Stay Tuned! Analysing Hyperparameters of a Wide-Kernel Architecture for Industrial Faults

Dan Hudson
Semantic Information Systems Group (SIS)
Osnabrück University
Osnabrück, Germany
daniel.hudson@uos.de

Jurgen van den Hoogen
Semantic Information Systems Group (SIS)
Osnabrück University
Osnabrück, Germany
jurgen.vandenhoogen@uos.de

Stefan Bloemheuvel
Tilburg University, CSAI
Jheronimus Academy of Data Science (JADS)
Tilburg, The Netherlands
s.d.bloemheuvel@jads.nl

Martin Atzmueller
Osnabrück University / SIS
German Research Center for Artificial Intelligence (DFKI)
Osnabrück, Germany
martin.atzmueller@uos.de

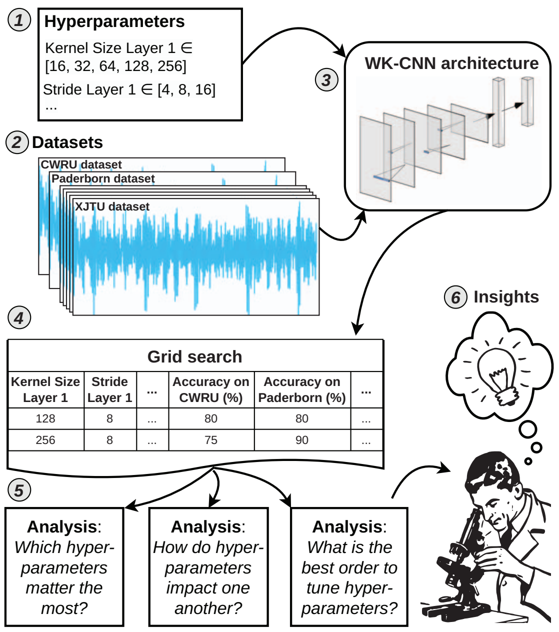
***Abstract*—The performance of a deep learning model depends heavily on its architectural hyperparameters. However, there is often little guidance on how to tune those hyperparameters. This paper provides insights into how to tune the architectural hyperparameters of a wide-kernel convolutional model for industrial fault detection, by analysing a grid search over 12,960 possible combinations of hyperparameter settings on seven benchmark datasets of vibration time series. By aggregating the results on these seven datasets, we are able to generalise across multiple industrial fault detection settings. We find that, generally speaking, the number of filters in the later convolutional layers and the hyperparameters associated with the first layer are the most important. Additionally, we analyse the relationships between hyperparameters and develop this analysis into a 'recommended sequence' for how to tune them one-at-a-time.**

***Index Terms*—deep learning, convolutional neural networks, hyperparameter analysis, time series classification, industrial fault detection, bearing fault detection**

I. Introduction

Deep learning models have succeeded at many time series analysis problems within industry, such as monitoring vibrations to see when a machine is likely to break down [1]–[3]. However, it is not easy to tell beforehand which hyperparameter settings are necessary for a model to be successful in a given use-case. As a result, users can be left frustrated because their deep learning model is not working and they do not know why, or what to try next. To tackle this issue, we present an extensive exploration of how to tune the hyperparameters of the wide-kernel convolutional neural network (WK-CNN) derived from [2], [4], [5] for industrial fault detection. The relatively few hyperparameters and state-of-the-art performance of this architecture make it an ideal choice for this study.

We demonstrate an analysis method to identify important hyperparameters, their pairwise interactions, and preferred choices for tuning hyperparameters sequentially. Specifically, a grid search over 12,960 hyperparameter combinations tells us which settings perform well and which ones perform poorly. This paper analyses the results after applying our grid search process to seven benchmark datasets in the field of fault detection, which extends the coverage of our previous analysis executed in [5]. We use the results to reach insights regarding how to effectively tune a wide-kernel architecture, when applied to vibration data from new industrial sources, e. g., with different machinery and sensor setup. Figure 1 depicts the full analysis workflow. By answering the questions posed in step 5, we make three contributions: (1) We identify which hyperparameters are most important for WK-CNN accuracy generally. (2) We analyse pairwise interactions between hyperparameters. (3) Based on empirical investigations, we concretely recommend an order in which to tune hyperparameters sequentially.


