



Hyperparameter analysis of wide-kernel CNN architectures in industrial fault detection: an exploratory study

Jurgen van den Hoogen^{1,2} · Dan Hudson³ · Stefan Bloemheugel^{1,2} · Martin Atzmueller^{3,4}

Received: 13 April 2023 / Accepted: 31 July 2023
© The Author(s) 2023

Abstract

Industrial fault detection has become more data-driven due to advancements in automated data analysis using deep learning. Such methods make it possible to extract useful features, e. g., from time series data retrieved from sensors, which is typically of complex nature. This allows for effective fault detection and prognostics that boost the efficiency and productivity of industrial equipment. This work explores the influence of a variety of architectural hyperparameters on the performance of one-dimensional convolutional neural networks (CNN). Using a multi-method approach, this paper focuses specifically on wide-kernel CNN models for industrial fault detection, that have proven to perform well for tasks such as classifying vibration signals retrieved from sensors. By varying hyperparameters such as the kernel size, stride and number of filters, an extensive hyperparameter space search was conducted; to identify optimal settings, we collected a total of 12,960 different combinations on three datasets into a model hyperparameter dataset, with their respective performance on the underlying fault detection task. Afterwards, this dataset was explored with follow-up analysis including statistical, feature, pattern and hyperparameter impact analysis. We find that although performance varies substantially depending on hyperparameter choices, there is no single simple strategy to optimise performance across the three datasets. However, an optimal setting in terms of performance can be found in the number of filters used in the later layers of the architecture for all datasets. Furthermore, hyperparameter importance differs across and within the datasets, and we found nonlinear relationships between hyperparameter settings and performance. Our analysis highlights key considerations when applying a wide-kernel CNN architecture to new data within the field of industrial fault detection. This supports practitioners who wish to apply and train state-of-the-art convolutional learning methods to apply to similar fault detection settings, e. g., vibration data arising from new combinations of sensors and/or machinery in the context of bearing faults.

Keywords Deep learning · Convolutional neural networks · Subgroup discovery · Hyperparameter analysis · Time series classification · Industrial fault detection · Bearing fault detection

Dan Hudson and Stefan Bloemheugel have contributed equally to this work.

✉ Martin Atzmueller
martin.atzmueller@uni-osnabrueck.de

Jurgen van den Hoogen
j.o.d.hoogen@jads.nl

Dan Hudson
daniel.dominic.hudson@uni-osnabrueck.de

Stefan Bloemheugel
s.d.bloemheugel@jads.nl

¹ Department of Cognitive Science and Artificial Intelligence, TSHD, Tilburg University, 5037 AB Tilburg, The Netherlands

1 Introduction

Advances in deep learning (DL) have revolutionised data analysis due to its capacity to work with raw data, automatically discover its structure and identify patterns [1–3], e. g., for image recognition, or time series analysis. Especially in industry, where data are retrieved using sensors, e. g., time series data, applying DL has become more prominent.

² Jheronimus Academy of Data Science, 5211 DA 's-Hertogenbosch, The Netherlands

³ Semantic Information Systems Group, Osnabrück University, Wachsbleiche 27, 49090 Osnabrück, Germany

⁴ German Research Center for Artificial Intelligence (DFKI), Berghoffstraße 11, 49090 Osnabrück, Germany

resulting in rather efficient and well-performing approaches [4, 5]. In addition, due to the availability of large datasets, automated DL methods provide strong performance in signal classification tasks. However, the optimal settings for these DL architectures are usually use-case specific and thus not transparent. Therefore, this work focuses on analysing hyperparameters of wide-kernel CNN architectures that are specifically applied to signals for industrial fault detection. Using a multi-method approach, we performed a large-scale exploratory study, for which we ran a total of 12,960 unique models. Each model has its own specific hyperparameter combination for our particular classification problem, being tested on three industrial datasets.

The purpose of this research is to get insights into architectural aspects of wide-kernel CNN models, in particular to shed light on the often opaque hyperparameter settings. This concerns, for example, the number of filters, strides and kernels in a wide-kernel CNN (WK-CNN) model, focusing on how they influence model performance. We explore the different hyperparameter settings to identify which configurations generate a good or bad result, and to obtain insights resembling ‘general rules’ for applying a one-dimensional WK-CNN in a signal classification context. In particular, we use the model architecture previously proposed by [3, 6], which is built upon the wide-kernel framework developed by [7]. This architecture has proven to be state-of-the-art in bearing fault detection tasks. The model is easily trainable due to only utilising five convolutional layers (other state-of-the-art models use much deeper architectures with more layers [8–11]), while achieving very high performance [3, 6, 7, 9]. Furthermore, the wide-kernel model can be scaled towards dimensionality changes of the data i.e., different amounts of sensors, and due to its compactness it is applicable in real-world settings.

The WK-CNN has a limited but non-trivial number of architectural hyperparameters, making an extensive hyperparameter search interesting and informative. Due to the above reasons, such an architecture is also applicable in different contexts and application domains where time series data sampled at a high frequency is often used, such as the industrial vibration data we specifically focus on.

Our contributions are summarised as follows:

1. Exploratory analysis in the context of time series fault detection with high frequency data:
 - (a) *Experimentation* We investigate the performance of a wide-kernel CNN architecture [3, 6, 7] with varying hyperparameter settings on multiple datasets by running a total of 12,960 models per dataset. This includes univariate and multi-variate time series data from sensors, measuring vibration signals from industrial machines, in fault detection scenarios.

- (b) *Statistical analysis* We perform statistical descriptive analysis, modelling and assessment on our collected model parameter dataset in order to obtain general insights about the hyperparameter space.

2. In-depth feature, pattern and impact analysis:

- (a) *Feature analysis* We identify within our hyperparameter search the most important features that influence model performance.
 - (b) *Pattern analysis* We search for patterns that identify specific hyperparameter combinations which optimise/yield performance using local pattern mining methods.
 - (c) *Impact analysis* We visualise the hyperparameter space and explore the impact of these hyperparameters on WK-CNN performance. Additionally, we use a graph-like representation of the results to determine how incremental changes to hyperparameters are related to changes in performance category (e.g. from ‘Low-’ to ‘High-’ performance).

Figure 1 depicts our analysis workflow; the individual steps are labelled according to the contributions above. The rest of the paper is structured as follows: related work is discussed in Sect. 2, where also the necessary background on deep learning is briefly summarised. Afterwards, we describe the experimental setting, introduce the used CNN model and the analysis techniques applied in Sect. 3. Next, we describe the results and our findings in depth in Sect. 4. Finally, Sect. 5 concludes and summarises our findings, followed by promising future prospects.

2 Related work

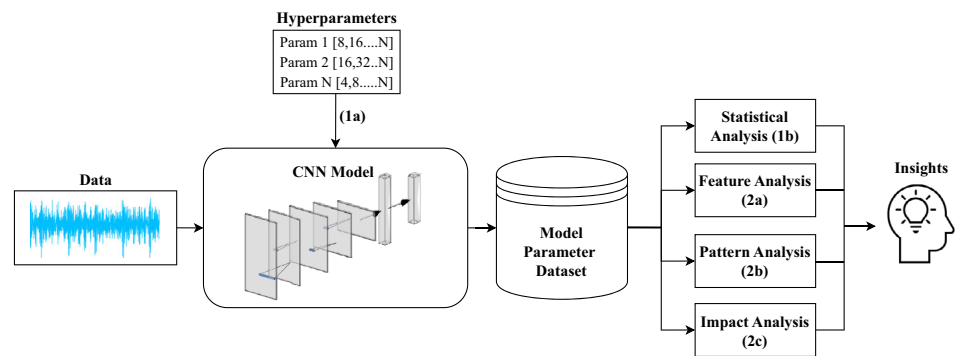
This section covers relevant work relating to fault detection on industrial machinery. In addition, we describe deep learning (DL) in general, specifically in the context of time series analysis using sensor data with a focus on convolutional neural networks (CNN). Furthermore, hyperparameter search for optimising DL architectures, as well as extended analysis via pattern mining using Subgroup Discovery is discussed.

2.1 Fault detection

Fault detection plays a vital role in preventing machine breakdowns, especially for rotating industrial equipment [12].

In the past, faults were detected using physics-based models, which call for prior understanding of the underlying mechanisms. These models do not adjust to changing environments or increases in the quantity of data and variables [13]. Therefore, the development of data-driven analytics and the rise of Industrial Internet of Things (IoT) has consider-

Fig. 1 Workflow towards deeper understanding and insights on the architectural hyperparameters of the proposed wide-kernel CNN model. The labels provided in parentheses align with the contributions described in Sect. 1



ably changed the way to detect faults in industrial equipment, leading to a more intelligent approach [5, 14]. Without previous knowledge of the technical components of industrial machinery, these technologies potentially enable automated data processing while being readily adaptive to shifting operational circumstances.

The data for fault detection tasks are usually gathered with sensors that measure vibrations from industrial equipment, which is represented as time series data. These sensors are able to capture phenomena such as the decay of the underlying mechanical components that are not immediately visible in the internal sensors of the machine [5, 7, 15]. However, many data-driven solutions using traditional machine learning (ML) algorithms cannot process the raw time series directly and therefore require substantial feature extraction [16–18].

These necessary preprocessing steps often take a significant amount of time and require high-level understanding in signal processing and data processing. Furthermore, extracting features is heavily dependent on the type of data gathered, meaning that the processing scheme must be re-evaluated with new data or use-cases.

For fault detection, many ML approaches have been researched, e. g., K-nearest neighbour (K-NN) [19, 20], random forest [21], support vector machine [17, 22–24] and artificial neural networks [25, 26]. However, across these techniques the performance is heavily dependent on quality of data as well as feature extraction and the design of the experiment. Thus, this research focuses on DL methods that can automatically extract features from raw data, simplifying the complex feature extraction process, and examines its respective hyperparameter space.

2.2 Deep learning

In the last decades, deep learning (DL) has become a more prominent technique for analysing complex (multi-dimensional) data. It uses multi-layered architectures to discover patterns in data, which are difficult to find manually.

These models extract important information and automatically transform them into valuable features.

2.2.1 Deep learning for fault detection

Initially, the multi-layer perceptron (MLP), where all layers are fully connected, was employed for fault detection [27]. However, the depth of these networks is constrained due to the large increase in computation time. In order to account for this, more sophisticated neural network topologies were created in recent years, for example, recurrent neural networks (RNN) that can take temporal dependencies into account. Therefore, their results on time series data were promising [28, 29]. However, RNNs require vast amounts of memory to store these time dependencies, making them less suited for high-frequency sensor data or with long sequences.

As a result, the field of fault detection started to use convolutional neural networks (CNN) [5, 14, 30, 31] which have demonstrated notable performance improvements. Additionally, CNN methods coupled with data transformations, such as spectrograms, have been utilised several times [4, 32]. Furthermore [33], was successful in building a one-dimensional (1D) CNN that can handle raw time series data directly. These 1D CNNs are robust against noise in time series data and can be trained using a small sample of data [34]. Afterwards, many optimized models for fault detection using CNNs were introduced, such as but not limited to; wide-kernel CNNs [3, 6, 7] (which is used in this work), attention CNNs [9], periodic CNNs [8] or CNN autoencoders for unsupervised learning tasks [11]. All of these models have their own characteristics in terms of complexity, computational requirements and tasks. The general mechanics behind the CNN are further discussed in the next paragraph.

2.2.2 Convolutional neural networks

Convolutional neural networks are essentially regularised MLPs that are initially designed to handle two-dimensional (2D) data, e. g., for processing images. These types of neural networks were first proposed by LeCun in 1989 [35]. The

main advantages of a CNN compared to a MLP are three-fold. First, these architectures use weight-sharing, enabling models to efficiently process the data through multiple layers. Second, CNNs utilise downsampling procedures that reduce the dimensionality of the data. Third, they contain local receptive fields for mapping data structures in a joint way. Therefore, CNNs require less memory and boost algorithmic performance. However, for a time, CNNs were less commonly used for time series data due to their 2D nature. Initially, when using CNNs for time series data, one had to convert the one-dimensional data into a 2D matrix using signal processing techniques, resulting in more computations. Then, [36] developed a CNN that is able to directly process raw 1D data, e.g., time series. With this development, DL became a prominent technique for time series analysis [3, 6, 7, 33, 34].

