A Deep Learning First Approach to Remaining Useful Lifetime Prediction of Filtration System With Improved Response to Changing Operational Parameters Using Parameterized Fully-Connected Layer

Con Tran Vu[1], Ashok Chandra-Sekaran[2], and Wilhelm Stork[3]

[1,3] *Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany*
con.vu@kit.edu

[2] *Helmut Fischer GmbH, 71069 Sindelfingen, Germany*

Abstract

For the remaining useful lifetime prediction, apart from the normal sensor data which is updated regularly, there are also operational parameters, which do not change during a cycle of operation. Different sets of parameters result in essentially different, but relevant systems and thus require the adaptation from the statistical model for better prediction. We noticed that neural networks could easily overfit into one set of operational parameters and demonstrate constant bias in the prediction for other sets (underfitting). An aspect of major contribution from our work is the use of Parameterized Fully-Connected Layer (PFL). The PFL builds the parameter dependency right into each layer, in this way the parameters act as "meta-inputs" which adapt the model of neural network models to the different operating conditions. In another aspect of contribution, our work demonstrated that, instead of using feature engineering, convolutional layers could be used to automatically learn the features which are relevant for the prediction. In this way, the deep learning architecture could be reused for different problems or systems. We conduct experiments on the filtration system datasets provided by the Data Challenge 2020 and received results that compare favorably to the prize winners.

1. Introduction

Predictive Maintenance relies on the prediction about the health condition of a system in order to optimize the maintenance and inspection activities (Gouriveau, 2016). As maintenance tasks are conducted when needed, not only a saving of time and cost of services can be achieved, but unexpected downtime could be avoided (Mobley, 2002).

A common metric for the condition of a system is the Remaining Useful Lifetime (RUL), which shows the time remaining until system failure or defect (Vachtsevanos, 2006). A variety of approaches could be used for the prediction of the RUL. Model-based approaches rely on physical-mathematical models or statistical models of the degradation phenomenon (Walter & Flapper, 2017). Data-based approaches rely on Conditional Monitoring, in which sensors and other data were used as input into an algorithm that produced the output prediction (Li et al., 2014).

Deep learning is a sub-field of machine learning, in which deep neural networks are trained to learn from a large amount of data the representations or patterns useful for tasks such as classification or regression (LeCun, Bengio, & Hinton, 2015). Deep Learning demonstrated superior performance in many areas including image, video, and natural language processing. Deep learning networks were also applied for the RUL prediction with great success (Babu, Zhao, & Li, 2016).

In this paper, we presented a deep learning approach for RUL prediction based on Convolutional Neural Network with two architectural innovations. We proposed the novel Parameterized Fully-Connected Layer, with which the deep learning model can adapt the weights of neural networks to changing operational parameters. We also proposed the multi-head predictor architecture, with each head adapting to a different stage of the degradation process. We trained our model with the data provided by the Prognostics and Health Management Society of Europe (PHME) Data Challenge 2020. The model was evaluated according to the guideline provided by the challenge and on the test set which was kindly provided by the challenge organizers. We saw a major improvement in performance in comparison to the base model without the innovations, and to the state-of-the-art models by the challenge winners of 2020.

The paper was organized as follows. Section 2 introduces the PHME dataset. Section 3 presents our approach for data preparation, and more importantly, a detailed description of the network architecture and the PFL. Section 4 recapitulates the training and the experiments we did on the network, as well as presents and discusses the results. Finally, section 5 concludes the paper and present future outlook.

2. DATA

For this research, we used the data set provided by the PHME 2020 Data Challenge. The data set was collected from an experimental setup of a liquid filtration system. The filter gets clogged over time by the suspension particle in the liquid. This leads to a growing pressure drop across the filter.

A prognostic model should be constructed to predict the RUL of the system. In the PHME 2020 Dataset, the failure condition was defined as the first point in time when the pressure drop across the filter exceeds 20 psi.

2.1. Experiment Setup Description

The main components of the filtration system, in the order of the flow of fluid, are:

- Source tank
- A peristaltic pump
- A pulsation damper
- Upstream pressure sensor
- Filter - the component whose defect should be predicted
- Downstream pressure sensor
- A flowmeter sensor
- Sink tank

The two pressure sensors and the flowmeter produced three sensor data streams. Data were recorded at the rate of 10 Hz.