1
Hyperparameters
Kernel Size Layer 1 ∈ [16, 32, 64, 128, 256]
Stride Layer 1 ∈ [4, 8, 16]
...
2
Datasets
CWRU dataset
Paderborn dataset
XJTU dataset
3
WK-CNN architecture
4
Grid search
5
Analysis: Which hyper-parameters matter the most?
Analysis: How do hyper-parameters impact one another?
Analysis: What is the best order to tune hyper-parameters?
6
Insights

Kernel Size Layer 1	Stride Layer 1	...	Accuracy on CWRU (%)	Accuracy on Paderborn (%)	...
128	8	...	80	80	...
256	8	...	75	90	...

Fig. 1: Analysis workflow.

The rest of the paper is organised as follows: Section II discusses related work. Next, Section III introduces the applied wide-kernel architecture, hyperparameter search setup, and the used datasets. Section IV presents our methodology before discussing the results in Section V. Finally, Section VI concludes with a summary and interesting options for future work.

II. Related Work

Fault detection in rotating industrial equipment is crucial for preventing breakdowns [6], and has changed drastically with the rise of the Industrial Internet of Things (IIoT) combined with data-driven analysis techniques [7], [8]. These methods do not require in-depth knowledge about the technical components of industrial machinery, enabling automated processing and automated adaptation to changing operational environments.

In typical industrial contexts, time series data is gathered from sensors measuring vibrations within the machinery under different operating conditions. However, traditional Machine Learning (ML) methods like K-NN [9], [10], Random Forest [11] and SVM [12], [13] are not able to process the raw time series data directly, therefore requiring extensive feature extraction. These obstructions in fault detection led to the development of automated learning techniques, i.e., Deep Learning specifically for time series.

Initially, a multilayer perceptron (MLP) was a common approach for this task [14]. However, its limited depth due to computational constraints led to the adoption of recurrent neural networks (RNNs), which are effective for time series data because of their ability to model temporal dependencies [15], [16]. However, due to the high sampling frequencies of vibration sensors, these RNNs were still computationally too costly. Therefore, the adoption of one-dimensional convolutional neural networks (CNNs), that are able to process time series effectively became more popular [7], [8], [17]–[19], showing state-of-the-art performance in fault detection. Especially utilising a wide-kernel in the first convolutional layer seemed to yield increased performance [1], [2], [4], [5].

In this paper, we investigate the influence of architectural hyperparameters on model performance, focusing on a wide-kernel CNN model known for its effectiveness in fault detection [1], [2], [4]. Through an extensive grid search we explore in total 12,960 model combinations on seven well-known benchmark datasets representing various fault detection tasks, also for supporting computational sensemaking [20] and explanation [21] onto modeling decisions.

Optimising CNN architectures is an established research area. For example, [22] examined the impact of batch sizes in CNNs, while [23] developed efficient hyperparameter optimisation methods for image classification, and [24] refined hyperparameters, for applications like COVID chest x-ray classification. Furthermore, [25] evaluated optimisation algorithms for sensor-based activity recognition using 1D CNNs, although with different model depths and objectives compared to this study. Additionally, the use of Bayesian optimisation in LSTM-RNN models for traffic prediction was explored in [26]. However, this optimisation algorithm assumes that the hyperparameters can be tuned in a continuous way, making it inapplicable to the integer values used in this work, such as, e.g., the number of filters in a layer. Therefore, this work focuses on an extensive grid search to identify the most effective hyperparameter combinations for its specific neural network model.

III. Background

This section describes the WK-CNN architecture, the settings used in the hyperparameter grid search, and the seven datasets.

A. Wide-Kernel Architecture

The wide-kernel architecture described in [1], [2], [4], [5] excels at detecting industrial faults from vibration data. It is relatively lightweight, consisting of an initial ‘wide’ convolutional layer with a large kernel length, followed by four more convolutional layers with shorter kernels. Layers include the ReLU activation, batch normalisation, dropout of 0.15, and average pooling to decrease the output from length T to $\frac{T}{2}$. The output from these convolutional layers is then flattened and routed into a fully-connected layer of 100 ReLU units, which in turn feeds into a softmax layer that classifies the vibration signal.

An important question is what kernel size, stride and number of filters are optimal across the different convolutional layers. We consider the *kernel size* and *number of filters* in layer 1, layer 2 and layers 3-5, and also the *stride* in layer 1. Taken together, this offers 7 different architectural hyperparameters which can be tuned in order to optimise the performance of the wide-kernel convolutional neural network (WK-CNN).

A grid search over combinations of values allows us to examine the relationship between these 7 hyperparameters and WK-CNN performance. Full details of the WK-CNN architecture and the experimental procedure are provided in Section 3.1 and 3.2 of [5]. Below, we extend on that related previous work by including four extra benchmark datasets, and by applying new follow-on analysis methods to extract insights from the results.

