VIBRATIONS-BASED MACHINE FAULT DIAGNOSIS AND PROGNOSIS USING CONVOLUTIONAL NEURAL NETWORKS

Jacob Hendriks

A thesis submitted in partial fulfillment of the requirements for the

MASTER OF APPLIED SCIENCE
degree in Mechanical Engineering

Ottawa-Carleton Institute for Mechanical and Aerospace Engineering
Faculty of Engineering
University of Ottawa

Abstract

This thesis addresses vibration-based machine health monitoring (MHM) by applying the fundamentals of machine learning (ML), convolutional neural networks (CNNs) and selected signal processing. The thesis first presents an exploration of the relationship between the hyperparameters of two-layer CNNs, the type of signal preprocessing used, and resulting diagnostic accuracy. For this, two popular bearing fault datasets and a gear fault dataset are used to reveal cross-domain trends. It is found that using time-frequency representations provided by the spectrogram transformation results in a reduced dependence on hyperparameter optimization and lays the foundation for the following work. Moreover, by applying ML theory and best practices, the thesis demonstrates shortcomings in currently accepted benchmarking practices to evaluate the domain adaptability of bearing fault diagnosis algorithms and proposes an alternative benchmarking framework to resolve them. A novel data preparation and transfer learning procedure that capitalizes on the use of multiple sensors and that achieves higher accuracy than state-of-the-art algorithms is demonstrated. In addition to fault diagnosis, the thesis addresses bearing health prognosis by applying CNNs to health indicator estimation using data from accelerated life testing. Several data augmentation methods adapted from other ML fields are compared. It is determined that methods proven in sound classification or image recognition fields are not guaranteed to benefit this task. Lastly, the thesis presents a 3D CNN designed for bearing health prognosis that uses a multi-sensor time-frequency input to improves upon single-sensor variants. The thesis explores the strengths, as well as the shortcomings, of CNNs for MHM, an emphasis is placed on network design, signal transformation, and experimental methodology.

Acknowledgments

I wish to thank my supervisor Dr. Patrick Dumond for his support, guidance, and tireless editing throughout the process of preparing my articles and this thesis. I also wish to thank my parents for their encouragement and financial backing, despite my own lack of confidence, without either of which I would certainly not have been able to pursue this degree. My wonderful siblings and dear friend Kat I thank for their unfailing encouragement and support of my quest to sort out the bad from the good vibes.

Table of Contents

List of Figures

List of Tables

Nomenclature

ANN:	Artificial Neural Network
CNN:	Convolutional Neural Network
CWRU:	Case Western Reserve University
FCF:	Fault Characteristic Frequency
FFT:	Fast Fourier Transform
FPT:	First Prediction Time
GN:	Gaussian Noise
GUI:	Graphical User Interface
HI:	Health Indicator
LSTM:	Long Short-Term Memory
MHM:	Machine Health Monitoring
ML:	Machine Learning
MN:	Masking Noise
PS:	Pitch Shifting
RM:	Region Masking
RMSE:	Root Mean Square Error
RUL:	Remainng Useful Lifetime

Chapter 1

Introduction

1.1 Background

Data-driven machine health monitoring (MHM) allows machine operators to improve capitalization of a mechanical asset's useful lifetime and avoid unexpected interruptions to machine operability by detecting mechanical faults and predicting component remaining useful life (RUL) in advance of a machine breakdown. Traditionally, signals gathered from machine-mounted sensors are analyzed by system experts to determine whether maintenance action should be taken. This involves first gathering and pre-processing the signals, extracting useful features from the signals, and then analyzing the features to determine whether a fault is present and the expected time delay before the fault becomes significant. Current research for advancing MHM is focused on improving and automating aspects of MHM, especially feature selection and analysis, to reduce the dependence on human experts [1].

Bearings and gears are common subjects of interest for MHM since they are nearly ubiquitous in machine design and are accountable for a large proportion of machine failures [2]. Vibration, temperature, current, oil analysis, and acoustic emissions are among the types of signals used in MHM. Vibration signals are the most widely investigated signal type for their convenience, affordability, and because vibration signals carry useful information about fault type and severity. Oil analysis is also a powerful paradigm for condition-based maintenance in cases where metal debris detected in lubrication can be related to the development of mechanical faults [3]. However, sensors for oil analysis incur greater upfront

and running costs and are more invasive than vibration sensors, which can be installed externally on many machines.

The main categories of MHM that use vibration data include model-based, data-driven, and combination systems [4]. Model-based systems are preferable where limited data is available to derive data-driven solutions and where the underlying system is sufficiently understood so as to be modelled well. Data-driven models are preferred for difficult-to-model systems and where it is easy to gather descriptive data. Combination systems involve a compromise of these considerations [5].

Machine learning (ML) is regarded as a suitable tool for use in data-driven MHM for its ability to detect complex patterns contained in vibration signals. Classical applications of ML using hand-crafted features such as k-nearest neighbours (kNN) and support vector machines (SVM) have been widely explored and perform well in many fault diagnosis problems [6]. However, the performance of these shallow networks is highly dependant on the quality of the selected features and whether they contain useful diagnostic information. Generally, some domain expertise is required in order to manually select appropriate features [7]. Performance has been shown to improve when using automated feature selection, which can be done by integrating feature selection into the training process (feature learning) [8], but can also be done separately using dimensional reduction techniques or, increasingly, separate ML algorithms like autoencoders [6].

Convolutional neural networks (CNNs) are a type of feature learning algorithm that perform well for abstract tasks such as image recognition and natural language processing [9]. For these types of tasks, the shared weights used in the convolutional layers improve generalization and computational efficiency compared with artificial neural networks (ANNs), allowing CNNs to efficiently learn features directly from high-dimensional inputs. This can allow them to be trained directly with large, complex inputs, including unprocessed

raw data, eliminating the need for manual feature engineering. CNNs can operate with inputs, such as colour images, 2D inputs, such as greyscale images or audio spectrograms, as well as 1D inputs, such as raw vibration signals or frequency spectra represented as vectors.

Given the robustness of CNNs in dealing with large inputs, providing them with raw data presents the most convenient option. However, there are still reasons for pre-processing input data to some extent. Bengio et al. [9] suggest that different representations of the same data can entangle and hide different explanatory factors of variation behind the data. They state that a good data representation can reduce the number of parameters to be trained, reduce the number of training samples required to achieve generalization, quicken network convergence, and improve overall accuracy of the network. Therefore, some studies have explored which input data representations, or input types, are most suitable for vibration-based fault diagnosis tasks.

1.2 Literature Review

Developing diagnostic or prognostic MHM solutions is especially challenging due to the nonlinear nature of gathered signals and non-stationary operation of many machines [10]. Traditional MHM techniques have been tailored towards linear systems and can suffer under uncertain operating conditions [11]. Models including nonlinear prediction that are able to deal with non-stationary signals are, therefore, highly sought after [12]. Si et al. [13] provide a review of statistical data-driven approaches for prognosis. Sikorska [14] provides a review of modeling techniques used for prognosis. An et al. [15] review and provide commentary regarding several methods including data-driven and physics-based MHM. These solutions are adequate for many monitoring situations, especially where operating conditions are less uncertain. However, certain key advantages motivate the transition towards data-driven methods, specifically ones using ML.

A key motivator is the capacity and adaptability of ML algorithms for continual improvement as new data becomes available. Coupled with the decreasing cost of sensors and access to edge and cloud-based computing systems, ML-based MHM solutions are able to use large amounts of data to solve problems of nonlinearity and operating condition uncertainty. Similar motivations have led to the adoption of ML as the primary engine behind image recognition, natural language processing, consumer analyitics, and many other computational tasks [16], [17]. CNNs are the most commonly used category of ML algorithm applied to MHM because of their feature-learning ability and computational efficiency [6]. Therefore, an important objective of this thesis is to assess previous uses of CNNs in MHM and attempt to improve them by looking at other applications of ML.

Raw vibration signals have been used in 1D CNNs with and without low-pass filters and downsampling to diagnose faults in bearings [18], [19] and gearboxes [20], [21]. Downsampling reduces the number of input parameters needed, but there is no apparent consensus on whether downsampling is generally beneficial. Many representations of vibration measurements have been demonstrated for diagnosis with CNNs. Chen et al. [22] perform bearing fault diagnosis using a low-pass filtered Fourier spectrum as a 1D input, presenting a novel CNN architecture that successfully detected natural faults when trained with artificial faults. Appana et al. [23] use the envelope spectrum of acoustic measurements for bearing diagnosis at various speeds, or rotations per minute (RPM). For 2D CNN inputs, Guo et al. [24] use the wavelet transform scalogram to detect bearing faults, Han et al. [25] use an adapted wavelet-based transformation to diagnose gear faults, and Lee et al. [26] use combined spectrograms from multiple accelerometers in a single 2D image, though it is difficult to compare the results of these studies directly since they were conducted with different validation frameworks. Pandhare et al. [27] compare CNNs trained with raw time-domain, envelope spectrum, and spectrogram inputs against SVM and kNN machines,

finding that the time-frequency input surpasses time-domain and frequency-domain in accuracy. Verstraete et al. [28] compare acceleration spectrograms, scalograms, and the Hilbert-Huang transformation for bearing fault diagnosis and noted the strong influence of network architecture on diagnostic performance, and achieved the best performance using the scalogram.

For prognosis, Li et al. use a 2D data representation created using spectrograms as inputs to a deep CNN used to estimate remaining useful life of bearings in the PRONOSTIA (FEMTO) dataset [29]. They demonstrate that their proposed method using a subset of measurements taken after reaching a kurtosis-based threshold, called the first predicting time (FPT), achieves higher accuracy compared to using measurements from the entire lifetime. The method for determining the FPT was first proposed by Li et al. [30], where it was also shown to improve the performance of degradation model-based estimations of RUL. Jiang et al. take advantage of the multiple accelerometers contained in the dataset by constructing an input with a height of two and a width of 2560 sample points of raw data sampled for each of the measurements in the PRONOSTIA dataset [31]. Their proposed solution uses convolution within a long short-term memory (LSTM) network to predict RUL.

Most work investigating data augmentation for creating datasets on which to train CNNs and similar deep feature learning algorithms are related to augmenting image datasets. Shorten et al. provide a comprehensive review of image data augmentation methods, this includes basic transformation-based techniques such as rotation, cropping, and erasing, as well as advanced techniques used to synthesize new data such as generative adversarial network-based data augmentation [32]. Salamon et al. present a deep CNN for environmental sound classification based on 1D time-series inputs and compared several augmentation methods, finding that randomized pitch shifting gave the greatest improvement to overall accuracy [33]. Recently, Li et al. demonstrate significant

improvements to bearing fault diagnosis using deep CNNs and raw time-series vibrations input [34]. They found that signal translation and time stretching transformations resulted in the highest accuracy, whereas amplitude shifting, adding Gaussian white noise, and masking noise all gave lower accuracy. No studies have been found to compare the effectiveness of different data augmentation techniques for bearing fault prognosis.

The studies described above represent the recent literature in MHM on which this thesis stands. Additional studies are introduced in each chapter if they are especially relevant to the chapter or used as a direct comparison. These additional studies are taken from the broader field of ML and are used to inform changes that can be made to improve MHM-based ML.

1.3 Motivation and Objectives

MHM can benefit from advancements in ML only insofar as ML principles are correctly understood and applied, and the degree to which established ML best practices are used. Surveying MHM publications reveals a focus on introducing "novel" diagnosis or prognosis algorithms that make some marginal improvement over other recent algorithms. Comparisons are sometimes made using the same public dataset. Though, often times, comparisons are made using different validation data or using different criteria. Moreover, inadequate validation is often used to support findings of ML papers within MHM [6]. Because of this, there is an absence of collectively shared practices around how ML should be applied to MHM, and where some exist, there are sometimes foundational flaws propagating through iterations of MHM works, as will be highlighted in Chapter 4.

This thesis addresses multiple distinct key areas at the intersection of MHM and ML using CNNs. Optimized input representations are understood to be critical in deep learning and, indeed, many MHM studies are dedicated to demonstrating novel or improved input

representations based on vibration signal transformations. However, few studies have compared the performance of different input representations under the same benchmarking framework, and none have systematically considered CNN architecture optimization for different input representations. The first objective of this thesis is, therefore, to explore the relationship between input representations and hyperparameter tuning in a diagnosis task.

A further, and perhaps even more critical, area that this thesis explores is how laboratory data, especially open-access data, are construed to represent "real life" or industrial scenarios, where new MHM techniques might eventually be applied. In these instances, it is critical to correctly apply accepted ML methodologies around constructing training and validation datasets for domain shift problems. Domain shift is related to operating condition uncertainty and is generally used to describe a ML validation framework where the training and validation data is deliberately selected so as to contain fundamental differences such as differences in RPM or bearing specifications. Therefore, a doman shift occurs when a validation dataset is "different enough" from the dataset used to train an algorithm that the algorithm cannot use dataset specific indicators to complete its task. However, there is no official definition of the term and the literature reflects a need for consensus on which situations constitude domain shift. It is also important to make honest appraisals of how well laboratory data can be made to reflect industrial measurements. To that end, this thesis aims to analyze the validity of a certain commonly used benchmark and correctly apply ML principles where they appear to be absent. Furthermore, there is an apparent need for comparing "novel" MHM CNNs to popular deep learning nets prevalent in more active ML fields, such as GoogleNet, AlexNet, and ResNet.

When considering key aspects of CNNs used for conventional applications, data augmentation is always demonstrated as an important contributor to CNN success. However, transformation-based data augmentation, especially advanced techniques, are not often

considered when applying ML to MHM problems. Indeed, these should have the greatest capacity to improve existing MHM solutions by aiding where these solutions currently struggle the most: in prognosis tasks. Therefore, this thesis seeks to determine whether existing data augmentation techniques used in other ML fields offer prognostic CNNs a similar performance boost.

In optimizing various sub-components involved in designing and training CNNs for MHM, it is possible to capture lessons learned in improving CNN effectiveness when sub-components are combined. Therefore, the final chapter of this thesis will seek to train a CNN to perform prognosis with a data representation using a 3D convolution architecture and basic data augmentation that improves upon conventional 2D kernels and representations. The aim is to justify improving overall performance by first focusing on individual aspects of the ML methodology and combining the most favorable characteristics into one algorithm.

1.4 Contributions

The experiments and results of this thesis are divided into different key aspects of applying CNNs to MHM and organized into four chapters. The highlights of these chapters are:

Chapter 3: Input Representations and Hyperparameters

- The results of training many variations of a 2-layer CNN are presented.
- Many variables are independently varied: input size, preprocessing method, kernel size and stride, and max pooling size and stride.
- Three open-access datasets are used to compare hyperparameter and preprocessing preferences from different domains.

Chapter 4: Benchmarking Domain Adaptation in Fault Diagnosis

- Two popular bearing fault diagnosis algorithms are compared with novel applications of AlexNet and ResNet in bearing fault diagnosis.
- A critical flaw in a commonly used benchmarking framework is identified, and an alternative benchmarking framework is proposed and demonstrated.
- The retrained ResNet outperforms state-of-the-art CNNs in both the original and proposed frameworks.

Chapter 5: Influence of Data Augmentation in Prognosis

- Several data augmentation methods are demonstrated in estimating the health indicator in accelerated life bearing tests.
- Methods that are useful in other applications are found to be less useful or negatively impact accuracy in this application.
- Region masking and masking noise are found to be the most useful augmentation methods, though room for improvement is identified.

Chapter 6: Prognosis Using a 3D CNN and Multi-Channel Spectrogram

- A 3D input feature is constructed from time-frequency representations of multiple signals.
- The proposed input and 3D convolutional architecture significantly outperforms single-sensor variants.
- A strong method for sensor fusion is established for health indicator estimations.

1.5 Organization of Thesis

The remainder of this thesis is organized as follows: Chapter 2 includes a theoretical background of supervised machine learning, a mathematical description of ANNs and CNNs, as well as vibration based diagnosis for bearings and gears. Chapters 3-6 contain the core experimental work of the thesis, each with sub-objectives and conclusions contributing to the

overall research objectives stated in section 1.4. Overall conclusions are given in Chapter 7 and recommendations for future areas of inquiry are discussed in Chapter 8.

Chapter 2

Theoretical Background

2.1 ML and Supervised Learning

ML is a topic in computer science and mathematics broadly defined as the study of algorithms that improve automatically through experience [35]. ML involves algorithms that learn to solve tasks based on example solutions without explicitly being programmed to solve the task. ML is used in areas where it is infeasible to explicitly write an algorithm to solve a particular task, often due to the outputs depending on a generalization of inputs.

Supervised learning is a task within ML involving the learning of a function that maps an input to an output based on examples of input-output pairs [36]. The function is learned during the training process, during which the function is inferred from a set of examples comprising the training set. In the training set, each example includes the input (a matrix or a vector) and its expected output (usually a vector or a single continuous number, sometimes a matrix). After successful training, the function is able to map inputs to correct outputs for unseen instances, that is, examples not included in the training set. These may be a part of a testing set or a validation set. This requires the function to generalize from the training data and infer upon underlying factors of variation.

Most work in applying ML to MHM can be classified as supervised learning. Generally, inputs to ML algorithms are created from indicators of machine health that can be measured directly using accelerometers, thermometers, electrical sensors, tachometers, or any number of other types of sensors. These measurements are sometimes used directly, but are more often transformed into a more condensed input vector used as an input to the ML algorithm.

The algorithm is trained to map these inputs to an output that represents either a current machine health state or a likely future health state.

ANNs and CNNs can be used as the basis for unsupervised learning, but are most commonly used in supervised learning. Thanks to the simplicity of some specialized software packages, it is possible to use them successfully having only a general understanding of their mechanisms, though it is prudent to review their mechanisms in some detail.

2.2 Mathematical Theory of CNNs

The following description is informed by the background given by Lei et al. [6], as well as Andrea Vedaldi's thorough primer [37]. CNNs are an evolution and an extension of ANNs. ANNs roughly imitate the activities of animal neurons in processing information, whereas CNNs imitate the animal visual cortex for processing images. It is appropriate to begin a description of CNNs with a description of the more basic ANN.

ANNs are multi-layer perceptron supervised learning networks with forward and backward propagation. Figure 1 shows a two-layer ANN during forward propagation, where the input vector is transformed by two hidden layers and mapped to the output layer.

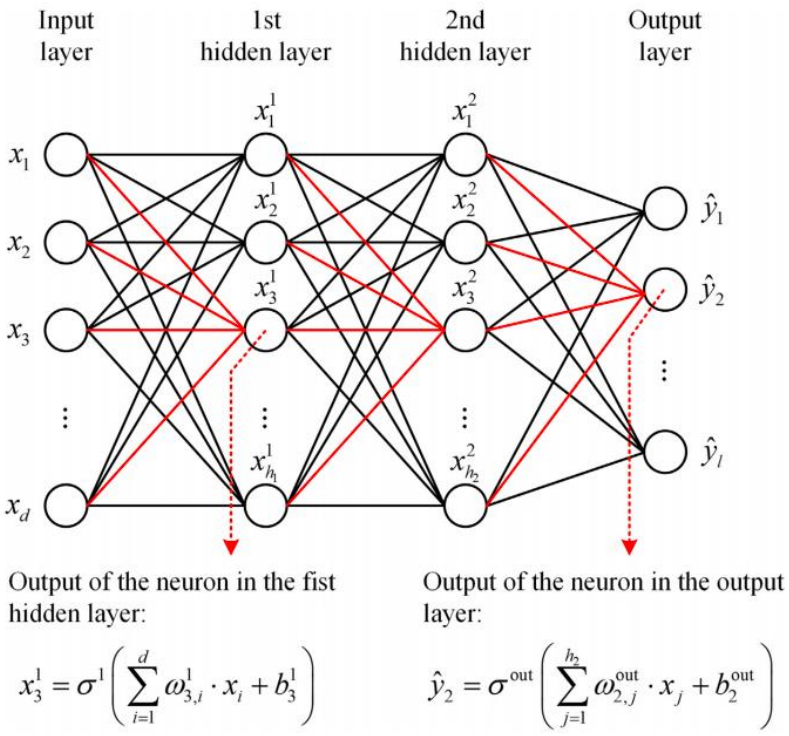


Figure 1: Architecture of an ANN with two hidden layers [6]

ANNs are trained such that the hidden layers transform the input to provide either a classification or a regression output. For classification, the output neuron corresponding to the correct class is close to one and all other output neurons are close to zero. For regression, there is usually a single output neuron that takes a continuous value. Provided a dataset of training vectors $\{x_i, y_i\}_{i=1}^{m}$ with m samples, where $x_i \in \mathbb{R}^d$ includes d features and $y_i \in \mathbb{R}^l$ contains l output classes, the output of the hth hidden layer is expressed as

$$\left(x_i^h\right)_j = \sigma^h\left(\sum_{i=1}^{n_h-1} \omega_j^h \cdot x_i^{h-1} + b_j^h\right), \qquad j = 1,2,\dots,n_h, \qquad h = 1,2,\dots,H. \qquad \text{Eq. (1)}$$

where $\left(x_i^h\right)_j$ is the output of the jth neuron in the hth hidden layer, and $x_i^0 = x_i, n_h$ is the number of neurons in the hth hidden layer, σ^h represents the activation function of the hth hidden layer, n_{h-1} is the number of neurons in the $(h-1)$th hidden layer, ω_j^h are[the weights between the neurons in the previous hidden layer and the jth neuron in the hth hidden layer,

and b_j^h is the bias of the hth hidden layer. The activation function serves as a regularizer and prevents very large or negative numbers from cascading through the layers and skewing training. The rectified linear unit (ReLU) function is the most common choice for activation functions in deep learning [38], which simply sets negative inputs to zero and leaves positive values unchanged.

The output layer is

$$(\hat{y})_k = \sigma^{out}\left(\sum_{i=1}^{n_H} \omega_j^{out} \cdot x_i^H + b_j^{out}\right), \qquad k = 1,2,\dots,l, \qquad \text{Eq. (2)}$$

where $(\hat{y})_k$ is the predicted output of the kth neuron in the output layer, σ^{out} is the activation function of the output layer, ω_k^{out} and b_j^{out} are the weights and bias of the output layer.

Provided a given training sample input x_i, the feed forward process gives the expected $\hat{y}_i$, which is compared with the expected response y_i to calculate the error and the neurons of the network. The back-propagation process is the procedure by which the network weights and biases are tuned to minimize the error for the training samples by

$$\min_{\omega,b} E_i = \frac{1}{2}\sum_{k=1}^{l}[(y_i)_k - (\hat{y}_i)_k]^2 \qquad \text{Eq. (3)}$$

To perform this minimization, gradient descent is used. This involves updating each weight and bias by a small amount in the direction indicated by the partial derivative of the error with respect to that weight and bias, expressed as

$$\omega \leftarrow \omega - \eta \cdot \frac{\partial E_i}{\partial \omega}, \qquad b \leftarrow b - \eta \cdot \frac{\partial E_i}{\partial b} \qquad \text{Eq. (4)}$$

where η is the learning rate. Each layer's weights and biases are updated beginning with the output layer and progressing with each hidden layer towards the input layer. Additional information about back propagation training is provided by Rumelhart et al. [39]. CNNs also have their parameters tuned using back propagation. The main differentiating feature of CNNs is their invariance to the location of features in the input layer. In this case, the regions

of the input are mapped to create an output feature map by sweeping a relatively small number of neurons, formed by a kernel, across the input space.

Generally, CNNs consist of several different types of layers combined in series, or sometimes combined in series and parallel, to create various CNN architectures. These layer types are convolutional layers, pooling layers, and fully connected layers. The convolutional and pooling processes are shown in Figure 2. In CNNs, the weights and biases are contained within kernels. Each kernel contains one bias and several weights depending on the kernel dimension. While the letter k is used for weights in kernels, they are similar in function to the weights ω used in ANNs. In convolutional layers, the filter kernels $k^c \in \mathbb{R}^{H\times L\times D}$ convolve the input vectors $x^{c-1} \in \mathbb{R}^{M\times N}$ from the previous $(c-1)$th layer, where H, L, and D are the height, length, and depth of the kernels, respectively. In the following diagram, positional subscripts are used to illustrate the selective connections made during convolution.

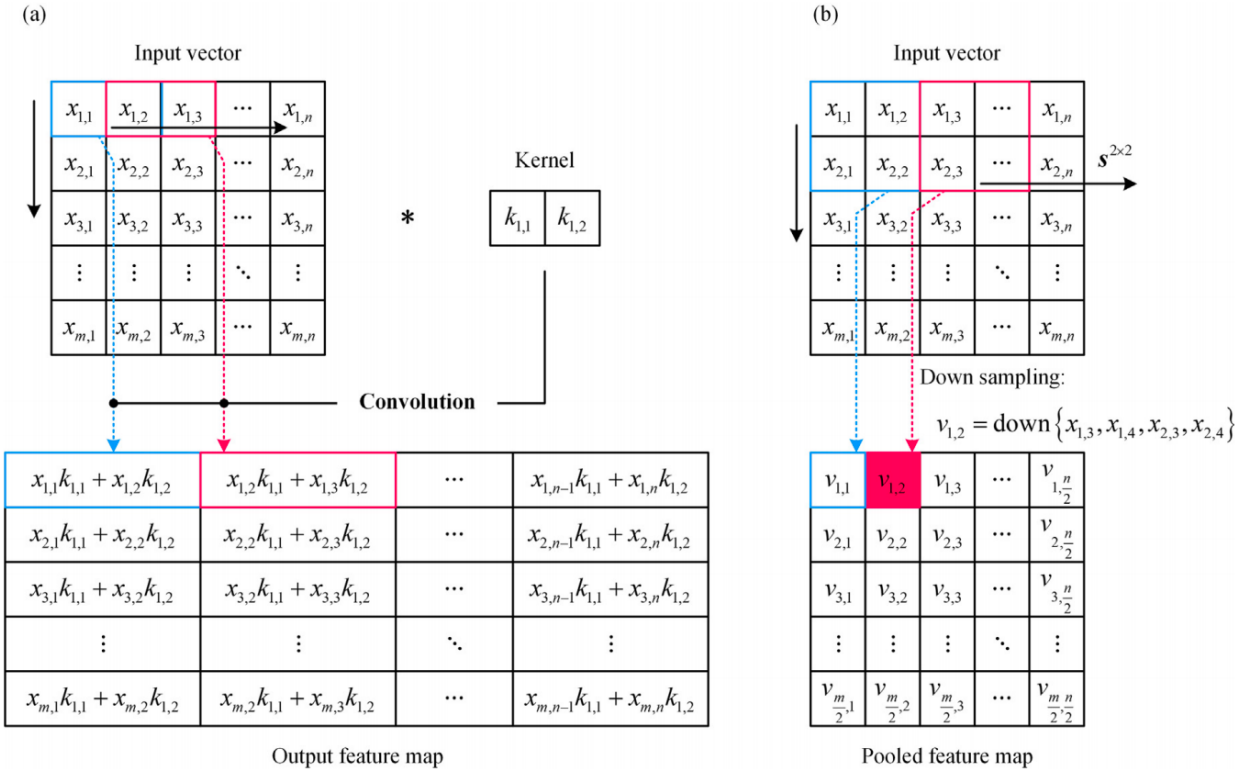


Figure 2: a) Convolutional process using kernel $k^c \in \mathbb{R}^{2\times1\times1}$ and b) pooling using filter $s^{2\times2}$ [6]

The output feature map of the cth layer is obtained as follows:

$$x_i^c = \sigma_r\left(x_i^{c-1} * k^c + b^c\right) \in \mathbb{R}^{(M-H+1)\times(N-L+1)\times D}, \quad c = 2,3,\ldots$$
$$\left(x_i^{c-1} * k^c\right)_{j,k,d} = \sum_{m=1}^{M}\sum_{h=1}^{H}\sum_{l=1}^{L} x_{(i),j+h-1,k+l-1,m}^{c-1} \cdot k_{h,l,d}^c \qquad \text{Eq. (5)}$$

where σ_r represents the activation function of the ReLU, j and k are the height and width of the stride respectively. The influence of stride selection is discussed in Chapter 3. Other activation functions exist, though only the ReLU is used in the present work. Pooling layers are used to downsample feature maps to reduce the number of training parameters used and reduce overfitting. In this work, max-pooling is always used as the pooling function. The pooled feature map is described as follows:

$$(v_i^p)_{m,n,d} = \max\left\{x_{(i),j,k,d}^{p-1} \middle| \forall x_{(i),j,k,d}^{p-1} \in x_i^{p-1}, j,k \in \mathrm{N}^+, s^{r\times t}\right\},$$
$$s.t. \quad s^r \le j \le s^r m, \qquad s^t(n-1) \le j \le s^t n \qquad \text{Eq. (6)}$$

where $s^{r\times t}$ are the filters in the pooling layers. Stacking convolutional and pooling layers allows a CNN to learn complex features from the input data. These features are then flattened into a vector and used as the input to one or more fully connected layers, which eventually maps them to target classes.