Throughout each run-to-failure experiment, one profile of suspension (i.e. the particle in liquid) was used, which corresponds to one operating condition of the filtration system. The suspension profile defined the two operational parameters:

- *Solid Ratio (%)*: possible values are 0.4, 0.425, 0.45 and 0.475.
- *Particle Size (μm)*: this parameter were given as a range. Possible values are 45-53, 53-63 and 63-75.

To simplify the calculation, *Mean particle size* was defined as an operational parameter instead of a range. Possible values for this parameter are 49, 58, and 69.

Table 1. Data samples in the Train and Validation Set.

Data Set	Sample number	Solid Ratio	Particle Size
Train Set	1, 2, 3, 4	0.4	45-53
	5, 6, 7, 8	0.425	45-53
	9, 10, 11, 12	0.45	45-53
	33, 34, 35, 36	0.4	63-75
	37, 38, 39, 40	0.425	63-75
	41, 42, 43, 44	0.45	63-75
Validation Set	13, 14, 15, 16	0.475	45-53
	45, 46, 47, 48	0.475	63-75

Table 2. Data samples in the Test Set.

Data Set	Sample number	Solid Ratio	Particle Size
Test Set	17, 18, 19, 20	0.4	53-63
	21, 22, 23, 24	0.425	53-63
	25, 26, 27, 28	0.45	53-63
	29, 30, 31, 32	0.475	53-63

2.2. Data Sets Description

The PHME 2020 Data Challenge provided the data set in multiple numbered .csv files. Each file contains the sensor data of one experiment. The whole collected data was divided into three sets. The Train Set was the first set to be published. The Validation Set was published later. According to the procedure of the Data Challenge, both sets can be used to train, validate and refine the models. The models were tested on a Test Set with different operating conditions than the other two. The operating conditions of each file of the Train and Validation Set were listed in Table 1 and of the Test Set in Table 2.

2.3. Evaluation Metric

Four prognostics models should be trained using 100%, 75%, 50% and 25% of data from the combined Train and Validation Set. Each model M_i (i.e. model trained on $i\%$ of data) was then evaluated with using the Mean Absolute Error (MAE) on:

- the whole Train and Validation Set (TV) yielding the *validation* score
- the Test Set (TE) yielding the *test* score

Those two scores (*validation* and *test*) will be used below (see Tables 6 and 7) to evaluate and compare the performance of different variations of neural network model architecture.

Moreover, those two scores of the four models (trained with different percentage of total data - see Table 4) will be used to calculate the *final* score. The final score is calculated as 1.5 times the total test score plus the total validation score.

$$\sum_{i \in 100,75,50,25\%} 1.5 \times MAE(M_i(TV)) + MAE(M_i(TE)) \tag{1}$$

As per the guidelines of Data Challenge 2020, this is used as the basis for comparing the proposed model and other state-of-the-art models in the literature (see Table 5).

3. Approach

The normal approach to construct a prediction system consists of two steps. First, the data scientist examined the data and used various methods to extract features such as mean, slope, and so on. Then the features deemed relevant for the prediction would be selected. A model was trained which takes those features as input to produce RUL prediction.

With our approach, we argued that feature extraction could be done automatically and the model could learn by itself which features were the most relevant. This is possible in the Convolutional Neural Network (CNN) architecture (Szegedy et al., 2014). In CNN, there are multiple convolution kernels. They act like filters with learnable parameters. In tandem with the kernels are the pooling layers, whose function is to keep only the most salient parts from the filtered data. In a normal CNN, there can be multiple kernels. The kernel count is often referred to as "channels" and represents the number of features extracted from the original data. When the model is trained, the kernel will learn the best parameter to extract the most relevant feature, all done automatically.

This approach brings several advantages. First, it may help uncover features in the data which were otherwise unrecognizable to human. Secondly, it also enables automatic machine learning in the sense that one type of model could be used on a class of similar prediction problems.

3.1. Data Preparation

First, the data should be imported into the program and transformed to the input form required by the model. The target (i.e. true) RUL should also be assigned to each input sample.

It was observed that in the first few seconds after the system started, the liquid flow had not yet been established (Łomowski & Hummel, n.d.). The pressure sensors and the flow sensor thus yielded non-sensible values. After that, the sensor readings instantly returned to the normal operational level. The data in these few seconds is not representative of the health state of the system (filter) and was easily eliminated by conventional methods. We calculated the sample-wise first derivative. The first data point after the point of highest derivative was marked as the start of useful data. As per the instruction by the Data Challenge, the first point when the pressure difference reached 20psi was marked as the end of useful data.