B. Datasets

Compared to our previous work described in [5], we consider the three datasets used there and also add four new datasets to investigate the hyperparameters when generalising across more kinds of vibration recordings. In all, the datasets we use are as follows.

CWRU bearing dataset [27], [28]: Vibration data from damaged bearings, sampled at 12 kHz and segmented into 2048-timestep sequences. The data is taken from the fan end experiment, and aggregated across the 1797 and 1750 RPM operating speeds, commonly used for fault detection and viewed as a benchmark due to its similarity to real-world applications. In total, there are 1,355 sequences across 13 nearly-balanced classes, which are used with a 20-80% train-test split.

Gearbox dataset [29]: Represents lab recordings of vibrations from an industrial gearbox, either in a healthy condition or with a broken tooth. We aggregated across operating loads, and divided into 978 samples of 2048 points, with a 20-80% train-test split.

Paderborn dataset [30]: From the Paderborn experiments we took the vibration data for 8 'real damaged' inner race faults, following [30]. The raw data was sampled at 64 kHz and we extracted 1,000 randomly sampled sequences of length 2048 for each condition, with a 20-80% train-test split.

Society for Machinery Failure Prevention Technology (MFPT) dataset [31]: We extracted 7 outer race and 7 inner race faults under different load conditions and a healthy baseline condition, resulting in 15 unique classes. Sequences of 2048 points were created from one of the vibration sensors.

The University of Conneticut (UoC) gear fault dataset [32]: Data recorded by one vibration sensor at 20 kHz. Includes 9 conditions: healthy, missing tooth, root crack, spalling, and chipping tip with 5 different levels of severity. The dataset is balanced across every condition, with 182 sequences per condition, resulting in a fairly small dataset.

The Southeast University (SEU) dataset [33]: we extracted the recordings from x, y and z-axis vibration sensors placed on a planetary gearbox, with 5 different types of damage. We used the full recordings, including the initial start-up period at the beginning. In total, there are 5110 sequences of 2048 data points, meaning that there are 1022 sequences per class since the data is balanced.

XJTU dataset [34]: We extracted recordings of 15 bearings obtained from accelerated degradation. Two variables sampled at 25.6 kHz, converted into 480 sequences of 2048 timesteps per bearing fault condition. The data is derived from the last 30 minutes of the run-to-failure experiments, during which the fault has developed.

IV. Method

This section describes the multi-part analysis methodology we use to study the grid search results. In contrast to previous research [5], we specifically focus on generalising the results across all seven use-cases instead of investigating the datasets individually.

A. Multiple Regression and Feature Importance

Our grid search covers 12,960 different combinations of 7 hyperparameters. For each one, we train and evaluate a WK-CNN on 7 different benchmark datasets, leading to 7 accuracy scores. This data is therefore suitable for performing a follow-on multiple regression, where the goal is to predict the accuracy on each benchmark dataset from the 7 hyperparameters that were used for the model. We pursue this idea by trying out several common ML regression algorithms. A least-squares linear regression, a random forest, a k-nearest neighbours regressor (k-NN) and a multilayer perceptron (MLP) – containing 3 hidden layers of 32 units with ReLU activation – are all trained to predict the 7 accuracy scores from the WK-CNN hyperparameters. We also include a baseline which involves simply predicting the mean value every time. To evaluate the predictions of each approach, we use the R-squared, mean absolute error (MAE) and mean absolute percentage error (MAPE) evaluation metrics.

As a final step in this piece of the analysis, we also take a 'deep dive' on the MLP regressor, using Shapley values as in related work [5], to get feature importance scores for the 7 input variables, i.e., the 7 hyperparameters. We gain overall insights about the relative importance of each hyperparameter by visualising how much they contribute to reducing the MAE of the MLP regressor.

B. Variability According to Hyperparameter

One key question is how much the WK-CNN performance changes in response to a hyperparameter. To get an idea of this, we can look at the different distributions of accuracy scores for each hyperparameter setting. For example, when the kernel size in layer 1 is set to 32, there is a distribution of scores obtained by testing all combinations of the other hyperparameters. We have such a distribution for each value of a hyperparameter (e.g. a distribution for each kernel size).