A convolutional layer uses filters to convolve the input to generate the output features. Each of these filters extracts regional characteristics from the input local area through weight-sharing. In this layer, the kernel size defines the dimensions of the data to extract these characteristics with the filters. Furthermore, the stride controls the window shift towards the next local area, defined by the kernel size, where the filters are again applied. The outputs of the convolutional layer are then fed to the activation unit, which creates the final output features for the follow-up layer. The embedded activation function accounts for nonlinear information from the input of the convolutional operation. Next, the output of the convolutional layer combined with an activation is then fed to a pooling layer. By downsampling the input features' spatial size, this pooling layer minimises the network's complexity by lowering the number of parameters and computations.

2.3 Hyperparameter search in deep learning

This work focuses on exploring how architectural hyperparameters of a neural network influence the performance of the model and how adjusting these hyperparameters could lead to better or worse performance. For this, we searched extensively through a set of hyperparameter combinations for a wide-kernel CNN model that has been proven to perform well on fault detections the task at hand [3, 6].

In various forms, searching for optimised settings in DL architectures has been researched before. For example, [37] assessed the effect of the batch size in a CNN model, while [38] searched for efficient methods for hyperparameter optimisation using a variable length genetic algorithm, both focusing on image classification tasks i.e., two-dimensional data. Also, [39] showed optimal hyperparameter selection for COVID chest x-ray classification. Their method focused on a discriminative fine-tuning approach to optimise the individual learning rates for every layer. In addition, [40] evaluated various parameter optimisation algorithms for sensor-based

human activity recognition using one-dimensional CNNs. However, their work focused on a shallow 1D CNN model with fewer convolutional layers and different parameter settings. Furthermore, their focus lies on finding the optimal parameter setting using optimisation algorithms, while we focus on understanding the underlying patterns and behaviour of our models' performance.

Additionally, [41] automated their parameter search for a LSTM-RNN model for highway traffic prediction. Their strategy for searching involved Bayesian search optimisation. This optimisation algorithm assumes that the parameters to be tuned are of continuous nature, making it difficult to assess its usability on integer values of ordinal nature as used in this work. Therefore, our focus for hyperparameter tuning involved an extensive automated grid search throughout lists of combinations that influence the model's architecture.

2.4 Pattern mining

Pattern mining aims to extract knowledge from data by identifying frequently-occurring 'patterns' that characterise the data. In the context of hyperparameter optimisation, this can be used to identify combinations of hyperparameters that are strongly associated with good performance.

Patterns are descriptions of a group of data points within the dataset and can take different forms depending on the application being considered. When analysing customer shopping data, for example, they might take the form of a combination of items that are often bought together [42]. In time series, patterns might be short subsequences that occur especially frequently in the data [43]. Which patterns are interesting is usually decided by reference to how statistically exceptional they are.

One specialised field in pattern mining is subgroup discovery [44, 45]. This approach finds descriptions of data points within the dataset using a collection of simple rules in the form $\text{Attribute_A} > K$, where K acts as a threshold. Various statistical properties can be used to evaluate the exceptionality of the data points identified by the subgroup description, starting from simply the mean of a variable of interest (i.e., an interesting subgroup will have a mean value that is different from the mean of the overall population). Based on such statistical properties, there exists a variety of metrics to quantify the exceptionality of subgroups, giving a range of objective functions to choose from. Subgroup discovery also includes algorithms for efficiently searching through candidate descriptions in order to find the best subgroups.

Subgroup discovery has been applied to various domains, including medical [46, 47], advertising [47], social interactions [48, 49] and electricity consumption [50]. In relation to computer science, subgroup discovery has been applied to the prediction of faulty code modules [51, 52]. Specifically

in relation to neural networks, subgroup discovery has been applied to hyperparameter optimisation data by [53], who investigated combinations of hyperparameters that helped a neural network optimise a multi-objective consisting of maximum residual error and explained variance. In contrast to this prior work, we apply subgroup discovery to data relating to architectural hyperparameters (like number of filters in a convolutional layer) rather than learning parameters (like the learning rate and degree of regularisation).

One key benefit of pattern mining approaches such as subgroup discovery is that they offer interpretability. In the context of our work, this means that we are able to derive meaningful descriptions of groups of data points, defined by their shared hyperparameter settings, which are indicative of particularly good or poor WK-CNN performance.

3 Materials and methods

This section describes the experimental setting of the hyperparameter search and discusses a multi-method approach to analyse the data of the architectural hyperparameter settings and the corresponding performance each model achieves. Furthermore, this section gives a detailed description of the wide-kernel CNN (WK-CNN) architecture used.

3.1 Experimental setting

All three datasets used in this work come from industrial fault experiments. One dataset looks at the gearbox of the machine, while the other two datasets investigate a specific element of the machine, which is called a bearing. An example of a bearing with its properties is described in [54]. In these experiments, sensors are placed on a test rig that record vibrations from the machine supplemented with an annotation of its current condition, i. e., a label. The data combined with the annotated condition are used to train the WK-CNN, so that later on the model is able to correctly classify machine conditions on unseen sensor data samples.

We tested our WK-CNN in univariate and multi-variate time series experiments with both multi-class and binary classification tasks. The sampling rate of the data also varies between the different datasets. However, the sequence length of the data remained the same across the experiments and was set to 2048 data points, a frequently used sequence length within the signal classification and bearing fault domain [7, 34, 55]. Figure 2 shows an example of two vibration signals with its respective sequence length. All models were trained for 100 epochs with an early stopping criteria set to 10, combined with k -fold cross-validation where $k = 3$ to improve the generalisability of the results. Each WK-CNN is trained and then evaluated on an unseen test set, resulting in test accu-

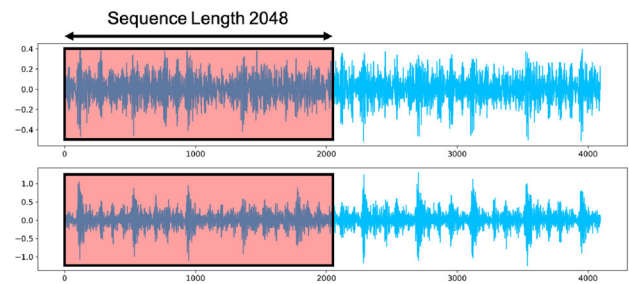


Fig. 2 Example signal of the vibrations from two sensors derived from the CWRU Bearing dataset. The red box indicates the length of one data sample (sequence)

racy scores. Next, we describe the wide-kernel architecture we use and the datasets used to train and test the models.

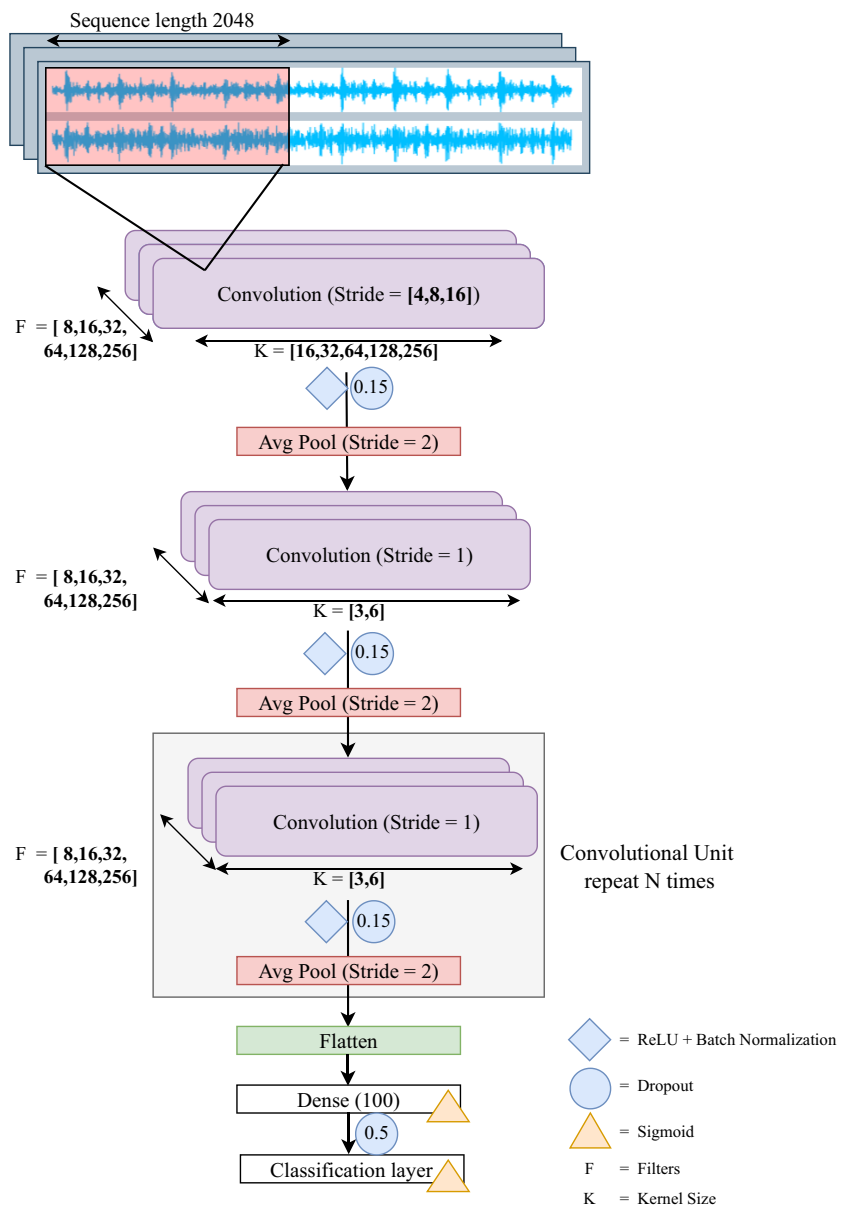
3.2 Model architecture

Previous work showed excellent performance when using a wide kernel in the first convolutional layer, followed by smaller kernels in the follow-up layers, for bearing fault detection [3, 6, 7]. Besides its high performance, the wide-kernel CNN is particularly suitable for fault detection tasks due to its reasonable size in terms of convolutional layers, and its ability to directly process the raw time series. However, there is no conclusive evidence that this particular model architecture has the optimal architectural hyperparameter settings, such as the number of filters and the stride, within its convolutional layers. Therefore, we experimented with a general model layout as proposed by [6, 7] and tested many versions of this model with different combinations of architectural hyperparameters, i. e., the filters, kernel sizes and strides were adjusted in the convolutional layers, changing the architecture and trainable parameters of the model.

Our architecture contains two convolutional layers, each utilising several combinations of stride, kernel size and filters. After that, a convolutional unit (see Fig. 3) is repeated *three* times, resulting in a total model of five convolutional layers. The three convolutional units all have the same hyperparameter combination in this study, relying on the observation that the deeper layers of the model should have the same properties for optimal learning, as described in previous work [3, 6, 7]. After each convolutional layer, the model utilises local average pooling to decrease the vector size of the convolutional output with length T divided by two, resulting in a pooled output length of $\frac{T}{2}$. At last, the reduced output from the convolutional layers is fed to two fully connected layers, of which the last one functions as the classification layer. Figure 3 shows the overall model architecture with two exemplary time series as input.