3.1.1. Labeling

As the data set only provides the sensor data and defines the sensor value-based condition for a failure, each data point must be labeled with the target (or true) RUL value. There are multiple approaches for RUL labeling (Ince, Sirkeci, & Genç, n.d.; Łomowski & Hummel, n.d.; Beirami et al., n.d.). We used the Piece-wise Linear Fault (Pwl_Fault) as the RUL assignment strategy (Ince et al., n.d.). It could be observed that the sensor values first changed slowly. After it approached and passed a "turning point", the sensor values decreased sharply (Łomowski & Hummel, n.d.). The Pwl_Fault strategy mimics this behavior. The RUL after the turning point is identical to the normal Linear RUL (i.e. the RUL decreases at the same rate as time flow). Before the turning point, the RUL decreases linearly from a known starting value. Further discussion and visualization for this RUL assignment strategy could be found in (Ince et al., n.d.).

The key part of this strategy is the identification of this turning point in data. It was done by fitting a line between the 500th and the 1000th sample of the pressure difference. The data point at which the sensor value deviated by more than 1psi from the fitted line was marked as the turning point.

3.1.2. De-noising

Data was de-noised to help the neural network model converge easier and to eliminate high-frequency noise. We denoised the flow rate, upstream pressure, and downstream pressure by convoluting them with a Hamming filter of window size 30.

The denoising process was different for the training phase and the testing phase. To yield the value for the current data point, the zero-centered Hamming convolution window extends into future values. During training, we allowed this so that the neural network model converges faster. For the validation and testing, we stopped the convolution when the edge of the filter reached the current (present-time) data point. The remaining data point was directly copied over from the original data stream. In this way, we made sure that the current prediction does not depend on future values.

3.1.3. Merging, Converting, Splitting

To create the input into the neural network model, we aggregated 50 data points into one input sample. Thus each input will be a tensor of the size 50x6 (current timestamp, up- and downstream pressure, flowmeter, solid ratio and mean particle size). The true training output was taken from the RUL of the last (i.e. present-time) data point.

The input samples were converted to PyTorch tensor. The whole data was shuffled and split into 80% training and 20%