To summarise this information, we look at the d-dimensional "earth mover's distance" (also known as the d-dimensional "Wasserstein distance") for one hyperparameter at a time . The earth mover's distance is a way to compare probability distributions to one another based intuitively on how much one distribution has to be moved or shifted in order to become the other [35]. The more dissimilar they are, the more moving is required, and the earth mover's distance is higher. The d-dimensional version generalises the method to work with more than 2 distributions [36]. With this method, we can assign a single number to each hyperparameter which gives a simplified summary of how much the hyperparameter affects the accuracy scores. Before calculating the earth mover's distances, we normalise the accuracy scores by quantilising, for each dataset, so that accuracy scores are comparable. We calculate d-dimensional earth mover scores on each dataset separately, leading to a table containing a number for each pair of hyperparameter and dataset.

An additional note is that the number of values that a hyperparameter can have in our experiments is not consistent, so, for example, we consider five different kernel sizes in the first layer but only three different strides. This makes it more difficult to directly compare the earth mover's distances of different hyperparameters, however it does not affect comparisons of the same hyperparameter across datasets. Nevertheless, to additionally consider the impact of some hyperparameters being tested with more settings than others, we provide a heatmap illustrating each earth mover's distance divided by the number of settings tested for that hyperparameter, to see if the general trends across hyperparameters remain unaffected.

C. *Influence of Hyperparameters on One Another*

Modifying the value of one hyperparameter might change what value is optimal for another hyperparameter. When this occurs, we can say that the first hyperparameter has influenced the optimal value of the second hyperparameter. We explore this idea by looking at how likely changing a hyperparameter is to affect the optimal setting of another, expressed as a probability. To calculate the influence of hyperparameter A on B, we apply the procedure shown in Algorithm 1 to each dataset, and then average across datasets.

Algorithm 1 Compute the influence of A on B

```
trialCount ← 0
differenceCount ← 0
for each configuration $\mathbf{c} \in (\mathbf{c_1}, \mathbf{c_2}, ..., \mathbf{c_{12960}})$ do
  $\mathbf{c}[B]$ ← optimise($\mathbf{c}$, $B$)
  for value $v \in$ options($A$) do
    $\mathbf{c}' \leftarrow \mathbf{c}$
    $\mathbf{c}'[A] \leftarrow v$
    $\mathbf{c}'[B]$ ← optimise($\mathbf{c}'$, $B$)
    trialCount ← trialCount $+1$
    if $\mathbf{c} \neq \mathbf{c}'$ then
      differenceCount ← differenceCount $+1$
    end if
  end for
end for each
return differenceCount / trialCount
```

After the scores have been calculated for each pair of hyperparameters, the results can be visualised. We visualise the results as a weighted, directed network where the nodes are hyperparameters and the edges reflect the influence one hyperparameter has on another. This provides a way to quickly evaluate how influential and how easily influenced each hyperparameter is.

This information naturally suggests a strategy for tuning the hyperparameters separately, whereby we try to minimise the impact that each hyperparameter has on previous hyperparameters. In other words, we wish to avoid the situation in which tuning the last hyperparameter means that previous hyperparameters need to be re-tuned. This approach is appealing because it provides an alternative to a full grid search, which is very computationally expensive when training with a larger dataset. Instead of a grid search, which tests every possible combination of the 7 hyperparameters, it might be possible to achieve high performance when tuning one hyperparameter at a time.

We empirically investigate how successful different strategies are. Each strategy takes the form of a sequence, stating in which order to tune the hyperparameters. From every possible starting configuration, we apply the tuning strategy, and then aggregate the results to understand how effective the strategy is. We aggregate by checking how frequently the final configuration appears in a given percentile; e.g., we consider how often the performance is in the 95th percentile after applying the tuning strategy. The results can be interpreted as saying how often the tuning strategy leads to performance of a certain level. For our investigations, we use the following percentiles: 99.9%, 99%, 95%, 90%, 80% and 50%.

V. Results

Here, we present the results of our analysis and the insights obtained.

A. *Multiple Regression and Feature Importance*

Of the various regression algorithms used to predict WK-CNN accuracy from the hyperparameters, Table I shows that the nonlinear models (Random forest, K-NN, MLP) perform the best. This suggests that there is some important nonlinear interaction between the hyperparameters and the fault detection performance of WK-CNNs.