We specifically chose to use a wide-kernel CNN architecture because it has been shown in previous work that such a

Fig. 3 Architecture of the used 1D wide-kernel CNN model, inspired by [6]. The model utilises two convolutional layers followed by a convolutional unit (in grey), which is repeated 3 times. The values marked in bold represent the hyperparameter combinations used in the search



model is capable of achieving near-perfect accuracy on the two bearing fault detection datasets (CWRU and Paderborn) [3, 6, 7], suggesting that it is well-suited for fault detection tasks and that a more complicated model gives little extra benefit. Furthermore, the architecture is lightweight in terms of computational resources and memory usage, which allows it to have a wider range of uses, in particular opening up the possibility of edge computing, which accelerates the transition to Industry 4.0. Additionally, the modest depth of the wide-kernel architecture, compared to other state-of-the-art CNN architectures [8–11], lends itself well for an extensive grid search of the hyperparameters due to the relatively low number of hyperparameters.

In addition to the model's architecture, some additional optimisation methods are utilised to improve performance

and computation time. These remained the same for every combination that was tested. First of all, between convolutional layers, a batch normalisation layer is added to speed up the training process. Second, we used Adam stochastic optimiser. This optimisation algorithm uses the strength of adaptive learning rates for each individual parameter and is well-suited for models that analyse high dimensional input and have a lot of trainable parameters, since it is memory-light and computationally efficient. According to [56], Adam is very effective with noisy signals. At last, for the CWRU and Paderborn dataset, we used mean squared error (MSE) as loss function for the model, that has proven to be beneficial in [57], while using binary cross-entropy for the Gearbox dataset.

Table 1 Characteristics of the three datasets in terms of sequences (seq), classes, sensors (sens) and sampling frequency (f_s)

Dataset	Seq	Classes	Sens	f_s
CWRU	1355	13	2	12 kHz
Gearbox	978	2	4	30 Hz
Paderborn	8000	8	1	48 kHz

Every sequence consist of 2048 data points

3.3 The data

For our experiments, we used three different datasets all measuring various fault conditions of industrial machinery. These datasets were chosen because they are good representations of industrial time series data retrieved from sensors, with proper annotations of the different classes available, which is difficult to find.

Even though the datasets all represent a similar task focusing on industrial faults, they differ significantly from one another. Table 1 gives an overview of the characteristics of the used datasets.

3.3.1 CWRU bearing dataset

The CWRU bearing dataset represents a bearing fault experiment conducted by the Case Western Reserve University, where damages were deliberately inflicted to a bearing element. To measure vibrations from the bearing element, sensors were placed on the drive-end and fan-end of the machine; we refer to [58, 59] for more detail. These vibrations are digitised into two time series variables that we segmented into sequences of 2048 data points (see Fig. 2). The data used in this study are retrieved from the 12k fan-end experiment, which is sampled at a rate of 12 kHz. We aggregated different machine operations with 1797 and 1750 motor speed (expressed in revolutions per minute, RPM).

The CWRU bearing dataset [58, 60] has been used various times in fault detection work [7, 32, 34, 61–63]. It can be seen as a benchmark dataset due to its similarity in structure to real-world industrial applications, which is uncommon for openly accessible data. Additionally, the well-documented experiment simplifies analysis and interpretation. In total, there are 1355 sequences and 13 different fault conditions, which are slightly unbalanced. Within these experiments, we follow a standard train, validation and test split of the dataset. First, we split the full dataset into 20% train and 80% test data, to make the classification task more complicated, as previously described in [3], and reduce the computation time. Furthermore, we provided the data in batches of size 64 to the model in order to speed up the training process.

3.3.2 Gearbox dataset

The Gearbox dataset for fault detection originates from the need to optimise and analyse industrial gearboxes. Especially when analysing wind turbines, faults often arise in this particular part of the machinery. For example, earlier research [64] has shown that the downtime of a wind turbine due to a gearbox failure is higher than for the other components. Recording and analysing vibration signals is the most commonly used approach for examining the faults of a wind turbine gearbox, which can be marked as non-stationary. The Gearbox dataset represents these particular vibration signals that are expected from an industrial gearbox. Using the SpectraQuest Gearbox Fault Diagnostics Simulator, researchers were able to simulate the behaviour of an industrial gearbox for diagnostics and prognostics research [65].

The simulated dataset consists of four vibration sensors placed in different directions of the gearbox. The experiment is performed under variations of load between 0 and 90%. The data consist of only two conditions, i. e., a healthy and broken tooth gearbox. We combined all the variations of load to make it a binary classification task, i. e., healthy vs broken. The dataset contains 978 sequences each of 2048 data points long with a sampling rate of 30 Hz. Within the 978 samples, 492 samples are marked as healthy and 486 samples as broken. Therefore, the dataset can be considered balanced. Considering this classification task, we assumed that the task would be relatively easy with the used architecture. Therefore, we reduced the amount of training data by applying a train/test split of 20/80% making the task more complicated and reducing computation times. We implemented a standard batch size of 32, since additional reduction of computation time was not necessary.

3.3.3 Paderborn bearing dataset

The Paderborn bearing dataset is a widely used and well-documented dataset for a variety of fault detection tasks. Deviating from the other datasets, the Paderborn bearing dataset contains both motor current and vibration recordings of damaged bearing elements, cf. [14, 30, 66].

The signals are extracted in the existing frequency inverters. Therefore, no additional sensors needed to be placed on the system, as was the case in the CWRU bearing and the Gearbox experiments, resulting in more resource-efficient experimentation, thus less expensive. In this work, we only extracted the vibration sensor, since this aligns with the vibration data retrieved from the other experiments.

The Paderborn experiment consists of many different faults, e. g., ‘healthy’, ‘real damaged’ and ‘artificially damaged’ bearings. For this work, we decided to only extract the data from the ‘real damaged’ experiments and more specifically for the ‘inner race’ faults, containing 8 condi-

Table 2 Architectural hyperparameters with their respective value domains used in the search

Hyperparameter	Value domain
Kernel size layer 1	{16, 32, 64, 128, 256}
Stride layer 1	{4, 8, 16}
Filters layer 1	{8, 16, 32, 64, 128, 256}
Kernel size layer 2	{3, 6}
Filters layer 2	{8, 16, 32, 64, 128, 256}
Kernel size layers 3–5	{3, 6}
Filters layers 3–5	{8, 16, 32, 64, 128, 256}

tions, similar to previous experiments conducted by [66]. The data are recorded for approximately 4 s with a rate of 64 kHz. This resulted in 80,000 sequences (10,000 for every condition), which would require substantial computational resources when executing the many CNN model configurations. Therefore, we decided to take an equally divided sample from the dataset of 10%, resulting in 1000 sequences per condition. Afterwards, similar to the other datasets, we made a train/test split of 20% and 80% where the data is fed to the model in batches of 64, reducing computation time. For an overview of the properties per dataset, see Table 1.

3.4 Hyperparameter search overview

Using the three datasets, we examined various hyperparameters of the wide-kernel CNN architecture that influence the dimensionality of the model and also potentially influence performance. Every possible combination was investigated within the hyperparameter search in order to get a thorough indication of the effects of the hyperparameters. Table 2 shows the hyperparameter search with respective value ranges. In total, we ran 12,960 combinations for each of the three datasets, resulting in 38,880 trained models with corresponding performance scores. This collection of 38,880 models and performance scores provides a dataset that we then analyse to gain insights into how architectural hyperparameter settings influence performance.

Due to the ordinal nature of the controlled hyperparameters, where exponential steps are being made between the values, for some analyses the features were recentred and rescaled using \log_2 , which is denoted as;

$$f(x) = \log_2(x) - E(\log_2(x)) \quad (1)$$

here x represents the data and E denotes the expectation (mean).

In this way, the numerical changes between the values are equally spaced. This makes the results interpretable, since a unit increase or decrease of a hyperparameter value means doubling or halving the numerical value. Table 3 shows a

snapshot of the final result file, including the (derived) hyperparameters and their accuracies.

3.5 Predicting performance

In general, the model parameter dataset derived from the hyperparameter search (described in the previous subsection) is too large to manually search for insights. Therefore, we used machine learning (ML) and DL algorithms to look for patterns in the data that explain the connection between the hyperparameters (independent variables) and WK-CNN performance (dependent variable). In this way, we gain understanding about the informativeness of the hyperparameters and their influence towards better WK-CNN performance. In addition, we compared the predictions of the models to a baseline level of performance to evaluate if the models learn meaningful information from the data. In order to formulate this regression task, using the hyperparameter search dataset, we take the accuracy score on the unseen test set as the (continuous) target variable. To predict the target variable, we utilised the following models: linear regression, random forest and K-NN, as well as a simple MLP with two layers (each utilizing 32 neurons with a ReLU activation function). All models utilize cross validation for improving generalizability of the predictions. The ML models are optimised using a cross-validated grid search.

3.6 Shapley values

Shapley values are derived from game theory as a fair method to award pay-outs to team members in a collaborative game [67]. When the game has an outcome that can be measured with a single number, Shapley values provide a way to quantify the contribution of each member to the team's outcome, represented in terms of a portion of the total outcome of the team. The method is quite general and can be applied to problems that are not typically thought of in terms of teams: for example, it is possible to use Shapley values to evaluate the contribution that different features make to the predictability of WK-CNN performance in a regression task, thus giving a measure of feature importance. This is the approach that we follow in order to understand which architectural hyperparameters of a CNN have the most influence on performance.

Like with Sect. 3.5, we base our Shapley value analysis on the results from the architectural hyperparameter search, i. e., on combinations of WK-CNN hyperparameters and corresponding performance (accuracy scores). This data is used to define a regression task, where the goal is to predict the accuracy. Unlike in Sect. 3.5, when performing the regression, we can choose to omit a hyperparameter from the inputs. Comparing the successfulness of the regression when the hyperparameter is omitted, to when it is included, indicates

Table 3 Snapshot of the result file with the controlled hyperparameters as columns and the measured performance (Acc.), for the respective datasets

Nr.	Kernel size layer 1	Stride layer 1	Filters layer 1	Filters layer 2	Kernel size layer 2	Filters layers 3–5	Kernel size layers 3–5	Acc.	Dataset
0	256	16	64	64	6	32	3	0.9141	cwru
1	256	16	64	64	6	16	6	0.6851	cwru
2	16	4	8	128	3	8	6	0.2913	cwru
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
N	128	4	8	8	3	16	3	0.9352	Paderborn

how much that hyperparameter contributes to the predictability of WK-CNN performance.

One way to think about the calculation of the Shapley values is to consider permutations of the architectural hyperparameters. First, it is decided which feature to calculate the Shapley value for. Then, a permutation is selected and two regressions are performed: (1) using hyperparameters in the permutation up to and including the one we are calculating the Shapley value for and (2) a regression that uses only the hyperparameters that appeared earlier in the permutation. The results are compared to see how much the hyperparameter contributes to lowering the error of the regression. Note that this is the contribution made to a regression which did not necessarily include all of the other hyperparameters as inputs. Averaging across all possible permutations gives the Shapley value for a feature. Using the notation seen in [68], the calculation is as follows:

$$\varphi_i = \frac{1}{|N|!} \sum_{\mathcal{O} \in \pi(|N|)} \left[v(\text{Pre}^i(\mathcal{O}) \cup \{i\}) - v(\text{Pre}^i(\mathcal{O})) \right] \quad (2)$$

where φ_i is the Shapley value of feature i , $|N|$ is the number of features, \mathcal{O} is a permutation of the features, $v(s)$ is the performance of a regression only using the set of features s , and $\text{Pre}^i(\mathcal{O})$ is the set of features that appear earlier than feature i in the permutation \mathcal{O} .