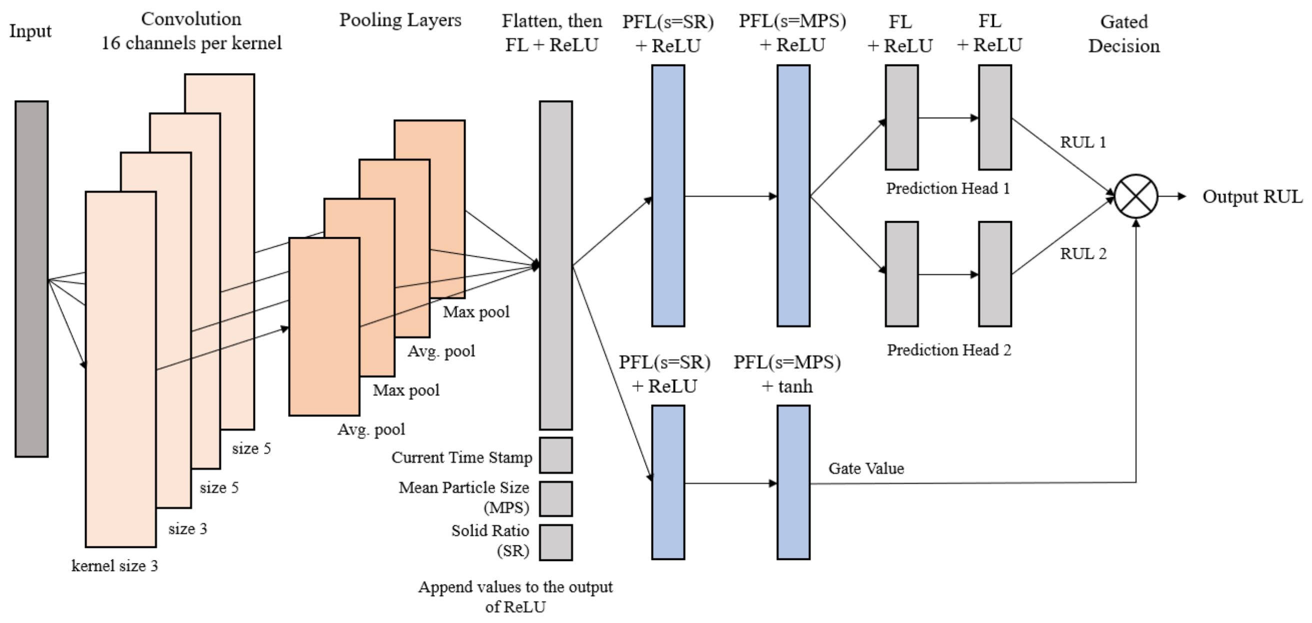

Figure 1. Architecture of the neural network model for RUL prediction. FLs are fully-connected layers. PFLs are parameterized fully-connected layer with weights and bias adapted to the parameter s.

validation set.

3.2. Setup Of Neural Network Model

The model for neural network can be described as composing of three stages. The first stage consists of four arrays of convolution kernels and pooling layers. Each array has `cnn_channels` number of channels. Two kernel sizes of 3 and 5 were used. Two types of pooling layers were applied: the `MaxPool` and `AvgPool`, which takes the max and average value from the output of the kernels.

The second stage consists of a series of FL and PFL and serves to aggregate and preprocess the data. A flatten layer then combines all the outputs of the pooling layers into one vector. This vector, which can have thousands of values were then condensed again using a fully-connected layer into a much shorter vector of size `D_aftercnn`, which represents all the information extracted from the sensor. The result is then appended with information necessary for prediction, namely the time stamp, the solid ratio, and the mean particle size. This vector is the input into two consecutive PFL layers. Each of them also has an additional input for the operational parameters, which adapts the weights and biases of the layer according to the parameter value.

The third stage are two parallel neural networks, each composed from `hidden_count` fully connected layers of size `D_hidden` and drop-out layers with probability `dropout`. Each of them gives an estimation of the RUL and thus can be considered as a prediction head. A third sub-neural network from PFLs acts as a gate, deciding which output value of the heads to be ”selected” as the final result.

3.2.1. Parameterized Fully-connected Layer

A simple Fully-connected layer can be represented by the equation (Paszke et al., 2019):

$$y = xW^T + b \tag{2}$$

where x is the input activation, y is the output, W is a matrix of weights and b is a vector of bias values. One can consider that W and b together is ”the model” which will be ”trained” to become corresponding to the physical system.

A special feature of the PHME 20 Dataset is the availability of system-wide parameters such as solid ratio or mean particle size. They are constant in each run-to-failure experiment, but changed between experiments. Not only are those parameters a part of the input data stream, but they also essentially change the system, resulting in a completely different, but related system. If the neural network for prediction would like to model the system exactly, it would have to have a new set of weights for every different set of operational parameters. This is intractable as the parameters are continuous valued, which would require an infinite number of neural networks.

To deal with this problem, we ”programmed” the network so that the weights can changed itself adapting to the different operational parameters. In effect, the resulting model is not a neural network, but is a prototype for a family of neural networks, whose weights are related by a rule. From the programming point of view, the weight matrix and bias vector should be a function of the operational parameters s.

$$\begin{aligned} W &= W(s) \\ b &= b(s) \end{aligned} \tag{3}$$

In this data set, the operational parameters s are the solid ratio and the mean particle size. The function form of $W(s)$ and $b(s)$ can be any type (polynomial, exponential etc.) which is suitable for the problem. In this paper, we model this dependency using simple linear function:

$$\begin{aligned} W &= W_{center} + sW_{slope} \\ b &= b_{center} + sb_{slope} \end{aligned} \tag{4}$$

where W_{center}, W_{slope}, b_{center} and b_{slope} are learnable parameters so that the model could create a brand new neural networks depending on the value of the operational parameter s.

Thus instead of having an unlimited number of weight sets for all the operational parameters, what the model learned is actually a ”rule” for calculating the neural network weights. One key consideration here is the complexity of the rule. A simple linear PFL with one operational parameter has double the number of learnable parameters as compared to an FL. The amount of learnable parameters increase by another-fold for each additional operational parameter and for an additional order of the polynomial function. A large size model will negatively impact the training time and resources, and will not likely achieve the best fit for the data.