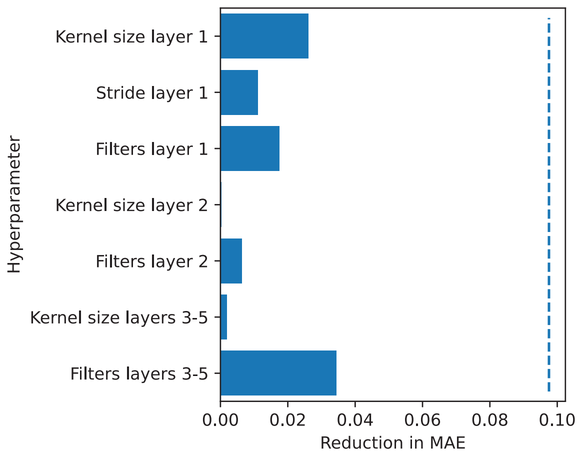


Fig. 2: Feature importance derived using Shapley values. The sizes of boxes depict how much each feature contributes to reducing the MAE of the MLP model. The dotted line indicates the total improvement achieved by using all hyperparameters as inputs.

Taking the MLP as an example model for predicting WK-CNN performance, we can see in Fig. 2 that some features are more important than others. The filters in layers 3-5, and the hyperparameters related to the first layer, appear to be the most important.

B. *Variability According to Hyperparameter*

Next, we take a deeper look at how hyperparameters impact the accuracy on the different datasets, before using this information to again draw general conclusions. As stated in Section IV-B, we summarise how the distribution of accuracy scores shifts when a given hyperparameter is changed, for each dataset,

TABLE I: Metrics of the multiple regression task to predict WK-CNN accuracy using the hyperparameters as input variables. For the task, we utilised a mean baseline, a linear model, two nonlinear models and a DL model (MLP). The MLP model contains three hidden layers each containing 32 ReLU units.

Model	**R2**	**MAE**	**MAPE (%)**
Mean Baseline	0.000	0.164	48.12
Linear Regression	0.437	0.120	32.40
Random Forest	0.966	0.072	18.26
K-NN	0.729	0.076	19.16
MLP	0.730	0.076	18.86

using the d-dimensional earth mover's distance. The resultant scores are presented in Table II.

All of the hyperparmeters show some variation across datasets. In the case of the kernel size in layer 1, for example, the earth mover's distance is 0.05 for Gearbox and 0.64 for SEU. By contrast, the distance score for the number of filters in layers 3-5 is much greater for Gearbox and much smaller for SEU. This implies there are differences between the datasets which can noticeably impact the relationship between a hyperparameter and test accuracy.

The kernel size in layer 2 and layers 3-5, and likewise the number of filters in layer 2, seem to have very little impact on the distribution of accuracy scores, since the earth mover's distances are close to the minimum of 0 across all datasets. The kernel size in layer 1 and the number of filters in layers 3-5 appear to have the largest earth mover's distances, suggesting that they both are important for determining what accuracy a wide-kernel CNN is likely to have.

As noted in Section IV-B, different hyperparameters have a different number of possible settings. This means that sometimes a greater number of distributions are being compared when calculating the earth mover's distance. To validate that this does not impact the conclusions we draw, we visualise the earth mover's distances when they are divided by the number of possible settings that each hyperparameter can have, in the heatmap in Figure 3. This confirms that the kernel size in layer 1 and the number of filters in layers 3-5 are generally the most important hyperparameters for deciding what level of test accuracy is likely to be obtained. To a slightly lesser extent, the stride in layer 1 and the number of filters in layer 1 also have an impact for some datasets. The remaining hyperparameters have very little impact by comparison.

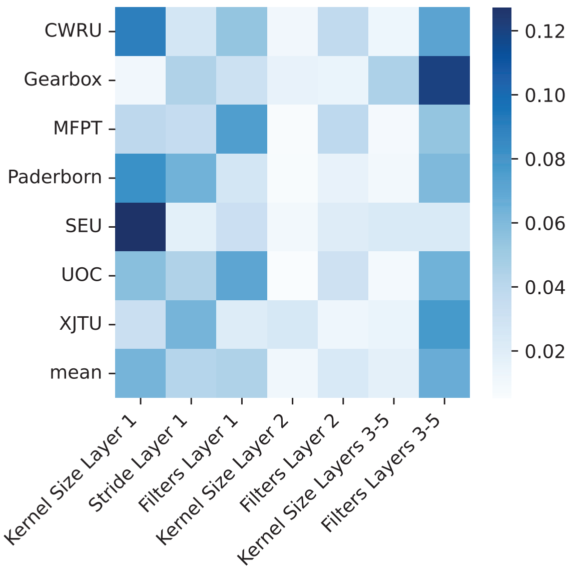


Fig. 3: Heatmap showing the earth mover's distance across hyperparameter settings and datasets. Normalised by dividing by the number of settings of the hyperparameter that were tested in the experiment. Higher scores indicate more change in the distribution of WK-CNN accuracy scores.