3.7 Subgroup discovery

Subgroup discovery allows us to enhance our understanding of how architectural hyperparameters of wide-kernel CNNs relate to their performance. We use subgroup discovery to identify regions within the hyperparameter space where the performance is particularly good or particularly poor. By combining several rules with the structure `Hyperparameter_A >= K` where K represents a threshold value, subgroup discovery provides an interpretable definition of a group of data points that are interesting with regard to a target variable (in our case, performance of wide-

kernel CNNs on the task). The goal of the search is to find subgroups (groups of models) with an extreme distribution of values for performance, either an exceptionally high or low mean value. This information is combined with the size of the subgroup, i.e., how many models it includes, via a “quality function” that makes it possible to compare the quality of subgroups. For this, we use the ‘Piatetski-Shapiro’ [69] and ‘Lift’ [70] quality functions.

3.8 Modelling individual changes to hyperparameters as a graph

Given the comprehensive nature of the hyperparameter search used in this work, it is possible to compare models that differ only by a simple change to a single hyperparameter. For example, it is possible to compare two models that differ only in the fact that one has a kernel size of 32 in the first layer, while the other has a kernel size of 64. This helps to build up a picture of the impact of small, incremental changes to the hyperparameters. To support our analysis, we create an additional dataset derived from the architectural hyperparameter search results. In this derived dataset, each entry represents the impact of changing a single hyperparameter, by either halving it or doubling it, capturing the difference from before to after the change. In this way, the derived dataset is modelled as a graph where each node is a specific model with a unique configuration; the edges connect models that only differ by a small modification of a single hyperparameter, comparable with a 1-hop neighbourhood.

3.9 Software and resources

We applied Python with Tensorflow and Keras to develop and train the models. For that, we used a dedicated server with two Intel Xeon CPUs (3.2 GHz), 256 GB RAM and an Nvidia Quadro RTX6000 (24 GB) GPU. Furthermore, the ML models used for the regression are implemented using the scikit-learn, Numpy and Pandas. The functionality used for subgroup discovery is available through the SD4Py

package [71]. In addition, a Github page¹ is made available containing the code for the used wide-kernel CNN model with the hyperparameter search and the full result datasets of the experiments as demonstrated in Table 3.

4 Results

This section describes the results of our analysis on the performance of all the hyperparameter combinations. Hence, emphasis lies on understanding and interpretation of the behaviour of the WK-CNNs' performance in relation to the controlled hyperparameters, and not directly on individual model performance itself. Similarly, we focus on analysing the performance of WK-CNNs with different hyperparameter combinations, not the original vibration signal data.

4.1 General overview

To get an initial understanding of the hyperparameter search results, the distributions of performance scores across all models are shown in Fig. 4. Evidently, the hyperparameters have a strong impact on the ultimate performance of a model, emphasising the importance of finding appropriate hyperparameter settings. However, the correlation coefficients between the number of *trainable* parameters of a model, which is influenced by its hyperparameter settings, are moderately negatively associated with model performance, i. e., $r = -0.17$ (CWRU), $r = -0.69$ (Gearbox), $r = -0.48$ (Paderborn). This indicates that the number of *trainable* parameters, determined by the hyperparameter settings, should not just be increased to attempt improving model performance. Furthermore, the magnitude of the correlation between the *trainable* parameters and model performance is heavily dependent on the dataset e. g., the fault detection use-case.

Figure 4 shows that the distributions of performance deviate between the three datasets. The performance on the CWRU data has a much larger spread of values than the other two, with some models having close to chance-level performance while others are near-perfect. The results on the Gearbox data show a series of clear peaks of accuracy. Like with CWRU, the worst-performing models have only chance-level performance, while the best are almost perfect. For the results on the Paderborn data, the majority of models have high accuracy, close to the theoretical maximum of 1.0, and an accuracy of below 0.9 is relatively uncommon. These figures suggest that there may be differences within the datasets themselves that make achieving a high performance score easier or harder. The Paderborn use-case, for example, is clearly the 'easiest' dataset, where the majority of hyperparameter configurations seem to be able to solve the task.

One potential explanation for this is the vast availability of data, as shown in Table 1.

To obtain a slightly more detailed view on the results, we look at how the models perform for each hyperparameter individually. In Figs. 5, 6 and 7, the distribution of the performance is visualised for each value of each hyperparameter.

The visualisations indicate that for certain hyperparameter settings in the model's architecture, an optimum for performance can be found, which suggests that afterwards the models tend to overfit, especially when looking at the number of filters. Moreover, we see that the trends in average performance on the CWRU and Paderborn datasets are more alike than on the Gearbox dataset, which could be explained by the nature of the data, e. g., Gearbox contains 4 vibration sensors, while CWRU and Paderborn contain 1 and 2, respectively. Also, the classification task for the Gearbox use-case varies from the others, since it only contains binary classes and the sampling rate is significantly lower.

Furthermore, the optimal value detected in a few hyperparameters indicates that the relationship between these features and model performance is of nonlinear nature. However, keep in mind that the initial analysis only looks at the hyperparameters separately and therefore does not capture the interaction between hyperparameters.

4.1.1 CWRU overview

Some general impressions can directly be made based on the visualisations in Fig. 5. First of all, it seems that the kernel size adaptations in layer 2 and layers 3–5 (representing the values 3 and 6) result in very similar performance. Therefore, it seems that these deviations barely influence model performance. Another interesting result is the model performance concerning the filters in all layers, e. g., layer 1–5. They show an initial optimum around filter setting 32, meaning that having more filters eventually results in overfitting. Furthermore, the average performance between different kernel sizes in the layer 1 shows an almost linear growth, suggesting that a larger kernel size clearly improves performance. This was not expected since the complexity of the task is quite high, and therefore, we instead anticipated that a narrow kernel would be more effective.

4.1.2 Gearbox overview

Looking at how specific hyperparameter values affect performance on the Gearbox dataset, we encounter different behaviour than for the CWRU use-case. For example, in layer 1, increasing the stride reduces performance, while increasing the kernel size does not seem to be that influential. Furthermore, increasing the filters in layer 1 increases performance on average, while adding more filters in layer 2 does not influence the average performance that much. How-

¹ <https://github.com/JvdHoogen/ParamAnalysisWCNN>.

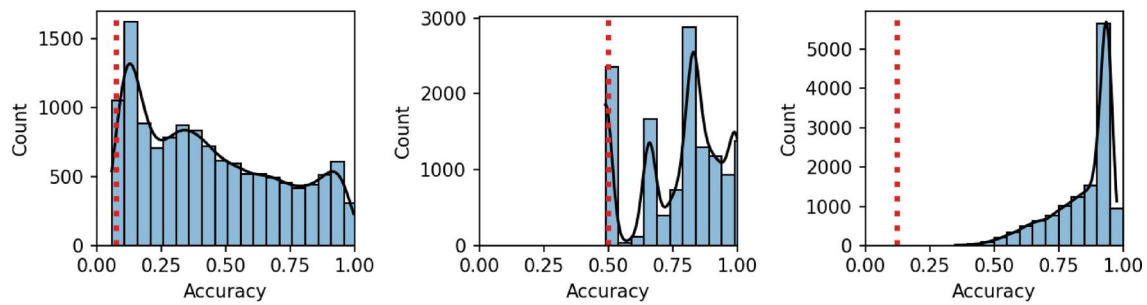


Fig. 4 Distribution of the performance expressed as accuracy of the wide-kernel CNNs for CWRU (left), Gearbox (middle) and Paderborn (right). The dotted red line indicates chance-level performance

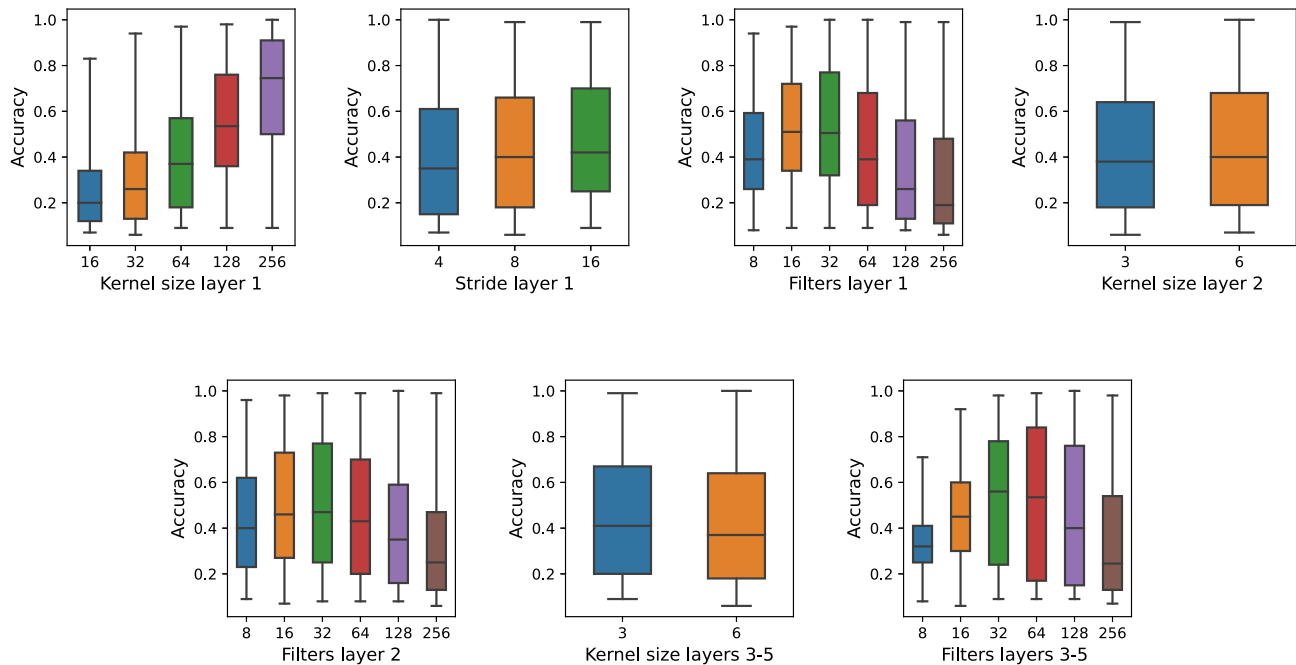


Fig. 5 Distribution of model performance, expressed as accuracy, for the CWRU use-case visualised for each individual hyperparameter (kernel size, stride, filters)

ever, when looking at the filters in layer 3–5, we see a drastic reduction of performance when increasing the amount above 32 filters, suggesting that there is an optimum for learning in the filters as seen in the CWRU dataset as well. At last, the average performance relating to the kernel sizes in layer 2 and layers 3–5 stay similar between the two conditions.

4.1.3 Paderborn overview

For the Paderborn use-case, Fig. 7 shows that deviations between the different hyperparameter settings are smaller compared to the CWRU and Gearbox results. However, we do see similar trends in the results as for the CWRU use-case. For example, increasing the kernel size and stride in layer 1 positively influences the performance of the model. Furthermore, a similar optimum in the number of filters for layers 1–5 can be detected around the value of 32, which is the same

finding as for CWRU. Also, the kernel sizes in layer 2 and layers 3–5 did not influence performance clearly.

4.2 Predicting performance

To get insights into the informativeness of the hyperparameters that are controlled in our hyperparameter search, we defined the performance of wide-kernel CNNs as the target of a regression task. In this case, we analysed in which way the hyperparameter settings contribute to the performance of the models. To perform regression, the performance expressed as classification accuracy is used as target variable and therefore of continuous nature, on a scale from 0 to 1.0. The variables described in Table 2 are used as independent variables. This results in a feature set of size 7 for predicting model performance. These features are then recentred and rescaled using \log_2 .