The outputs of the fully connected layers are represented as:

$$x_i^f = \sigma_r\left(\omega^f \cdot x_i^{f-1} + b^f\right), \qquad f = 2,3,\cdots \qquad \text{Eq. (7)}$$

where x_i^f is the flattened feature map generated by the convolutional sequence and $\theta^f = \{\omega^f, b^f\}$ represents the training parameters of the fully connected layers. Any number of fully connected layers may follow the convolutional and pooling layers, though usually fewer than four are used. The output of a CNN is mapped to a classification vector or a regression unit in the same manner as that of an ANN.

For the work done in this thesis, MATLAB has been used with the Deep Learning Toolbox to design and train CNNs. This toolbox, especially with the graphical user interface DeepNetworkDesigner, simplifies the network design process significantly. Where many different CNN architectures needed to be trained, a custom set of functions were written to construct and train CNN models based on generalized instructions. Appendix A contains the MATLAB code used to perform experiments, where each numbered section contains the code for a .mat file. CNNs with 1D, 2D, and 3D inputs are trained within the various chapters, with classification outputs for diagnosis tasks, or regression outputs for prognosis. Some MATLAB functions were developed specifically for preparing the training data and for interpreting results. Several key functions were written for data augmentation.

2.3 Data Augmentation

Data augmentation is a critical tool in ML used when the available training data is limited or when it is necessary to improve the network's invariance to certain types of transformations. Data augmentation effectively expands the original training dataset by performing some transformation to the training samples and yields a greater number of unique new training samples. The transformation used must alter the original data in a way that makes it distinguishable from other transformed data, but without completely obscuring the underlying factors of variation. The class labels are usually preserved during data augmentation. For image classification datasets, augmentation techniques include cropping, rotating, and obscuring random sub-regions of an image, which improve translational and rotational invariance as well as overall generalization. Each of these transformations can involve randomized parameters to allow multiple new images to be generated from each original image.

In Chapter 3, a sliding window is used to extract many short-duration training samples from each original experimental measurement in the time-domain. Preprocessing transformations, such as the fast Fourier transform (FFT), are then performed as needed on the extracted sample. This method has several benefits: it is simple and easily reproduced, it offers a many-fold increase in available training samples, it dramatically reduces the dimensionality of the training samples, and it promotes translational invariance in the resulting trained network. The data augmentation technique is illustrated in Figure 3. A sliding window of some prescribed length is used to extract a sample from the original experimental measurement. The window is advanced in time by a set increment to extract the next training sample. Some overlap between windows is provisioned to increase the yield of new training samples, improving translational invariance. In machine learning, translational invariance is defined as an insensitivity of a model to the position of features within the input space. For example, in image recognition, an algorithm intended to detect the presence of cats in photos would be said to possess translational invariance if it successfully identifies cats appearing in any given sub-region of the photo. This technique of sample segmentation is the perhaps simplest possible form of data augmentation. Some disregard this as a data augmentation technique since it does not transform the data but merely segment it. Regardless, this segmentation can useful and important for training networks when the available training data comprises a smaller number of longer measurements.

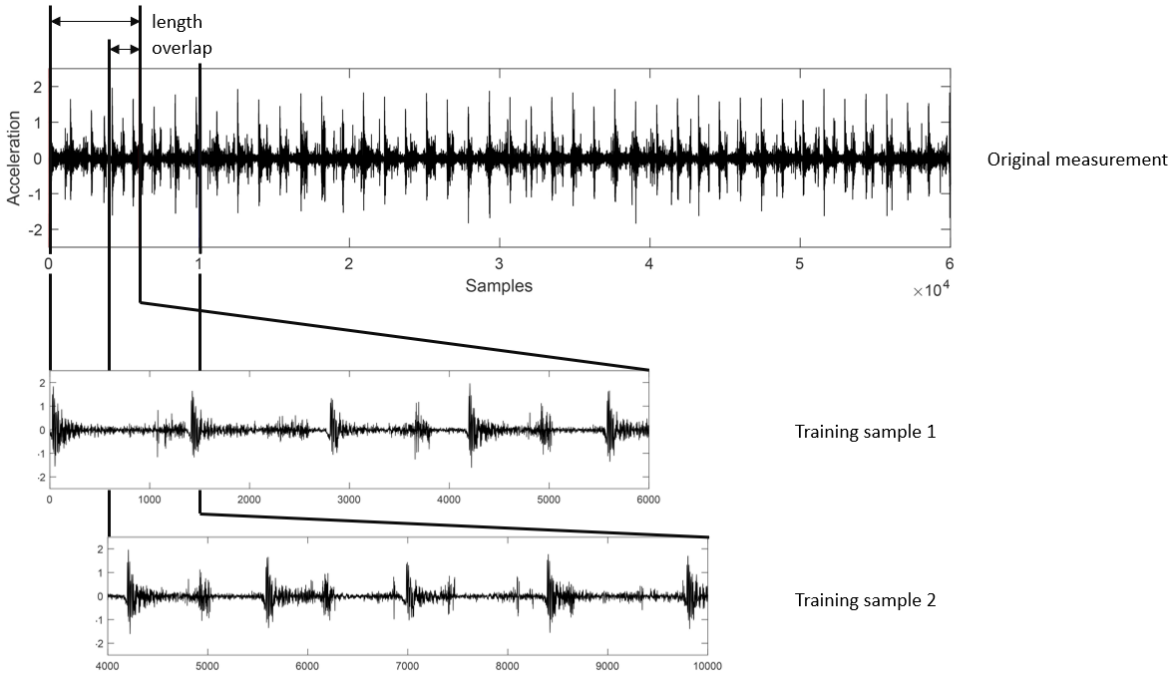


Figure 3: Sliding window using the overlap data augmentation technique applied to time-series data

Data augmentation is applied to all data used as inputs to CNN algorithms in this thesis. More advanced forms of data augmentation used in Chapter 5 are described in section 5.4. The augmentation methods studied are tailored for vibration-based inputs. Therefore, fundamentals of vibration-based bearing and gear fault diagnosis are reviewed.

2.4 Vibration-based bearing fault diagnosis

Although CNN algorithms determine outputs without prior knowledge of the problem at hand, an understanding of bearing faults and their diagnosis methods undoubtedly help in evaluating the performance of these algorithms, while ensuring that outcomes do not contradict theory. Therefore, this section describes the fundamental nature of the observation data and some of the known factors of variation behind it. Additional useful resources include

a model for single-point defect vibration signals given by McFadden and Smith [40] and a detailed bearing fault diagnosis tutorial from Randall and Antoni [41].

Many diagnostic algorithms focus on localized bearing faults, since these are the easiest to study in labs and are the type of fault present in almost all publicly available research datasets. Localised bearing faults are also useful since they are more easily distinguished by their vibrational signature and usually precede destructive component failure. Localized faults consist of geometric irregularities, such as pits or spalls that inhibit the normal smooth motion of the bearing components relative to each other. These faults may arise due to any number of factors, such as overheating, lubrication failure, overloading, material defects, misalignment, and through natural material wear or fatigue. Some of the former factors can occur simultaneously and/or behave as catalysts to accelerate the natural rate of fatigue.

As the inner race and outer race rotate relative to each other, the contact zones between the different bearing components pass over the fault. This short-duration contact is similar to an impact and causes a burst of vibrations at frequencies known as the fault characteristic frequencies (FCFs) that propagate through the bearing housing to be detected elsewhere on the machine. The most basic form of bearing fault diagnostics involves checking vibration signals for repeated impacts at the FCFs.

Several other factors combine to obfuscate the diagnostic information within the signal. Among these is the contamination of the signal by background noise, which is a prevalent issue in industrial applications, where many machines interfere with each others' sensors through common transmission paths. Because bearing resonant frequencies tend to be quite high, it is generally easiest to find wide frequency bands dominated by bearing signals in the higher end of the spectrum rather than the lower end, as the lower end of the frequency spectrum is more likely to contain contamination from distant background machinery. Another complicating factor is the cyclical variation of transmission paths within the bearing

itself caused by the relative rotation of bearing components. The influence of this amplitude modulation is illustrated in Figure 4.

A basic and powerful tool to simplify signal analysis and reveal the underlying shape of the vibration waveform involves taking the envelope of the signal, which effectively smooths the resonance waves induced during impacts against the fault zone into a single wave. This may be regarded as the effective instantaneous amplitude, and it more clearly reveals the FCF than the raw signal. Figure 4 shows how different fault locations involve different amplitude modulation patterns as the fault moves relative to the sensor.

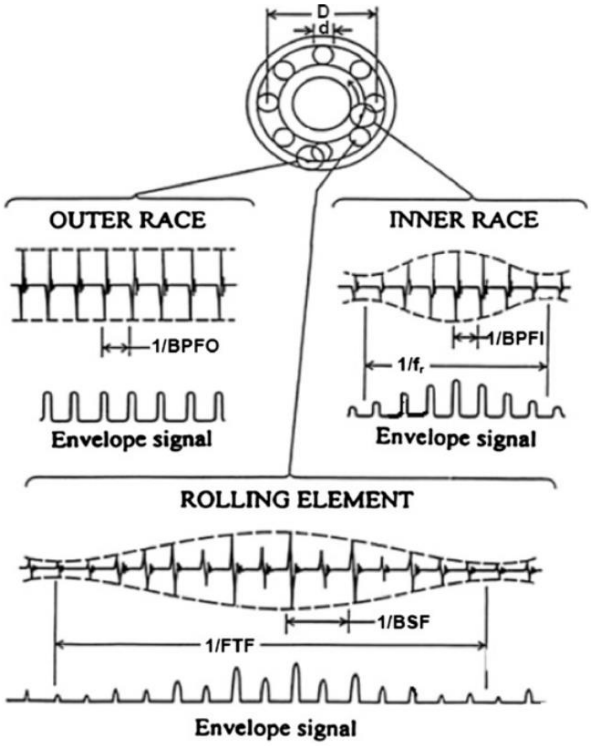


Figure 4: Typical signals and envelope signals from local faults in rolling element bearings [41]

BPFO, BPFI, BSF, and FTF stand for ball pass frequency outer, ball pass frequency inner, ball spin frequency, and fundamental train frequency respectively. Frequently changing operating conditions pose a challenge for in-situ bearing fault diagnosis. A bearing within a machine may be subject to different speed and loading conditions depending on the machine's mode of operation. Varying the speed will cause a change in the FCF being activated by a

given fault and a change in loading conditions will change the vibration magnitude and can also change the modulation and attenuation characteristics of the vibration response. This motivates the development of diagnostic algorithms that are unaffected by changes in operating conditions or limits them to work on machines with consistent operating conditions.

Although laboratory test benches attempt to simulate real industrial situations, artificially damaged bearings and gears are imperfect approximations of real damage endured gradually through normal wear processes. The resulting difference in fault geometry induces a different fault signal during operation. Figure 5 shows test bench measurements from identical bearings under the same loading conditions with a seeded fault introduced to the outer race with electron discharge machining (left) and outer race spalling by natural wear occurring after accelerated life testing (right). The raw signals and frequency spectra are somewhat different in appearance, but the differences are minimized in the envelope spectra.

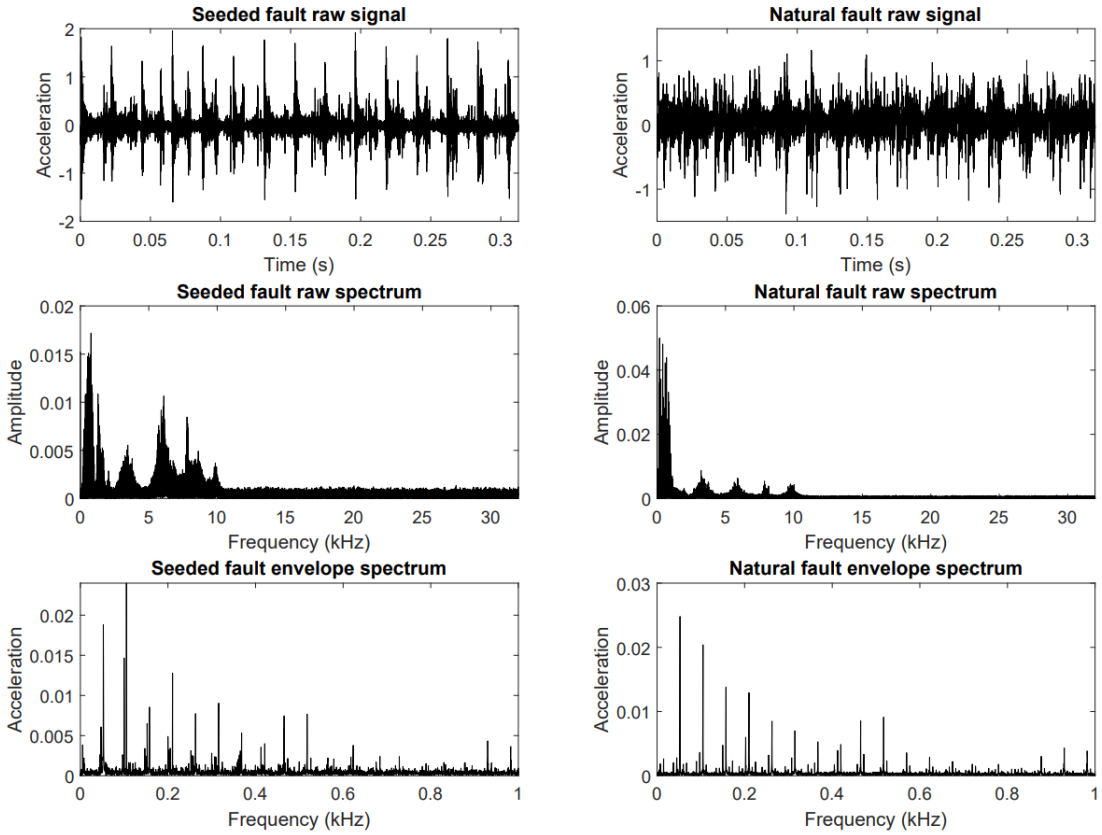


Figure 5: Comparison of natural faults (left column) and real fault signals (right column) with respect to raw signal (top row), raw frequency spectrum (middle row) and envelope spectrum (bottom row) for identical bearings under the same operating conditions. Signals obtained from the Paderborn University dataset [42]

2.5 Vibration-based gear fault diagnosis

A useful resource describing gear fault detection using vibration spectral analysis is given by P. D. McFadden [43]. Planetary gearboxes have unique characteristics and are prevalent enough to warrant individual consideration; an informative tutorial is given by Guo et al. [44], which also reviews conventional methods employing time-synchronous averaging and narrowband demodulation.

Unlike bearing vibration signals, which contain no periodic impulses in the healthy state, healthy gear signals contain sharp periodic impulses due to the repeated engagement and disengagement of the meshing teeth. Therefore, gear faults cannot be diagnosed simply by

detecting the presence of new periodic fault signals, but must be diagnosed by detecting other changes in the gear signal. The frequency of tooth engagement is known as the gear meshing frequency (GMF) and is calculated as the product of the shaft frequency and the number of teeth on the gear mounted to that shaft.

Gear faults may arise due to several factors, including inadequate lubrication, contamination, manufacturing or installation error, and overloading. These factors may give rise to fatigue pitting on a tooth surface or cracks at the base of a tooth where stresses are concentrated. Spectral analysis reveals that sidebands often appear about the GMF and its harmonics when a local tooth defect is present in a gear. In systems with many pairs of meshing gears, it can be very difficult to distinguish the components in the spectrum, complicating the diagnosis process. Additionally, the presence of sidebands does not always indicate fault severity.

It is not possible to obtain a pure recording of the gear meshing signal. Practical constraints require accelerometers to be placed on the external gearbox housing, this results in a variable transmission path that passes through other gears, shafts, bearings, and the housing itself. The resulting modulation can considerably distort the signal and cause sidebands to arise even for healthy gears.

Figure 6 shows the raw signals and envelope spectra for a two-stage gear reducer when all gears are healthy and when the input gear is chipped.

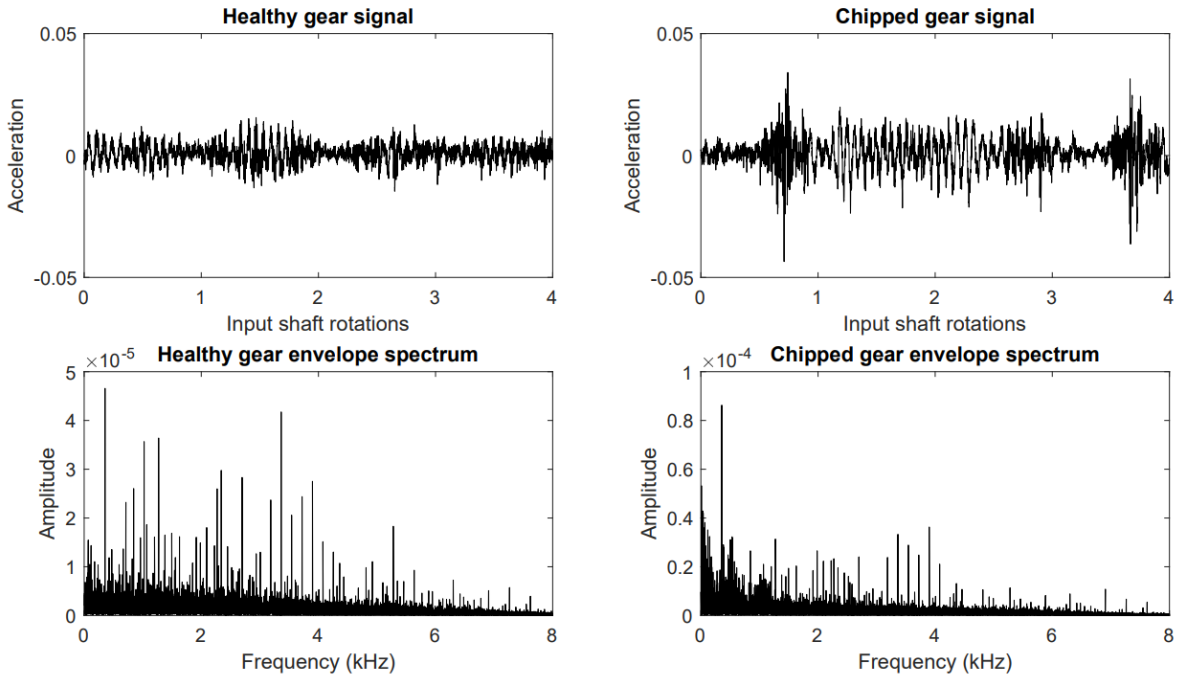


Figure 6: Raw signal (top) and spectrum of the envelope signal (bottom) for a healthy gear train (left) and one with a chipped tooth at the input gear (right). Signals obtained from the 2009 PHM data challenge dataset [45, p. 200]

With respect to gear fault diagnosis, one of the objectives of the following chapter is to explore whether CNN architectures that are effective for bearing fault diagnosis are also effective for gear fault diagnosis, or whether gear fault diagnosis requires an entirely different configuration of hyperparameters.

Chapter 3

Input Representations and Hyperparameters

Parts of this chapter have been submitted for publication.

ABSTRACT

This chapter presents the results of training different configurations of classical CNNs for vibration-based bearing fault and combined bearing and gear fault diagnosis using three popular benchmark datasets. Different preprocessors are studied to explore the relationship between preprocessing and hyperparameter selection. Raw temporal measurement, Fourier spectrum, envelope spectrum, and spectrogram input types are individually used to train CNNs. Many configurations of CNNs are trained, with variable input sizes, convolutional kernel sizes and stride. The experiment is repeated using three datasets to evaluate whether the results are consistent over different, but related domains, including two bearing fault datasets and a gear fault dataset. The results show that each input type favors different combinations of hyperparameters, and that each of the datasets studied yield different performance characteristics. It is demonstrated that CNNs trained with spectrograms are less dependant on hyperparameter optimization over all three datasets.

3.1 Introduction

Many of the studies referenced in the literature review introduce new combinations of CNN architectures and data representations not previously used for fault diagnosis. Few of the referenced studies compare the effectiveness of different input data representations within an unchanged CNN architecture. Though it was noticed by Verstraete et al. [28], no studies have been found that focus on the interaction between data representation and hyperparameter optimization. None of the works reviewed have elaborated on the process by which CNN hyperparameter selection was conducted, nor was the sensitivity to changes in hyperparameter values, task domain, and preprocessing methods investigated. Furthermore, in the opinion of the author, too few previous works have validated novel applications of deep learning using multiple datasets available to the public, which leads to questions about overfitting. This chapter aims to address these shortcomings by investigating the effects of changing hyperparameters, while using various common data representations and data from three popular datasets in the public domain. The goal is to compare the difference between different bearing datasets as well as between bearing datasets and a combined bearing and gear system within the same range of CNN configurations. The results will provide a template framework for designing CNN-based diagnosis algorithms, as well as determine whether some data representations are universally stronger or stronger only when paired with certain CNN configurations and datasets.

3.2 CNN input types

Since CNNs can theoretically extract useful features directly from raw data, very good accuracy should be achievable without using complex data preprocessing. In practice, however, strong performance may require manipulation of the observation data, especially

in cases where data is initially of high dimensionality and the availability of training data is limited. One important cause for this is related to a phenomenon known as "the curse of dimensionality". When the number of dimensions increase, the volume of occupied space increases more quickly and the distance between available datapoints becomes much greater, leading to a sparse dataset. Reducing the number of dimensions used to represent the data while preserving factors of variation that maintain a statistically significant relationship between the observations and their labels (e.g. health condition) can make it far easier for ML algorithms to learn features from the data and overcome sparsity.

Reducing dimensionality is not the only objective of pre-processing. Useful transformations to the raw data can highlight or accentuate explanatory factors, making classification easier. A common example of this is the Fourier transform, which is very useful when the observed data is a time-series measurement, but obvious explanatory factors are present in the frequency spectrum. Transformations may be selected by the algorithm designer using domain-specific expertise, or they may be performed by generic data reduction operations such as principal component analysis or autoencoders. This study is focused on common expert-chosen transformations, since they preserve spatial patterns that will be detectable by the CNN.

The following sub-sections introduce each of the input types that are used to train CNNs in all of the case studies of sections 3.4-3.6. The aim is to provide the reader with an intuitive understanding of the transformations and resulting representations without relying on mathematical descriptions.

3.2.1 1D raw time input

As the simplest possible input type, raw time provides only a few parameters that may be chosen by the algorithm designer to characterize this 1D input. Chiefly, one must consider

the length or duration of the raw signal that is used as an input feature. In general, a longer signal duration is more likely to contain useful diagnostic information. If a signal is too short, it may not contain enough instances of periodic fault-induced emissions to establish a high probability of those emissions being the result of a fault-related pattern. However, increasing the signal duration also increases the computational power required to train the network. Larger inputs also increase the probability of the network becoming overfit.

Another motivation for using a shorter window arises when training samples are extracted from a longer experimental measurement via moving window data augmentation. Shorter windows allow more unique training sample to be extracted from the available measured data.

When machine-mounted accelerometers operate at higher sample rates, more data is generated to describe the machine's vibrations over a given span of time. The two signals shown in Figure 7 show how a different number of samples are needed to span a full shaft rotation, and how some fault related signals are manifested over multiple full rotations.

Generally, the resonant frequencies of bearing components are known, so it is possible to determine the upper bound beyond which it is no longer useful to increase the sample rate. However, for this work, no re-sampling of the signals is performed; all benchmark signals will be considered with their native sampling frequencies as stated in each case study.

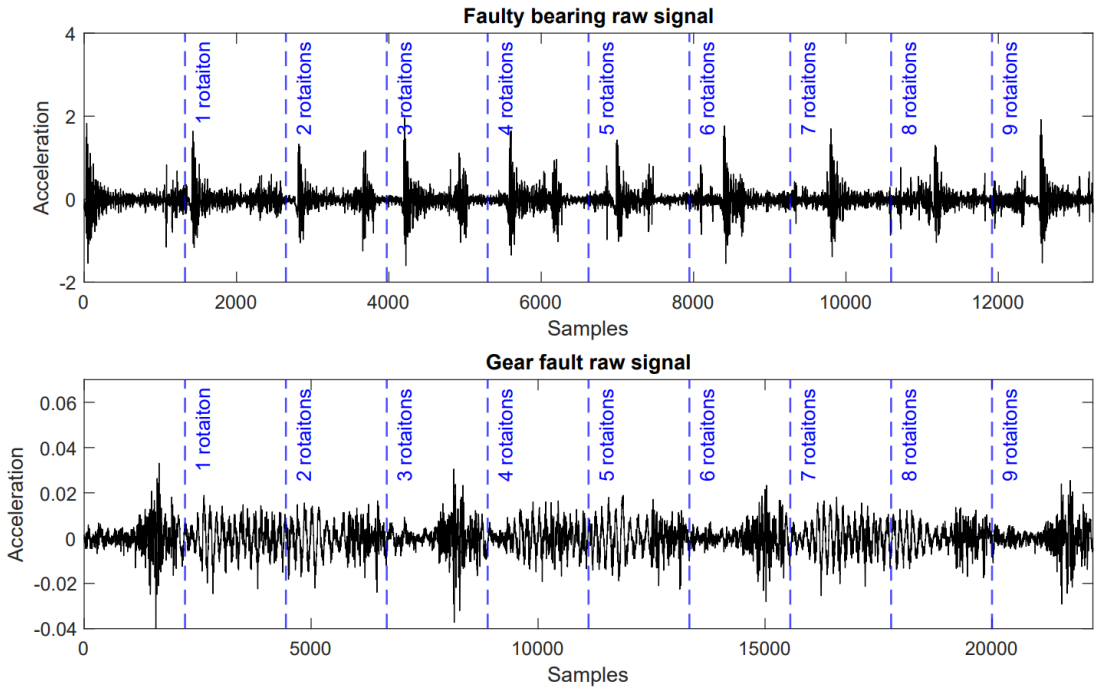


Figure 7: Raw vibration signal for an outer race bearing fault operating at 2900 RPM and sampled at 64 kHz (top) and a chipped tooth fault and a compound fault gear running at 1800 RPM and sampled at 66.6667 kHz. Data obtained from Paderborn University and 2009 PHM Data Challenge [42], [45]

The experiments conducted here include a study of the influence of input duration against performance in a range of different CNN configurations to determine when it may be advantageous to use a longer or shorter input.

3.2.2 1D frequency spectrum

Data augmentation is performed prior to using the FFT to obtain the 1D frequency spectrum samples. The absolute value of the single-sided spectrum is kept. Since no low-pass filter is used, the upper bound of the spectrum is defined by the Nyquist frequency, itself simply determined by halving the sample rate. The frequency resolution achieved is dependant on the duration of the signal extracted during data augmentation. In this case, the resulting input vector will contain half the number of elements contained within the original time domain window.

The overlapping sliding windows used during data augmentation ensure that each spectrum obtained contains variations of the original signal. As with using a raw time input, the probability that a successful diagnosis can be performed on any given spectrum will depend on whether a window coincides with transient fault-induced signals; a very short window has a low probability of containing such information. Figure 8 shows two such spectra obtained from the same experimental measurement with a 75% overlap. The first sample (bottom left) coincides with an instance of a ball passing over an outer race fault, resulting in a spectrum that more clearly indicates the presence of the fault. Assuming the original measurement is stationary, extracted spectrums become increasingly similar to the spectrum of the whole measurement as the window size is made larger. Therefore, using longer windows to extract data results in a more homogeneous dataset.