C. *Influence of Hyperparameters on One Another*

This section describes the influence that hyperparameters have on one another. It focuses on the question "if Y is tuned, and then afterwards X is tuned, how likely is it that Y will need to be re-tuned?" In other words, this means "how much does changing X affect what value Y should have?" The analysis is done in a pairwise fashion, leading to probabilities reflecting how much each hyperparameter affects each of the others. After calculating the results between pairs of variables, the results are shown in Fig. 4.

As shown in Fig. 4, there are noticeable differences in how influential and how easily influenced the hyperparameters are. Both the kernel size in layer 1 and the number of filters in layers 3-5 are highly influential, unlike, for example, the kernel size in layer 2. Generally, the kernel size hyperparameters are less easily influenced, perhaps because they are affected more by properties of the data (such as the sampling frequency) than by other the hyperparameters. The optimal number of filters to use in a layer is more likely to be influenced by relevant kernel sizes. One possibility is that a large number of filters and a large kernel size leads to over-parameterisation of the model, and thus the risk of overfitting, meaning that the number of filters must change to compensate. Overall, two of the most important hyperparameters seem to be the kernel size in layer 1 and the number

TABLE II: The d-dimensional Wasserstein/earth mover's distances calculated per hyperparameter per dataset.

Dataset	Kernel Size Layer 1	Stride Layer 1	Filters Layer 1	Kernel Size Layer 2	Filters Layer 2	Kernel Size Layers 3-5	Filters Layers 3-5
CWRU	0.45	0.08	0.33	0.02	0.23	0.02	0.43
Gearbox	0.05	0.13	0.19	0.03	0.08	0.09	0.72
MFPT	0.19	0.11	0.45	0.01	0.23	0.02	0.32
Paderborn	0.41	0.19	0.16	0.01	0.09	0.02	0.36
SEU	0.64	0.05	0.19	0.02	0.13	0.05	0.14
UOC	0.29	0.13	0.42	0.01	0.18	0.02	0.38
XJTU	0.16	0.19	0.13	0.05	0.07	0.03	0.46
Average	0.31	0.13	0.27	0.02	0.15	0.04	0.40

TABLE III: Probability of obtaining a certain performance level if tuning hyperparameters individually, according to different orders of tuning.

Ordering	99.9th percentile	99th	95th	90th	80th	50th
Minimising influence order	0.19	0.49	0.75	0.89	0.96	1.00
Maximising influence order	0.12	0.29	0.62	0.80	0.94	0.99
Random order	0.14	0.38	0.69	0.84	0.95	1.00

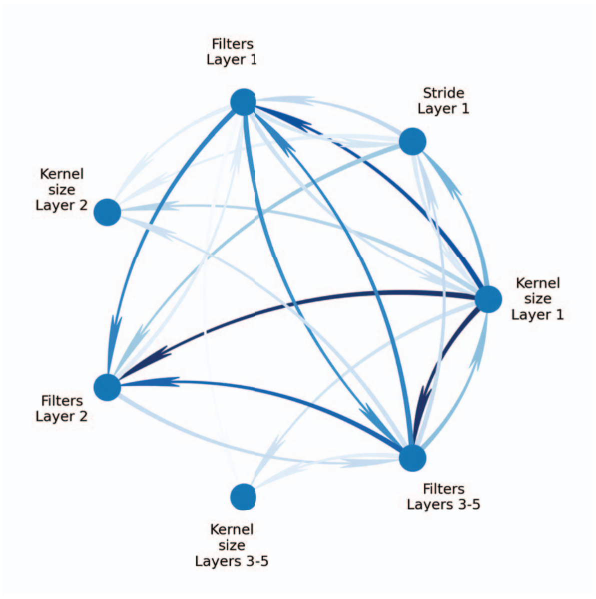


Fig. 4: Network visualisation of hyperparameter pairwise influence. The arrows indicate how likely it is that changing one hyperparameter (the origin) will mean that another hyperparameter (the destination) will need to be re-tuned. Darker arrows imply a greater likelihood. Probabilities less than 0.15 are not shown.

of filters in layers 3-5, since changing these will have the highest chance of making it necessary to re-tune other hyperparameters.