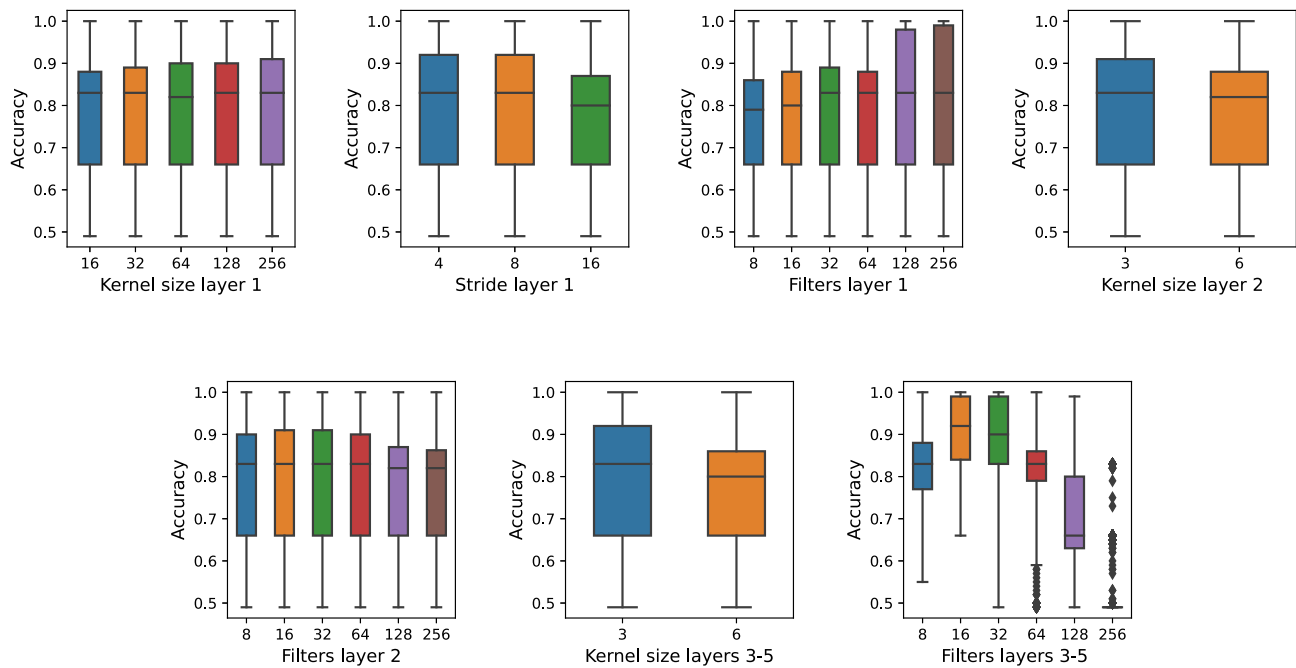


Fig. 6 Distribution of model performance, expressed as accuracy, for the Gearbox use-case visualised for each individual hyperparameter (kernel size, stride, filters)

We assessed regression performance based on mean absolute error (MAE) and mean absolute percentage error (MAPE) since they show the average absolute distance between the predicted accuracy of a wide-kernel CNN and the actual accuracy of a wide-kernel CNN. These metrics are more understandable regarding the scale of the target variable compared to mean squared error.

For the regression task, we compared several machine learning (ML) models with a multi-layer perceptron (DL) model, and a baseline. The results in Table 4 indicate that the ML and DL models were able to learn from the data due to the large performance gap with the baseline model. Therefore, the regression models emphasise that the data collected from the experiments are related to the performance of the model, validating the importance of these hyperparameters for wide-kernel CNNs.

Random forest and MLP show the best performance compared to the other models. However, the random forest explains the variance in the data, expressed as R-squared, clearly better compared to any of the other used models. Furthermore, we see that the K-NN algorithm achieves almost equal results, suggesting that distance-based models are also well-suited for the task at hand.

Interestingly, we find a particularly low performance on the linear regression model in terms of MAE and MAPE, also showing a low explained variance, thus indicating that the data cannot be modelled linearly. Compared with the performance from the other nonlinear methods, we are able

Table 4 Metrics of the regression task to predict wide-kernel CNN model performance using the hyperparameters as input variables

Dataset	Model	R2	MAE	MAPE (%)
CWRU	Mean baseline	0.0000	0.2368	98.40
	Linear regression	0.4038	0.1693	58.93
	Random forest	0.9740	0.0830	27.29
	K-NN	0.8030	0.0849	27.96
	MLP	0.8245	0.0819	26.70
Gearbox	Mean baseline	0.0000	0.1384	20.50
	Linear regression	0.5119	0.0995	13.79
	Random forest	0.9823	0.0401	5.37
	K-NN	0.8405	0.0490	6.75
	MLP	0.8387	0.0510	6.99
Paderborn	Mean baseline	0.0000	0.0936	12.34
	Linear regression	0.4217	0.0703	9.06
	Random forest	0.9613	0.0423	5.48
	K-NN	0.7034	0.0455	5.91
	MLP	0.7363	0.0421	5.42

For the task, we utilised a mean baseline, a linear model, two nonlinear models and a DL model (MLP)

to conclude that the relationship between the dependent and independent variables is nonlinear.

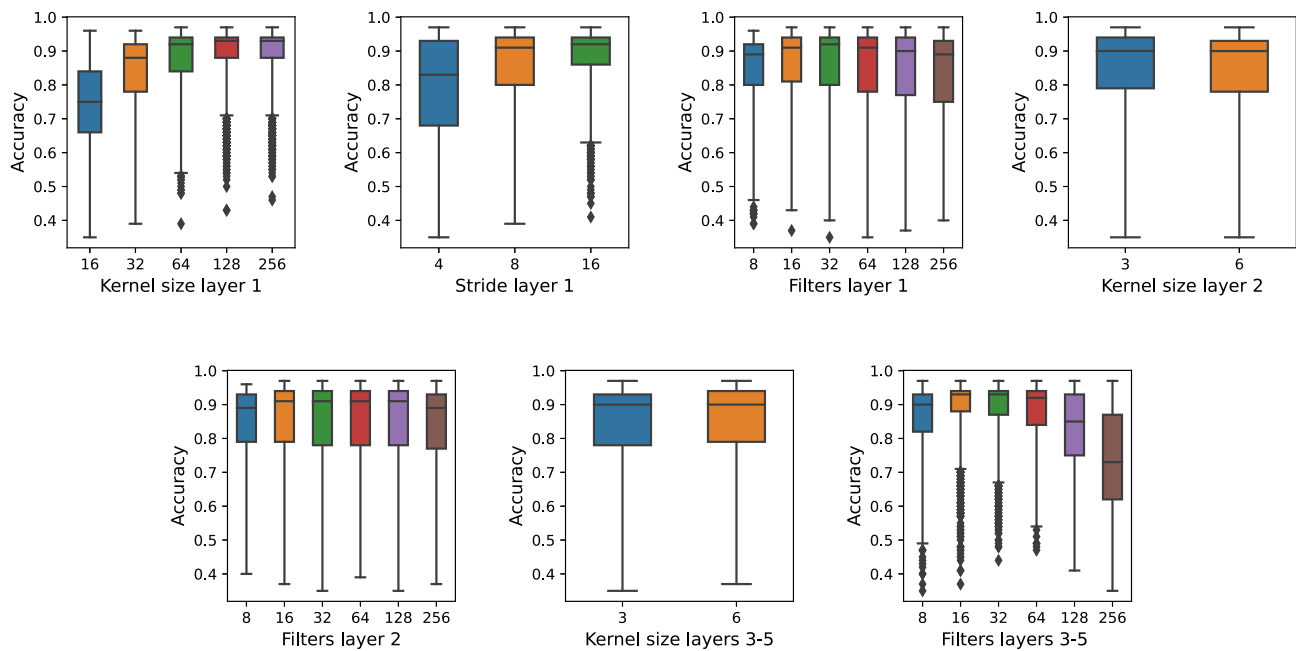


Fig. 7 Distribution of model performance, expressed as accuracy, for the Paderborn use-case visualised for each individual hyperparameter (kernel size, stride, filters)

4.3 Feature importance

As described in Sect. 3.6, we also investigate feature importance by computing Shapley values. We compute Shapley values from the random forest, to assess how important each architectural hyperparameter is for the model. We specifically chose this model for calculating Shapley values due to its relatively low computational costs while still performing very well on all datasets, as can be seen in Sect. 4.2. Features with a larger Shapley value contribute more to the random forest, meaning that they are beneficial for predicting model performance. This procedure involves (re)training the random forest with different combinations of WK-CNN hyperparameters as inputs. To quantify the predictive power, we used the reduction in MAE that each hyperparameter contributes. The importance of hyperparameters varies across the three datasets. Therefore, we review the scores for the datasets individually, which are shown in Fig. 8.

Concerning the Shapley values for the CWRU use-case, the kernel size in layer 1 appears to be the most important feature, accounting for more than half of the reduction in MAE across all features. The number of filters across the network also has an impact, while the stride and the kernel size from layer 2 onwards are unimportant.

When looking at the results for the Gearbox task, the number of filters in layers 3–5 is by far the most important feature, accounting for almost all of the reduction in MAE. Other features have very little value by comparison.

At last, when analysing the Shapley values for Paderborn, the kernel size in layer 1, the stride in layer 1, and the number of filters in layers 3–5 all have a considerable impact. This is similar to what was observed for the CWRU task, although for Paderborn, the stride in layer 1 also appears to have some predictive value.

Across the three datasets, the most consistently valuable feature is the number of filters in layers 3–5, which was in the top three Shapley values for all datasets. This feature has more predictive value than the others combined when considering the Gearbox task. For the other two use-cases, the kernel size in layer 1 also had relatively high Shapley values. Besides these, the other features had relatively small and inconsistent importance. Overall, the results suggest that hyperparameter importance depends on the properties of the data; for example, the importance of the kernel size in layer 1 for CWRU and Paderborn might reflect the fact that these datasets have a similar sampling rate.

4.4 Local maxima

The previous section examined hyperparameters individually to estimate their importance, providing information that can be useful for deciding which hyperparameter to tune first. However, there may be limitations to an approach that tunes hyperparameters individually. To this end, we consider whether the results include local maxima, i. e., configurations which cannot be improved by tuning any individual model

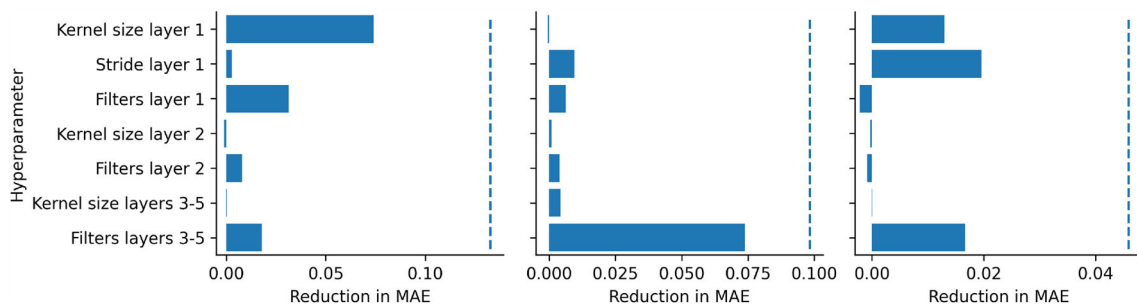


Fig. 8 Shapley values for the contribution each hyperparameter makes to the performance of Random Forest, measured as MAE, for CWRU (left), Gearbox (middle) and Paderborn (right). The dotted line indicates the total reduction in MAE when using all features together

hyperparameter, but which are sub-optimal, meaning there is another configuration which is better.

We search for local maxima and find that they are present within each dataset (for CWRU there are 88, for Gearbox there are 482 and for Paderborn there are 235). Most of these perform comparably to the global optimum, but not necessarily: notably, on the CWRU data, the worst-performing local maximum has less than half the accuracy of the global maximum (0.44 compared to 1.00). This means that there are some models which perform sub-optimally but cannot be improved by changing any single hyperparameter.

The existence of local maxima could be due to the nonlinear interactions between hyperparameter settings and performance, although it could also be impacted by stochastic effects, like the initialisation and the stochastic nature of training neural networks. All of this suggests that the analysis can be developed further by considering the interactions between combinations of hyperparameters, and how they impact performance. This is what we do in the following section, via subgroup discovery.

