular for data sets with a low proportion of full run-to-failure samples. We also conclude that the proposed sequence ordering self-supervision scheme can be used to learn representations of data that are useful for remaining useful life prediction.

Future endeavours should include evaluating this approach on other network architectures, other data sets, and comparing to other SSL pretraining tasks.

ACKNOWLEDGEMENTS

The authors would like to thank the iRel4.0 Horizon 2020 ECSEL Joint Undertaking project, which is financed by the European Commission and the work performed in this paper was also financed by the Swedish innovation agency Vinnova. The authors would also like to thank the KKS (Swedish Knowledge Foundation) which is financing the Industrial PhD school IndTech. In addition, the authors would like to thank the knowledge platform DIGIPROD which is a project funded by RISE.

REFERENCES

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., & Joulin, A. (2021). Emerging properties in self-supervised vision transformers. In *Proceedings of the international conference on computer vision (iccv)*.

Chicco, D. (2021). Siamese neural networks: An overview. In H. Cartwright (Ed.), *Artificial neural networks* (pp. 73–94). New York, NY: Springer US. Retrieved from https://doi.org/10.1007/978-1-0716-0826-5_3 doi: 10.1007/978-1-0716-0826-5_3

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *Bert: Pre-training of deep bidirectional transformers for language understanding.* arXiv. Retrieved from https://arxiv.org/abs/1810.04805 doi: 10.48550/ARXIV.1810.04805

Jing, L., & Tian, Y. (2021, nov). Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, *43*(11), 4037-4058. doi: 10.1109/TPAMI.2020.2992393

Kefalas, M., Santiago Rojo, J. d., Apostolidis, A., van den Herik, D., van Stein, B., & Bäck, T. (2022). Explainable artificial intelligence for exhaust gas temperature of turbofan engines. *Journal of Aerospace Information Systems*, *19*(6), 447-454. Retrieved from https://doi.org/10.2514/1.I011058 doi: 10.2514/1.I011058

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization..

Krokotsch, T., Knaak, M., & Gühmann, C. (2022). Improving semi-supervised learning for remaining useful lifetime estimation through self-supervision. *International Journal of Prognostics and Health Management*, *13*(1). doi: 10.36001/ijphm.2022.v13i1.3096

Listou Ellefsen, A., Bjørlykhaug, E., Æsøy, V., Ushakov, S., & Zhang, H. (2019). Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. *Reliability Engineering & System Safety*, *183*, 240-251. Retrieved from https://www.sciencedirect.com/science/article/pii/S0951832018307506 doi: https://doi.org/10.1016/j.ress.2018.11.027

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization..

McInnes, L., Healy, J., Saul, N., & Großberger, L. (2018). Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, *3*(29), 861. Retrieved from https://doi.org/10.21105/joss.00861 doi: 10.21105/joss.00861

Misra, I., Zitnick, C. L., & Hebert, M. (2016). Unsupervised learning using sequential verification for action recognition. *CoRR*, *abs/1603.08561*. Retrieved from http://arxiv.org/abs/1603.08561

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

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* (p. 1-9). doi: 10.1109/PHM.2008.4711414

Van Rossum, G., & Drake Jr, F. L. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Yoon, A. S., Lee, T., Lim, Y., Jung, D., Kang, P., Kim, D., ... Choi, Y. (2017). *Semi-supervised learning with deep generative models for asset failure prediction.* arXiv. Retrieved from https://arxiv.org/abs/1709.00845 doi: 10.48550/ARXIV.1709.00845

Zhou, M., Li, Z., & Xie, P. (2021, 07). Self-supervised Regularization for Text Classification. *Transactions of the Association for Computational Linguistics*, *9*, 641-656. Retrieved from https://doi.org/10.1162/tacl_a_00389 doi: 10.1162/tacl_a_00389