Using the above information, we can put the hyperparameters in a sequence that minimises the likelihood of each element in the sequence causing hyperparameters earlier in the sequence to need re-tuning. Doing this, the order is:

1) Kernel Size Layer 1
2) Filters Layers 3-5
3) Stride Layer 1
4) Filters Layer 1
5) Filters Layer 2
6) Kernel Size Layers 3-5
7) Kernel Size Layer 2

Tuning the hyperparameters in this order leads to performance within the top 1% of models roughly half the time. By contrast, tuning in the reverse order achieves the same level of performance less than one third of the time. Table III shows the likelihood of achieving different levels of performance depending on the order in which the hyperparameters are tuned. Particularly when targeting high performance, above the 90th percentile, it is beneficial to tune such that the likelihood of needing re-tuning is minimised.

VI. Conclusions

How best to tune hyperparameters is an often-neglected question when working with neural networks. This paper extended on our analysis proposed in [5] and used information from a grid search across 12,960 hyperparameter combinations to discover useful information about how to tune a wide-kernel CNN architecture ([1], [2], [4]) for industrial fault detection. In the process, we examined the link between individual hyperparameters and the final performance of trained models, and additionally we investigated what impact each hyperparameter has on tuning other hyperparameters. Looking at how they individually contribute to model performance, the hyperparameters in the first layer and the number of filters in layers 3-5 seem to be the most important. Developing our analysis further, we also proposed a strategy for tuning hyperparameters sequentially, when applying the WK-CNN architecture to new datasets in situations where a full grid search would be too computationally expensive. The strategy we suggested is one which minimises the likelihood that hyperparameters will need to be re-tuned during the process. The results from the benchmark datasets suggest that this performs better than tuning hyperparameters in either a random order, or the reversed order. Future confirmatory work could seek to establish this conclusion more firmly.

ACKNOWLEDGEMENTS

This work was supported by funds of zukunft.niedersachsen, Volkswagen Foundation (project "HybrInt – Hybrid Intelligence through Interpretable Artificial Intelligence in Machine Perception and Interaction").

REFERENCES

[1] W. Zhang, G. Peng, C. Li, Y. Chen, and Z. Zhang, "A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals," *Sensors*, vol. 17, no. 2, p. 425, 2017.

[2] J. van den Hoogen, S. Bloemheuvel, and M. Atzmueller, "An improved wide-kernel cnn for classifying multivariate signals in fault diagnosis," in *ICDMW 2020*, 2020, pp. 275–283.

[3] S. Vollert, M. Atzmueller, and A. Theissler, "Interpretable machine learning: A brief survey from the predictive maintenance perspective," in *Proc. IEEE International Conference on Emerging Technologies and Factory Automation*, 2021.

[4] J. van den Hoogen, S. Bloemheuvel, and M. Atzmueller, "Classifying multivariate signals in rolling bearing fault detection using adaptive wide-kernel cnns," *Applied Sciences*, vol. 11, no. 23, 2021.

[5] J. van den Hoogen, D. Hudson, S. Bloemheuvel, and M. Atzmueller, "Hyperparameter analysis of wide-kernel cnn architectures in industrial fault detection: an exploratory study," *Int. J. Data Sci. Anal.*, pp. 1–22, 2023.

[6] M. R. W. Group *et al.*, "Report of large motor reliability survey of industrial and commercial installations, part i," *IEEE Trans. Ind Appl.*, vol. 1, no. 4, pp. 865–872, 1985.

[7] V. Pandhare, J. Singh, and J. Lee, "Convolutional neural network based rolling-element bearing fault diagnosis for naturally occurring and progressing defects using time-frequency domain features," in *2019 Prognostics and System Health Management Conference (PHM-Paris)*, 2019, pp. 320–326.

[8] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring," *Mechanical Systems and Signal Processing*, vol. 115, pp. 213–237, 2019.

[9] D. Pandya, S. Upadhyay, and S. P. Harsha, "Fault diagnosis of rolling element bearing with intrinsic mode function of acoustic emission data using apf-knn," *Expert Syst. Appl.*, vol. 40, no. 10, pp. 4137–4145, 2013.

[10] Z. Zhou, C. Wen, and C. Yang, "Fault detection using random projections and k-nearest neighbor rule for semiconductor manufacturing processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 28, no. 1, pp. 70–79, 2015.

[11] Z. Wang, Q. Zhang, J. Xiong, M. Xiao, G. Sun, and J. He, "Fault diagnosis of a rolling bearing using wavelet packet denoising and random forests," *IEEE Sensors Journal*, vol. 17, no. 17, pp. 5581–5588, 2017.

[12] P. Santos, L. F. Villa, A. Reñones, A. Bustillo, and J. Maudes, "An svm-based solution for fault detection in wind turbines," *Sensors*, vol. 15, no. 3, pp. 5627–5648, 2015.