In this data set, the two operational parameters are the solid ratio and the mean particle size. In our model, we modeled their influence by using two consecutive PFL, each with a single operational parameter. This setup allowed us to train the model in a reasonable amount of time and allow a good fit to data, while still be able to model complex non-linear interactions between sensor data and those parameters.

3.2.2. Multi Prediction Heads Architecture

Our neural network model with two prediction heads is a modification of the Multi-task architecture (Caruana, 1997). In this kind of network, each prediction head is specialized in one task, such as predicting the failure probability in one failure mode.

Upon examination of the data set, it can be observed that the sensor readings have different behaviors during the two different stages of the lifetime. The beginning stage takes up a major duration of the lifetime. In this stage, the sensor data changes very slowly with small dynamics. In the second stage, the sensor data changes very quickly leading to final malfunction.

In our model, each of the two prediction heads is responsible for the RUL prediction of each stage. In another word, one head gives the RUL prediction for the slow-degradation stage and the other predicts the RUL of the fast-degradation stage. The final RUL is decided by a gate function.

$$\begin{aligned} RUL_{final} = \sigma((gate - threshold_{gate}) \times \\ threshold_{\sigma}) \times (RUL_1 - RUL_2) + RUL_2 \end{aligned} \tag{5}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, RUL_1 and RUL_2 are the predictions of the two heads and $threshold_{gate}$ is a learnable parameter. $threshold_{\sigma}$ is a hyper-parameter that determines the width of the sigmoid shape. Most importantly, the $gate$ value controls the gate function so that it picks RUL_1 or RUL_2 or some value in between as the final output RUL. This $gate$ value is produced by a separate sub-network consisting of two PFL layers.

The sigmoid function is essentially a ”softened” step function. This function was used for two reasons. First, the function is smoothly differentiable, allowing back-propagation to work. The model (including the gate function, the sub-network calculating $gate$ value, and both prediction heads) was trained in one go and only the final output value was used for the loss function. As a result the head-selecting mechanism must allow back-propagation backwards to the weights and the normal `if..else` for selecting between heads simply won't work. Secondly, in comparison to Heaviside step function as an alternate, the sigmoid allows a smooth transition of RUL prediction from the slow- to the fast-degradation stage. Otherwise, the predicted RUL in this transition stage would be very noisy.

4. EXPERIMENTS AND RESULTS

A number of experiments have been carried out to examine the performance of the architecture. The first experiment followed the PHME2020 Data Challenge procedure and yielded the score for comparison with other state-of-the-art prediction models. In the second experiment, variations of the standard architecture were evaluated and compared to examine the relative performance gain resulting from each architectural innovation. The third experiment examined closely the performance gain due to the PFL, which gave an intuition in understanding its advantage.

4.1. Model Training and Overall Benchmark

Four models were trained from 25%, 50%, 75%, and 100% of the train and validation data set. The prototype for the

Table 3. Hyper-parameters used for the training of the neural network model

Hyper-parameter	Value	Description
learning rate	0.000375	Starting value for learning rate scheduler
hidden_count	2	Number of hidden layers per prediction head
*D_hidden*n	20	Number of nodes per hidden layer in each prediction head
dropout	0.002	Probability of dropout in each Dropout layer
D_aftercnn	18	Width of the condensed vector after the pooling layer
cnn_channels	16	Number of CNN channels per CNN array
$threshold_\sigma$	10	Width of the sigmoid function

models was constructed using the PyTorch library in Python. The data for training and validation was selected according to heuristics so that every operating condition was equally represented.

Two evaluation metrics were calculated for each model. The ”validation” score is the MAE when the models were tested against 100% of the training and validation set. The ”test” score is the performance of the models with the test set. The ”validation” score was always better because the model has ”seen” at least a part of the data it was scored against. In contrast, each model ”saw” the test set only once during the final evaluation.