4.5 Subgroup discovery

In this work, subgroup discovery is used to identify combinations of hyperparameter settings that are particularly indicative of good (or bad) performance on average. The results are presented for the three fault detection use-cases separately. In each case, we show a selection of the best subgroups, omitting worse-performing subgroups and redundant subgroups that overlap highly with those already shown.

Looking at the subgroups for the CWRU use-case (which has an average performance of 0.44) in Table 5, the kernel size of layer 1 is used repeatedly in the definitions of the subgroups. One of the largest subgroups simply applies the condition that the kernel size should be at least 128 and that by itself is already indicative of improved performance at the task. Excluding models which only have 8 filters in layers 3–5 gives a further average improvement, perhaps due to the fact that using only 8 filters gives insufficient modelling capacity. Increasing these hyperparameters (kernel size in layer 1, and

the number of filters in layers 3–5) leads to a higher average performance, suggesting that these two hyperparameters are closely related to performance on this dataset. Looking at the worst subgroups, kernel size in layer 1, when sufficiently small, seems to imply low performance.

Subgroup discovery on the results of the Gearbox experiments (which has an average performance of 0.77) suggests that high performance depends on restricting the number of filters in layers 3–5 (see Table 6). For example, one of the smaller but best-performing subgroups requires that there should be 128 or 256 filters in layer 1 but only 16 or 32 filters in layers 3–5. Having many filters in the latter layers of the model does not appear to be necessary in order to achieve a good score, but instead increases the risk of poor performance, which could be due to overfitting on a limited amount of data. This is consistent with Fig. 6, where performance is generally higher with 16 or 32 filters in the last layers. This visualisation also shows that, although the median performance values are similar for all settings of this hyperparameter, having 128 or 256 filters in layer 1 leads to a higher 75% percentile, suggesting that more of the best-performing models have 128 or 256 filters in layer 1—something that subgroup discovery also indicates.

Amongst WK-CNNs trained on the Paderborn dataset (with an average performance of 0.85), the best subgroups involve a larger kernel size in layer 1 and a smaller number of filters in layers 3–5. An excessively high number of filters in layers 3–5 is suggestive of poor performance. A selection of subgroups is shown in Table 7.

When comparing the results of subgroup discovery across the three use-cases, there are some common themes that emerge. In general, it seems that having only 8 filters in layers 3–5 is insufficient and leads to poor performance. On the Gearbox and Paderborn data, setting this hyperparameter too high also has a tendency to reduce performance on average. The kernel size in layer 1 was another important hyperparameter, with a high kernel size being a condition for high performance on the CWRU and Paderborn datasets, although it is less important for the Gearbox data. On CWRU

Table 5 A selection of the most informative subgroups, and associated statistics, for wide-kernel CNNs trained on CWRU data

Subgroup	Average performance	Size	Quality function
Kernel size layer 1 ≥ 128	0.66	4320	Piatetski-Shapiro
AND filters layers 3–5 ≥ 16			
Kernel size layer 1 ≥ 128	0.62	5184	Piatetski-Shapiro
Kernel size layer 1 = 256	0.82	1152	Lift
AND filters layer 1 ≥ 32			
AND filters layers 3–5 ≥ 32			
Kernel size layer 1 ≤ 64	0.27	5184	Piatetski-Shapiro*
AND filters layer 1 ≥ 32			
Kernel size layer 1 ≤ 64	0.21	3888	Piatetski-Shapiro*
AND filters layer 1 ≥ 64			

Asterisks by the quality function indicate a search for worst-performing subgroups

Table 6 A selection of the most informative subgroups, and associated statistics, for wide-kernel CNNs trained on Gearbox data

Subgroup	Average performance	Size	Quality function
Filters layers 3–5 ≤ 64	0.86	8640	Piatetski-Shapiro
Filters layer 1 ≥ 128	0.95	1440	Lift
AND 16 \leq Filters layers 3–5 ≤ 32			
Filters layers 3–5 ≥ 128	0.60	4320	Piatetski-Shapiro*

Asterisks by the quality function indicate a search for worst-performing subgroups

Table 7 A selection of the most informative subgroups, and associated statistics, for wide-kernel CNNs trained on Paderborn data

Subgroup	Average performance	Size	Quality function
Kernel size layer 1 ≥ 32	0.90	8640	Piatetski-Shapiro
AND Filters layer 3–5 ≤ 128			
Kernel size layer 1 ≥ 128	0.93	1728	Lift
AND 16 \leq Filters layers 3–5 ≤ 32			
Filters layers 3–5 ≥ 128	0.73	2880	Piatetski-Shapiro*
AND stride layer 1 ≤ 8			

Asterisks by the quality function indicate a search for worst-performing subgroups

and Paderborn, the best results were observed when optimising these two hyperparameters in conjunction.

4.6 Visualising hyperparameter space based on performance

To get a better understanding of similar model configurations that have a highly different performance, we visualise the hyperparameter space. For this, we use the 7 controlled hyperparameters as described in Table 2. To visualise the data, we use T-SNE algorithm [72] to reduce the dimensional space, which makes it possible to plot the data in 2D. The colour represents the performance of a configuration, shifting from dark blue (low performance) to dark red (high performance). Figure 9 shows these lower dimensional model configurations for the three use-cases.

The cloud visualisations in Fig. 9 show that compressing the data for all datasets results in very little clustering, which is normally expected when reducing the dimensionality of

data with T-SNE. One particular reason for this is that the input for the T-SNE algorithm, i. e., the 7 hyperparameters controlled in this study, are of ordinal nature and are evenly spaced out across the seven-dimensional grid space, meaning that there are no regions of higher or lower density which would form clusters. However, the visualisation still provides insights based on the colour-coded performance of every data point.

For the CWRU visualisation, we see that most configurations are assigned to the lower performance boundaries, i. e., blue coloured. Good-performing models (coloured in red) are scattered throughout the whole cloud, suggesting that good and bad configurations in terms of model performance are closely related.

For Gearbox, we see different behaviour since the general performance is much higher than for CWRU. A clear separation between high- and low-performing models is evident based on the colours, e. g., it seems that low performance is more situated in the left part of the data cloud while in the

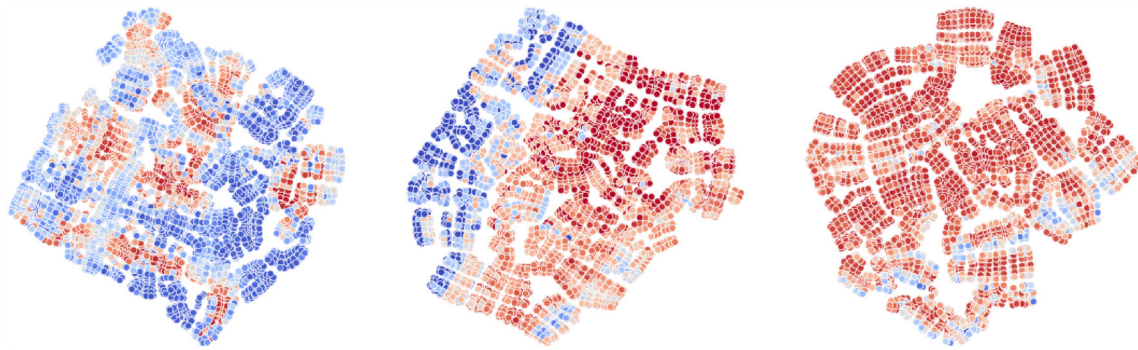


Fig. 9 Cloud visualisations of the hyperparameter space compressed to two dimensions using T-SNE algorithm for CWRU (left), Gearbox (middle) and Paderborn (right). The colours shift from lowest performance (dark blue) to highest performance (dark red)

centre and right position of the two-dimensional space, we mostly find high-performing models. This suggests that there are differences in the hyperparameter configurations between low and high performance.

Lastly, for the Paderborn use-case, we mostly see high performing models. Nonetheless, the poor-performing models are mostly positioned in the lower part of the data cloud. We also find that medium-performing configurations tend to appear between regions of good and bad performance.

The cloud visualisations indicate that some good- and bad-performing models are close by, suggesting that small nuances in the configurations of the model, i. e., hyperparameter adaptations, can result in drastic performance change. This becomes more evident in Fig. 10, where we zoomed in on the hyperparameter space and highlighted two very similar model configurations in terms of hyperparameters, while having a considerable difference in performance. The characteristics of these examples are further displayed in Table 8.

Looking at the examples, we see a drastic performance change when adjusting the hyperparameters for the use-cases where data are sampled at a high frequency (CWRU and Paderborn), while being less evident for Gearbox. However, many other examples of these so-called edge cases can be detected in Fig. 9, due to the close proximity between high- and low-performing models. It is therefore required to handle the hyperparameter tuning of a wide-kernel CNN model with care.

4.7 Categories of performance

Categorising the target variable (model performance) can be seen as a preprocessing step for follow-up analysis based on “good”- and “bad”-performing groups, as further described in Sect. 4.8. In order to convert the target variable, i. e., the accuracy, into a discrete variable, we fit a mixture of Gaussians. This acts as a form of clustering based purely on the accuracy. We use this method because it is flexible and capable of approximating a range of distributions.

After selecting a value for parameter n , the expectation maximisation algorithm is used to fit a Gaussian mixture model to find the means and standard deviations for n Gaussian distributions that together fit the data best [73]. We computed the silhouette score to find the optimal value for this parameter n . On the results from the CWRU data, this procedure leads to 6 categories. On the accuracies obtained when training on the Gearbox data, 4 categories were found. Applying the same approach to the Paderborn use-case, 3 categories were found. The categories for each dataset with their respective mean and standard deviation are described in Table 9.

4.8 Modelling hyperparameter space as a graph

Visualising the model performance in Sect. 4.6 indicated that many edge cases at the border between high and low performance exist, suggesting that their hyperparameters, i. e., model configuration, are closely related. Thus, small adjustments to hyperparameters can have a large effect on model performance. We now investigate this further. Specifically, we look in more detail at how likely it is to observe a change in the performance as a result of a small change in a single hyperparameter. If this is unlikely, then it seems that most wide-kernel CNNs exist in a region of the hyperparameter space where performance is quite consistent, and not affected by small perturbations to the hyperparameter settings. For this, we take the categorised performance described in Sect. 4.7 to show how many steps are necessary to jump from one category to another. For example, we examine how many small changes can be made to a ‘high’-performing WK-CNN before it becomes ‘low’-performing or some other change occurs.