[13] D. You, X. Gao, and S. Katayama, "Wpd-pca-based laser welding process monitoring and defects diagnosis by using fnn and svm," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 1, pp. 628–636, 2014.

[14] A. Hajnayeb, A. Ghasemloonia, S. Khadem, and M. Moradi, "Application and comparison of an ann-based feature selection method and the genetic algorithm in gearbox fault diagnosis," *Expert Syst. Appl.*, vol. 38, no. 8, pp. 10 205–10 209, 2011.

[15] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[16] P. Yao, S. Yang, and P. Li, "Fault diagnosis based on rsenet-lstm for industrial process," in *Proc. IEEE Advanced Information Technology, Electronic and Automation Control Conference*, vol. 5, 2021, pp. 728–732.

[17] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, "Real-time motor fault detection by 1-d convolutional neural networks," *IEEE Trans. Ind. Electron.*, vol. 63, no. 11, pp. 7067–7075, 2016.

[18] Y. Liu, X. Yan, C.-a. Zhang, and W. Liu, "An ensemble convolutional neural networks for bearing fault diagnosis using multi-sensor data," *Sensors*, vol. 19, no. 23, 2019.

[19] W. Zhang, G. Peng, and C. Li, "Rolling element bearings fault intelligent diagnosis based on convolutional neural networks using raw sensing signal," pp. 77–84, 2017.

[20] M. Atzmueller, "Declarative Aspects in Explicative Data Mining for Computational Sensemaking," in *Proc. International Conference on Declarative Programming*. Heidelberg, Germany: Springer, 2018.

[21] M. Atzmueller and T. Roth-Berghofer, "The Mining and Analysis Continuum of Explaining Uncovered," in *Proc. AI-2010*. Springer, 2010.

[22] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT Express*, vol. 6, no. 4, pp. 312–315, 2020.

[23] X. Xiao, M. Yan, S. Basodi, C. Ji, and Y. Pan, "Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm," 2020.

[24] A. P. Adedigba, S. A. Adeshina, O. E. Aina, and A. M. Aibinu, "Optimal hyperparameter selection of deep learning models for covid-19 chest x-ray classification," *Intelligence-Based Medicine*, vol. 5, p. 100034, 2021.

[25] S. Raziani and M. Azimbagirad, "Deep cnn hyperparameter optimization algorithms for sensor-based human activity recognition," *Neuroscience Inf.*, vol. 2, no. 3, p. 100078, 2022.

[26] H. Yi and K.-H. N. Bui, "An automated hyperparameter search-based deep learning model for highway traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 9, pp. 5486–5495, 2020.

[27] "Cwru dataset; case western reserve university bearing data center," available: https://csegroups.case.edu/ bearingdatacenter/home.

[28] H. Ocak and K. A. Loparo, "Estimation of the running speed and bearing defect frequencies of an induction motor from vibration data," *Mechanical Systems and Signal Processing*, vol. 18, no. 3, pp. 515–533, 2004.

[29] Y. Pandya, "Gearbox fault diagnosis data," 06 2018. [Online]. Available: https://data.openei.org/submissions/623

[30] C. Lessmeier, J. K. Kimotho, D. Zimmer, and W. Sextro, "Condition monitoring of bearing damage in electromechanical drive systems by using motor current signals of electric motors: A benchmark data set for data-driven classification," *Proc. PHM Society European Conference*, vol. 3, no. 1, 2016.

[31] Society For Machinery Failure Prevention Technology, "Fault Data Sets," https://mfpt.org/fault-data-sets/, Online, accessed: July 2023.

[32] P. Cao, S. Zhang, and J. Tang, "Gear Fault Data," 4 2018. [Online]. Available: https://figshare.com/articles/dataset/Gear_Fault_Data/6127874

[33] Southeast University (SEU), "Gearbox mechanical datasets," https://github.com/cathysiyu/Mechanical-datasets, Online, accessed: July 2023.

[34] B. Wang, Y. Lei, N. Li, and N. Li, "A hybrid prognostics approach for estimating remaining useful life of rolling element bearings," *IEEE Transactions on Reliability*, pp. 1–12, 2018.

[35] V. M. Panaretos and Y. Zemel, "Statistical aspects of wasserstein distances," *Annual review of statistics and its application*, vol. 6, pp. 405–431, 2019.

[36] J. Kline, "Properties of the d-dimensional earth mover's problem," *Discrete Appl. Math.*, vol. 265, pp. 128–141, 2019.