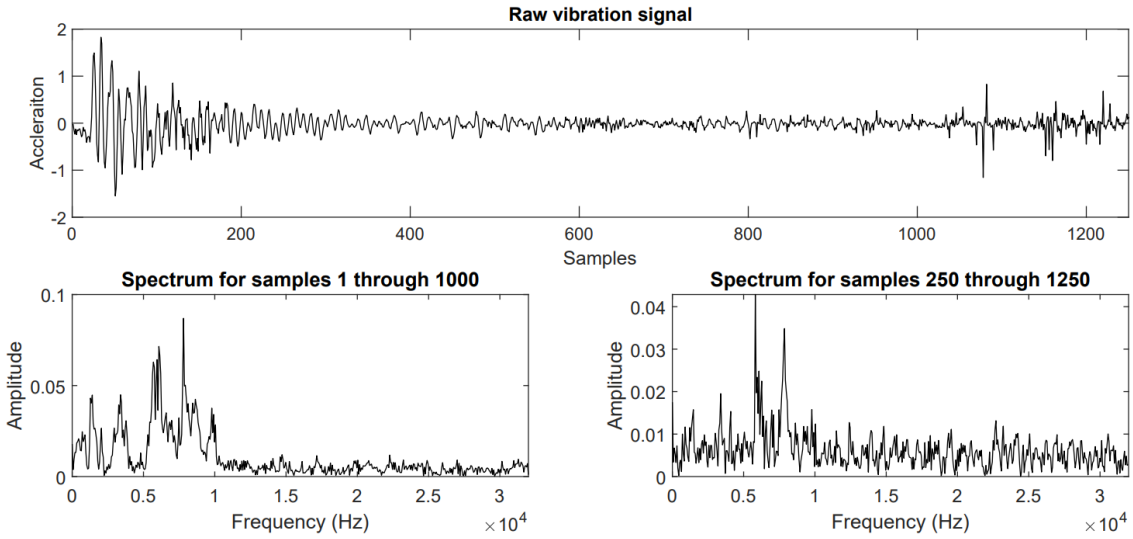


Figure 8: Raw vibration measurement from a bearing fault test bench with two frequency spectra obtained from windows of 1000 samples with 75% overlap, data from CWRU [46]

3.2.3 1D envelope spectrum

The envelope transformation simplifies a signal to reveal the overall shape of the signal. It emphasizes lower frequency modulating signal elements such as low frequency fault related impacts. As a result, the spectrum of the envelope signal can more easily outline FCFs

and their harmonics. This eliminates information about the bearing component's resonant frequencies, that may not be useful for the model's ability to learn simpler and more distinct patterns. The transformation can yield an upper and lower envelope. Since vibration signals tend to be symmetric, the upper and lower envelopes will have very similar frequency spectra. This work uses a standard MATLAB function *envspectrum(x,fs),* where x is the signal and fs is the sample rate in Hz, to obtain the envelope spectrum.

Figure 9 illustrates the transformation in multiple steps: from the original raw signal to the upper envelope signal to the envelope spectrum. The spectrum of the original signal is also shown for comparison. The upper envelope spectrum clearly illustrates the fault characteristic frequency with its harmonics, whereas high frequency vibrations around bearing component resonance frequencies dominate the raw spectrum.

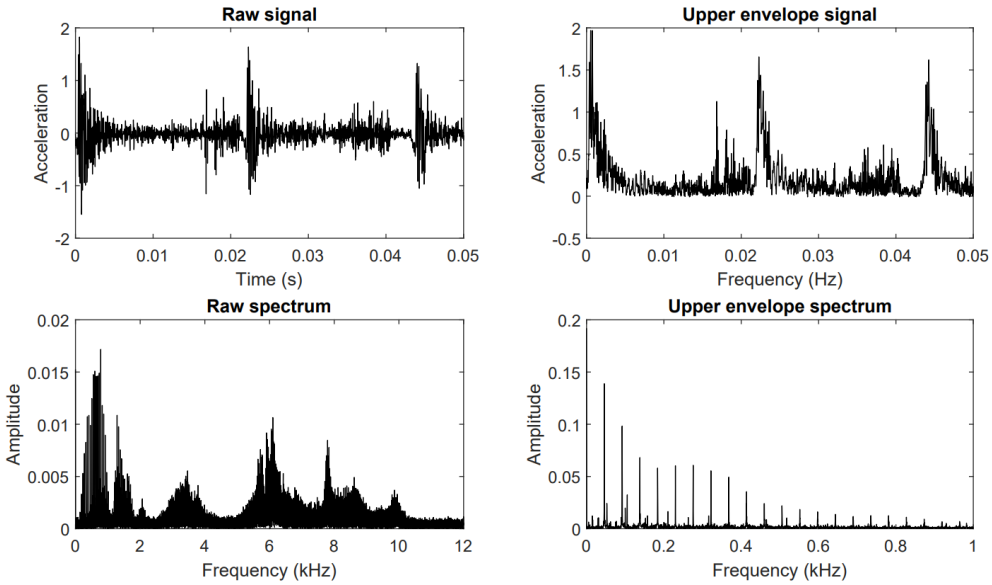


Figure 9: Comparison of raw vibration signal, envelope signal, raw spectrum, and envelope spectrum for an outer race fault sampled at 64 kHz from the Padderborn University dataset

Several qualities of bearing fault signals are illustrated in Figure 9. First, the raw signal shows that the high frequency response to the fault impact is quickly dissipated by the structural damping of the material. The envelope signal discards the high frequency

information by effectively tracing the upper portion of the waveform, preserving the lower frequency information more closely related to the FCFs. The horizontal range in the two spectrum plots reflect this discarding of high frequency content well. The peaks in the raw spectrum show the resonant frequencies of the bearing components that are activated by the impacts. The regular peaks in the envelope spectrum are harmonics of the FCF. Information about fault severity could be inferred from the amplitude of the impacts; higher amplitudes could be expected from larger faults. Information about the fault type is most easily taken from the FCF found in the envelope spectrum.

3.2.4 2D short time Fourier transform spectrogram

The spectrogram provides a very useful time-frequency representation of the signal; it includes the benefits of spectral analysis as well as time localization of frequency components. The spectrogram is obtained by dividing the original signal into many shorter windows and taking the FFT of each one in sequence. The result is a 2D matrix with time progression along one direction and the frequency scale along the other.

Two parameters used in defining the transformation allow some flexibility in tailoring the spectrogram to suit a given application: window length and overlap. The window length is the result of the original signal length divided by the number of desired samples prior to taking the FFT. A shorter window provides finer division of the signal in time, and thus improves time localization, increasing the time resolution of the spectrogram. The penalty for increased time resolution is decreased frequency resolution due to the innate restrictions of the Fourier transform. The frequency resolution is proportional to the size of the windowed signal and is thus improved by increasing the window length. This creates a tradeoff between time and frequency resolution. The overlap used in obtaining the spectrogram lends some improvement to the time resolution by fitting more windows into the signal, with some

samples shared by adjacent windows. For simplicity, and to avoid training CNNs with very large inputs, no overlap is used in this study.

3.3 Network architectures

This particular study aims to investigate the relationship between the basic CNN hyperparameters and the choice of signal preprocessing method. Therefore, the same sequence of layers is used in all experiments with experimental variables that include input type, input size, convolutional kernel size and stride.

All network configurations use two convolutional layers, each one followed by a ReLU layer, a 50% dropout layer, and a max pooling layer. The size of the convolutional kernel is varied to investigate its influence. The same kernel size is used in the first and second layers for simplicity and to reduce the number of experimental network configurations. Max pooling for 1D inputs use a 2×1 filter and a 2×1 stride while max pooling for 2D inputs use a 2×2 filter and a 2×2 stride. All convolutional layers have 16 filters. The output from the second max pooling layer is flattened and followed by a fully connected layer with 7 output neurons. A softmax layer is used to classify the output. Here, network configurations are simply denoted by the convolutional kernel size and stride and the input size used. As an example, Table 1 details the network architecture for a network with kernel size of 8×1, stride of 2×1, and input size of 1248×1.

Table 1: Example CNN specifications for a 8×1 filter size, a 2×1 stride, and a 1248×1 input.

#	Layer type	Activations	Learnables
1	Input layer	1248×1×1	-
2	Convolutional layer	624×1×16	Weights 8×1×1×16 Bias 1×1×16
3	ReLU	624×1×16	-
4	Dropout (50%)	624×1×16	-
5	Max pooling	312×1×16	-
6	Convolutional layer	312×1×16	Weights 8×1×16×16 Bias 1×1×16
7	ReLU	312×1×16	-
8	Dropout (50%)	312×1×16	-
9	Max pooling	312×1×16	-
10	Fully connected	1×1×7	Weights 7×4992 Bias 7×1
11	Softmax	1×1×7	-
12	Class output	-	-

In addition to the network architecture, the parameters for training a CNN greatly influence the final accuracy. Unless otherwise indicated, the same training parameters are used across all case studies. Training is ended after eight epochs for Case Study 1, 3 epochs for Case study 2, and 40 epochs for Case Study 3, since this was observed to provide network convergence across all data types and network configurations. All configurations are trained using the ADAM optimizer for gradient descent. 1-D CNNs are trained with an initial learning rate of 0.01, while 2D CNNs are trained with an initial learning rate of 0.001. The number of training samples used for each configuration varies depending on the signal duration used in any given configuration. The duration of the original signal, as well as the dimensions of the processed input data, are presented along with the accuracies for each configuration.

3.4 Case study 1: Case Western Reserve University bearing fault dataset

This case study explores the use of CNNs for identifying healthy bearings and bearings having various fault types. Key parameters for preparing the training data and CNN hyperparameters are varied to identify important trends. The objective here is to identify which variables strongly influence diagnostic accuracy and to identify the most accurate combination of input type and CNN configuration.

3.4.1 Dataset description

The Case Western Reserve University (CWRU) dataset [46] is commonly used to benchmark new network architectures for bearing fault diagnosis. A useful exploration of the dataset as a tool for benchmarking is given by Smith and Randall [47]. It contains vibration measurements from multiple accelerometers mounted on an electric motor containing bearings in various states of health. Bearing faults were artificially introduced using electro-discharge machining. Different fault types were simulated separately on the inner raceway, outer raceway, or one of the rolling elements. The dataset cannot be used directly for training prognosis algorithms since it does not contain natural degradation or gradual changes of health during operation. Damaged bearings were installed onto the fan end or drive end positions of a motor's shaft. Thus, seven fault states are created; one for which all bearings are healthy, and three different damage states for each of the two bearing locations. Using these seven states as labels, the trained CNN shall have to accurately diagnose both the fault type and fault location for a prediction to be counted as correct by the error function.

The motor was operated with loads of 0 to 3 HP and speeds ranging from 1720 to 1797 rpm. Three fault severities were simulated by machining defects of different sizes into the

bearings. Vibration data was sampled at 12 kHz for a duration of 10 seconds for each configuration of motor load, fault severity, and fault type. Additional recordings were conducted with outer race faults in which the bearing was installed by placing the static fault at different locations relative to the accelerometer. That is, directly below the accelerometer, orthogonal to the accelerometer, or opposite from the accelerometer.

3.4.2 Data preparation

Data from different runs of the test bench are divided between training and testing groups prior to performing data augmentation. This ensures an accurate appraisal of the testing accuracy of the trained networks, data from the training set must not overlap with data from the test set, including by being obtained from the same damaged bearing specimen. Importantly, dividing the data prior to data augmentation and order shuffling prevents data from a given run from appearing in both training and testing sets. K-fold cross-validation is used to verify results with $k = 4$. Thus, each network configuration is trained with one quarter of the original experimental data. Rather than using a fully random division into k groups, a constraint is applied such that equal proportions of each class appear in each subset, eliminating the class imbalance problem from consideration. Each testing and validation accuracy is the average obtained over the k trained CNNs.

Though the experimental data contains two channels of vibration data, data from only one accelerometer is used. The fault type and faulty bearing location are used as labels, for a total of seven labels. The three different fault severities and four different loading conditions are lumped into each of these seven labels. The goal of the network is to diagnose the fault type and location irrespective of these factors.

The data augmentation method described in section 2.3 is used for all experiments, with 25% overlap. Six input lengths are chosen, using multiples of the number of samples needed

to span a full shaft rotation at the given sample rate and average rotation rate over all experiments. Prior to training, the data is normalized to have a mean of zero.

3.4.3 Results and discussion

Tables **2** to **5** tabulate the average training and testing accuracies taken over 4-fold cross-validation for various network input sizes, convolutional filter sizes and strides. The results obtained cover a broad range of accuracies, highlighting significant sensitivity to changing hyperparameters. The best accuracy obtained here (80.1%) still falls short of some other researchers' findings using shallow classical CNNs. This seems likely to be a result of their allowing samples extracted from a given experimental run to exist in both the training and testing dataset.

Table 2: Training and testing accuracies for various CNN architectures with no preprocessing

Input type: raw time series

Training accuracy: (%)

Kernel size, stride						
[4,1], [2,1]	72.7	68.8	64.8	71.9	68.0	72.7
[8,1], [2,1]	82.0	86.7	83.6	59.4	65.6	49.2
[16,1], [4 1]	89.1	86.7	89.8	82.8	83.6	67.2
[32,1], [4,1]	93.8	87.5	97.7	86.7	41.4	46.1
[64,1], [4,1]	89.8	**93.8**	96.1	83.6	56.3	50.8
[128,1], [4,1]	81.3	61.7	54.7	61.7	42.2	31.3
[256,1], [4,1]	89.8	87.5	68.8	40.6	28.9	34.4
Input length:	416	832	1248	2496	4992	9984
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1], [2,1]	50.9	49.4	47.7	49.7	41.2	33.5
[8,1], [2,1]	62.6	58.8	57.2	43.2	40.6	31.8
[16,1], [4 1]	65.0	66.7	64.4	54.6	58.3	39.4
[32,1], [4,1]	66.9	67.5	65.3	63.3	33.1	35.0
[64,1], [4,1]	65.9	**71.2**	63.3	53.1	37.4	36.6
[128,1], [4,1]	60.0	48.2	50.8	47.7	35.9	27.0
[256,1], [4,1]	63.1	56.7	49.4	33.1	29.3	35.4
Input length:	416	832	1248	2496	4992	9984
Preprocessed length:	416	832	1248	2496	4992	9984

Table 3: Training and testing accuracies for various CNN architectures with FFT preprocessing

Input type: Fourier spectrum

Training accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	96.9	97.7	**98.4**	100.0	99.2	99.2
[8,1] [2,1]	99.2	96.9	98.4	99.2	97.7	97.7
[16,1] [4 1]	85.9	88.3	93.0	93.8	94.5	69.5
[32,1] [4,1]	71.9	88.3	73.4	82.0	79.7	22.7
[64,1] [4,1]	40.6	45.3	39.1	45.3	34.4	23.4
[128,1] [4,1]	52.3	48.4	47.7	30.5	30.5	20.3
[256,1] [4,1]	42.2	39.8	32.0	38.3	31.3	27.3
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	75.4	74.3	**79.1**	71.4	75.6	66.5
[8,1] [2,1]	71.4	71.9	76.1	71.9	70.6	68.4
[16,1] [4 1]	68.3	69.0	75.1	74.2	68.5	50.5
[32,1] [4,1]	56.2	65.2	53.9	56.8	62.7	25.1
[64,1] [4,1]	32.3	32.6	37.0	34.9	30.3	25.1
[128,1] [4,1]	45.7	40.7	34.9	25.6	30.1	25.1
[256,1] [4,1]	37.0	34.2	26.8	34.1	25.2	25.2
Input length:	208	416	832	1248	2496	4992
Preprocessed length:	416	832	1248	2496	4992	9984

Table 4: Training and testing accuracies for various CNN architectures with envelope spectrum preprocessing

Input type: envelope spectrum

Training accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	71.1	78.1	82.8	83.6	78.1	64.8
[8,1] [2,1]	78.1	85.9	84.4	88.3	82.0	65.6
[16,1] [4 1]	61.5	78.1	81.3	83.6	82.0	78.1
[32,1] [4,1]	79.7	85.2	91.4	87.5	82.0	72.7
[64,1] [4,1]	78.9	90.6	93.0	86.7	82.8	63.3
[128,1] [4,1]	75.0	94.5	93.0	95.3	89.1	80.5
[256,1] [4,1]	87.5	92.2	96.1	**95.3**	93.0	87.5
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	59.4	68.1	61.3	62.9	67.6	55.7
[8,1] [2,1]	53.9	67.2	64.0	72.0	67.0	49.9
[16,1] [4 1]	68.0	67.8	67.6	58.5	65.1	59.7
[32,1] [4,1]	65.5	67.5	71.0	66.0	59.7	53.9
[64,1] [4,1]	66.9	74.6	72.4	66.9	67.5	47.0
[128,1] [4,1]	67.7	73.4	75.7	67.7	68.6	58.5
[256,1] [4,1]	68.8	76.9	76.7	**78.0**	64.5	66.7
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Table 5: Training and testing accuracies for various CNN architectures with spectrogram preprocessing

Input type: spectrogram (window = 104)

Training accuracy: (%)

Kernel size, stride						
[2,2], [1,1]	100.0	100.0	100.0	100.0	100.0	100.0
[4,4], [1,1]	100.0	100.0	100.0	100.0	100.0	100.0
[6,6], [1,1]	100.0	100.0	100.0	100.0	100.0	100.0
[8,8], [1,1]	100.0	99.2	100.0	100.0	61.7	53.1
Input size:	129×7	129×15	129×23	129×47	129×95	129×191
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

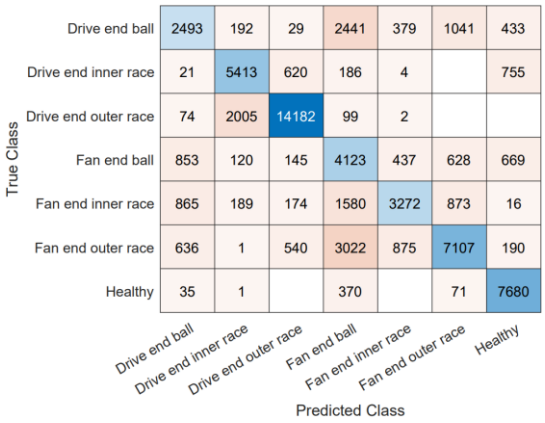
Kernel size, stride						
[2,2], [1,1]	80.0	76.1	70.5	75.1	75.5	70.9
[4,4], [1,1]	75.8	73.7	71.8	71.1	74.8	66.7
[6,6], [1,1]	80.0	**80.1**	71.2	74.6	71.2	64.7
[8,8], [1,1]	75.1	73.2	76.9	71.7	69.7	67.9
Input length:	129×7	129×15	129×23	129×47	129×95	129×191
Original signal length:	416	832	1248	2496	4992	9984

Evidently, different input types favor different hyperparameters for maximizing validation accuracy. Using raw temporal measurements appears best paired with mid-sized kernels and a smaller input space. Using FFT preprocessing achieves poor accuracy if large kernels are used and does not show a strong dependence on input size, whereas envelope spectrum preprocessing works best with a larger kernel and mid-sized input. Using the spectrogram input type provides the most consistently strong diagnostic accuracy, with less sensitivity to changes in input size and kernel size and stride than other input types.

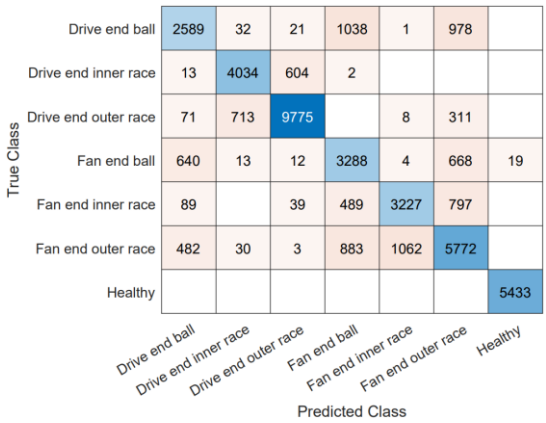
Testing accuracy is consistently lower than training accuracy, suggesting that overfitting remains a significant problem, even with the use of dropout. This might be addressed with a more extensive and more sophisticated data augmentation method. If only the strongest configurations for each input type are taken, the Fourier spectrum, envelope spectrum, and spectrogram seem to perform approximately equally.

Figure 10 shows confusion matrices from the best performing configurations under each input type. Since k-fold cross validation is used with $k = 4$, four confusion matrices can be produced from each configuration. The summation of these four confusion matrices are presented for each input type.

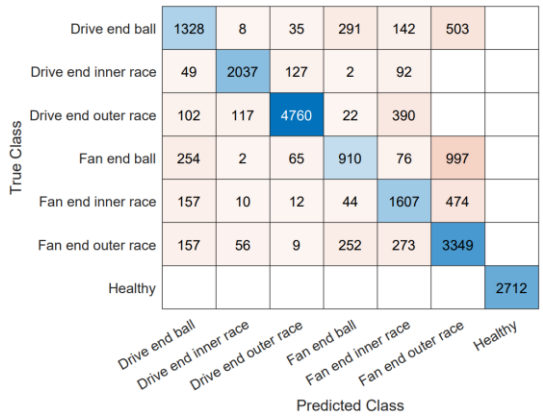
a) Raw time input (71.2 % accuracy)



b) Fourier spectrum input (79.1 % accuracy)



c) Envelope spectrum input (78.0 % accuracy)



d) Spectrogram input (80.1 % accuracy)

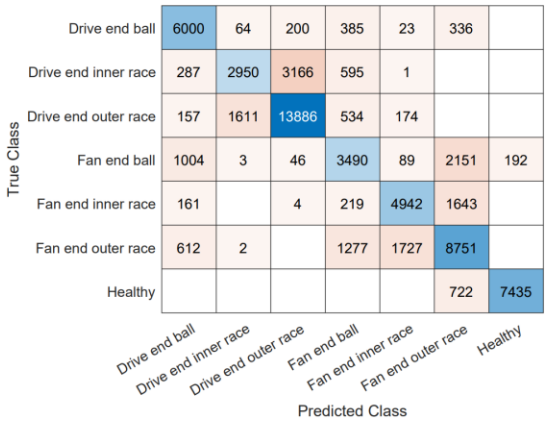


Figure 10: Confusion matrices for a) raw time, b) Fourier spectrum, c) envelope spectrum, and d) spectrogram as inputs to the CNN.

Considering that practical users of diagnosis algorithms may not be concerned whatsoever with the fault type and simply need to know whether the bearing is healthy or needs replacing, it is clear that the best preprocessor is the envelope spectrum. The envelope spectrum preprocessor resulted in no instances of misclassified healthy bearings, nor any faulty bearings falsely classified as healthy. Combined with the study of different CNN architectures, it is suggested that the envelope spectrum is the best input type provided that a suitable network architecture is used.

3.5 Case study 2: Paderborn University bearing fault dataset

This case study contains two parts. In part 1, the same CNN configurations used in section 3.4 are trained with data from artificially damaged bearings and tested with bearings with real damages gained during accelerated life testing. In part 2, these CNN configurations are again used with training and testing datasets both originating from artificially damaged bearing experiments. The purpose of part 1 is to evaluate the cross-domain applicability of the studied CNN configurations in a situation that reflects a real scenario. Part 2 aims to determine the extent to which the inaccuracy observed in part 1 can be attributed to the domain difference. This case study will also reveal whether the trends in hyperparameter selection for each input type are consistent across these two datasets.

3.5.1 Dataset description

The Paderborn University bearing dataset [42] is another popular benchmarking dataset used for bearing fault diagnosis algorithms. An important differentiating characteristic of this dataset is that it includes bearings with seeded faults, as well as bearings with natural faults. Three different methods are used to simulate bearing damage: electric discharge machining, drilling (various diameters), and electric engraving. Outer race and inner race fault types are studied.

3.5.2 Data preparation

For part 1, bearings with artificially introduced bearing damage are used in the training set while testing is done with bearings with natural wear induced during accelerated life testing. The datasets in part 1 are selected to match that of researchers Chen et al. [22] so

that the performance of various traditional CNNs can be directly compared to that of their novel CNN architecture. In the second part, training and testing are both done with bearings having artificial damage. Tables 6 and 7 indicate which experimental runs are included in each dataset for the two parts. The data augmentation procedure described for Case Study 1 is reused for Case Study 2 with the same overlap and input durations. Since the experiments in this dataset use different rotational speeds and sample rates, the input durations used do not correspond with integer numbers of shaft rotations here. However, the reuse of the same input durations allows for direct comparison of the same CNN configurations between datasets.

Table 6: Division of experimental data between training and testing datasets for Case Study 2 part 1

Dataset	Class	Fault origin	Bearing code
Training	Healthy	None	K002
Training	IR damage	Artificial damage	KI01
Training	IR damage	Artificial damage (electric engraver)	KI05
Training	IR damage	Artificial damage (electric engraver)	KI07
Training	OR damage	Artificial damage (EDM machining)	KA01
Training	OR damage	Artificial damage (electric engraver)	KA05
Training	OR damage	Artificial damage (drilled)	KA07
Testing	Healthy	None	KA001
Testing	IR damage	Overload, wrong viscosity, contamination	KI14
Testing	IR damage	Overload, wrong viscosity, contamination	KI16
Testing	IR damage	Overload, wrong viscosity, contamination	KI17
Testing	IR damage	Overload, wrong viscosity, contamination	KI18
Testing	IR damage	Overload, wrong viscosity, contamination	KI21
Testing	OR damage	Overload, wrong viscosity, contamination	KA04
Testing	OR damage	Overload, wrong viscosity, contamination	KA15
Testing	OR damage	Overload, wrong viscosity, contamination	KA16
Testing	OR damage	Overload, wrong viscosity, contamination	KA22
Testing	OR damage	Overload, wrong viscosity, contamination	KA30

Table 7: Division of experimental data between training and testing datasets for Case Study 2 part 2

Dataset	Class	Fault origin	Bearing code
Training	Healthy	None	K004
Training	Healthy	None	K005
Training	Healthy	None	K006
Training	IR damage	Artificial damage	KI01
Training	IR damage	Artificial damage (electric engraver)	KI03
Training	IR damage	Artificial damage (electric engraver)	KI05
Training	OR damage	Artificial damage (electric engraver)	KA06
Training	OR damage	Artificial damage (drilled)	KA07
Training	OR damage	Artificial damage (drilled)	KA08
Testing	Healthy	None	K001
Testing	Healthy	None	K002
Testing	Healthy	None	K003
Testing	IR damage	Artificial damage (electric engraver)	KI07
Testing	IR damage	Artificial damage (electric engraver)	KI08
Testing	OR damage	Artificial damage (EDM machining)	KA01
Testing	OR damage	Artificial damage (electric engraving)	KA03
Testing	OR damage	Artificial damage (electric engraving)	KA05

3.5.3 Results and discussion

3.5.3.1 Part 1: artificial to natural damage

The following results show the accuracy of the previously studied CNN configurations for the cross-domain task described above.