The models were trained with a custom training loop with automatic learning rate scheduling for better convergence. In each training epoch, the root-mean-square error was used for training so that the model will converge faster. At the end of each epoch, validation was done using MAE. Xavier initialization was used for the weights and the biases were set to zero (Glorot & Bengio, 2010). For training, we used the Adam optimizer (Kingma & Ba, 2014). The hyper-parameters required for training of the model are listed in Table 3

The starting value in our RUL assignment scheme (PwL_Fault) has a considerable impact on the prediction performance. A higher RUL start value means the RUL curve approach that of linearly decreasing RUL, and gives useful information earlier to the user regarding the actual time point of defect. On the other hand, a lower RUL value means that in the first stage, RUL decreases slowly (slower than the flow of time), and then in the second stage it decreases with the flow of time approaching the actual defect. In the second case, the actual time point of defect could still be identified albeit a little bit later in the system lifetime. In our experiment, we trained two sets of models corresponding to the starting RUL value of 150 and 100 seconds.

Table 4. Results of the model when trained with different percentage of train and validation set.

Models		RUL Start = 150	RUL Start = 100
25%	validation	1.660	1.169
	test	3.925	2.408
50%	validation	1.264	0.956
	test	3.708	2.394
75%	validation	0.974	1.080
	test	3.717	2.131
100%	validation	1.124	0.863
	test	3.153	2.491
Final score		**26.777**	**18.204**

Table 5. Comprison with other state-of-the-art algorithms. The scores are from the prize winners of the PHM Data Challenge 2020.

Algorithm	Score
Our Neural Network Model RUL Start = 100	**18.204**
Our Neural Network Model RUL Start = 150	**26.777**
Kernel Regression (Łomowski & Hummel, n.d.)	49.67
Neural Network (Beirami et al., n.d.)	57.24
Gradient Boosting (Ince et al., n.d.)	86.74

The results and the final score were given in the table 4. The performance of the model is better with a lower starting RUL value. It is easily explainable given the discussion so far. Lower starting RUL means the RUL value decreases slowly in the beginning and then faster later. The sensor data also show similar behavior: they changed relatively little in the first stage and then evolved very quickly towards the defect time. The similarity means the model can achieve better fitting without going too much into the highly nonlinear mapping region, which improved both the validation and test score.

Regardless of the starting RUL value, the model architecture was able to demonstrate stellar performance. Table 5 shows the comparison with other approaches of the prize winners of the Data Challenge. All the models here was trained with the same training set and tested with the same test set provided by and following the rule of the PHM Data Challenge 2020. The other authors do the feature extraction and selection manually, before feeding those features into a prediction model. With our approach, we simply feed the (denoised) sensor data stream to the model and rely on CNN kernels to

Table 6. Results of Model Variations when trained with 100% of Train and Validation Set

RUL Start = 150, 100% data		With PFL	Without PFL
2 prediction heads	validation	1.124	2.352
	test	**3.153**	3.184
1 prediction head	validation	**0.925**	2.471
	test	3.873	4.021

RUL Start = 100, 100% data		With PFL	Without PFL
2 prediction heads	validation	**0.863**	1.422
	test	**2.491**	2.847
1 prediction head	validation	0.959	1.603
	test	3.009	3.444

Table 7. Results of model variations when trained with 25% of train and validation set. For the validation score, the model was benchmarked against 100% of train+validation set.

RUL Start = 150, 25% data		With PFL	Without PFL
2 prediction heads	validation	1.660	3.556
	test	**3.925**	4.579
1 prediction head	validation	**1.374**	3.219
	test	4.124	5.062

RUL Start = 100, 25% data		With PFL	Without PFL
2 prediction heads	validation	**1.169**	1.976
	test	**2.408**	2.718
1 prediction head	validation	1.211	2.133
	test	3.320	3.879

learn to extract the relevant features. As would be shown in the next subsection, even without PFL and two-head architecture, the results are already very good. Nevertheless, those two innovations are the origin of the performance gain.

4.2. Evaluation of Variations of Model Architecture

To examine the specific performance gain contributed to the architectural innovations, we trained additional models with and without the architectural changes. For the first innovation with PFL, we created models in which the PFLs were replaced by the normal fully-connected layer. For the second innovation, we create models with only one prediction head instead of two and consequently eliminated the gate subnetwork. In combination, it gave rise to four architecture variations.

We trained the models exactly as earlier, with the same hyperparameters. And similar to the first experiment, we also considered the two cases of starting RUL values of 150 and 100 seconds. Regarding the amount of training data, we also experimented with the case when only 25% of the whole data was provided, so that we can have an evaluation for the adaptability of the models.