For our analysis, we use the graph-like representation of the data (described in Sect. 3.8), in which each WK-CNN is a node and models that change by halving/doubling a single hyperparameter are connected by edges. Using this representation of the results we perform random walks, which



Fig. 10 Zoomed-in visualisations of the hyperparameter space compressed using T-SNE algorithm for CWRU (left), Gearbox (middle) and Paderborn (right). The two highlighted dots exemplary show edge cases

with low distance in the two-dimensional space based on their similarity in terms of hyperparameter configuration, while having a considerable performance difference. These edge cases are further detailed in Table 8

Table 8 Example of two edge case model configurations for all datasets in terms of model performance, as highlighted in Fig. 10

Dataset	Hyperparameters							Acc.
	Kernel size Layer 1	Stride Layer 1	Filters Layer 1	Filters Layer 2	Kernel size Layer 2	Filters Layers 3–5	Kernel size Layers 3–5	
CWRU	256	16	64	16	3	32	3	0.83
	256	16	64	16	3	16	6	0.49
Gearbox	256	16	64	16	3	32	3	0.99
	256	16	64	16	3	16	6	0.85
Paderborn	256	16	64	16	3	32	3	0.75
	256	16	64	16	3	16	6	0.48

The hyperparameters in bold show the change in configuration between the examples

Table 9 Categorising the results from the three datasets

Dataset	Category	Mean performance	Std	Count
CWRU	Very low	0.13	0.0009	3166
	Low	0.27	0.0045	2753
	Medium low	0.42	0.0037	2132
	Medium high	0.58	0.0043	1737
	High	0.75	0.0049	1621
	Very high	0.92	0.0019	1551
Gearbox	Very Low	0.49	0.000002	2251
	Low	0.67	0.0026	2476
	High	0.84	0.0016	5500
	Very high	0.97	0.0009	2733
Paderborn	Low	0.64	0.0076	2154
	Medium	0.83	0.0037	4188
	High	0.93	0.0002	6618

'Count' refers to the number of data points, i.e., the different wide-kernel CNN configurations, assigned to a category

progress from a starting wide-kernel CNN and terminate when the walk reaches a model with a different performance category from the initial wide-kernel CNN. The length of such walks gives an idea of how many small changes can be performed without the category being changed. A longer walk means that more small changes have been performed before the category changes (e.g. before moving from a low-performing model to a medium-performing model). In order to aggregate this information, we first take the median path length per starting WK-CNN and then we take the mean across each performance category. We have 10 random walks per model configuration (a total of 12,960 times 10), and the median is used to avoid the impact of outliers. A baseline to compare with is established by randomly shuffling the category labels and then repeating the analysis. Shuffling removes any tendency for similar models to have the same performance category, and thus provides a baseline path length. This gives a first set of results from the graph-like representation.

A second analysis is to look at whether path lengths tend to correlate with performance within a category, which gives an insight into whether the stability (robustness to small changes) of the hyperparameter settings is indicative of performance, even within a region of similar hyperparameter settings. For each model, we look at the path length and the performance score it achieved, and calculate the correlation of these two variables. A high path length implies that a wide-kernel CNN is far from other performance categories, and thus relatively central within its performance category. Therefore, the correlations give an indication whether models that are ‘further inside’ a stable region tend to have better performance on average. These correlation coefficients, combined with the path lengths, show how small changes to the hyperparameters are related to model performance.

The results are fairly consistent across the datasets. The average path lengths are higher than the corresponding baseline values, as would be expected if models that only differ by a single hyperparameter change have similar performance. As shown in Table 10, this is especially true for the worst-performing and best-performing categories in each dataset. In these categories, multiple small changes are usually needed before the performance noticeably increases or decreases. The best- and worst-performing regions of hyperparameter space are the most stable in this respect, meaning that multiple small changes can be made without seriously affecting performance. These categories also have the highest correlation between path length and performance, meaning that, for example, the most stable configurations of hyperparameters within the best-performing category are even better than less stable configurations belonging to the same category.

For the Paderborn use-case, for example, models within the worst-performing category displayed a correlation of -0.56 between path length and performance, and the best-

performing category gave a correlation of 0.45 . A similar phenomenon is observed for the other use-cases. This suggests that models that are ‘further inside’ a good-performing region have even better performance, and the contrary is true for the worst-performing regions of hyperparameter space, as can be seen in Table 10.

5 Conclusions

This work investigated the influence of architectural hyperparameters for a one-dimensional CNN in signal classification tasks. More specifically, the work focused on identifying faults in industrial machinery. For this, we executed an extended hyperparameter search using a wide-kernel CNN structure derived from previous work by [3, 6] which has been proven to perform very well in signal classification tasks involving bearing element faults. The models with differing hyperparameter configurations were applied to three datasets, with varying characteristics, as shown in Table 1. Nonetheless, the underlying task for each dataset was equivalent; they all measured vibrations from industrial equipment.

Given this base wide-kernel CNN architecture, this work showed the impact of architectural hyperparameters such as the number of filters, kernel size and stride in multiple convolutional layers. The hyperparameter search was performed to identify how much influence individual hyperparameters had on the performance of the model, but also to enable follow-on analysis such as a search for interesting combinations of hyperparameters in the form of patterns. In total, there were 12,960 different hyperparameter combinations tested for each of the three available datasets, leading to 38,880 trained wide-kernel CNNs altogether. The results were thoroughly analysed for each individual dataset. We specifically chose to keep the results separated in this way, since the outcomes were dataset-dependent.

In the analysis, we first visualised the distribution of performance of the models for each individual hyperparameter setting, using box plots. Based on these visualisations, an optimum could be detected for the amount of filters, especially in the deeper part of the network (layers 3–5). Building on this, we formulated model performance as a regression task using the controlled hyperparameters as independent variables and the corresponding performance of a wide-kernel CNN as the target. First, the trained algorithms performed better compared to a simple baseline, implying that there is a learnable relationship between the hyperparameters and model performance. Second, the results of the regression task showed that there are nonlinear relationships in the data of the hyperparameter search due to the significant performance gap between the linear and nonlinear algorithms.

Table 10 Median length of random walks before the performance category changes, averaged across all models belonging to the category that the walks started from

Dataset	Category		Average path length	Average baseline	Correlation to performance
CWRU	Very low	(0.12)	4.60	1.02	−0.44
	Low	(0.26)	1.68	1.02	0.04
	Medium low	(0.40)	1.23	1.02	−0.02
	Medium high	(0.54)	1.18	1.01	0.03
	High	(0.72)	1.34	1.02	0.08
	Very high	(0.91)	2.28	1.02	0.40
Gearbox	Very Low	(0.49)	3.72	1.15	−0.16
	Low	(0.67)	1.52	1.16	−0.17
	High	(0.84)	2.58	1.16	0.10
	Very high	(0.98)	2.34	1.16	0.39
Paderborn	Low	(0.64)	1.49	1.32	−0.56
	Medium	(0.83)	1.61	1.33	−0.04
	High	(0.93)	4.92	1.34	0.45

Each walk begins at a model with a specific performance category and finishes when a model with a different category is reached. Correlation between path length and performance within each category is shown in the final column

Furthermore, a closer look into the importance of individual features, using random forest to calculate Shapley values, showed that especially the filters in the convolutional layers 3–5 generally influence model performance the most.

In addition, the kernel size in layer 1 also influences model performance. For the CWRU and Paderborn task in particular, it appears that a very high kernel size (around $\frac{1}{8}$ of the total input signal) yields the best performance, while not being influential for the Gearbox use-case. An explanation for this lies with the sampling frequency of the vibration data, which is much lower for the latter dataset than the other two, suggesting that a wide-kernel becomes more relevant when the input data are more densely sampled.

Extending the analysis, we searched for subgroups in the data to see if any specific model configuration, i.e., combinations of hyperparameters, are more likely to result in a good performance. This yielded results that were consistent with the feature importance analysis and the box plots of the distribution of performance scores for each hyperparameter setting individually. The number of filters in layers 3–5 is important across all use-cases, and there is an optimal value for it in each case, around 32. The kernel size in layer 1 was important for the higher sampling rate CWRU and Paderborn data, and increasing this hyperparameter is beneficial. On the Gearbox data, the number of filters in layer 1 appears to interact with other hyperparameters; by itself, changing this hyperparameter does not increase median performance; however, having many filters in layer 1 does give noticeable benefits when combined with an optimal number of filters in layers 3–5.

Furthermore, visualising the hyperparameter space in 2D using the T-SNE algorithm showed that good- and bad-

performing models are close by, suggesting that many edge cases can be detected with large performance gaps, while having similar hyperparameter configurations. This emphasises on the importance of thorough CNN modelling and testing when searching for an optimal performing model.

Finally, this paper also considered how robust the performance of a wide-kernel CNN is to small changes to the hyperparameters, when categorising model performance. We found that there is some consistency in the performance of similar combinations of hyperparameters, meaning that numerous small changes might be needed to go from a poor-performing to a well-performing model and vice versa. In other words, there seem to be relatively consistent regions of hyperparameter space, such as poor-performing regions. Additionally, the correlation between path length and model performance indicated that wide-kernel CNNs which were further inside a poor-performing region had the worst performance; likewise wide-kernel CNNs further inside a good-performing region were the best.

In general, our results indicate that the filters in the last layers of the model are most influential on model performance and that in certain cases utilising a wide kernel in the first convolutional layer, with an extremely high value, can improve model performance.

For future work, we aim to extend the hyperparameter search towards more signal classification tasks. Furthermore, to generalise across different types of neural network architectures, another interesting direction to consider involves the investigation of the behaviour of hyperparameters in attention-based models [9]. Finally, we are interested in additional data recording methods that could benefit the field of industrial fault detection. For example, acoustic recordings or

(thermal/infrared) images retrieved from machinery are two ubiquitous forms of data that are often used in data analysis, which lend themselves for automated learning techniques such as wide-kernel CNNs.

Author Contributions JH, SB and MA contributed in conceptualising the idea and design of the study. JH, SB and DH designed the methodology of the study. JH retrieved, prepared and processed the data used for the experiments. JH and DH collaborated on the analysis of the experiment results. JH and DH drafted the manuscript. All authors reviewed and edited the manuscript. JH implemented the methods and algorithms and ran the experiments, in collaboration with SB and DH. All authors read, reviewed and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability Data are available at [58] (CWRU), [65] (Gearbox) and [66] (Paderborn).

Code availability There is a Github page available with the corresponding code (<https://github.com/JvdHoogen/HyperparameterAnalysisWCNN>).