Table 8: Training and testing accuracies for various CNN architectures with no preprocessing

Input type: raw time series

Training accuracy: (%)

Kernel size, stride						
[4,1], [2,1]	62.5	68.8	87.5	81.3	84.4	87.5
[8,1], [2,1]	81.3	75.0	81.3	78.1	84.4	90.6
[16,1], [4 1]	81.3	93.8	87.5	81.3	84.4	93.8
[32,1], [4,1]	75.0	75.0	87.5	93.8	87.5	87.5
[64,1], [4,1]	84.4	84.4	81.3	100.0	100.0	93.8
[128,1], [4,1]	81.3	87.5	93.8	100.0	100.0	100.0
[256,1], [4,1]	75.0	84.4	81.3	96.9	90.6	100.0
Input length:	416	832	1248	2496	4992	9984
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1], [2,1]	47.0	46.1	47.1	37.8	43.5	35.9
[8,1], [2,1]	49.7	46.6	49.3	48.6	48.0	31.5
[16,1], [4 1]	49.6	50.2	51.7	49.5	52.2	52.5
[32,1], [4,1]	47.7	49.0	48.7	49.4	51.5	52.7
[64,1], [4,1]	47.9	49.2	49.1	49.4	**55.0**	50.0
[128,1], [4,1]	45.7	45.5	46.9	46.6	47.9	51.8
[256,1], [4,1]	45.8	45.4	48.7	49.0	48.9	49.7
Input length:	416	832	1248	2496	4992	9984
Preprocessed length:	416	832	1248	2496	4992	9984

Table 9: Training and testing accuracies for various CNN architectures with FFT preprocessing

Input type: Fourier spectrum

Training accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	75.0	93.8	87.5	90.6	90.6	90.6
[8,1] [2,1]	75.0	90.6	90.6	90.6	93.8	93.8
[16,1] [4 1]	68.8	84.4	84.4	96.9	96.9	96.9
[32,1] [4,1]	78.1	84.4	84.4	96.9	87.5	90.6
[64,1] [4,1]	75.0	87.5	90.6	93.8	84.4	90.6
[128,1] [4,1]	84.4	87.5	75.0	84.4	90.6	87.5
[256,1] [4,1]	90.6	78.1	81.3	87.5	90.6	75.0
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	37.8	39.0	41.8	38.9	39.1	41.3
[8,1] [2,1]	38.0	36.3	40.5	43.3	37.4	43.1
[16,1] [4 1]	36.9	42.8	40.7	34.7	48.4	44.0
[32,1] [4,1]	37.3	35.2	42.1	41.0	48.3	46.9
[64,1] [4,1]	38.2	43.4	40.7	39.5	32.3	**55.8**
[128,1] [4,1]	40.2	41.1	42.3	41.6	39.7	41.0
[256,1] [4,1]	37.8	42.2	37.3	41.4	40.8	50.5
Input length:	208	416	832	1248	2496	4992
Preprocessed length:	416	832	1248	2496	4992	9984

Table 10: Training and testing accuracies for various CNN architectures with envelope spectrum preprocessing

Input type: envelope spectrum

Training accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	53.1	71.9	68.8	84.4	78.1	87.5
[8,1] [2,1]	59.4	71.9	59.4	75.0	78.1	78.1
[16,1] [4 1]	65.6	68.8	62.5	78.1	87.5	90.6
[32,1] [4,1]	56.3	71.9	81.3	81.3	71.9	90.6
[64,1] [4,1]	75.0	59.4	68.8	75.0	84.4	75.0
[128,1] [4,1]	53.1	90.6	78.1	84.4	71.9	81.3
[256,1] [4,1]	59.4	78.1	75.0	78.1	90.6	78.1
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	42.2	39.3	37.0	37.2	36.7	36.4
[8,1] [2,1]	42.3	40.3	38.1	36.8	37.4	36.5
[16,1] [4 1]	42.0	39.5	38.2	37.5	37.1	36.7
[32,1] [4,1]	41.9	39.8	37.5	36.7	36.9	36.6
[64,1] [4,1]	41.9	40.0	37.2	37.2	36.5	36.6
[128,1] [4,1]	41.9	40.2	37.6	37.4	36.8	36.5
[256,1] [4,1]	41.7	39.6	37.0	36.9	36.5	36.6
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Table 11: Training and testing accuracies for various CNN architectures with spectrogram preprocessing

Input type: spectrogram (window = 104)

Training accuracy: (%)

Kernel size, stride						
[2,2], [1,1]	81.3	87.5	84.4	78.1	93.8	96.9
[4,4], [1,1]	78.1	78.1	84.4	87.5	96.9	96.9
[6,6], [1,1]	68.8	84.4	93.8	90.6	93.8	90.6
[8,8], [1,1]	75.0	87.5	84.4	84.4	90.6	96.9
Input size:	129× 7	129× 15	129× 23	129× 47	129× 95	129× 191
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[2,2], [1,1]	39.5	40.3	38.4	36.6	48.9	30.4
[4,4], [1,1]	39.5	40.1	34.6	32.8	49.2	49.7
[6,6], [1,1]	39.1	40.4	32.0	30.5	51.9	46.6
[8,8], [1,1]	43.5	37.4	39.0	49.6	50.0	**52.2**
Input length:	129× 7	129× 15	129× 23	129× 47	129× 95	129× 191
Original signal length:	416	832	1248	2496	4992	9984

In general, it is found that all CNN configurations have significantly lower testing accuracy compared to those found using the CWRU dataset. While this is certainly attributable, to some extent, to the disparity between measurements of artificial and natural bearing damage, other factors might make the Paderborn dataset more difficult to learn from compared to the CWRU dataset. Foremost among these factors, is the reduced number of experimental runs from which to learn, as this leads to dataset sparsity.

Unlike Case Study 1, the envelope spectrum appears to be the poorest choice in preprocessor based on overall accuracy. The remaining three preprocessing methods seem to be approximately equal, though all seem too poor to be particularly useful.

The results of Part 1 give broader confirmations of the findings of Chen et al. [22], who demonstrate that classic implementations of CNNs are not able to learn enough useful features from artificially damaged bearings to accurately diagnose real faults.

3.5.3.2 Part 2: artificial to artificial damage

Table 12: Training and testing accuracies for various CNN architectures with no preprocessing

Input type: raw time series

Training accuracy: (%)

Kernel size, stride						
[4,1], [2,1]	81.3	71.9	81.3	93.8	96.9	87.5
[8,1], [2,1]	71.9	71.9	87.5	90.6	90.6	96.9
[16,1], [4 1]	87.5	87.5	90.6	93.8	100.0	100.0
[32,1], [4,1]	87.5	100.0	96.9	100.0	96.9	100.0
[64,1], [4,1]	87.5	87.5	96.9	100.0	100.0	100.0
[128,1], [4,1]	90.6	96.9	90.6	100.0	100.0	100.0
[256,1], [4,1]	90.6	96.9	100.0	100.0	96.9	96.9
Input length:	416	832	1248	2496	4992	9984
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1], [2,1]	33.8	43.2	39.7	48.5	52.3	53.6
[8,1], [2,1]	43.8	50.3	52.5	52.0	53.2	48.8
[16,1], [4 1]	46.3	50.6	50.1	56.2	51.7	48.3
[32,1], [4,1]	50.6	57.3	60.5	54.2	59.6	52.1
[64,1], [4,1]	55.0	55.1	57.6	64.2	60.1	47.2
[128,1], [4,1]	53.6	57.1	**70.0**	62.1	54.3	58.7
[256,1], [4,1]	52.5	58.7	57.1	64.2	62.5	51.7
Input length:	416	832	1248	2496	4992	9984
Preprocessed length:	416	832	1248	2496	4992	9984

Table 13: Training and testing accuracies for various CNN architectures with FFT preprocessing

Input type: Fourier spectrum

Training accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	68.8	90.6	87.5	96.9	96.9	100.0
[8,1] [2,1]	65.6	87.5	93.8	96.9	100.0	100.0
[16,1] [4 1]	56.3	87.5	93.8	93.8	96.9	100.0
[32,1] [4,1]	75.0	87.5	96.9	100.0	100.0	93.8
[64,1] [4,1]	84.4	84.4	87.5	96.9	96.9	100.0
[128,1] [4,1]	78.1	87.5	90.6	90.6	93.8	100.0
[256,1] [4,1]	68.8	90.6	90.6	100.0	96.9	100.0
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	44.7	55.6	58.9	63.7	66.1	62.2
[8,1] [2,1]	43.9	55.3	59.4	68.3	61.9	65.2
[16,1] [4 1]	46.2	52.0	56.0	63.9	**68.7**	67.8
[32,1] [4,1]	47.9	55.7	58.9	61.8	66.4	63.6
[64,1] [4,1]	47.1	50.3	58.0	55.2	65.2	69.1
[128,1] [4,1]	47.9	57.7	55.2	53.9	47.2	56.0
[256,1] [4,1]	42.4	57.6	61.9	63.5	64.0	49.8
Input length:	208	416	832	1248	2496	4992
Preprocessed length:	416	832	1248	2496	4992	9984

Table 14: Training and testing accuracies for various CNN architectures with envelope spectrum preprocessing

Input type: envelope spectrum

Training accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	46.9	50.0	71.9	78.1	71.9	84.4
[8,1] [2,1]	65.6	56.3	68.8	84.4	87.5	90.6
[16,1] [4 1]	59.4	78.1	78.1	75.0	93.8	84.4
[32,1] [4,1]	65.6	59.4	65.6	87.5	81.3	81.3
[64,1] [4,1]	65.6	68.8	78.1	81.3	84.4	78.1
[128,1] [4,1]	53.1	59.4	81.3	87.5	90.6	96.9
[256,1] [4,1]	62.5	53.1	68.8	84.4	81.3	87.5
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	31.8	25.6	22.2	23.8	27.0	31.5
[8,1] [2,1]	28.8	26.5	18.2	20.5	19.5	15.8
[16,1] [4 1]	32.2	30.6	32.3	21.9	23.8	31.1
[32,1] [4,1]	23.5	23.8	21.3	32.2	38.3	38.3
[64,1] [4,1]	29.8	32.4	29.4	37.9	38.3	37.6
[128,1] [4,1]	25.6	27.5	30.9	32.8	31.1	31.2
[256,1] [4,1]	25.2	36.7	24.7	39.2	32.5	**51.6**
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Table 15: Training and testing accuracies for various CNN architectures with spectrogram preprocessing

Input type: spectrogram (window = 104)

Training accuracy: (%)

Kernel size, stride						
[2,2], [1,1]	90.6	96.9	100.0	100.0	84.4	96.9
[4,4], [1,1]	84.4	90.6	96.9	96.9	100.0	100.0
[6,6], [1,1]	93.8	100.0	100.0	100.0	100.0	100.0
[8,8], [1,1]	96.9	93.8	100.0	100.0	100.0	100.0
Input size:	129×7	129×15	129×23	129×47	129×95	129×191
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: : (%)

Kernel size, stride						
[2,2], [1,1]	49.1	57.3	58.7	56.8	59.7	60.9
[4,4], [1,1]	49.9	57.9	62.8	62.8	60.9	**69.5**
[6,6], [1,1]	50.4	57.6	60.6	61.5	60.1	63.6
[8,8], [1,1]	49.6	57.5	60.8	66.8	58.5	69.0
Input length:	129×7	129×15	129×23	129×47	129×95	129×191
Original signal length:	416	832	1248	2496	4992	9984

Part 2 eliminates the underlying domain difference between natural and artificial bearing damage measurements by exclusively using artificial bearing damage. Validation accuracy is improved overall with the notable exception of CNNs trained with envelope spectrum data. This indicates that the inherent difference between artificial and natural bearing damage are a significant, but not sole, contributor to the poor accuracy achieved in Part 1.

As with Case Study 1, results obtained using spectrogram preprocessing appear to be the least sensitive to changes in kernel and input sizes. Other trends linking accuracy and hyperparameter values differ between case study 1 and case study 2. This suggests that different underlying factors including experimental procedure and physical setup influence hyperparameter optimization. This implies that CNNs may need to be tuned for different industrial applications if a universally applicable architecture and training scheme is not developed.

3.6 Case study 3: 2009 PHM challenge gear fault dataset

This case study explores the effectiveness of the previously described CNN configurations for diagnosing various health conditions of a two-stage gear box using the 2009 PHM Challenge dataset *[45]*. The objective of this case study is to determine whether architectures successful for bearing fault detection by CNNs are also able to perform gearbox fault diagnosis.

3.6.1 Dataset description

This dataset contains eight unique health states for the gearbox, each having a different combination of sub-components that are either healthy or artificially damaged. This gives rise to health states with multiple faults, leading to a more complicated diagnosis problem.

Table 16 summarizes the states of the various gears, bearings, and shafts for each of the health states. The dataset contains two channels of vibration measurements and a tachometer signal. For this experiment, only the first channel is used. For all states, four seconds are sampled at a sampling frequency of 66.67 kHz.

Table 16: Summary of gearbox component condition in various labeled states [7]

	Gear				Bearing						Shaft	
Case	**32T**	**96T**	**48T**	**80T**	**IS:IS**	**ID:IS**	**OS:IS**	**IS:OS**	**ID:OS**	**OS:OS**	**Input**	**Output**
Spur 1	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good	Good
Spur 2	Chipped	Good	Eccentric	Good	Good	Good	Good	Good	Good	Good	Good	Good
Spur 3	Good	Good	Eccentric	Good	Good	Good	Good	Good	Good	Good	Good	Good
Spur 4	Good	Good	Eccentric	Broken	Ball	Good	Good	Good	Good	Good	Good	Good
Spur 5	Chipped	Good	Eccentric	Broken	Inner	Ball	Outer	Good	Good	Good	Good	Good
Spur 6	Good	Good	Good	Broken	Inner	Ball	Outer	Good	Good	Good	Imbalance	Good
Spur 7	Good	Good	Good	Good	Inner	Good	Good	Good	Good	Good	Good	Keyway Sheared
Spur 8	Good	Good	Good	Good	Good	Ball	Outer	Good	Good	Good	Imbalance	Good

IS = Input Shaft; IS = Input Side; ID = Idler Shaft; OS = Ouput Side; OS = Output Shaft

3.6.2 Data preparation

The same data augmentation procedure described above is used here to generate many more training and testing samples of different sizes form the original measurements. Again, k-fold cross validation is used, with k = 3. One difference in procedure was mandated for the process using envelope spectrum preprocessing; data was normalized to have a zero mean as well as having a standard deviation of one. This was necessary to achieve network convergence under the same learning parameters as all other preprocessing methods.

Unlike the datasets used in the previous two case studies, there is only one measurement obtained for each health state. This means that training and testing data cannot be obtained from different sets of damaged components. This leads to a simpler ML problem for which less generalization is needed to achieve high testing accuracies. The results below indeed

show that the testing accuracies achieved here are much higher than the previous case studies.

3.6.3 Results and discussion

Table 17: Training and testing accuracies for various CNN architectures with no preprocessing

Input type: raw time series

Training accuracy: (%)

Kernel size, stride						
[4,1], [2,1]	41.7	56.3	54.2	89.6	100.0	100.0
[8,1], [2,1]	45.8	77.1	83.3	100.0	100.0	100.0
[16,1], [4 1]	39.6	66.7	79.2	85.4	95.8	100.0
[32,1], [4,1]	68.8	85.4	81.3	95.8	100.0	100.0
[64,1], [4,1]	72.9	93.8	85.4	93.8	100.0	100.0
[128,1], [4,1]	77.1	97.9	97.9	93.8	100.0	100.0
[256,1], [4,1]	89.6	97.9	100.0	97.9	100.0	100.0
Input length:	416	832	1248	2496	4992	9984
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1], [2,1]	34.3	27.7	31.4	27.1	27.2	20.3
[8,1], [2,1]	44.7	53.5	52.2	46.4	31.5	24.7
[16,1], [4 1]	53.4	64.0	67.7	68.1	55.3	49.2
[32,1], [4,1]	64.2	75.7	80.0	74.4	80.2	51.6
[64,1], [4,1]	73.2	84.9	93.8	93.1	83.6	59.7
[128,1], [4,1]	78.2	90.9	96.1	91.7	93.2	69.0
[256,1], [4,1]	79.2	91.5	96.4	98.1	96.7	92.6
Input length:	416	832	1248	2496	4992	9984
Preprocessed length:	416	832	1248	2496	4992	9984

Table 18: Training and testing accuracies for various CNN architectures with FFT preprocessing

Input type: Fourier spectrum

Training accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	85.4	95.8	100.0	100.0	100.0	100.0
[8,1] [2,1]	79.2	95.8	100.0	100.0	100.0	100.0
[16,1] [4 1]	77.1	97.9	100.0	100.0	100.0	100.0
[32,1] [4,1]	77.1	91.7	97.9	100.0	100.0	100.0
[64,1] [4,1]	81.3	95.8	97.9	100.0	100.0	100.0
[128,1] [4,1]	87.5	95.8	100.0	100.0	100.0	100.0
[256,1] [4,1]	83.3	95.8	100.0	97.9	95.8	100.0
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	81.3	94.1	97.5	99.7	99.9	**100.0**
[8,1] [2,1]	82.3	95.1	97.9	99.6	99.8	**100.0**
[16,1] [4 1]	81.4	93.8	98.6	99.2	99.0	99.8
[32,1] [4,1]	84.4	95.1	98.3	99.3	99.5	99.8
[64,1] [4,1]	85.6	96.2	98.8	99.6	99.7	99.0
[128,1] [4,1]	87.3	96.7	98.9	99.0	99.0	99.9
[256,1] [4,1]	88.4	97.0	99.1	98.0	98.0	99.7
Input length:	208	416	832	1248	2496	4992
Preprocessed length:	416	832	1248	2496	4992	9984

Table 19: Training and testing accuracies for various CNN architectures with envelope spectrum preprocessing

Input type: envelope spectrum

Training accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	31.3	52.1	60.4	89.6	100.0	100.0
[8,1] [2,1]	20.8	39.6	56.3	97.9	100.0	100.0
[16,1] [4 1]	31.3	31.3	56.3	89.6	100.0	100.0
[32,1] [4,1]	31.3	52.1	45.8	81.3	97.9	100.0
[64,1] [4,1]	31.3	47.9	54.2	77.1	100.0	100.0
[128,1] [4,1]	25.0	47.9	52.1	83.3	97.9	100.0
[256,1] [4,1]	35.4	54.2	70.8	89.6	95.8	100.0
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[4,1] [2,1]	20.8	24.9	28.1	34.3	43.0	58.6
[8,1] [2,1]	21.9	24.6	27.3	32.5	40.5	55.6
[16,1] [4 1]	21.8	26.2	29.6	35.8	46.8	58.8
[32,1] [4,1]	22.8	27.3	29.8	37.0	46.0	59.1
[64,1] [4,1]	22.7	27.1	30.8	36.3	46.0	63.0
[128,1] [4,1]	21.7	25.8	30.1	37.8	45.4	65.4
[256,1] [4,1]	19.5	26.0	29.1	36.6	46.1	63.3
Input length:	208	416	832	1248	2496	4992
Original signal length:	416	832	1248	2496	4992	9984

Table 20: Training and testing accuracies for various CNN architectures with spectrogram preprocessing

Input type: spectrogram (window = 104)

Training accuracy: (%)

Kernel size, stride						
[2,2], [1,1]	95.8	97.9	100.0	100.0	100.0	100.0
[4,4], [1,1]	95.8	97.9	100.0	100.0	100.0	100.0
[6,6], [1,1]	95.8	100.0	100.0	100.0	100.0	100.0
[8,8], [1,1]	91.7	97.9	100.0	100.0	100.0	100.0
Input size:	129× 7	129× 15	129× 23	129× 47	129× 95	129× 191
Original signal length:	416	832	1248	2496	4992	9984

Testing accuracy: (%)

Kernel size, stride						
[2,2], [1,1]	75.9	87.4	95.8	96.1	98.4	97.2
[4,4], [1,1]	81.7	92.5	97.3	98.8	98.6	99.0
[6,6], [1,1]	82.3	93.6	97.8	99.2	**99.2**	98.2
[8,8], [1,1]	83.9	92.2	97.2	99.1	98.7	92.8
Input length:	129× 7	129× 15	129× 23	129× 47	129× 95	129× 191
Original signal length:	416	832	1248	2496	4992	9984

Despite the increased complexity of the mechanical system studied, the diagnostic abilities of the studied CNNs appear much greater for this dataset. This is almost certainly a result of the fact that the data from all experimental runs appear in both training and testing, even if unique samples created during data augmentation do not appear in both datasets. As with the other case studies, different preprocessing methods yield different patterns in which kernel sizes and input sizes perform best.

Using the raw time input type appears to only be successful when larger kernels are used. Moreover, performance is somewhat improved by using mid-sized inputs. FFT input

types appear to work very well irrespective of kernel size and benefit somewhat by using larger inputs. The results from envelope spectrum contrast starkly with those of other preprocessing types – the average testing accuracy achieved is much poorer for the configurations trained here. It appears that a longer input duration than studied here would be needed for the envelope spectrum to be accurate, based on the upward trend in the left-to-right direction in Table 19. Spectrogram preprocessing gives strong results overall with performance peaking when paired with mid-sized kernels and larger inputs.

3.7 Chapter Discussion and Conclusions

Some insights can be extrapolated with respect to which CNN configurations are best suited to each input type. The only commonality between all three datasets studied is the low sensitivity to hyperparameter selection in achieving high accuracy for models trained with spectrograms. This makes them the safest choice if extensive hyperparameter optimization is not possible.

When varying input size and kernel size on two-layer CNNs, distinct patterns emerge showing different trends in validation accuracy when different input data types are used. Furthermore, the three datasets studied yield different patterns and vastly different accuracies from the same CNNs. It is clear that there is a complex relationship between input type and hyperparameter optimization that varies for each dataset, so it will be difficult to develop a single CNN architecture and preprocessor combination that provides a maximum accuracy for any given dataset. Case Study 2 highlights the ineffectiveness of classical CNNs for cross-domain problems involving training with artificially damaged bearings and testing with real damage, suggesting that they would not be adequate for real industrial applications.

The comparatively high performance of the studied CNNs with the PHM 2009 dataset in Case Study 3 show how extracting training and testing samples from the same experimental run leads to a far easier problem. This supports the findings of Pandhare et al. [27], who demonstrate that diagnostic accuracy can drop from 95-100% when experimental data is mixed to around 60% when experimental data contained in training and testing datasets is mutually exclusive. It seems probable that researchers finding greater accuracies on the CWRU dataset using classically shallow CNNs achieve such results by "contaminating" validation datasets in this way, a possibility explored in the next chapter. A frequent claim in studies such as this is that the application of deep learning for machine fault diagnosis will eliminate the need for a human expert, presumably an expert on the mechanical system being diagnosed. However true that claim might be, it neglects the fact that a different sort of expert is needed to design a viable solution using deep learning methods. Clearly, useful diagnoses cannot be achieved without a preprocessor and CNN architecture that is well suited for the target domain. It is also clear that misleadingly high diagnosis accuracies can be achieved if data augmentation is performed on the experimental data before the data is randomly shuffled and split into training and testing datasets, leading to contamination of the testing set.

If the diagnostic problem is fairly constructed to reflect real-world challenges, it appears that classical shallow CNNs are, irrespective of kernel size and input size, not able to perform in a manner that would motivate industrial implementation. However, the present study is not an exhaustive exploration of CNNs; other hyperparameters can be altered to give many more network architectures.

Furthermore, this chapter only looked at simple 2-layer CNNs. There exist very deep CNNs for image recognition tasks, with tens of convolutional layers and finely tuned hyperparameters. It might be better to apply these existing networks to fault diagnosis than

to invest effort in designing diagnosis CNNs from scratch. This only requires the development of an input feature that is useful and compatible with existing deep CNNs. This interesting notion is also explored in the following chapter.

Chapter 4

Benchmarking Domain Shift in Fault Diagnosis

Parts of this chapter have been submitted for publication.

ABSTRACT

This chapter investigates the use of the Case Western Reserve University (CWRU) bearing dataset for benchmarking bearing fault diagnosis convolutional neural networks (CNNs) in a domain shift problem. The common method for using the CWRU dataset for demonstrating domain shift is described and a potential flaw is identified. It is argued that the accepted procedure of constructing training and testing datasets with different operating conditions does not constitute a useful domain shift problem since the same physical bearing specimens exist in both training and testing sets. To address this while using the CWRU dataset, an alternative benchmarking framework is proposed that constructs training and testing datasets with independent sets of bearing specimens. The original and the proposed benchmarking frameworks are compared by training a set of commonly cited diagnosis CNNs within each framework. The results indicate that the original framework allows CNNs to learn features related to specific bearing specimens and may not be able to generalize for different bearings. It is also found that using existing state-of-the-art deep CNNs from other fields in machine learning research may currently present a more efficient option than developing custom CNN architectures for diagnosis when large machine fault datasets are unavailable.

4.1 Introduction

This chapter is focused on a particular benchmarking practice that is often used with the CWRU dataset. In the previous chapter, this dataset was used to explore relative performance of various CNN configurations and data types. In benchmarking, where absolute performance metrics are of utmost importance, it is most common to use the same validation scheme that is used by previous authors so that direct comparisons can be drawn across works by other researchers. The CWRU dataset is large and the experimental conditions are many-faceted, leading to many possible use cases for researchers. The problematic use case that is explored here was not intended by the dataset's originators; it arises from the fact that the details of the experimental methodology used to obtain the laboratory vibration signals may have been misunderstood and the fact that researchers have simply followed the methods of previous studies without careful consideration of the stated experimental parameters. The key characteristics of the CWRU dataset are summarized again in the following paragraph.

The dataset contains vibration data measured from an electric motor within which either of the two roller bearings have been replaced with artificially damaged bearings. The motor is run at loads of zero to three HP, with increments of one HP, and different speeds ranging from 1730 RPM to 1797 RPM. For each bearing location and operating condition, three different sizes of artificial faults are studied, with faults being machined onto either the inner raceway, outer raceway, or a rolling element. Importantly, the bearing specimens containing different sized faults are reused to generate data at the different speeds studied. This fact is ignored or unknown to users of the original benchmarking framework. Two accelerometers are mounted to the motor casing near the two bearing locations. The accelerometers are

sampled at 12 kHz for the majority of experiments, while some additional experiments were conducted at 48 kHz, but not used here.

The existence of different operating conditions in the dataset allow researchers to use it to test the robustness of their algorithms when training on one operating condition and testing it on each of the other operating conditions independently. This framework is thought to present a greater challenge, compared to building training and testing datasets with overlapping operating conditions. Furthermore, this benchmarking framework is supposed to be more reflective of real industrial scenarios, where monitored assets operate at a range of speeds. A handful of state-of-the-art diagnostic algorithms have been validated using this framework. These include the deep neural network-based algorithm by Jia et al. [48], Deep Convolutional Neural Networks with Wide First-layer Kernels (WDCNN) by Zhang et al. [49], Convolution Neural Networks with Training Interference (TICNN) by Zhang et al. [50], and a CNN convolutional neural network based on a capsule network with an inception block (ICN) by Zhu et al. [51]. Zhu et al. also used this framework to benchmark AlexNet, ResNet, and a deep inception net with atrous convolution (ACDIN), which was originally introduced by Chen et al. [22].

The results in this paper indicate a possible flaw in the generally accepted framework for constructing training and testing datasets. This flaw arises from the fact that the measurements at different operating conditions provided in the CWRU dataset are created using the same faulted bearing specimens, re-measured at different speeds. This allows networks trained in this framework to learn RPM-independent features related to specific bearing specimens and not necessarily features generalized to the fault type in unseen bearings. The result is that trained networks have an advantage when compared to networks that have been trained on differently organized datasets. To demonstrate the difference between this framework and one that is perhaps more realistic, a new framework is proposed

that groups datasets by fault size, preventing bearing specimens from appearing in both training and testing datasets. A visualization of this difference in dataset construction is presented in Figure 11. The proposed framework is regarded as being more realistic since, in industrial situations, encountered data may involve known fault types but the faulted bearings themselves will always be unseen.

4.2 Theory of transfer learning

Transfer learning is generally defined as the application of knowledge learned in one task to a different but related task. Transfer learning is expected to be a very important tool for developing more robust fault diagnosis algorithms that can, for example, diagnose both motors and generators without reconfiguration. Lei et al. [6] summarize the current state of knowledge in transfer learning and describe scenarios and factors giving rise to transfer learning in the field of fault diagnosis. Here, a simple technique is used that retrains AlexNet and ResNet, which have been pre-trained for image classification in large image datasets. Using pre-trained deep networks can greatly reduce the time spent training and make it possible to use deeper network structures than would otherwise be practical, often resulting in stronger performance. The details of the technique are described in section 4.3.4.

4.3 Methodology

4.3.1 Dataset preparation

The overall objective of dataset preparation is to use the available laboratory data to construct a problem that reflects the real diagnosis problem as closely as possible. The original framework mainly considers the case wherein the target for diagnosis operates at a different RPM from the training data. Therefore, it uses sub-datasets at each speed to train networks and test networks with data at different operating speeds. Each dataset in the

original framework contains measurements from each of the created bearing specimens. However, if one were to consider a real industrial situation, it would be far more likely to have information about the target domain operating speed than to have experimental information pertaining to specific faulted bearing specimens, or even to have very close approximations of these faulted bearing specimens. Addressing this, the proposed framework trains networks on complete datasets composed of data from a minimum number of bearing specimens, and tests networks with datasets containing an entirely different collection of bearing specimens.