The results for the case of 100% data were given in table 6 and for the case of 25% of data in table 7. From the results, three main aspects could be pointed out.

First, the variations with PFL always performed better than their normal counterpart. We would examine this improvement in closer detail in the next sub-section.

Secondly, the normal "baseline" variation (i.e. without PFL, one prediction head) performs on a similar level as the variations with innovations, albeit being the one with the worst performance. This demonstrated the learning power of the neural network and the ability of convolutional kernels to learn to extract the necessary features without explicit human intervention.

Thirdly, for the case of RUL start value of 150 and with PFL, the variation with one prediction head performed better in the validation than the two-head counterpart. With the test set, the variation with full architecture still performed better. One possible explanation could be given to this unexpected result. With a high RUL start value, the model must conduct mapping from slow-changing sensor data to fast-decreasing RUL in the first stage during the system's lifetime and from fast-changing data to fast-decreasing RUL in the second stage. This operation is highly non-linear. A model with two prediction heads and a gate would strive towards linear mapping in each stage using a separate prediction head and then blending with sigmoid function. Its ability to fit non-linear function could then be worse than that in which the non-linearity is built into the weights of the one prediction head. However, this non-linearity also has downsides. A highly non-linear model tends to overfit and perform worse with unforeseen data.

4.3. Specific Performance Improvement From Parameterized Fully-connected Layer

We conducted a close-up examination on the performance improvement provided by the PFL. Two model variations with and without PFL were compared. Both of them have two prediction heads, were trained with 100% of data whose RUL label starting from the value of 150.

For the experiment, we used four sets of input data with varying solid ratios of 0.4, 0.425, 0.45, and 0.475. The other parameter, namely mean particle size, was kept the same in the four sets. The error terms (not the absolute value) of every single prediction were recorded. We then calculated the mean and standard deviation of the error terms for each of the four input sets.

Results for the non-PFL model were shown in Figure 2. Two observations could be made here. First, there is a constant

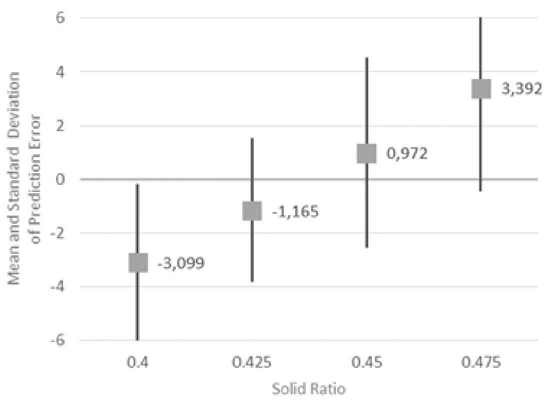

Figure 2. Prediction error statistics in model without PFL. Solid ratio is the changing operational parameter.

bias in the error term. The distribution of error is not zero-centered for the solid ratios of 0.4 and 0.475. This means the network was not able to fit at these parameter values (i.e. underfit). Secondly, the means of the distribution varies in a quasi-linear manner with changing solid ratio. This suggested that if the model itself could adapt linearly to this parameter, then the error terms would be ”canceled”.

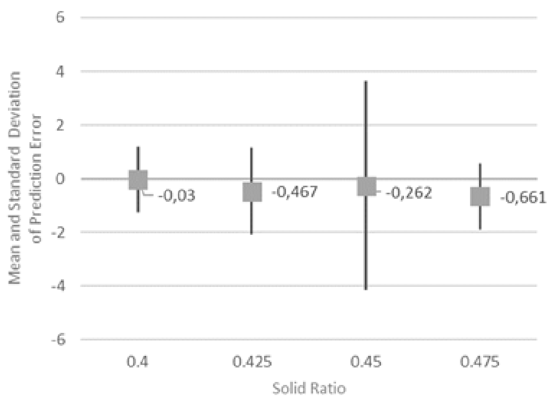

Figure 3. Prediction error statistics in network model with PFL with changing solid ratio.

Results for the model architecture with PFL were shown in Figure 3. As the model create essentially a different neural network for each ”solid ratio”, a better adaptation and thus clear improvement could be seen. The error distribution is now zero-centered. Furthermore, except for the solid ratio of 0.45, the standard deviation of error is also greatly reduced. Without inherent error bias, the model managed to fit better and thus the MAE was reduced.