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. *Nature* **323**(6088), 533–536 (1986)
- LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. *Handbook Brain Theory Neural Netw.* **3361**(10), 1995 (1995)
- van den Hoogen, J., Bloemheuvel, S., Atzmueller, M.: Classifying multivariate signals in rolling bearing fault detection using adaptive wide-kernel CNNs. *Appl. Sci.* **11**(23), 11429 (2021). <https://doi.org/10.3390/app112311429>
- Zhao, D., Wang, T., Chu, F.: Deep convolutional neural network based planet bearing fault classification. *Comput. Ind.* **107**, 59–66 (2019)
- Zhao, R., Yan, R., Chen, Z., Mao, K., Wang, P., Gao, R.X.: Deep learning and its applications to machine health monitoring. *Mech. Syst. Signal Process.* **115**, 213–237 (2019)
- van den Hoogen, J.O.D., Bloemheuvel, S.D., Atzmueller, M.: An improved wide-kernel cnn for classifying multivariate signals in fault diagnosis. In: 2020 International Conference on Data Mining Workshops (ICDMW), pp. 275–283 (2020). <https://doi.org/10.1109/ICDMW51313.2020.00046>
- Zhang, W., Peng, G., Li, C., Chen, Y., Zhang, Z.: A new deep learning model for fault diagnosis with good anti-noise and domain adaptation ability on raw vibration signals. *Sensors* **17**(2), 425 (2017)
- Li, R., Wu, J., Li, Y., Cheng, Y.: Periodnet: Noise-robust fault diagnosis method under varying speed conditions. *IEEE Transactions on Neural Networks and Learning Systems*, 1–15 (2023). <https://doi.org/10.1109/TNNLS.2023.3274290>
- Zhao, Z., Jiao, Y., Zhang, X.: A fault diagnosis method of rotor system based on parallel convolutional neural network architecture with attention mechanism. *J. Signal Process. Syst.* 1–13 (2023)
- Qiang, R., Zhao, X.: An intelligent diagnosis method for rolling bearings based on ghost module and adaptive weighting module (2023)
- Chen, S., Yu, J., Wang, S.: One-dimensional convolutional auto-encoder-based feature learning for fault diagnosis of multivariate processes. *J. Process Control* **87**, 54–67 (2020). <https://doi.org/10.1016/j.jprocont.2020.01.004>
- IEEE Motor Reliability Working Group.: Report of large motor reliability survey of industrial and commercial installations, Part I. *IEEE Trans. Ind. Appl.* **1**(4), 865–872 (1985)
- Yin, S., Li, X., Gao, H., Kaynak, O.: Data-based techniques focused on modern industry: an overview. *IEEE Trans. Ind. Electron.* **62**(1), 657–667 (2014)
- Pandhare, V., Singh, J., Lee, J.: 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), pp. 320–326 (2019). <https://doi.org/10.1109/PHM-Paris.2019.00061>
- Jing, L., Zhao, M., Li, P., Xu, X.: A convolutional neural network based feature learning and fault diagnosis method for the condition monitoring of gearbox. *Measurement* **111**, 1–10 (2017)
- Lei, Y., Jia, F., Lin, J., Xing, S., Ding, S.X.: An intelligent fault diagnosis method using unsupervised feature learning towards mechanical big data. *IEEE Trans. Ind. Electron.* **63**(5), 3137–3147 (2016)
- You, D., Gao, X., Katayama, S.: Wpd-pca-based laser welding process monitoring and defects diagnosis by using fnn and svm. *IEEE Trans. Ind. Electron.* **62**(1), 628–636 (2014)
- Guo, S., Yang, T., Gao, W., Zhang, C.: A novel fault diagnosis method for rotating machinery based on a convolutional neural network. *Sensors* **18**(5), 1429 (2018)
- Pandya, D., Upadhyay, S., Harsha, S.P.: Fault diagnosis of rolling element bearing with intrinsic mode function of acoustic emission data using apf-knn. *Expert Syst. Appl.* **40**(10), 4137–4145 (2013)
- Zhou, Z., Wen, C., Yang, C.: Fault detection using random projections and k-nearest neighbor rule for semiconductor manufacturing processes. *IEEE Trans. Semicond. Manuf.* **28**(1), 70–79 (2015). <https://doi.org/10.1109/TSM.2014.2374339>
- Wang, Z., Zhang, Q., Xiong, J., Xiao, M., Sun, G., He, J.: Fault diagnosis of a rolling bearing using wavelet packet denoising and random forests. *IEEE Sensors J.* **17**(17), 5581–5588 (2017)
- Huang, J., Hu, X., Yang, F.: Support vector machine with genetic algorithm for machinery fault diagnosis of high voltage circuit breaker. *Measurement* **44**(6), 1018–1027 (2011)
- Konar, P., Chattopadhyay, P.: Bearing fault detection of induction motor using wavelet and support vector machines (svms). *Appl. Soft Comput.* **11**(6), 4203–4211 (2011)
- Santos, P., Villa, L.F., Reñones, A., Bustillo, A., Maudes, J.: An svm-based solution for fault detection in wind turbines. *Sensors* **15**(3), 5627–5648 (2015)
- Chow, M.Y., Mangum, P.M., Yee, S.O.: A neural network approach to real-time condition monitoring of induction motors. *IEEE Trans. Ind. Electron.* **38**(6), 448–453 (1991)
- Cococcioni, M., Lazzarini, B., Volpi, S.L.: Robust diagnosis of rolling element bearings based on classification techniques. *IEEE Trans. Ind. Inform.* **9**(4), 2256–2263 (2013)

27. Hajnayeb, A., Ghasemloonia, A., Khadem, S., Moradi, M.: Application and comparison of an ann-based feature selection method and the genetic algorithm in gearbox fault diagnosis. *Expert Syst. Appl.* **38**(8), 10205–10209 (2011)
28. Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G.: Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148* (2016)
29. Yao, P., Yang, S., Li, P.: Fault diagnosis based on rsetnet-lstm for industrial process. In: 2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), vol. 5, pp. 728–732 (2021). <https://doi.org/10.1109/IAEAC50856.2021.9391030>
30. Liu, Y., Yan, X., Zhang, C.-A., Liu, W.: An ensemble convolutional neural networks for bearing fault diagnosis using multi-sensor data. *Sensors* **19**(23), 5300 (2019). <https://doi.org/10.3390/s19235300>
31. Zhang, W., Peng, G., Li, C.: Rolling element bearings fault intelligent diagnosis based on convolutional neural networks using raw sensing signal, 77–84 (2017)
32. Chen, R., Huang, X., Yang, L., Xu, X., Zhang, X., Zhang, Y.: Intelligent fault diagnosis method of planetary gearboxes based on convolution neural network and discrete wavelet transform. *Comput. Ind.* **106**, 48–59 (2019)
33. Ince, T., Kiranyaz, S., Eren, L., Askar, M., Gabbouj, M.: Real-time motor fault detection by 1-d convolutional neural networks. *IEEE Trans. Ind. Electron.* **63**(11), 7067–7075 (2016)
34. Zhang, A., Li, S., Cui, Y., Yang, W., Dong, R., Hu, J.: Limited data rolling bearing fault diagnosis with few-shot learning. *IEEE Access* **7**, 110895–110904 (2019)
35. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324 (1998)
36. Kiranyaz, S., Ince, T., Hamila, R., Gabbouj, M.: Convolutional neural networks for patient-specific ecg classification. In: *Proc. IEEE EMBC*, pp. 2608–2611 (2015). IEEE
37. Kandel, I., Castelli, M.: The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express* **6**(4), 312–315 (2020). <https://doi.org/10.1016/j.icte.2020.04.010>
38. Xiao, X., Yan, M., Basodi, S., Ji, C., Pan, Y.: Efficient Hyperparameter Optimization in Deep Learning Using a Variable Length Genetic Algorithm (2020)
39. Adedigba, A.P., Adeshina, S.A., Aina, O.E., Aibinu, A.M.: Optimal hyperparameter selection of deep learning models for covid-19 chest x-ray classification. *Intell. Based Med.* **5**, 100034 (2021). <https://doi.org/10.1016/j.ibmed.2021.100034>
40. Raziani, S., Azimbagirad, M.: Deep CNN hyperparameter optimization algorithms for sensor-based human activity recognition. *Neurosci. Inform.* **2**(3), 100078 (2022)
41. Yi, H., Bui, K.-H.N.: An automated hyperparameter search-based deep learning model for highway traffic prediction. *IEEE Trans. Intell. Transp. Syst.* **22**(9), 5486–5495 (2020)
42. Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. *Data Min. Knowl. Discov.* **15**(1), 55–86 (2007)
43. Fournier-Viger, P., Lin, J.C.-W., Kiran, R.U., Koh, Y.S., Thomas, R.: A survey of sequential pattern mining. *Data Sci. Pattern Recognit.* **1**(1), 54–77 (2017)
44. Atzmueller, M.: Subgroup discovery. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **5**(1), 35–49 (2015)
45. Herrera, F., Carmona, C.J., González, P., Del Jesus, M.J.: An overview on subgroup discovery: foundations and applications. *Knowl. Inf. Syst.* **29**, 495–525 (2011)
46. Atzmueller, M., Puppe, F., Buscher, H.-P.: Towards knowledge-intensive subgroup discovery. In: *LWA*, pp. 111–117 (2004). Citeseer
47. Lavrač, N., Cestnik, B., Gamberger, D., Flach, P.: Decision support through subgroup discovery: three case studies and the lessons learned. *Mach. Learn.* **57**, 115–143 (2004)
48. Atzmueller, M.: Compositional subgroup discovery on attributed social interaction networks. In: *Discovery Science: 21st International Conference, DS 2018, Limassol, Cyprus, October 29–31, 2018, Proceedings 21*, pp. 259–275 (2018). Springer
49. Hudson, D., Wiltshire, T.J., Atzmueller, M.: Local exceptionality detection in time series using subgroup discovery: an approach exemplified on team interaction data. In: *Discovery Science: 24th International Conference, DS 2021, Halifax, NS, Canada, October 11–13, 2021, Proceedings 24*, pp. 435–445 (2021). Springer
50. Jin, N., Flach, P., Wilcox, T., Sellman, R., Thumim, J., Knobbe, A.: Subgroup discovery in smart electricity meter data. *IEEE Trans. Ind. Inform.* **10**(2), 1327–1336 (2014)
51. Rodríguez, D., Ruiz, R., Riquelme, J.C., Harrison, R.: A study of subgroup discovery approaches for defect prediction. *Inf. Softw. Technol.* **55**(10), 1810–1822 (2013)
52. Rodríguez, D., Ruiz, R., Riquelme, J.C., Aguilar-Ruiz, J.S.: Searching for rules to detect defective modules: a subgroup discovery approach. *Inf. Sci.* **191**, 14–30 (2012)
53. Millot, A., Cazabet, R., Boulicaut, J.-F.: Exceptional model mining meets multi-objective optimization. In: *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pp. 378–386 (2021). SIAM
54. Harris, T.A.: *Rolling Bearing Analysis*. Wiley, New York (1991). OCLC: 21373436
55. Tong, Z., Li, W., Zhang, B., Zhang, M.: Bearing fault diagnosis based on domain adaptation using transferable features under different working conditions. *Shock and Vibration* **2018** (2018)
56. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014)
57. Hui, L., Belkin, M.: Evaluation of Neural Architectures Trained with Square Loss versus Cross-Entropy in Classification Tasks (2021)
58. CWRU Dataset; Case Western Reserve University Bearing Data Center. Available: <https://csegroups.case.edu/bearingdatacenter/home>
59. Ocak, H., Loparo, K.A.: Estimation of the running speed and bearing defect frequencies of an induction motor from vibration data. *Mech. Syst. Signal Process.* **18**(3), 515–533 (2004)
60. Ocak, H., Loparo, K.A.: Estimation of the running speed and bearing defect frequencies of an induction motor from vibration data. *Mech. Syst. Signal Process.* **18**(3), 515–533 (2004). [https://doi.org/10.1016/S0888-3270\(03\)00052-9](https://doi.org/10.1016/S0888-3270(03)00052-9)
61. Neupane, D., Seok, J.: Bearing fault detection and diagnosis using case western reserve university dataset with deep learning approaches: a review. *IEEE Access* **8**, 93155–93178 (2020)
62. Liu, X., Huang, H., Xiang, J.: A personalized diagnosis method to detect faults in a bearing based on acceleration sensors and an fem simulation driving support vector machine. *Sensors* **20**(2), 420 (2020)
63. Piltan, F., Kim, J.-M.: Svm-based hybrid robust pio fault diagnosis for bearing. In: *International Conference on Intelligent and Fuzzy Systems*, pp. 858–866 (2020). Springer
64. Malik, H., Pandya, Y., Parashar, A., Sharma, R.: Feature extraction using emd and classifier through artificial neural networks for gearbox fault diagnosis. In: *Applications of Artificial Intelligence Techniques in Engineering: SIGMA 2018, Volume 2*, pp. 309–317 (2019). Springer
65. Pandya, Y.: Gearbox Fault Diagnosis Data (2018). <https://data.openei.org/submissions/623>
66. Lessmeier, C., Kimotho, J.K., Zimmer, D., Sextro, W.: 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* **3**(1) (2016)
67. Shapley, L.S., et al.: A value for n-person games (1953)
 68. Brandsæter, A., Glad, I.K.: Shapley values for cluster importance. *Data Mining and Knowledge Discovery*, 1–32 (2022)
 69. Piatetsky-Shapiro, G., Matheus, C.J.: The interestingness of deviations. In: *Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases*, vol. 1, pp. 25–36 (1994)
 70. Atzmueller, M., Lemmerich, F.: Fast subgroup discovery for continuous target concepts. In: *Foundations of Intelligent Systems: 18th International Symposium, ISMIS 2009, Prague, Czech Republic, September 14–17, 2009. Proceedings* 18, pp. 35–44 (2009). Springer
 71. Hudson, D., Wiltshire, T.J., Atzmueller, M.: Visualization methods for exploratory subgroup discovery on time series data. In: *Bio-inspired Systems and Applications: from Robotics to Ambient Intelligence*, pp. 34–44. Springer, Cham (2022)
 72. Maaten, L.V.D., Hinton, G.: Visualizing data using t-sne. *J. Mach. Learn. Res.* **9**, 2579–2605 (2008)
 73. Daumé, H.: *A Course in Machine Learning*. Hal Daumé III, Online (2017). <http://ciml.info/>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.