Samples are labeled with one of seven classes that correspond to a fault type, if any, and location of the faulty bearing, which may be at the drive or fan end of the motor. The three fault types are defined by the location of the fault (on the inner race, outer race, or rolling element) and may be present at either of the two bearing locations. This gives six fault scenarios in addition to the healthy state, for a total of seven classes.

The CWRU dataset contains additional measurements for outer race faults in which the fault itself is placed at zero degrees, 90 degrees, and 180 degrees with respect to the accelerometer. For simplicity and to balance the dataset, only one of these is used for each operating condition and fault size combination. Wherever possible, the zero degree measurement is used for the outer race fault samples. For the few instances where a zero degree measurement does not exist, a 90 degree measurement is substituted instead. In this way, the same number of samples for each fault type exists in the datasets.

Datasets for the original framework are labeled A, B, and C in keeping with the practice used by previous studies [48]–[51] to allow direct comparisons to be made with this work. The datasets used in the proposed framework are labeled D, E, and F.

Figure 11 shows the fundamental difference in how the original and proposed frameworks divide data into subsets using an example of a single fault class. Although the original

framework does use independent measurements in each of dataset A, B, and C, these measurements originate from a shared set of damaged bearings. Moreover, the actual RPM changes in the dataset are not significant (Table 21). This makes the transfer of knowledge between these datasets relatively easy for machine learning algorithms to solve. It must be acknowledged that it may not have been clear to authors using the original framework that the same bearings are present in datasets A, B, and C, as this is not obvious without a close inspection of the CWRU dataset's download page. Furthermore, it should be noted that the original framework is a product of iterative research using the CWRU dataset in different ways, and not likely the intention during the original development of the CWRU dataset to be used in this way.

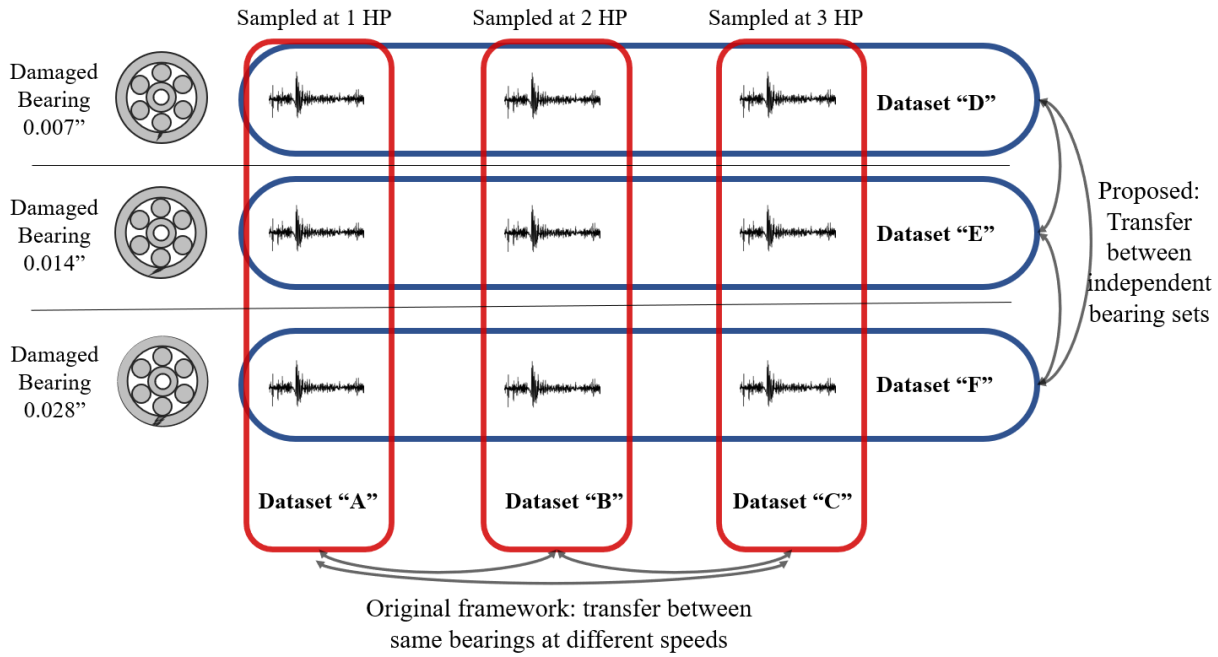


Figure 11: Basis of dividing bearings between subsets in the original framework (red) and the proposed framework (blue) for a single fault class

The parameters used to define the datasets in the original framework and in the proposed framework are shown in Table 21 and Table 22.

Table 21: Dataset names and parameters for the original framework

Rotational speed (RPM)	Load (HP)	Damage Size (in)	Name
1772	1	0.007, 0.014, 0.021	A
1750	2	0.007, 0.014, 0.021	B
1730	3	0.007, 0.014, 0.021	C

Table 22: Dataset names and parameters for the proposed framework

Rotational speed (RPM)	Load (HP)	Damage Size (in)	Name
1772, 1750, 1730	1, 2, 3	0.007	D
1772, 1750, 1730	1, 2, 3	0.014	E
1772, 1750, 1730	1, 2, 3	0.021	F

4.3.2 Data augmentation

Zhang et al. show that increasing the number of training samples by increasing the overlap not only improves network performance, but also decreases the standard deviation in accuracy by diminishing the influence of the random initial network weights [49]. For all of the trained networks, a sliding window with 97% overlap is used, meaning that the stride length is 3% of the window length. Different window lengths are used, per the requirements of each network architecture studied. The window lengths are listed in section 4.3.4.

4.3.3 Data preprocessing

Two network architectures that were purpose-built for machine fault diagnosis are recreated: ACDIN and WDCNN. Both are designed to use a one-dimensional input vector, which can be either raw temporal data or the single-sided Fourier spectrum obtained with the FFT. Results are presented for both input types. To match the input size required by the network architecture, the window size used in data augmentation is adjusted. For spectra input types, the window size is double the length of the final input vector since spectra

produced by the FFT contain only half the elements of the raw input used. The training samples created by the FFT have the added step of having their standard deviations adjusted to one and their means adjusted to zero.

The novel implementation of transfer learning using pretrained deep CNNs is demonstrated using AlexNet and ResNet. This novelty arises from the way that data from both accelerometers are processed and fed into the networks using an additional convolutional layer that occurs prior to the input of the pre-trained networks. Input features are constructed using two spectrograms layered depth wise, which are obtained from the two accelerometer channels. The size of the time-frequency representation created by the spectrogram can be controlled using parameters of the spectrogram transformation: the length of the original signal, the window size, the amount of overlap, and the number of discrete Fourier transform (DFT) points used. A window size of 104 points is used, with a 54 point overlap and 452 DFT points. The resulting image (of size [227, 227]) is cropped to the required network input size in the case of ResNet, keeping the lower-left portion. Table 23 summarizes the key parameters of the data preprocessing used to create the training samples.

Table 23: Summary of data preprocessing inputs and outputs for the network configurations studied

Network	Input type	Original signal length	Accelerometer channels used	Processed input size
ACDIN	Time	5118	1	[5118, 1]
ACDIN	Spectra	10236	1	[5118, 1]
WDCNN	Time	2048	1	[2048, 1]
WDCNN	Spectra	4096	1	[2048, 1]
AlexNet	Spectrogram	11500	1, 2	[227, 227, 2]
ResNet	Spectrogram	11500	1, 2	[224, 224, 2]

4.3.4 Network architectures and training parameters

ACDIN and WDCNN are reconstructed per the descriptions provided in their respective original publications [22], [49]. The versions of AlexNet and ResNet used herein have been pre-trained on large datasets to classify 1000 types of images. To prepare the networks to be used in this new fault diagnosis task, some modifications where required prior to re-training. Firstly, the final fully connected layers of the network are reinitialized. Moreover, the final fully connected layer made up of 1000 neurons, corresponding to the classes in the original task, are replaced with fully connected layers with 7 neurons, for the new task.

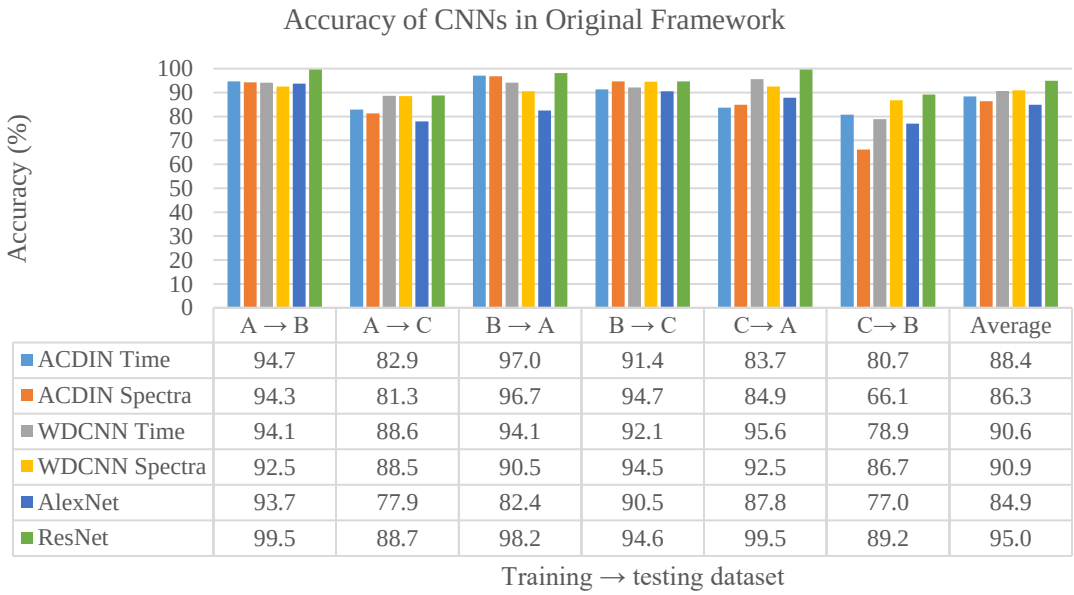
In addition to modifying the final layers of the network, the inputs of these two pre-trained networks must be made to match the input features created by preprocessing. AlexNet and ResNet are trained using 227 and 224-pixel square RGB images, respectively. The RGB channels correspond to three channels of depth. However, the source data used herein contains only two channels originating from the two accelerometers. To create a third channel of meaningful information, the present study uses an approach that has, to the author's knowledge, not yet been used in fault diagnosis. An additional convolutional layer is added before the first layer of AlexNet and ResNet. This new layer has 3 kernels of size [5, 5, 2] with stride [1, 1]. The output of this layer is the convolution of the original two spectrograms, has the same height and width as the original image, and has three depth wise channels. The output from the new layer is used directly as the input to the pre-trained networks.

The Adam optimizer for stochastic gradient descent is used for training all networks with an initial learning rate of 0.0005. Since a high degree of overlap is used during data augmentation, a smaller number of training epochs can be used to achieve network convergence. The networks were trained for as many iterations as were required to achieve

a constant gradient; ACDIN is trained for three epochs, WDCNN and AlexNet are trained for four epochs, and ResNet is trained for just one epoch. The relatively low number of epochs required is mainly the consequence of the extensive data augmentation used.

4.4 Results and discussion

Figure 12 presents the accuracies calculated for all the studied networks in the original framework trained on each of the three datasets (A, B, and C) and tested on the remaining datasets.



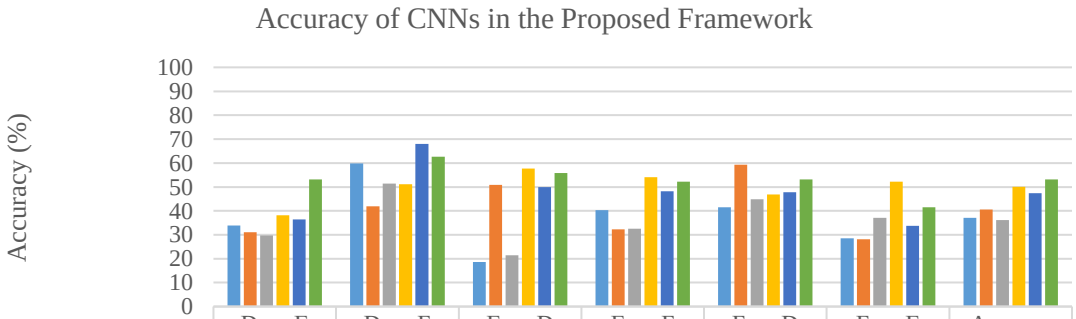
	A → B	A → C	B → A	B → C	C→ A	C→ B	Average
ACDIN Time	94.7	82.9	97.0	91.4	83.7	80.7	88.4
ACDIN Spectra	94.3	81.3	96.7	94.7	84.9	66.1	86.3
WDCNN Time	94.1	88.6	94.1	92.1	95.6	78.9	90.6
WDCNN Spectra	92.5	88.5	90.5	94.5	92.5	86.7	90.9
AlexNet	93.7	77.9	82.4	90.5	87.8	77.0	84.9
ResNet	99.5	88.7	98.2	94.6	99.5	89.2	95.0

Figure 12: Testing accuracy of all networks for each cross-domain case in the original framework

On average, all networks perform well and correspond closely to the findings of other authors, using the same analysis assumptions used in the original studies. Neither time nor spectra input types show a clear advantage over the other. ResNet appears to be the strongest, on average performing 4.1% better than WDCNN trained with frequency spectra. It may be worth remarking on the variability of scores when benchmarked by different researchers. The authors of WDCNN [49] found an average accuracy of 90.0% using spectra inputs, while Zhu et al. [51] found that it scored 88.0% on average using raw time samples.

The difference in input types is possibly due to a misinterpretation of WDCNN, as it was originally proposed to be trained with spectra. Zhu et al. also found an average accuracy of 84.0% for ACDIN when trained with raw time segments, providing a slightly larger discrepancy. Some amount of variability can be attributed to random network weight initialization, but there are many other factors that could have given rise to these differences. If all authors recreated network architectures accurately, differences in batch sizes, data normalization, learning rate and, most importantly, data augmentation can make a significant difference to the final accuracy. Resnet's average score matches that of Zhu et al.'s implementation of ResNet, though surprisingly, AlexNet did not perform as well as Zhu et al.'s version.

Next, the testing accuracies for networks trained in the proposed framework are shown in Figure 13.



	D → E	D → F	E → D	E → F	F → D	F → E	Average
ACDIN Time	33.9	59.8	18.7	40.4	41.5	28.5	37.1
ACDIN Spectra	31.0	41.9	50.8	32.2	59.3	28.2	40.6
WDCNN Time	29.7	51.5	21.5	32.5	44.9	37.1	36.2
WDCNN Spectra	38.1	51.2	57.7	54.1	46.8	52.2	50.0
AlexNet	36.5	68.0	50.0	48.2	47.7	33.8	47.4
ResNet	53.2	62.6	55.9	52.3	53.2	41.4	53.1

Figure 13: Testing accuracy of all networks for each cross-domain case in the proposed framework

The average accuracies of networks in the proposed framework decrease significantly. As in the original framework, the retrained ResNet appears to outperform all other networks overall. However, in the proposed framework, AlexNet now outperforms most of the ACDIN

and WDCNN networks. In general, the proposed framework seems to slightly favor state-of-the-art networks when they are trained with spectra rather than time measurements, especially for case E → D.

To explain why CNNs perform poorly in the proposed framework compared to the original framework, it is useful to view the statistics of the sub-datasets used in training and testing. Table 24 contains statistical measures of the raw vibration signals of each faulty class within each dataset. The statistical measures used are kurtosis, crest factor, shape factor, and margin factor, defined in equations (8)-(11). These were selected because they are sensitive to the bearing condition [52].

Kurtosis:

$$k = E\left[\left(\frac{x-\mu}{\sigma}\right)^4\right] \quad \text{Eq. (8)}$$

where k is the kurtosis of input vector x, μ is the mean, and σ is the standard deviation.

Crest factor:

$$C = \frac{x_{peak}}{x_{rms}} \quad \text{Eq. (9)}$$

where C is the crest factor of vector x, x_{peak} is the maximum value in x, and x_{rms} is the root mean square of x.

Shape factor:

$$SF = \frac{x_{rms}}{\frac{1}{N}\sum_{i=1}^{N}|x_i|} \quad \text{Eq. (10)}$$

where SF is the shape factor of x, or equivelantly, the RMS divided by the mean of the absolute value of x.

Margin factor:

$$MF = \frac{max|x_i|}{\left(\frac{1}{N}\sum_{i=1}^{N}\sqrt{|x_i|}\right)^2} \quad \text{Eq. (11)}$$

where MF is the margin factor of *x*.

It is most interesting to see how these features vary, or not, between the original sets A, B, and C and the proposed sets D, E, and F. The overall homogeneity of datasets in the original framework is exemplified in the lower standard deviations obtained from the original sets when compared to the proposed sets. This explains some of the increased difficulty in reaching high accuracy in the proposed framework.

The decrease in perfromance by algorithms from the original framework when compared to the proposed framework is notable. Even more so considering that ACDIN and WDCNN algorithms were originally developed for domain shift, that is, to diagnose bearings when the training domain differs from the target domain.

Table 24: Vibration signal parameters in each faulty class for each dataset

FOR CLASS: DE_BALL							Standard deviation	
Dataset	**A**	**B**	**C**	**D**	**E**	**F**	**(ABC)**	**(DEF)**
Kurtosis	8.22	6.88	5.55	3.37	10.69	6.09	1.09	3.02
Crest Factor	13.05	16.30	15.56	5.68	16.38	14.05	1.39	4.59
Shape Factor	1.35	1.32	1.29	1.27	1.39	1.29	0.03	0.05
Margin Factor	13.05	16.30	15.56	5.68	16.38	14.05	1.39	4.59
FOR CLASS: DE_IR								
Dataset	**A**	**B**	**C**	**D**	**E**	**F**	**(ABC)**	**(DEF)**
Kurtosis	13.26	15.88	14.15	5.20	23.68	12.59	1.09	7.59
Crest Factor	14.15	13.23	13.61	6.18	15.21	10.61	0.38	3.69
Shape Factor	1.55	1.59	1.54	1.35	1.53	1.57	0.02	0.10
Margin Factor	14.15	13.23	13.61	6.18	15.21	10.61	0.38	3.69
FOR CLASS: DE_OR								
Dataset	**A**	**B**	**C**	**D**	**E**	**F**	**(ABC)**	**(DEF)**
Kurtosis	30.45	34.46	32.45	11.37	3.23	36.79	1.63	14.29
Crest Factor	17.12	18.68	18.51	7.24	6.11	15.76	0.70	4.31
Shape Factor	1.93	1.96	1.94	1.64	1.27	2.08	0.02	0.33
Margin Factor	17.12	18.68	18.51	7.24	6.11	15.76	0.70	4.31
FOR CLASS: FE_BALL								
Dataset	**A**	**B**	**C**	**D**	**E**	**F**	**(ABC)**	**(DEF)**
Kurtosis	27.63	15.63	14.05	12.14	14.04	15.74	6.06	1.47
Crest Factor	23.99	17.06	17.89	25.65	15.98	22.22	3.09	4.00
Shape Factor	1.47	1.52	1.42	1.32	1.54	1.36	0.04	0.10
Margin Factor	23.99	17.06	17.89	25.65	15.98	22.22	3.09	4.00
FOR CLASS: FE_IR								
Dataset	**A**	**B**	**C**	**D**	**E**	**F**	**(ABC)**	**(DEF)**
Kurtosis	8.22	7.85	7.20	7.44	4.98	6.43	0.42	1.01
Crest Factor	8.76	11.36	8.21	8.31	9.18	8.95	1.37	0.37
Shape Factor	1.42	1.41	1.40	1.41	1.34	1.38	0.01	0.03
Margin Factor	8.76	11.36	8.21	8.31	9.18	8.95	1.37	0.37
FOR CLASS: FE_OR								
Dataset	**A**	**B**	**C**	**D**	**E**	**F**	**(ABC)**	**(DEF)**
Kurtosis	22.15	19.97	25.34	22.49	18.58	4.50	2.21	7.73
Crest Factor	13.45	12.01	12.07	11.98	11.05	7.23	0.67	2.06
Shape Factor	1.61	1.71	1.87	1.74	1.73	1.32	0.11	0.20
Margin Factor	13.45	12.01	12.07	11.98	11.05	7.23	0.67	2.06

4.5 Chapter Conclusions

In this chapter, previous results that show ACDIN, WDCNN, AlexNet, and ResNet performing well in the original diagnostic framework have been reproduced. However, it has also been shown that diagnostic accuracy decreases on average by approximately 45% in the proposed benchmark framework. An argument that the proposed framework is a closer representation of real situations than the original framework is presented. The primary difference between the original and proposed framework lies in the fact that the original framework contains repeated measurements of the same faulted bearing specimens across training and testing datasets, while the proposed framework does not.

Considering that the goal of developing cross-domain capable CNNs is to create diagnostic algorithms that can detect faults in settings different from the original training test bench, this poses a noteworthy shortcoming. If machine learning algorithms are to succeed in practical domain shift scenarios, they must be trained and tested with datasets that better represent problems encountered in the field. When using the CWRU dataset, the proposed framework is an improvement in this regard, though algorithms capable of achieving higher accuracies in the proposed framework will need to be developed.

The need for this investigation ultimately arises from the limited amount of data available to researchers describing machine faults. Other datasets, such as the Paderborn University bearing fault dataset [53] can also provide a useful domain shift framework, since it allows for training on artificial or seeded faults and testing with naturally generated faults over accelerated life testing, as was done by Chen et al. [22]. Ideally, future work will focus on training algorithms with easily obtainable test bench data and attempting to apply learned models to field measurements, as was done by Yang et al. [54]. Domain shift problems between artificial and natural faults, or between laboratory and field data, should eventually

become the dominant form of domain shift in condition monitoring. The CWRU dataset can still be useful to researchers, though this paper has demonstrated that insufficient consideration of its experimental parameters has been made in previous studies.

Besides considering different datasets for domain shift problems, it might also be beneficial for researchers to transition away from considering MHM as a diagnosis problem and to formulate MHM as a prognosis problem. For this, ML can be used to estimate the current machine health in terms of a continuous number reflecting the degree of wear. This might offer several advantages. Firstly, it acknowledges that actual maintenance decisions are not often made on the basis of fault type, but simply on whether a bearing is healthy or not. Secondly, it accounts for the spectrum of conditions between healthy and faulted, and would allow operators to leave a bearing in service while it is only partially damaged. Continually estimating the bearing's health over its lifetime would allow one to forecast the time when it will cross some pre-determined health threshold and require replacement. Compared to a simple diagnosis framework, a prognostic system would lend greater flexibility and a wider range of possible maintenance strategies. This leaves out the possibility of diagnostic-prognostic ensemble algorithms, which are discussed later as possible future work. The following two chapters present two methods for performing health indicator estimations.

Chapter 5

Influence of Data Augmentation in Prognosis

Parts of this chapter have been submitted for publication.

ABSTRACT

This chapter demonstrates various data augmentation techniques that can be used when working with limited run-to-failure data to estimate health indicators related to the remaining useful life of roller bearings. The PRONOSTIA bearing prognosis dataset is used for benchmarking data augmentation techniques. The input to the networks are multi-dimensional frequency representations obtained by combining the spectra taken from two accelerometers. Data augmentation techniques are adapted from other ML fields and include adding Gaussian noise, region masking, masking noise, and pitch shifting. Augmented datasets are used in training a conventional CNN architecture comprising two convolutional and pooling layer sequences with batch normalization. Results from individually separating each bearing's data for the purpose of validation shows that all methods, except pitch shifting, give improved validation accuracy on average. Masking noise and region masking both show the added benefit of dataset regularization by giving results that are more consistent after repeatedly training each configuration with new randomly generated augmented datasets. It is shown that gradually deteriorating bearings and bearings with abrupt failure are not treated significantly differently by the augmentation techniques.

5.1 Introduction

Due to the expensive and time consuming nature of collecting faulty vibrational data in the lab, much work has been done with the small number of publicly available datasets, such as the Case Western Reserve Dataset for fault diagnosis and the PRONOSTIA dataset (also called the FEMTO dataset) for prognosis [6], [47]. To train accurate deep learning models from large inputs like raw frequency spectrums, it is necessary to have a very large number of training samples, and the small size of these commonly used datasets, small relative to the datasets used in training deep image nets, hinders this process. A common and economical way to increase the number of unique training samples is through data augmentation, which can be achieved using a variety of techniques. Data augmentation grows a dataset by either adding slightly modified copies of existing data or by adding synthetic data created from existing data. This can improve performance overall and reduce the overfitting that often occurs when training deep networks on a limited number of training samples.

While data augmentation for image and sound data has been studied extensively [32], few researchers have investigated the influence of data augmentation on vibration-based fault diagnosis and no previous studies have been found pertaining to fault prognosis.

5.2 Methodology

5.2.1 Dataset Description

The PRONOSTIA dataset is used to evaluate the proposed data augmentation techniques. The dataset can be obtained from [55] and is described in detail in [56]. In it, bearings are run to failure under different accelerated life conditions, as described in Table 25. Vibration measurements having a duration of 0.1 seconds using two accelerometers are repeated every

10 seconds throughout the lifetime of the experiments. A sample rate of 25,600 Hz is used, yielding 2560 data points for each measurement.

Table 25: PRONOSTIA dataset details

Conditions	Load (N)	Speed (RPM)	Bearings
1	4000	1800	Bearing1-1, Bearing1-2, Bearing1-3, Bearing1-4, Bearing1-5, Bearing1-6, Bearing1-7
2	4200	1650	Bearing2-1, Bearing2-2, Bearing2-3, Bearing2-4, Bearing2-5, Bearing2-6, Bearing2-7
3	5000	1500	Bearing3-1, Bearing3-2, Bearing3-3

The bearings begin in healthy states and gradually develop faults. The degradation can be seen in their increase in vibrational amplitude shown in Figure 1. The examples shown in this figure highlight some qualities of the dataset: that the two accelerometer channels capture similar, but different, vibrational signatures, that the bearings have significantly different lifespans, and that the vibrational amplitude sometimes suggests gradual health degradation, and sometimes shows sudden rapid degradation. The accelerated lifetime tests are terminated when the acceleration exceeds a safety threshold.

Information about modes of failure, typically investigated in bearing fault diagnosis (i.e. inner race, outer race, ball faults), are not provided; prognosis in this dataset is intended to be performed blindly with respect to the failure mode. However, it is possible to distinguish between bearings with early signs of failure and bearings that appear to fail abruptly with little warning. Whether these differentiable subsets of bearings are substantially different in terms of how they are impacted by data augmentation will be investigated.

5.3 Data preprocessing and labeling

Input features are obtained from the original raw vibration measurements containing 2560 data points using the FFT algorithm to give the frequency spectrum of each channel. The two acceleration channels are concatenated to form a single input feature of size [1280 × 2 × 1]. By convention, this follows the $[h \times w \times c \times N]$ format for image CNN inputs where h is image height, w is image width, c is the number of channels, and N is the number of samples in the dataset. The frequency spectrum was chosen for the input representation since it condenses the input signal into a smaller number of features compared to the spectrogram or raw signal. As has been discussed, there are several advantages to using a smaller input feature and these are especially true for smaller datasets.

Rather than using the entire bearing lifetime to train and test algorithms, this chapter uses the portion near the end of a bearing's lifetime where changes in bearing health appear to occur. For each bearing, only samples following the first prediction time (FPT) are kept for training and testing using the method for determining the FPT described by Naipeng Li et al. [30] and first used for the FEMTO dataset by Xiang Li et al. [29]. This method is thought to be useful since it provides emphasis on correctly identifying end-of-life behaviour and ignors the larger portion of uneventful run-in data.

Labels for training and validation purposes are applied to all of the measurements after the FPT. Instead of directly labeling samples with the RUL, the labels are normalized by using a health indicator (HI), which corresponds to the RUL, between one (healthy) and zero (failed). Labels follow a linear descent. Samples taken before the FPT are not used in training or in calculating validation root mean square error (RMSE), though the performance of the trained network over the entire duration of the experiment is important and evaluated qualitatively. It is expected that test data from prior to the FPT yields a predicted HI close

to one, since it is believed that no significant change in bearing health is occurring. Having the algorithm train with data that is more likely to contain useful fault information is expected to result in an easier task.