It should be reminded that the model has never seen the data with solid ratio of 0.45 but still perform very well. This illustrates a huge advantage of PFL regarding unseen test data. Due to the fact that not the exact weights but the rule for calculating weights are learn, the model could calculate a whole new network, adapting to a completely new set of operating conditions. In another words, the trained model could in effect ”extrapolate” the neural network.

5. Conclusion and Outlook

We presented a deep-learning first approach for the RUL prediction of the filtration system. Instead of manual feature engineering, automatic feature extraction was deployed using the convolutional neural network. The resulting ”baseline” deep network already performed very well. Additional performance gain was possible with two architectural innovations. The Parameterized Fully-connected Layer gave the network the possibility to adapt itself to the operational parameter and resulting in a performance boost in both validation and test. The architecture with two prediction heads and a blending gate helped the model to adapt to different stages of system lifetime and improved model linearity. The final model with all improvements shown stellar performance in validation and testing in comparison with state-of-the-art models.

In order to achieve automatic machine learning, all the stages of the model building pipeline should be automated. Our approach has shown that the feature extraction stage could be replaced by automatic feature learning from the CNN. Another approach for this could be the use of unsupervised learning. The representations learned by unsupervised techniques (autoencoder, deep belief network etc.) could be used as the input into a supervised network and trained to output the required e.g. RUL value. However, work still has to be done to automate the data preparation and labeling, which still depends heavily on human intervention. Some solutions are already available for text, image, and tabular data which could be used as starting point for stream type data (Howard & Gugger, 2020). Only then can an automatic end-to-end prediction pipeline be realizable.

References

Babu, G. S., Zhao, P., & Li, X.-L. (2016). Deep convolutional neural network based regression approach for estimation of remaining useful life. In *Database systems for advanced applications* (pp. 214–228). Springer International Publishing. doi: 10.1007/978-3-319-32025-0_14

Beirami, H., Calzà, D., Cimatti, A., Islam, M., Roveri, M., & Svaizer, P. (n.d.). A data-driven approach for rul prediction of an experimental filtration system. *PHM Society European Conference*, *5*(1). doi: 10.36001/phme.2020.v5i1.1318

Caruana, R. (1997). Multi-task learning. *Machine Learning*, *28*(1), 41–75. doi: 10.1023/a:1007379606734

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & D. M. Titterington (Eds.), *Aistats* (Vol. 9, p. 249-256). JMLR.org.

Gouriveau, R. (2016). *From prognostics and health systems management to predictive maintenance 1 : monitoring and prognostics*. Hoboken, NJ: Wiley.

Howard, J., & Gugger, S. (2020, February). Fastai: A layered API for deep learning. *Information*, *11*(2), 108. doi: 10.3390/info11020108

Ince, K., Sirkeci, E., & Genç, Y. (n.d.). Remaining useful life prediction for experimental filtration system: A data challenge. *PHM Society European Conference*, *5*(1). doi: 10.36001/phme.2020.v5i1.1317

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization.*

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017, May). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. doi: 10.1145/3065386

LeCun, Y., Bengio, Y., & Hinton, G. (2015, May). Deep learning. *Nature*, *521*(7553), 436–444. doi: 10.1038/nature14539

Li, H., Parikh, D., He, Q., Qian, B., Li, Z., Fang, D., & Hampapur, A. (2014, August). Improving rail network velocity: A machine learning approach to predictive maintenance. *Transportation Research Part C: Emerging Technologies*, *45*, 17–26. doi: 10.1016/j.trc.2014.04.013

Mobley, R. (2002). *An introduction to predictive maintenance*. Amsterdam New York: Butterworth-Heinemann.

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 *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2014). *Going deeper with convolutions.*

Vachtsevanos, G. (2006). *Intelligent fault diagnosis and prognosis for engineering systems*. Hoboken, N.J: Wiley.

Walter, G., & Flapper, S. D. (2017, December). Condition-based maintenance for complex systems based on current component status and bayesian updating of component reliability. *Reliability Engineering & System Safety*, *168*, 227–239. doi: 10.1016/j.ress.2017.06.015

Łomowski, R., & Hummel, S. (n.d.). A method to estimate the remaining useful life of a filter using a hybrid approach based on kernel regression and simple statistics. *PHM Society European Conference*, *5*(1). doi: 10.36001/phme.2020.v5i1.1316