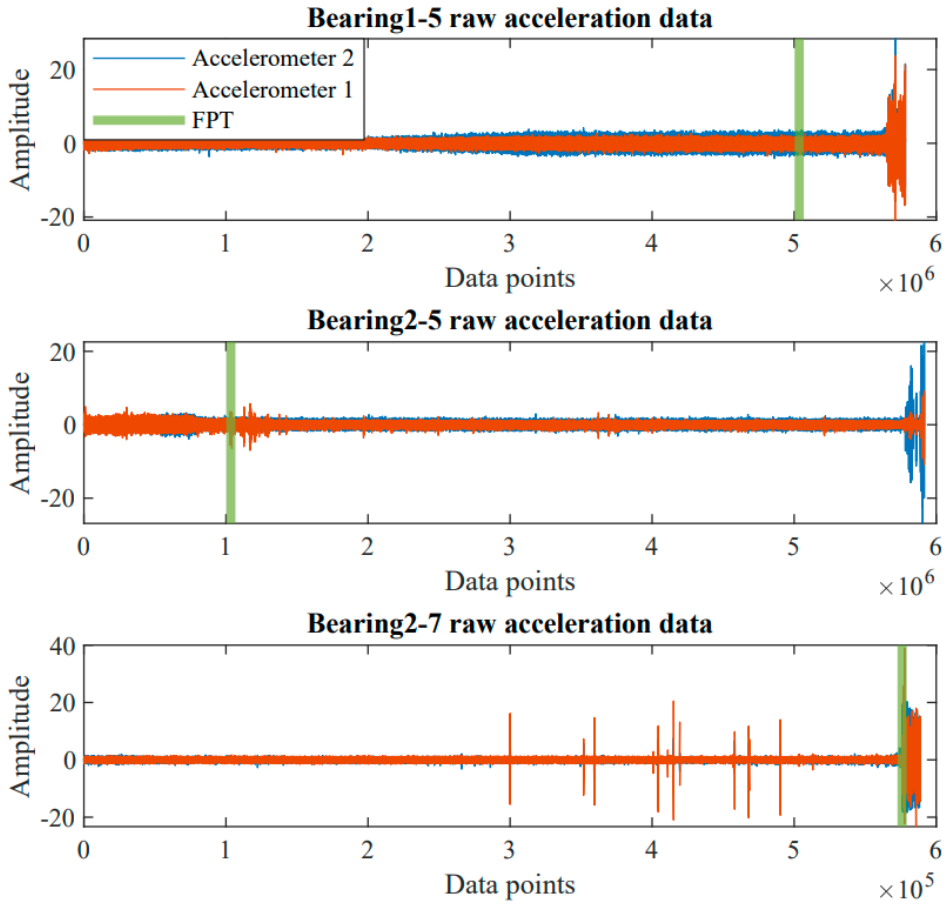


Figure 14: Example multichannel vibration data from bearing lifetimes with FPT indicated

Bearing1-5 of Figure 14 shows the most typical behavior of bearing degradation in the dataset. Specifically, there are some signs of growing wear midway through the bearing's lifetime and significant increase in acceleration amplitude and kurtosis just prior to its end of life. Bearing2-5 and Bearing2-7 show significantly different behavior compared to the majority of the experiments. Their FPTs occur very early or very late in the lifetime. Bearing2-5 shows surprising levels of degradation from very close to the beginning of the test with a longer period of reduced amplitude and kurtosis following. This causes its FPT to occur

earlier. The opposite is true for Bearing2-7, which shows the FPT triggering just prior to the termination of that experiment. The result is the majority of the measurements from Bearing2-7, those where the kurtosis had not yet grown significantly, are discarded and not used for training. Table 26 shows how many measurements were recorded for each bearing's entire lifetime and how many are kept following the FPT. Note that there is a wide range in how early or late the FPT is triggered. The rightmost column represents the earliness of the FPT with the number of observations after the FPT divided by the number of lifetime observations.

Table 26: Number of original measurements (or observations) for each bearing during the entire lifetime and after FPT, ordered by FPT fraction

Bearing	Lifetime Observations	Observations After FPT	FPT Fraction
Bearing1-5	2463	65	0.026
Bearing3-3	434	13	0.030
Bearing3-1	515	21	0.041
Bearing2-7	230	11	0.048
Bearing2-6	701	39	0.056
Bearing2-2	797	49	0.061
Bearing3-2	1637	208	0.127
Bearing1-7	2259	298	0.132
Bearing1-3	2375	351	0.148
Bearing1-2	871	163	0.187
Bearing1-4	1428	349	0.244
Bearing1-1	2803	691	0.247
Bearing1-6	2448	821	0.335
Bearing2-4	751	418	0.557
Bearing2-5	2311	1913	0.828
Bearing2-3	1955	1694	0.866
Bearing2-1	911	882	0.968
Total:	24889	7986	

A histogram of the lifetime/FPT measurements reveal distinct groups of bearings which displayed signs of wear relatively early in life, and those which did not trigger the FPT until

very late in life (Figure 15). The different behavior of bearings in these different groups is discussed in the results.

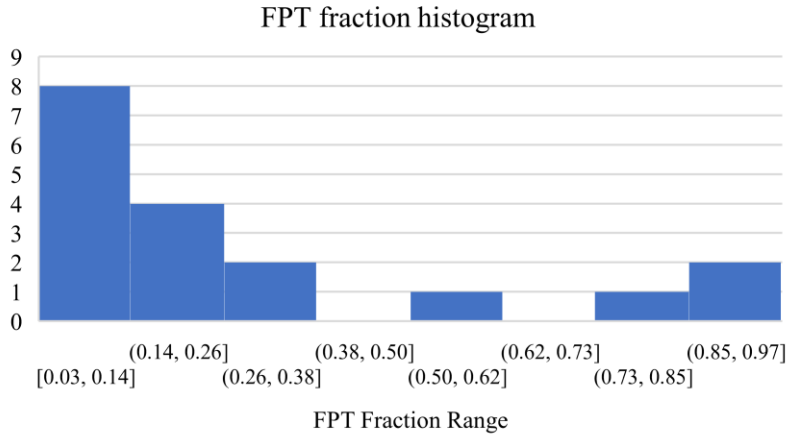


Figure 15: Histogram of FPT fraction of all bearings

5.4 Proposed data augmentation techniques

All of the proposed data augmentation techniques have parameters adjusted so that the augmented dataset is grown by a factor of four. For each sample in the original dataset, four unique copies are generated by the various augmentation methods. Time stretching, as used by Li et al. [34], is not applicable to Fourier spectra as input features, so it has not been included. Likewise, Li et al.'s use of time shifting is not applicable. Their shifting of samples forward or backward created empty regions and regions where data was removed from the beginning or end of the samples. Since CNNs inherently possess properties of translational invariance, the shifting of samples is not likely to cause improved performance. Conversely, erased regions of data are more likely to influence performance. Therefore, a method using region masking is used instead of time shifting.

- **Gaussian noise (GN):** The sample is combined with random noise from the standard normal distribution, scaled by a factor of 0.3. Each sample is combined with four randomly generated noise signals to give four unique new signals.

- **Masking noise (MN):** 20% of each channel's data points are set to zero by random selection. This works similarly to dropout layers sometimes used in the architecture of deep neural networks. Random selection of datapoints is performed four times for each original sample to generate the four new unique samples.
- **Region masking (RM):** Region masking randomly selects adjacent data-points to set to zero. Specifically, two regions, each of 10% of the input length, are randomly selected from each channel. Random selection of masking regions is repeated four times for each original sample.
- **Pitch shifting (PS):** Prior to transforming the raw signal to the frequency spectrum using the FFT, the signals have their pitches shifted using the *shiftPitch* function in MATLAB. Each signal from the original dataset creates four new samples by shifting its pitch by different amounts: [-0.5, 0, 0.5, 1], measured in semitones. The pitch is shifted without altering the signal duration. The new samples are then converted to the frequency domain, giving the same [1280 × 2 × 1] input as all the other augmentation types.

5.5 CNN architecture and training parameters

A simple 2-layer CNN is used for all experiments. The CNN's architecture and hyperparameters are outlined in Table 27. CNNs are trained to ten epochs in every experiment. Training is performed with the ADAM optimizer for gradient descent. An initial learning rate of 0.001 is used. The learning rate is decreased using a multiplication factor of 0.2 after every 2 epochs. Mini batches of 32 samples are used with batch normalization. Similar to using a smaller input size, there are advantages to using networks with fewer layers. Compared to the deep networks shown in the previous chapter, shallower networks are far less prone to become overfit.

Table 27: Details of CNN architecture used for all data augmentation types

Layer	Type	Activations	Learnables
1	Input	1280×2×1	-
2	Convolution Filters: 8 Size: 6×2×1 Stride: [1,1]	1280×2×1	Weights: 6×2×1×8 Bias: 1×1×8
3	Max pool Size: 3×1 Stride: [3,1]	247×2×8	-
4	reLU	247×2×8	-
5	Batch normalization	247×2×8	Offset: 1×1×8 Scale: 1×1×8
6	Convolution Filters: 8 Size: 6×2×8 Stride: [1,1]	247×2×8	Weights: 6×2×8×8 Bias: 1×1×8
7	Max pool Size: 3×1 Stride: [3,1]	143×2×8	-
8	reLU	143×2×8	-
9	Batch normalization	143×2×8	Offset: 1×1×8 Scale: 1×1×8
10	Fully connected	1×1×1	Weights: 1×2288 Bias: 1×1
11	Regression output	-	-

For training and validation datasets, one bearing is set aside for validation and the remaining 16 bearings are used for training. The validation set is rotated through the bearings so that each one is given the opportunity to be isolated for validation. Non-augmented data is used for validation, whereas new augmented training data is generated for each training cycle. The process is repeated four times so that the average validation score for each bearing can be taken in addition to the standard deviation found between repetitions under the same conditions.

5.6 Results and discussion

Overall results are presented in Figure 16, where the average validation error achieved after four repeated trials for each bearing and each augmentation method is given. GN, RM, and MN offer similar comparative advantages over the non-augmented case, while PS does not result in a reduced error.

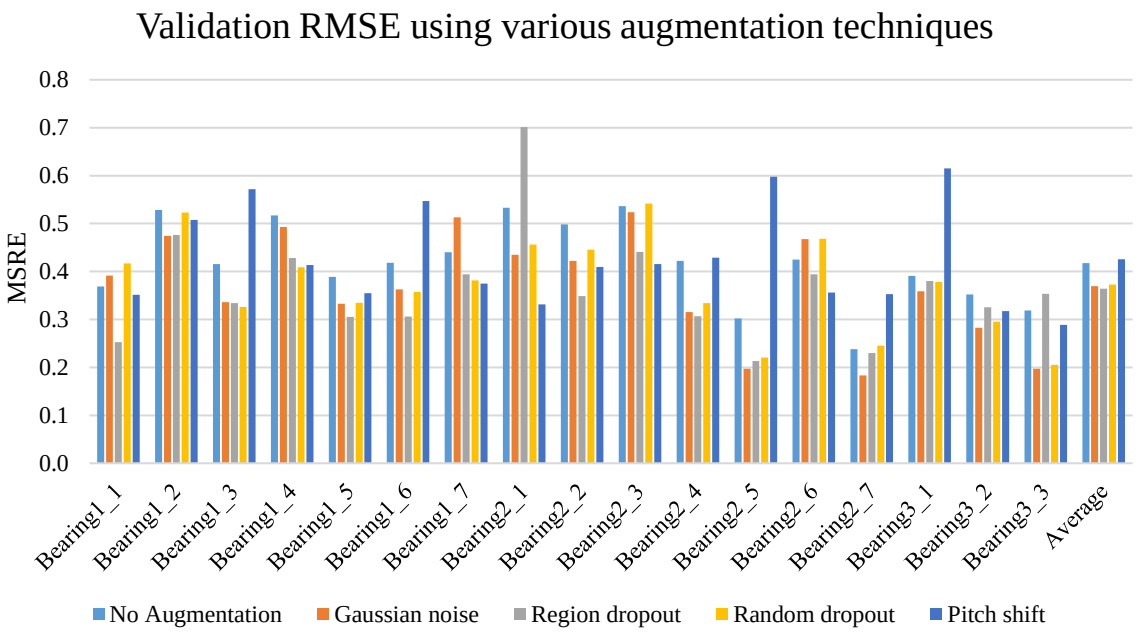


Figure 16: Validation RMSE for each bearing for all proposed augmentation methods

An example result for a bearing in the training set is shown in Figure 17. This figure shows measurements from the entire lifetime, though training was only performed using measurements after the FPT, which corresponds with the region where the HI begins to decline.

The different methods have varying degrees of success in modeling the HI prior to FPT. The case where no data augmentation is used is unacceptable since its HI intercepts zero near the beginning of the test. Because estimations vary considerably between consecutive measurements, the curves in this and the following HI estimation plots have been smoothed

using MATLAB's 'lowess', or linear regression functions with a trailing window of 20 data points.

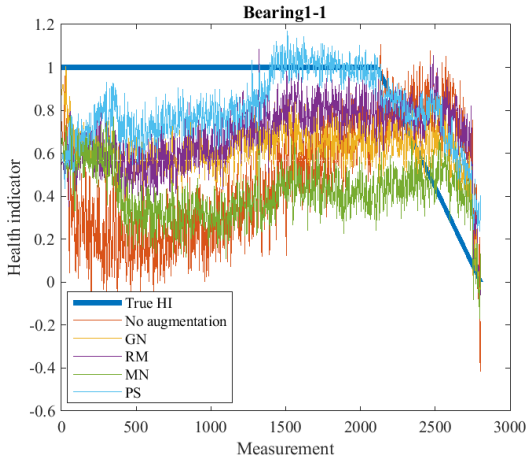


Figure 17: Smoothed HI predictions and true HI for Bearing1-1

More important for actual applications than the RMSE is whether the HI estimations correctly indicate ultimate component failure by crossing the zero-threshold close to the end of life and, importantly, do not cross it prematurely. In the case of Bearing1-7 in Figure 18, no augmentation provides sporadic HI estimations that dip near zero in early life, while estimations using PS augmentation do not reach HI = 0 at the end of the test.

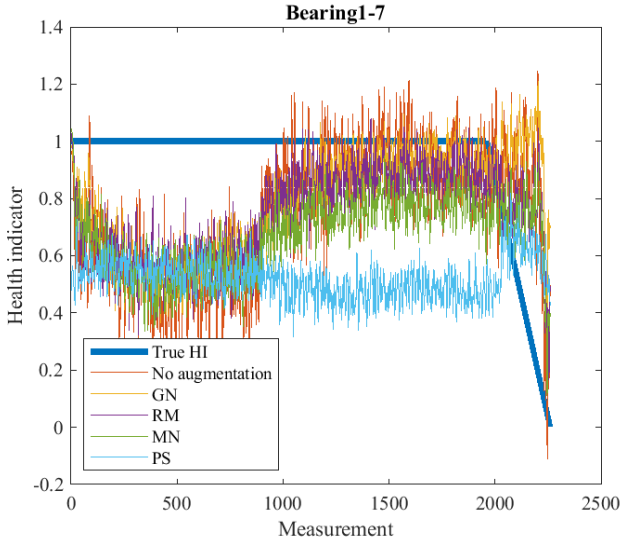


Figure 18: Smoothed HI predictions and true HI for Bearing1-7

To extract a reliable estimation of remaining useful life from the HI, the HI estimations would need to show a consistent decline trend that is apparent well in advance of failure and does not deviate greatly before reaching HI = 0. For most bearings, the obtained estimations during the early life, or run-in phase, could not be used to predict RUL. Usually, the decline in HI is sudden and only begins directly prior to the end of life. This makes the resulting algorithms more useful for failure detection rather than remaining useful life predictions, which still presents some practical value. Bearing3-3 in Figure 19 is an exemplary result that shows most methods closely following the "true HI" from one to zero.

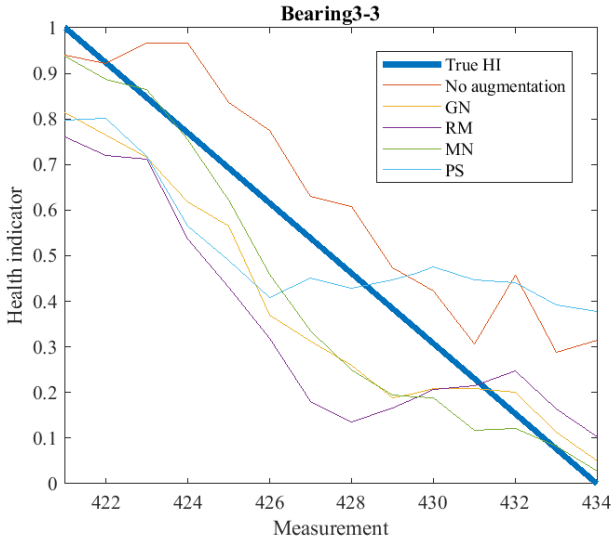


Figure 19: Smoothed HI predictions and true HI for Bearing3-3 after FPT

The raw vibrational measurements for Bearing1-5, Bearing 2-5, and Bearing2-7 were shown in Figure 14 and their respective HI estimations are provided in Figure 20 through Figure 22. Curves are generated using data smoothing already described.

Raw data for Bearing1-5 is suggestive of a fault that grows steadily throughout the bearing's lifetime, though the FPT occurrs somewhat later in life. The resulting HI predictions (Figure 20) show little coherence and do not indicate failure as they should. It is

possible that Bearing1-5 undergoes a mode of failure that cannot be modeled easily based on the spectra of all the other bearings that exist within the validation dataset.

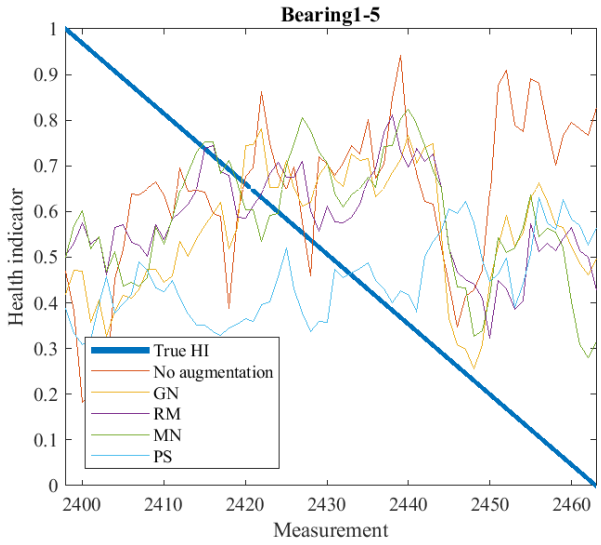


Figure 20: Smoothed HI predictions and true HI for Bearing1-5 for the entire lifetime

The raw measurements from Bearing2-5 and very early FPT indicate that the bearing shows signs of having a fault almost immediately after having started the test and continued to operate with the minor fault for some time before the fault grew enough to trigger the end of the test. The resulting estimations (Figure 21) are somewhat scattered and, for most methods, sharply rise instead of decline just prior to the end of the test. Interestingly, in this instance, it is PS that appears to give the most useful HI since it is the only one that drops to zero.

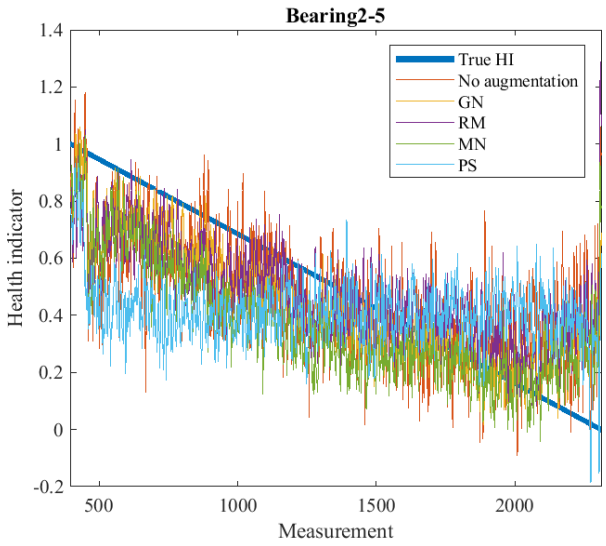


Figure 21: Smoothed HI predictions and true HI for Bearing2-5 for the entire bearing lifetime

The raw measurements for Bearing 2-7 show few signs of bearing damage throughout the bearing's lifetime until its abrupt failure, corresponding with a very late FPT. After FPT, all methods except for PS properly intercept HI = 0 at the end of life.

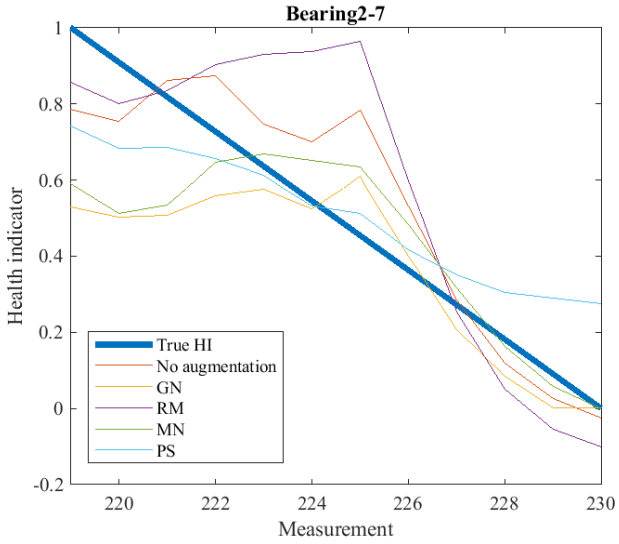


Figure 22: Smoothed HI predictions and true HI for Bearing2-7 for the entire bearing lifetime

While the inability of all methods to predict failure of Bearing1-5 and generally high RMSE undercuts confidence in the resulting algorithm, it is still interesting to note the influence of the various data augmentation methods. It appears that any of the methods, besides PS, can be used to lend some improvement to the RMSE. While data augmentation with PS has been found to be very useful in sound recognition applications, it does not appear to be useful in the domain of bearing fault detection and prognosis based on Fourier spectra input features.

Since the bearings can be separated into distinct groups based on FPT fraction, it is interesting to view average accuracies of groups of bearings based on how early their FPT was triggered. Table 28 summarizes the bearings in each group and Figure 23 shows for these groups and each method the change in RMSE versus no data augmentation.

Table 28: Bearing groups based on FPT fraction

Early FPT Group	Middle FPT Group	Late FPT Group
Bearing1-5	Bearing1-1	Bearing2-1
Bearing2-2	Bearing1-2	Bearing2-3
Bearing2-6	Bearing1-3	Bearing2-4
Bearing2-7	Bearing1-4	Bearing2-5
Bearing3-1	Bearing1-6	

Each method appears to bring about a different change in each FPT segment. While augmentation for the late FPT group shows the lowest error overall, it is also the least improved by augmentation, mainly because this group was better estimated even without augmentation. Pitch shifting causes an increase in RMSE compared to no data augmentation for the middle and late FPT groups, while it is slightly beneficial for early FPT bearings. The largest improvement achieved by any group was the early FPT bearings with GN augmentation.

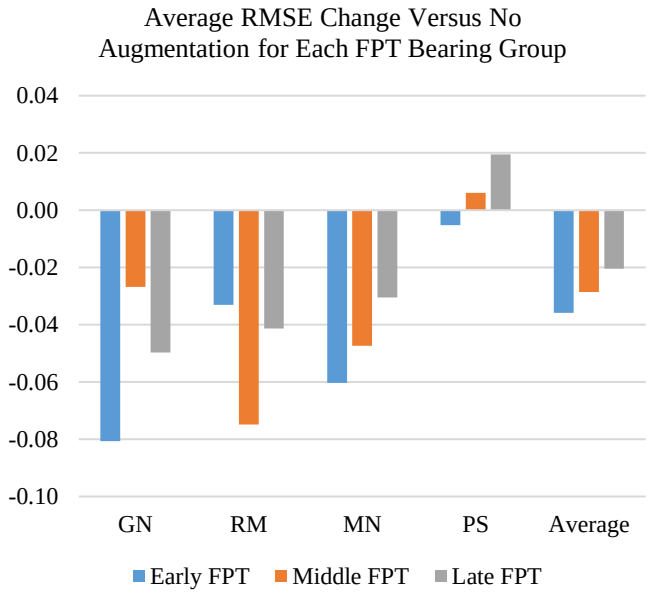


Figure 23: Average RMSE change versus no augmentation for all methods with bearings separated into early, middle, and late FPT groups

In Figure 24,the RMSE of all bearings plotted against their FPT fraction, for each augmentation method, indicates that error increases slightly for bearings with a greater FPT fraction, that is, for bearings with signs of wear earlier in life. This gives the counterintuitive impression that bearings that fail suddenly are easier to predict, but this is not the reality. Bearings with an FPT relatively close to their end of life show a lower RMSE in large part because that RMSE is calculated only with data following the FPT. This is one potential flaw with this procedure of using FPT to select shorter segments of the lifetime measurements with which to train and calculate error.

The statistical significance of the trend shown in Figure 24 can be assessed by the R-squared, a measure of how well the data points can be modeled by a linear fit. Figure 25 shows that the average R-squared for all methods are relatively low. This indicates that, despite appearances, there is not a significant correlation between FPT fraction and RMSE.

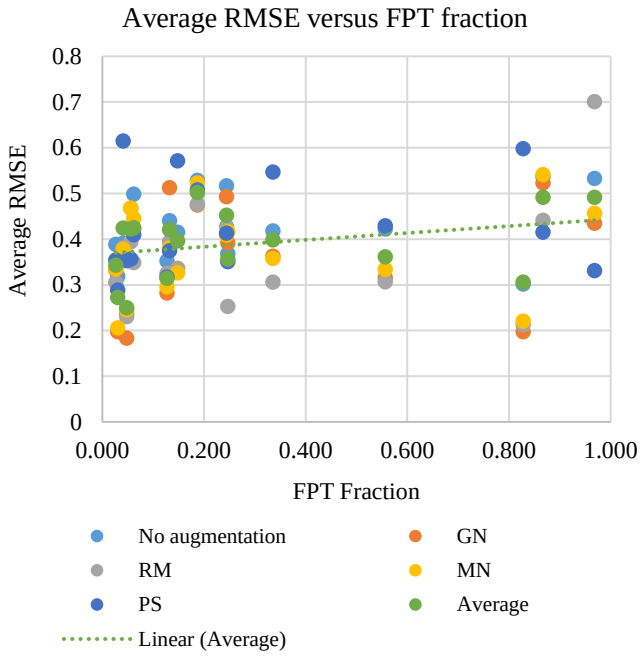


Figure 24: Scatter plot of average RMSE against FPT fraction for all bearings using each augmentation method

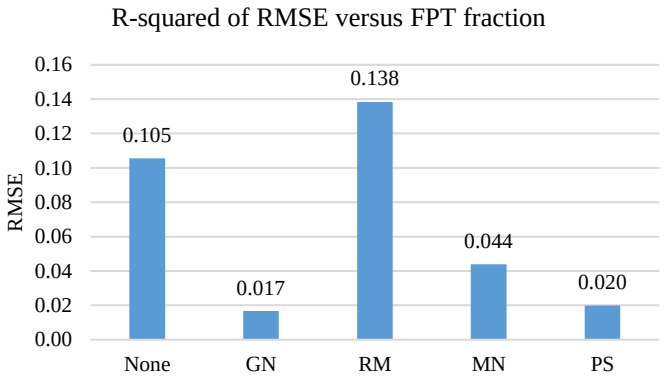


Figure 25: R-squared for linear fit of RMSE versus FPT for each method

Unsurprisingly, there is some variance between repeatedly training new networks with newly generated augmented datasets. This is a result of variation in the randomized parameters of each augmentation technique, as well as the randomized network initialization and stochastic gradient descent. The standard deviation for each bearing and augmentation method is calculated to reveal which augmentation methods give the most consistent results for a particular bearing. The average standard deviation for each method across all bearings

is provided in Figure 26. Region masking and masking noise are shown to be the most consistent, while pitch shifting is worse than using no augmentation at all. A reduction in standard deviation is beneficial since it improves confidence in the accuracy of the augmentation method across different fault types and operating conditions, important for highly variable industry applications.

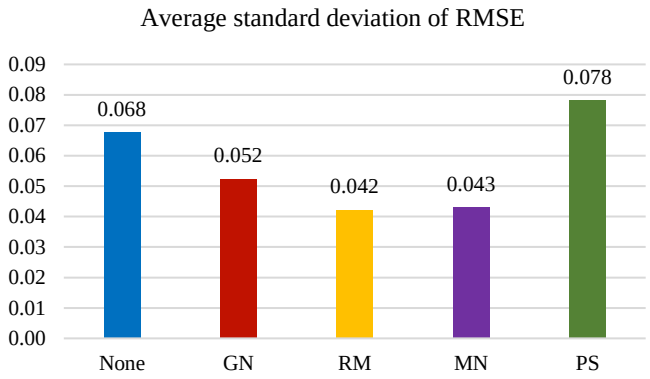


Figure 26: Average standard deviation for the validation accuracy of all bearings achieved using each method after repeatedly training each configuration four times

From the outset, the methodology developed for this paper assumed that the use of FPT, based on the work in [29], [30], would provide some advantage over using training data from the entire lifetime of measurements using a fully linear labeling scheme. Considering the results, in which early life HI estimations are not very useful, and estimations after FPT are only sometimes able to provide useful RUL indications, it seems fitting to question that assumption. The FPT is intended to be an objective method of excluding the many run-in measurements from training data, whose inclusion might cause a skewed dataset. This results in improved modeling of degradation at the end of life, but also leads to poor beginning of life estimations.

Perhaps a more significant improvement can be achieved by using a different input representation other than raw time or frequency data. Many authors have noted that time-frequency representations such as spectrograms can give improved diagnostic results [6]. It

is suggested that future work should be done to establish better input representations in conjunction with labeling schemes and data augmentation methods for prognosis.

5.7 Chapter Conclusions

This chapter found that all the studied data augmentation methods except for pitch shifting offer improvement to performance, while reducing variability in training results. However, even with the slight improvement gained through augmentation, the resulting HI curves do not follow very closely a predictable descent towards zero. Therefore, they would not be very useful for making informed maintenance decisions. Data augmentation, in general, has been found to offer more significant performance improvements for other applications [32]–[34]. It follows that specially designed augmentation methods tailored for the task and input representations used in HI estimations might be discovered to offer a much greater improvement than the methods studied here. That remains subject for future work, as the following chapter explores how a different input representation and CNN architecture influences performance. An additional significant difference in the following experiments is the omission of the FPT. Xiang Li et al. found FPT to be beneficial under their proposed CNN architecture. However, too few studies have considered the FPT method to confirm that it is universally beneficial, and the FPT was not obviously helpful in this chapter.

Chapter 6

Prognosis Using a 3D CNN and Multi-Channel Spectrogram

Parts of this chapter have been accepted for publication in the *27th International Congress on Sound and Vibration* [57].

ABSTRACT

In this chapter, a novel algorithm is proposed using fused data from multiple sensors to present a CNN with rich time-frequency and spatial information to enhance prognosis. Spectrograms are computed from signals taken at multiple locations on a machine and layered to create a 3D input matrix. This allows the network to learn patterns in the time-frequency plane, as well as patterns relating the two signals through cuboidal convolution. The network's performance is demonstrated with data gathered from accelerated lifetime tests. The inherent noise-canceling capabilities of the algorithm are evaluated with gaussian noise added at various levels to the validation data. The resulting CNN is found to be much more consistent in estimating the HI than the algorithm proposed in the previous chapter.

6.1 Introduction

Bearing faults cause unique vibrational emissions, but fault-induced signals measured at the early stage of wear are often obfuscated by background noise. CNNs have been investigated to diagnose artificially damaged bearings in low-noise settings with very high accuracy [58], [58]–[62]. To better satisfy industry demands, there is a need for an algorithm which can estimate remaining useful life (RUL) from the early stages of wear even when operating in noisy environments.

This work leverages data fusion by using two vibration signals obtained simultaneously from bearings during accelerated life testing. Layering the two 2D matrices to form a 3D input places contemporaneous time-frequency elements observed from different locations adjacent to each other along the depth direction. While 2D convolution of spectrograms is sensitive to patterns in the time-frequency plane, cuboidal convolution allows for sensitivity to patterns in time-frequency and space. This method of data fusion is herein demonstrated to have strong inherent noise resilience when used to estimate a non-dimensional health indicator (HI), which is directly related to a bearing's RUL.

6.2 Data preparation

Refer to section 5.2.1 for a detailed description of the unprocessed PRONOSTIA dataset.

6.2.1 Spectrogram pre-processing

The spectrogram is a useful visualization of the short-time Fourier transform (STFT), which presents the sinusoidal frequency content of a signal as a function of time [63]. By transforming the 1D vibration data into a 2D image, it allows a human viewer to quickly recognize transient time-frequency patterns, such as regularly repeated impulses at a specific

frequency. Similarly, rendering vibration signals this way lends itself to image pattern recognition algorithms such as CNNs.

The time and frequency resolution of the spectrogram are constrained by certain parameters used in calculating the short-time Fourier transform. Chief among these is the window length, here measured in number of samples, which can be selected to give either sharp frequency resolution or good time localization. In this instance, a compromise window length is selected to give a well-rounded image. The effect of various window lengths is shown in Figure 27 with, from left to right, good time resolution, average time and frequency resolution, and good frequency resolution.

Another important parameter for the short-time Fourier transform is window overlap, which allows one to augment the resolution without changing the window length. Figure 28 shows the influence of different degrees of overlap while keeping a constant window length of 50 samples.

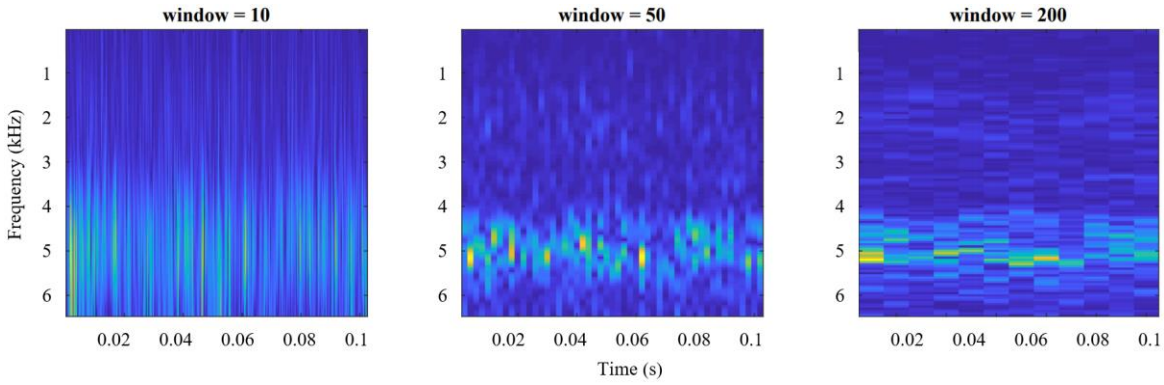


Figure 27: Effect of window length on spectrogram time-frequency resolution

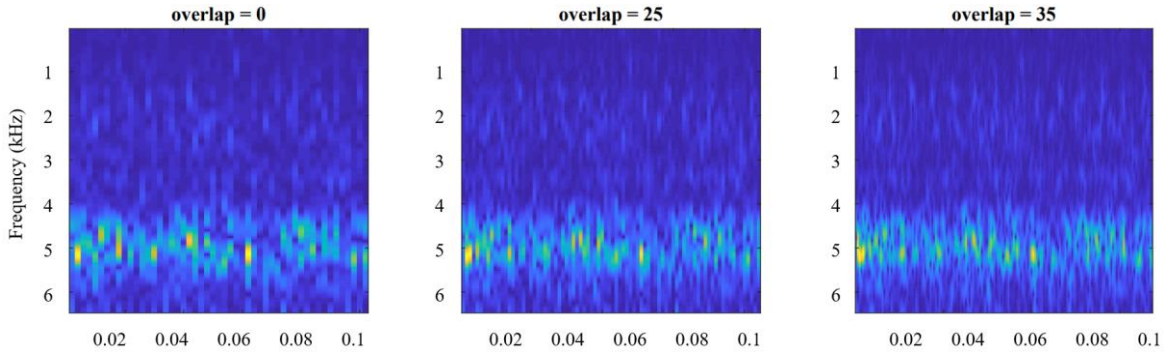


Figure 28: Effect of overlap on time resolution with constant window length = 50

Based on visual inspection of the resulting spectrograms, a combination of a 50-sample window with a 35 sample overlap is selected for all further experiments. For each instance of a 0.1 s recording, one spectrogram is computed from each channel. The two are then combined into a 3D matrix and used to train and validate the CNN.

6.2.2 Artificial noise

Compared to real industrial applications, laboratory-obtained vibration datasets contain very low levels of noise such that fault-induced indicators are clearly distinguishable. In real applications, other machinery and sources generate vibration that is transmitted through the structure, floor, and the machine itself and contaminate the signal measured at the bearing being monitored. These contaminating signals obscure fault-induced vibration and confuse the algorithm, reducing diagnostic and prognostic accuracy. Therefore, it is important for a successful algorithm to provide accurate prognosis even when signals are contaminated. For the purpose of repeatability, the noise source in this work is simply modeled as white Gaussian noise. The signal-to-noise ratios (SNR_{dB}) reported are always expressed in decibels and are calculated using the average signal power for the 34 run-to-failure recordings (17 experiments × 2 channels). Figure 29 shows the influence of noise in the spectrogram, making it easy to see how a noisy signal is more difficult to read and, in general, appears more damaged than a clean signal.

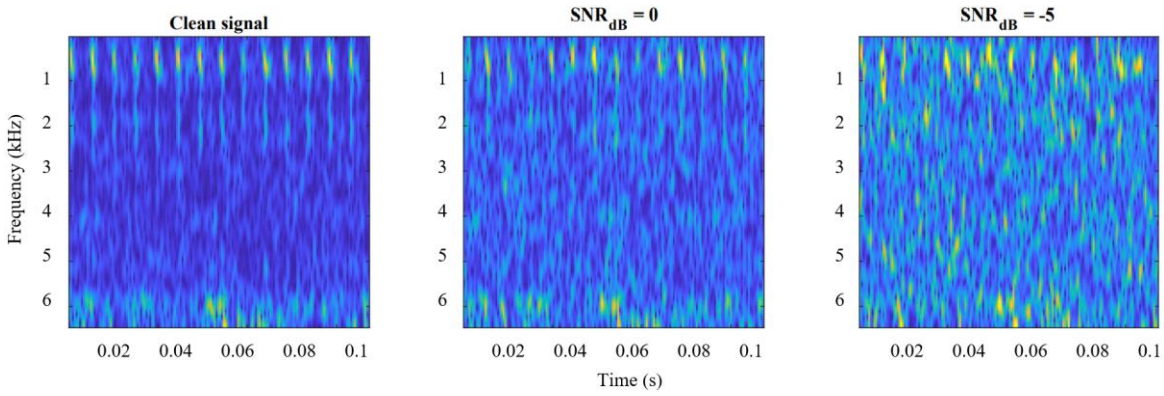


Figure 29: Spectrograms with various levels of white noise added to the same signal

6.2.3 Training labels

To normalize training labels, labels are applied as scalar values between one and zero, corresponding to healthy and critically failed states respectively. Labels are blindly applied according to a simple linear descent. This presents a challenging scheme for the CNN to learn since bearings which show obvious signs of gradual wear are labelled in the same way as bearings which rapidly deteriorate at the end of the experiment. During validation, one can translate the generic HI into the predicted RUL using Eq. ().

$$RUL = \frac{elapsed\ time}{1-HI} - elapsed\ time. \qquad \text{Eq. (12)}$$

6.3 Convolutional neural network

The CNN architecture used in this study is adapted from Zhang et al.'s spectrogram-based CNN for bearing diagnosis [60]. Fundamentally, the CNN presented herein differs in that it outputs a scalar HI rather than a diagnostic category. Otherwise, the novel difference between this and other CNNs applied in machine health monitoring is the use of cuboidal convolution layers. The sequence and parameters of the convolutional and pooling layers are

given in Figure 30. For simplicity, the layer diagram does not show the batch normalization and ReLU layers, which together follow each convolution layer.

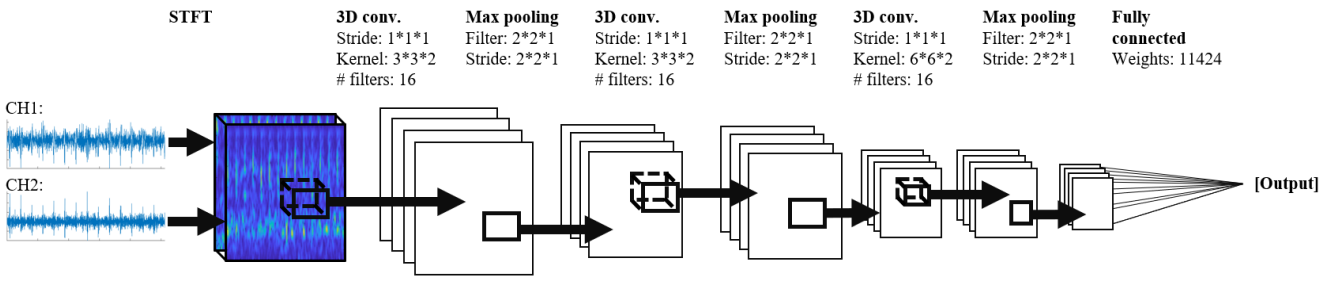


Figure 30: 3D-CNN layer diagram

To assess the advantage of cuboidal convolution in this scheme, some results are presented against 2D CNNs. In these instances, a similar architecture is used in creating both networks with the main difference for the 2D network being that the input is a spectrogram from one of the two channels rather than both channels, as in the 3D network of Figure 30. The height and width of convolution kernels and pooling filters are identical between both network architectures. For the 2D CNN, separate networks are trained entirely from one channel or the other, and their results are presented separately.

6.4 Results and analysis

6.4.1 Estimation of health indicator

In keeping with the original rules of the PRONOSTIA Data Challenge, the training set is comprised of the first two experiments from each operating condition. The results displayed come from the remaining 11 experiments in the validation set. The results shown in Figure 31 typify performance for gradual-onset failure (a) and sudden failure (b). In some instances, there is considerable variance between instantaneous predictions; a smoothed result is overlain based on a moving-average. Even so, the 3D CNN, using fused spectrograms as

inputs, has far less prediction variance and more accurately reflects the bearing's health state. In the two instances shown, the 2D CNN results sometimes erroneously show an increasing HI when approaching failure, as a result of the algorithm becoming confused by chaotic data.

The results generally show some decline in the HI. However, in some experiments, it can also be seen to rise somewhat over longer periods. Importantly, catastrophic failure is always detected by the 3D CNN, whereas failure is only sometimes detected by the 2D CNNs. In some cases, the 2D CNN erroneously shows a sharp HI increase upon failure. Therefore, the algorithm is somewhat useful as an advanced-warning tool and quite strong as a failure-detection thresholding-based tool.

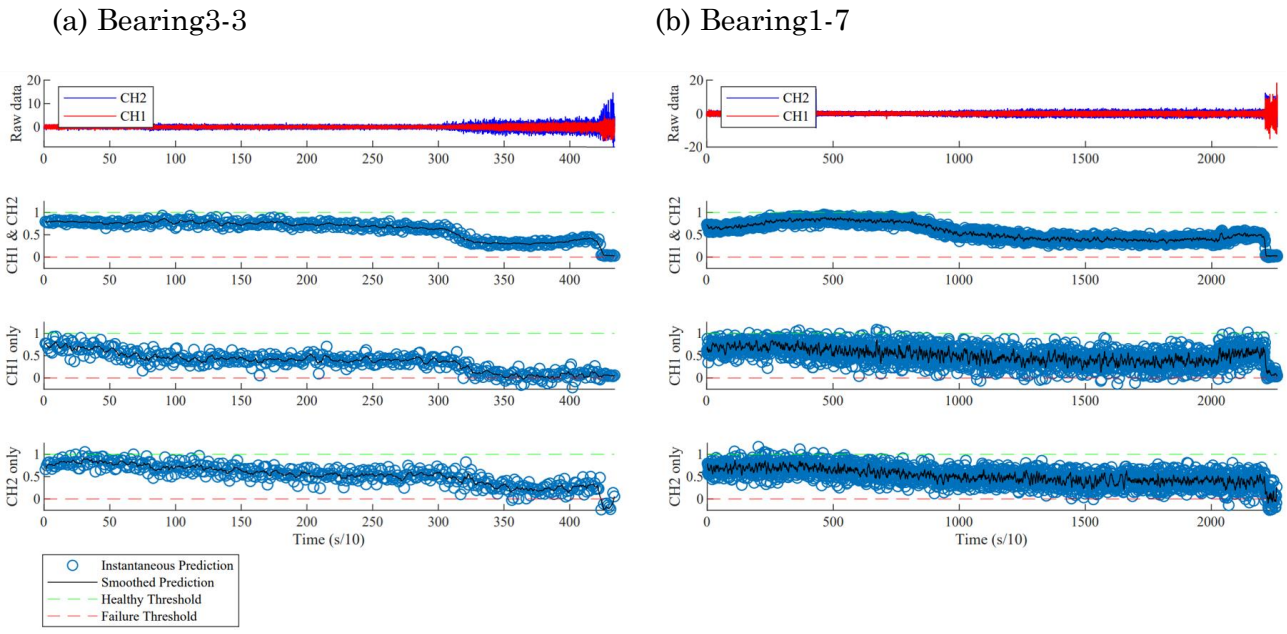


Figure 31: Top to bottom: raw vibration signals, HI from fused spectrogram and 3D CNN, HI from CH1 and 2D CNN, HI from CH2 and 2D CNN

Overfitting occurs when an algorithm fits training observations too well and as a result is unable to accurately predict future observations. Generally, using a smaller training set or a deeper algorithm will increase the likelihood of overfitting. The extension from a 2D to a

3D convolution and increased input dimensionality result in an inherently deeper network, even when the number of layers match. Therefore, it is imperative to check for overfitting. The accuracy of the CNNs studied are evaluated with the root-mean-square error (RMSE). Figure 32 shows that the 3D CNN has the lowest RMSE for both the validation and test sets. However, there may be some overfitting causing the validation RMSE to be higher. The 2D CNNs do not appear to be overfit, as their validation RMSEs are equivalent or lower than that of the training sets.

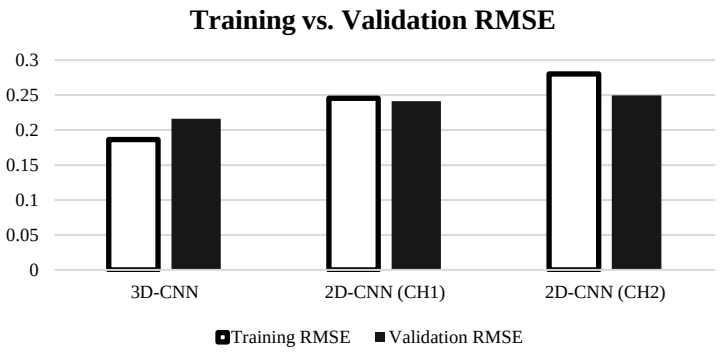


Figure 32: Validation and training RMSE for the three CNN configurations

While the RMSE shows that the 3D CNN has a modest improvement in accuracy, Figure 33 shows that precision is significantly improved. Variance is calculated from the instantaneous HI predictions made during the bearings' lifetimes and averaged for each training or validation group.

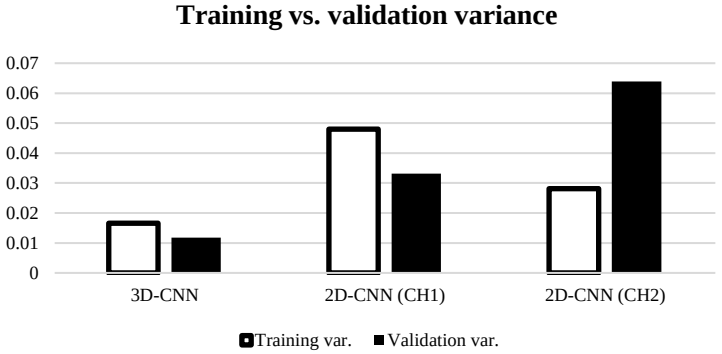


Figure 33: Validation and training average variance for the three CNN configurations

6.4.2 Noise tolerance

Figure 34 demonstrates the different manners in which the 3D CNN responds to noisy spectrograms when compared to the 2D CNNs using Bearing3-3 and Bearing 1-7 as a typical response. Noise is added at four different SNRs and prediction curves are generated from the entire experiment duration. A moving-average smoothing function is applied to the instantaneous prediction points to make the plots interpretable.

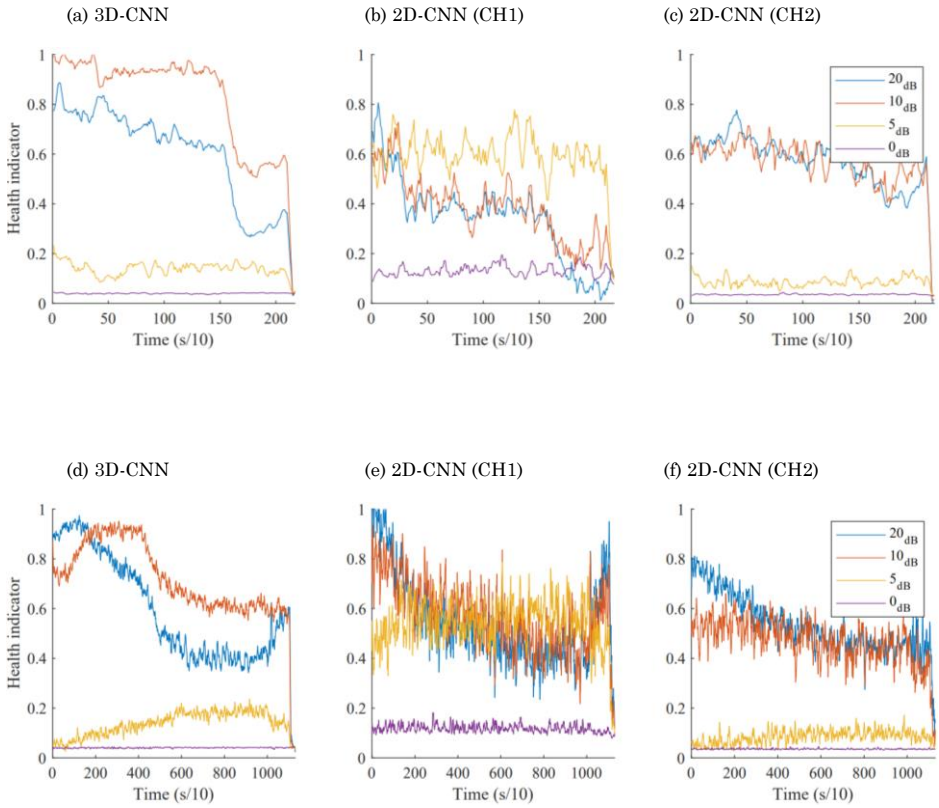


Figure 34: Smoothed HI predictions for Bearing3-3 (a) – (c) and Bearing1-7 (d) – (e) using 3D and 2D CNNs

Stronger noise, in general, gives the impression of a reduced HI from the onset of the experiment. Still, the 3D CNN predictions generally decline to intercept the failure threshold at the end of the experiment, whereas 2D CNN predictions have much higher variance during mid-life. The two CNN formats seem comparable in identifying critically failed states when

the noise level is not yet high enough to completely obscure the stronger vibrations generated shortly before failure. Noise performance should be improved by using a denoising pre-process and using dropout during training.

6.5 Chapter Conclusions

A new technique for continual HI estimation is demonstrated combining time-frequency information from multiple accelerometers into a 3D matrix as a CNN input. The 3D CNN offers improved accuracy and precision over a similar 2D CNN that uses only one channel of vibration data. The method is particularly successful in identifying critically failed bearings, while prognosis from early and mid-life varies. All investigated methods see a reduction in accuracy when noise levels increase, but the 3D CNN still shows stronger overall performance when compared to the 2D CNN at similar noise levels, especially in the retention of precision and critical failure identification.

Compared with the results achieved in the previous chapter, this chapter shows a significant improvement in detecting end of life and avoiding prematurely crossing the HI = 0 threshold. However, since many aspects of the CNN were altered to achieve these results, a more systematic approach should be used to isolate which changes resulted in the improvement. Furthermore, it is inferred that the 3D CNN is stronger because of the combined spectrogram inputs and the 3D kernel. An additional experiment could be conducted with the combined spectrogram input and a 2D kernel to confirm that the kernel dimension is useful.

Chapter 7

Conclusions

Although the experimental chapters focused on individual subprocesses in the overall procedure of designing and validating a CNN for MHM tasks, the findings allow for some general conclusions to be drawn. First, the main conclusions of each chapter are briefly reviewed:

Chapter 3: Input Representations and Hyperparameters

Each input type and dataset studied yield different optimal combinations of hyperperameters. Spectrograms are found to be less sensitive to changes in hyperparameter selection and result in high diagnostic accuracy overall. When training on bearings with seeded faults, the best diagnostic accuracy decreases by about 30% when testing on naturally damaged bearings versus other seeded faulty bearings, calling into question the ability of lab data to represent real applications.

Chapter 4: Benchmarking Domain Adaptation in Fault Diagnosis

A problem with the commonly used framework for testing CNN algorithms is described, demonstrating how this framework artificially inflates the testing accuracy of domain shift diagnostic algorithms. The proposed method corrects this problem, and the overinflation of diagnostic scores is confirmed as diagnostic accuracy of common algorithms is found to decrease dramatically. Generic image recognition deep CNNs are retrained and found to be competitive or better than hand-build fault diagnosis CNNs, calling into question the practice of proposing novel diagnosis CNNs more often than implementing existing state-of-the-art CNNs when there exists a lack of available data for MHM.

Chapter 5: Influence of Data Augmentation in Prognosis

Most data augmentation methods, which were taken from other ML applications, provided some improvement in RMSE for HI estimation tasks, though less improvement than they provide in other domains. No significant correlation between failure mode and change in accuracy after augmentation was found. Data augmentation did not raise performance to the extent expected and performance remained somewhat poor, raising questions about the validity of the frequency-domain input and the FPT for piece-wise HI labeling. It becomes apparent that unique transformation-based data augmentation methods need to be developed for vibration-based MHM.

Chapter 6: Prognosis Using a 3D CNN and Multi-Channel Spectrogram

A multi-channel 3D spectrogram based input, used in conjunction with 3D convolutional kernels, is found to improve HI estimations when compared with single-channel inputs and 2D kernels. The proposed input and 3D CNN combination also surpasses the more traditional 2D CNN when tested with various levels of white noise added to the testing data. HI estimations for early-life operation and at end-of-life appear to be significantly better than the results achieved in Chapter 5, suggesting that accuracy can be more easily improved using better input representations and without using the FPT method than via data augmentation.

All chapters

In this thesis, the optimization and combination of individual aspects of applying ML to MHM was explored. Overall conclusions may be drawn from the sum of the efforts in these different areas. In general, it may be concluded that the field of MHM, like natural language processing, could benefit from using the spectrogram representation as the default data representation, rather than raw time or frequency spectra representations. Finding low accuracies in a seeded to natural fault domain shift as well as in the proposed CWRU domain shift framework highlights the weakness of the studied approaches in dealing with this kind of problem. This weakness must be overcome by researchers aiming to develop industry solutions using laboratory data.

Considering that pre-trained image networks taken from mainstream ML research fields prevailed over custom diagnosis networks, and also that time-frequency representations are superior for vibration-based MHM, as in speech recognition, it is tempting to conclude that MHM researchers should change as little as possible when adapting methods from mainstream ML studies. This notion is reinforced by the observation that, on their own, MHM studies have created and continue to use a rather poor framework to demonstrate domain shift. Contradicting this possible conclusion is the fact that data augmentation methods that are useful in environmental sound classification do not appear to be useful for MHM. Therefore, researchers in MHM should consider all aspects of the ML algorithm carefully, taking into account domain-specific considerations in certain areas. Areas requiring special consideration include data augmentation, input representation and formatting, and formulating training/testing scenarios that reflect realistic MHM problems. It may not be time-efficient for MHM researchers to focus on CNN architecture design since CNN optimization in other areas of ML will likely continue to exceed the performance of CNNs designed by MHM researchers.

Chapter 8

Recommendations for Future Work

In light of the preceeding work, the author would recommend the following suggestions for designing a CNN for MHM:

- Use a pretrained CNN such as ResNet and add a new convolutional layer with three filters per the method used in 4.3.4, with the exception of using a 1×1 kernel size.
- If N channels of vibration data are available, concatenate resulting N spectrograms into a $[224 \times 224 \times N]$ input size, using appropriate spectrogram prameters.
- The filters of the added layer will learn channel-wise features and transform the N-channel original input into the 3-channel input required for the ensuing pre-trained layers.
- For prognosis problems, it might be beneficial to explore a custom loss function that gives more weight to samples from the end of the bearing's lifetime than from the beginning. This would effectively be a compromise between the FPT method in Chapter 5 and the equal weighting/keeping of all training samples as used in Chapter 6.
- For diagnosis problems, it seems unhelpful to distinguish between different fault types when bearings tend to be replaced rather than repaired, so it would be logical to condense the class space to include healthy or faulty bearing location information only.

Work arising from areas touched on but not fully explored in the thesis include:

- In Chapter 4, ResNet and AlexNet are the only pre-trained deep image classification networks used, more recent iterations of ResNet or other pre-trained networks might achieve higher accuracy.
- Time-frequency representations, other than the spectrogram (ex. scalogram), should be explored as inputs to pre-trained networks.
- Multiple factors in the approach to prognosis were changed between Chapter 5 and Chapter 6, including CNN architecture, input representation, training/validation division, noise tolerance, data augmentation, and importantly the use of the FPT. It may be worth isolating some of these changes to determine what contributed most towards the greater performance in the later chapter.
- White noise was used to evaluate noise tolerance. Real noise measurements could be taken from industrial settings and superimposed on test data. Results may validate or invalidate the use of white noise for this purpose.

Finally, some areas are not touched on in this thesis, but offer promising tools for applying ML to MHM. Perhaps the most promising emergent tool is one for data augmentation; generative adversarial networks (GAN) can be used to synthesize additional training samples based on a small training dataset. GANs could be especially advantageous in MHM for synthesizing data representing faulty signals in industrial environments by learning transformations for laboratory data.

While it is easiest to make HI predictions from a single measurement, practical applications will also allow for the comparison of present-moment sensor readings with previous measurements from the same machine and sensor location within the learning algorithm. Recurrent networks have been demonstrated for MHM based on this premise, and

many of the lessons learned for CNNs could be transferred to recurrent network architectures.

When considering methods to reduce the input size required to contain relevant time or frequency data, autoencoders were considered. Experiments using autoencoders were omitted from the thesis because of time and length considerations, and partially because they were unsuccessful. Experiments were conducted to attempt to encode frequency spectra from the FEMTO dataset into smaller vectors to be used as input features to recurrent networks. In every experiment, the decoder suffered from posterior collapse. This is thought to be a result of the high noise input and inadequate number of training samples, giving low dataset homogeneity [64]. Autoencoders have been used with success in other areas of ML research, and more work could be done to explore how they might successfully be trained for MHM tasks.

Finally, all the algorithms used in this thesis are tailored for either diagnosis or prognosis. Regression-based prognosis algorithms are not explicitly trained to differentiate between fault types. It is conceivable that superior prognosis performance could be achieved by a process that combines diagnosis and prognosis. For example, a diagnosis algorithm might differentiate between fault types, after which a specialized prognosis algorithm is called that has been trained on a specific fault type. Alternatively, a diagnosis algorithm could be used to raise an alarm if a fault is detected. Then a prognosis algorithm is activated. Similar to FPT, this would reduce the range of health states over which prognosis needs to be performed, possibly making the task easier for an algorithm to learn.

References

[1] A. Stetco *et al.*, "Machine learning methods for wind turbine condition monitoring: A review," *Renew. Energy*, vol. 133, pp. 620–635, Apr. 2019, doi: 10.1016/j.renene.2018.10.047.

[2] S. Nandi, H. A. Toliyat, and X. Li, "Condition Monitoring and Fault Diagnosis of Electrical Motors—A Review," *IEEE Trans. Energy Convers.*, vol. 20, no. 4, pp. 719–729, Dec. 2005, doi: 10.1109/TEC.2005.847955.

[3] S.-H. Hong, "Literature Review of Machine Condition Monitoring with Oil Sensors -Types of Sensors and Their Functions," *Tribol. Lubr.*, vol. 36, no. 6, pp. 297–306, Dec. 2020, doi: 10.9725/KTS.2020.36.6.297.

[4] Y. Peng, M. Dong, and M. J. Zuo, "Current status of machine prognostics in condition-based maintenance: a review," *Int. J. Adv. Manuf. Technol.*, vol. 50, no. 1–4, pp. 297–313, Sep. 2010, doi: 10.1007/s00170-009-2482-0.

[5] M. S. Kan, A. C. C. Tan, and J. Mathew, "A review on prognostic techniques for non-stationary and non-linear rotating systems," *Mech. Syst. Signal Process.*, vol. 62–63, pp. 1–20, Oct. 2015, doi: 10.1016/j.ymssp.2015.02.016.

[6] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, and A. K. Nandi, "Applications of machine learning to machine fault diagnosis: A review and roadmap," *Mech. Syst. Signal Process.*, vol. 138, p. 106587, Apr. 2020, doi: 10.1016/j.ymssp.2019.106587.

[7] L. Jing, M. Zhao, P. Li, and X. Xu, "A convolutional neural network based feature learning and fault diagnosis method for the condition monitoring of gearbox," *Measurement*, vol. 111, pp. 1–10, Dec. 2017, doi: 10.1016/j.measurement.2017.07.017.

[8] V. Bolón-Canedo, N. Sánchez-Maroño, and A. Alonso-Betanzos, "A review of feature selection methods on synthetic data," *Knowl. Inf. Syst.*, vol. 34, no. 3, pp. 483–519, Mar. 2013, doi: 10.1007/s10115-012-0487-8.

[9] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, Aug. 2013, doi: 10.1109/TPAMI.2013.50.

[10] O. Cardona-Morales, L. D. Avendaño, and G. Castellanos-Domínguez, "Nonlinear model for condition monitoring of non-stationary vibration signals in ship driveline application," *Mech. Syst. Signal Process.*, vol. 44, no. 1, pp. 134–148, Feb. 2014, doi: 10.1016/j.ymssp.2013.08.029.

[11] J. Sun, H. Zuo, W. Wang, and M. G. Pecht, "Prognostics uncertainty reduction by fusing on-line monitoring data based on a state-space-based degradation model," *Mech. Syst. Signal Process.*, vol. 45, no. 2, pp. 396–407, Apr. 2014, doi: 10.1016/j.ymssp.2013.08.022.

[12] W. M. Azmy, N. E. Gayar, A. F. Atiya, and H. El-Shishiny, "MLP, Gaussian Processes and Negative Correlation Learning for Time Series Prediction," in *Multiple Classifier Systems*, Jun. 2009, pp. 428–437, doi: 10.1007/978-3-642-02326-2_43.

[13] X.-S. Si, W. Wang, C.-H. Hu, and D.-H. Zhou, "Remaining useful life estimation – A review on the statistical data driven approaches," *Eur. J. Oper. Res.*, vol. 213, no. 1, pp. 1–14, Aug. 2011, doi: 10.1016/j.ejor.2010.11.018.

[14] J. Z. Sikorska, M. Hodkiewicz, and L. Ma, "Prognostic modelling options for remaining useful life estimation by industry," *Mech. Syst. Signal Process.*, vol. 25, no. 5, pp. 1803–1836, Jul. 2011, doi: 10.1016/j.ymssp.2010.11.018.

[15] D. An, N. H. Kim, and J.-H. Choi, "Options for prognostics methods : A review of data-driven and physics-based prognostics," *Annu. Conf. PHM Soc.*, pp. 1–14, 2013.
[16] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.
[17] J. Hirschberg and C. D. Manning, "Advances in natural language processing," *Science*, vol. 349, no. 6245, pp. 261–266, 2015, doi: 10.1126/science.aaa8685.
[18] L. Eren, "Bearing Fault Detection by One-Dimensional Convolutional Neural Networks," *Math. Probl. Eng.*, vol. 2017, pp. 1–9, 2017, doi: 10.1155/2017/8617315.
[19] L. Eren, T. Ince, and S. Kiranyaz, "A Generic Intelligent Bearing Fault Diagnosis System Using Compact Adaptive 1D CNN Classifier," *J. Signal Process. Syst.*, vol. 91, no. 2, pp. 179–189, Feb. 2019, doi: 10.1007/s11265-018-1378-3.
[20] G. Jiang, H. He, J. Yan, and P. Xie, "Multiscale Convolutional Neural Networks for Fault Diagnosis of Wind Turbine Gearbox," *IEEE Trans. Ind. Electron.*, vol. 66, no. 4, pp. 3196–3207, Apr. 2019, doi: 10.1109/TIE.2018.2844805.
[21] R. Huang, Y. Liao, S. Zhang, and W. Li, "Deep Decoupling Convolutional Neural Network for Intelligent Compound Fault Diagnosis," *IEEE Access*, vol. 7, pp. 1848–1858, 2019, doi: 10.1109/ACCESS.2018.2886343.
[22] Y. Chen, G. Peng, C. Xie, W. Zhang, C. Li, and S. Liu, "ACDIN: Bridging the gap between artificial and real bearing damages for bearing fault diagnosis," *Neurocomputing*, vol. 294, pp. 61–71, Jun. 2018, doi: 10.1016/j.neucom.2018.03.014.
[23] D. K. Appana, A. Prosvirin, and J.-M. Kim, "Reliable fault diagnosis of bearings with varying rotational speeds using envelope spectrum and convolution neural networks," *Soft Comput.*, vol. 22, no. 20, pp. 6719–6729, Oct. 2018, doi: 10.1007/s00500-018-3256-0.
[24] S. Guo, T. Yang, W. Gao, and C. Zhang, "A Novel Fault Diagnosis Method for Rotating Machinery Based on a Convolutional Neural Network," *Sensors*, vol. 18, no. 5, p. 1429, May 2018, doi: 10.3390/s18051429.
[25] Y. Han, B. Tang, and L. Deng, "Multi-level wavelet packet fusion in dynamic ensemble convolutional neural network for fault diagnosis," *Measurement*, vol. 127, pp. 246–255, Oct. 2018, doi: 10.1016/j.measurement.2018.05.098.
[26] W. J. Lee, "Learning via acceleration spectrograms of a DC motor system with application to condition monitoring," *Int J Adv Manuf Technol*, p. 14, 2020.
[27] V. Pandhare, J. Singh, and J. Lee, "Convolutional Neural Network Based Rolling-Element Bearing Fault Diagnosis for Naturally Occurring and Progressing Defects Using Time-Frequency Domain Features," in *2019 Prognostics and System Health Management Conference (PHM-Paris)*, Paris, France, May 2019, pp. 320–326, doi: 10.1109/PHM-Paris.2019.00061.
[28] D. Verstraete, A. Ferrada, E. L. Droguett, V. Meruane, and M. Modarres, "Deep Learning Enabled Fault Diagnosis Using Time-Frequency Image Analysis of Rolling Element Bearings," *Shock Vib.*, vol. 2017, pp. 1–17, 2017, doi: 10.1155/2017/5067651.
[29] X. Li, W. Zhang, and Q. Ding, "Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction," *Reliab. Eng. Syst. Saf.*, vol. 182, pp. 208–218, Feb. 2019, doi: 10.1016/j.ress.2018.11.011.
[30] N. Li, Y. Lei, J. Lin, and S. X. Ding, "An Improved Exponential Model for Predicting Remaining Useful Life of Rolling Element Bearings," *IEEE Trans. Ind. Electron.*, vol. 62, no. 12, Dec. 2012.
[31] J.-R. Jiang, J.-E. Lee, and Y.-M. Zeng, "Time Series Multiple Channel Convolutional Neural Network with Attention-Based Long Short-Term Memory for Predicting Bearing Remaining Useful Life," *Sensors*, vol. 20, no. 1, p. 166, Dec. 2019, doi: 10.3390/s20010166.

[32] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J. Big Data*, vol. 6, no. 1, p. 60, Dec. 2019, doi: 10.1186/s40537-019-0197-0.
[33] J. Salamon and J. P. Bello, “Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification,” *IEEE Signal Process. Lett.*, vol. 24, no. 3, pp. 279–283, Mar. 2017, doi: 10.1109/LSP.2017.2657381.
[34] X. Li, W. Zhang, Q. Ding, and J.-Q. Sun, “Intelligent rotating machinery fault diagnosis based on deep learning using data augmentation,” *J. Intell. Manuf.*, vol. 31, no. 2, pp. 433–452, Feb. 2020, doi: 10.1007/s10845-018-1456-1.
[35] “Machine Learning textbook.” http://www.cs.cmu.edu/~tom/mlbook.html (accessed Jan. 19, 2021).
[36] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Englewood Cliffs, N.J: Prentice Hall, 1995.
[37] A. Vedaldi, “A convolutional neural network primer,” p. 56.
[38] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for Activation Functions,” *ArXiv171005941 Cs*, Oct. 2017, Accessed: Jan. 02, 2021. [Online]. Available: http://arxiv.org/abs/1710.05941.
[39] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, Art. no. 6088, Oct. 1986, doi: 10.1038/323533a0.
[40] P. D. McFadden and J. D. Smith, “Model for the vibration produced by a single point defect in a rolling element bearing,” *J. Sound Vib.*, vol. 96, no. 1, pp. 69–82, Sep. 1984, doi: 10.1016/0022-460X(84)90595-9.
[41] R. B. Randall and J. Antoni, “Rolling element bearing diagnostics—A tutorial,” *Mech. Syst. Signal Process.*, vol. 25, no. 2, pp. 485–520, Feb. 2011, doi: 10.1016/j.ymssp.2010.07.017.
[42] C. Lessmeier, J. K. Kimotho, D. Zimmer, and W. Sextro, “Condition Monitoring of Bearing Damage in Electromechanical Drive Systems by Using Motor Current Signals of Electric Motors: A Benchmark Data Set for Data-Driven Classification,” p. 17, 2016.
[43] P. D. McFadden, “Detecting Fatigue Cracks in Gears by Amplitude and Phase Demodulation of the Meshing Vibration,” *J. Vib. Acoust.*, vol. 108, no. 2, pp. 165–170, Apr. 1986, doi: 10.1115/1.3269317.
[44] Y. Guo, L. Zhao, X. Wu, and J. Na, “Vibration separation technique based localized tooth fault detection of planetary gear sets: A tutorial,” *Mech. Syst. Signal Process.*, vol. 129, pp. 130–147, Aug. 2019, doi: 10.1016/j.ymssp.2019.04.027.
[45] “Public Data Sets | PHM Society.” https://www.phmsociety.org/references/datasets (accessed Aug. 24, 2020).
[46] “Case Western Reserve University Bearing Data Center Website.” (http://csegroups.case.edu/bearingdatacenter/home).
[47] W. A. Smith and R. B. Randall, “Rolling element bearing diagnostics using the Case Western Reserve University data: A benchmark study,” *Mech. Syst. Signal Process.*, vol. 64–65, pp. 100–131, Dec. 2015, doi: 10.1016/j.ymssp.2015.04.021.
[48] F. Jia, Y. Lei, J. Lin, X. Zhou, and N. Lu, “Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data,” *Mech. Syst. Signal Process.*, vol. 72–73, pp. 303–315, May 2016, doi: 10.1016/j.ymssp.2015.10.025.
[49] W. Zhang, G. Peng, C. Li, Y. Chen, and Z. Zhang, “A New Deep Learning Model for Fault Diagnosis with Good Anti-Noise and Domain Adaptation Ability on Raw Vibration Signals,” *Sensors*, vol. 17, no. 2, p. 425, Feb. 2017, doi: 10.3390/s17020425.

[50] W. Zhang, C. Li, G. Peng, Y. Chen, and Z. Zhang, "A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load," *Mech. Syst. Signal Process.*, vol. 100, pp. 439–453, Feb. 2018, doi: 10.1016/j.ymssp.2017.06.022.

[51] Z. Zhu, G. Peng, Y. Chen, and H. Gao, "A convolutional neural network based on a capsule network with strong generalization for bearing fault diagnosis," *Neurocomputing*, vol. 323, pp. 62–75, Jan. 2019, doi: 10.1016/j.neucom.2018.09.050.

[52] W. Caesarendra and T. Tjahjowidodo, "A Review of Feature Extraction Methods in Vibration-Based Condition Monitoring and Its Application for Degradation Trend Estimation of Low-Speed Slew Bearing," *Machines*, vol. 5, no. 4, p. 21, Sep. 2017, doi: 10.3390/machines5040021.

[53] C. Lessmeier, J. K. Kimotho, D. Zimmer, and W. Sextro, "Condition Monitoring of Bearing Damage in Electromechanical Drive Systems by Using Motor Current Signals of Electric Motors: A Benchmark Data Set for Data-Driven Classification," p. 17, 2016.

[54] B. Yang, "An intelligent fault diagnosis approach based on transfer learning from laboratory bearings to locomotive bearings," *Mech. Syst. Signal Process.*, p. 15, 2019.

[55] NASA Ames Prognostics Data Repository (http://ti.arc.nasa.gov/project/prognostic-data-repository), "FEMTO Bearing Data Set," *NASA Ames Res. Cent. Moffett Field CA*.

[56] P. Nectoux *et al.*, "PRONOSTIA: An experimental platform for bearings accelerated degradation tests.," p. 9.

[57] J. Hendriks and P. Dumond, "Vibration-Based Prognosis of Rolling Element Bearings With a 3D Convolutional Neural Net," Prague, Accepted 2020 2021, p. 8.

[58] H. Pan, X. He, S. Tang, and F. Meng, "An Improved Bearing Fault Diagnosis Method using One-Dimensional CNN and LSTM," *Stroj. Vestn. - J. Mech. Eng.*, Jun. 2018, doi: 10.5545/sv-jme.2018.5249.

[59] M. Xia, T. Li, L. Xu, L. Liu, and C. W. de Silva, "Fault Diagnosis for Rotating Machinery Using Multiple Sensors and Convolutional Neural Networks," *IEEEASME Trans. Mechatron.*, vol. 23, no. 1, pp. 101–110, Feb. 2018, doi: 10.1109/TMECH.2017.2728371.

[60] W. Zhang, F. Zhang, W. Chen, Y. Jiang, and D. Song, "Fault State Recognition of Rolling Bearing Based Fully Convolutional Network," *Comput. Sci. Eng.*, vol. 21, no. 5, pp. 55–63, Sep. 2019, doi: 10.1109/MCSE.2018.110113254.

[61] Z. Zilong and Q. Wei, "Intelligent fault diagnosis of rolling bearing using one-dimensional multi-scale deep convolutional neural network based health state classification," in *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, Zhuhai, Mar. 2018, pp. 1–6, doi: 10.1109/ICNSC.2018.8361296.

[62] J. Pan, Y. Zi, J. Chen, Z. Zhou, and B. Wang, "LiftingNet: A Novel Deep Learning Network With Layerwise Feature Learning From Noisy Mechanical Data for Fault Classification," *IEEE Trans. Ind. Electron.*, vol. 65, no. 6, pp. 4973–4982, Jun. 2018, doi: 10.1109/TIE.2017.2767540.

[63] E. Sejdić, I. Djurović, and J. Jiang, "Time–frequency feature representation using energy concentration: An overview of recent advances," *Digit. Signal Process.*, vol. 19, no. 1, pp. 153–183, Jan. 2009, doi: 10.1016/j.dsp.2007.12.004.

[64] "python - What is 'posterior collapse' phenomenon?," *Data Science Stack Exchange*. https://datascience.stackexchange.com/questions/48962/what-is-posterior-collapse-phenomenon (accessed Jan. 26, 2021).

Appendix A: MATLAB Code

1. CWRU_data_augmentation_from_uni.mat

```
function augmented_data = CWRU_data_augmentation_from_uni(unified_data,data_window,data_aug_stride,data_ch)

augmented_data = cell(1,2);

unified_row_index = 1;

while unified_row_index <= length(unified_data(:,1))
   %augmented_row = cell(1,2);
   current_label = char(unified_data{unified_row_index,2});
   raw_row_signal = cell2mat(unified_data(unified_row_index,1));
   raw_duration = length(raw_row_signal);

   starting_index = 1;
   while starting_index < raw_duration - data_window

      if data_ch == 1 || data_ch == 2
         current_segment = raw_row_signal(starting_index:starting_index+data_window-1,:,data_ch);
      else
         current_segment = raw_row_signal(starting_index:starting_index+data_window-1,:,:);
      end

      new_row = [{current_segment},{current_label}];
      augmented_data = [augmented_data ; new_row];
      starting_index = starting_index + data_aug_stride;
   end
   unified_row_index = unified_row_index + 1;
end

augmented_data = augmented_data(2:end,:);
%{
if data_ch == 1 || data_ch == 2
   %master_data = zeros(length(augmented_data{1,1}),1,1,length(augmented_data));
   %master_labels = strings(1,length(augmented_data));
   j = 1;

   while j <= length(augmented_data)

      augmented_data(j,1) = {augmented_data{j,1}(:,:,data_ch)};

      %both_channels_data = augmented_data{j,1};
      %master_data(:,:,:,j) = both_channels_data(:,:,data_ch,:);
      %master_labels(j) = augmented_data{j,2};

      j = j+1;
   end
end

%{
if data_ch == 3

   %master_data = zeros(length(augmented_data{1,1}),1,2,length(augmented_data));
   %master_labels = strings(1,length(augmented_data));
   j = 1;

   while j <= length(augmented_data)
```

```
        master_data(:,:,:,j) = augmented_data{j,1};
        master_labels(j) = augmented_data{j,2};
        j = j+1;
    end
    %}
end
end
```

2. CWRU_data_unifier.mat

```
% Specify the folder where the labeled sub-folders live.
root_folder = 'D:\CWRU_data\Paper_3\EF_D\Training\';

% Get a list of all files and folders in this folder.
files = dir(root_folder);
% Get a logical vector that tells which is a directory.
dirFlags = [files.isdir];
% Extract only those that are directories.
subFolders = files(dirFlags);
sub_folder_index = 3;

unified_data = {};

while sub_folder_index <= length(subFolders)

    myFolder = subFolders(sub_folder_index).name;
    myFolder = strcat(root_folder,myFolder,'\');

    % Get a list of all files in the folder with the desired file name pattern.
    filePattern = fullfile(myFolder, '*.mat');
    theFiles = dir(filePattern);
    theFolders = strings(1,length(theFiles));

    master_data = cell(length(theFiles),2);
    master_labels = cell(length(theFiles),1);
    master_index = 1;

    for k = 1 : length(theFiles)
        baseFileName = theFiles(k).name;
        fullFileName = fullfile(myFolder, baseFileName);
        %fprintf(1, 'Now reading %s\n', fullFileName);
        theFolders(k) = get_folder_name(fullFileName);

        raw_measurement = load(fullFileName);

        raw_de_struct = load(fullFileName,'*_DE_time');
        raw_fe_struct = load(fullFileName,'*_FE_time');

        raw_de_field = fields(raw_de_struct);
        raw_fe_field = fields(raw_fe_struct);

        raw_de = raw_de_struct.(char(raw_de_field));
        raw_fe = raw_fe_struct.(char(raw_fe_field));
        raw_data = zeros(length(raw_de),1,2);
        raw_data(:,1,1) = raw_de;
        raw_data(:,1,2) = raw_fe;

        master_data(k,1) = {raw_data};
        master_data(k,2) = {theFolders(k)};

        master_index = master_index + 1;

    end

    unified_data = [unified_data; master_data];
    sub_folder_index = sub_folder_index + 1;


end

save('D:\CWRU_data\Paper_3\EF_D\Training\unified_data.mat','unified_data')

function folder_name = get_folder_name(fullFileName)
```

```
    index_1 = 1;
    index_2 = 1;
    j = 1;
    while j <= length(fullFileName)-1
        temp_char = fullFileName(length(fullFileName)-j);
        if temp_char == '\'
            if index_1 == 1
                index_1 = j;
            elseif (index_1 ~= 1) && (index_2 == 1)
                    index_2 = j;
            end
        end
        j = j + 1;
    end
    folder_name = fullFileName(length(fullFileName)-index_2+1:length(fullFileName)-index_1-1);
end
```

3. CWRU_gather_data.mat

```
function [train_data, Train_labels, validation_data, Validation_labels] = CWRU_gather_data(data_path, numTrainingFiles, numTestingFiles, read_channel)


% Older function, after my ICSV paper I stopped using MATLAB's "image datastore" objects to hold training % data since using large numerical arrays gives more flexibility, though I had to make some custom % functions to support the switch

% Read file names and folder names in data path, create and shuffle an Image Datastore to
% using file names as labels

%Uses readFcn1_1D_ch1 to read first channel
if read_channel == 1

    imds = shuffle(imageDatastore(data_path,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames'));

    [imdsTrain,imdsTest] = splitEachLabel(imds,numTrainingFiles, numTestingFiles,'randomize');

    % Convert labels for training set and testing set into catagorical arrays
    imdsTrain.Labels = categorical(imdsTrain.Labels);
    imdsTrain.ReadFcn = @readFcn1_1D_ch1;
    imdsTest.Labels = categorical(imdsTest.Labels);
    imdsTest.ReadFcn = @readFcn1_1D_ch1;

    % Read filenames from datastores and write samples to unified 4D arrays
    fprintf('Reading data files \n');
    tic
    i = 1;
    train_data = zeros(1200,1,1,length(imdsTrain.Files)-1);
    train_labels = strings(length(imdsTrain.Files)-1,1);
    while i < length(imdsTrain.Labels)
        train_data(:,:,:,i) = readFcn1_1D_ch1(string(imdsTrain.Files(i)));
        train_labels(i) = imdsTrain.Labels(i);
        i = i + 1;
    end
    Train_labels = categorical(train_labels);
    i = 1;
    validation_data = zeros(1200,1,1,length(imdsTest.Files)-1);
    validation_labels = strings(length(imdsTest.Files)-1,1);
    while i < length(imdsTest.Labels)
        validation_data(:,:,:,i) = readFcn1_1D_ch1(string(imdsTest.Files(i)));
        validation_labels(i) = imdsTest.Labels(i);
        i = i + 1;
    end
    toc
    Validation_labels = categorical(validation_labels);

end

%Uses readFcn1_1D_ch2 to read second channel
if read_channel == 2

    imds = shuffle(imageDatastore(data_path,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames'));

    [imdsTrain,imdsTest] = splitEachLabel(imds,numTrainingFiles, numTestingFiles,'randomize');

    % Convert labels for training set and testing set into catagorical arrays
```

```
    imdsTrain.Labels = categorical(imdsTrain.Labels);
    imdsTrain.ReadFcn = @readFcn1_1D_ch2;
    imdsTest.Labels = categorical(imdsTest.Labels);
    imdsTest.ReadFcn = @readFcn1_1D_ch2;

    % Read filenames from datastores and write samples to unified 4D arrays
    fprintf('Reading data files \n');
    tic
    i = 1;
    train_data = zeros(1200,1,1,length(imdsTrain.Files)-1);
    train_labels = strings(length(imdsTrain.Files)-1,1);
    while i < length(imdsTrain.Labels)
        train_data(:,:,:,i) = readFcn1_1D_ch2(string(imdsTrain.Files(i)));
        train_labels(i) = imdsTrain.Labels(i);
        i = i + 1;
    end
    Train_labels = categorical(train_labels);
    i = 1;
    validation_data = zeros(1200,1,1,length(imdsTest.Files)-1);
    validation_labels = strings(length(imdsTest.Files)-1,1);
    while i < length(imdsTest.Labels)
        validation_data(:,:,:,i) = readFcn1_1D_ch2(string(imdsTest.Files(i)));
        validation_labels(i) = imdsTest.Labels(i);
        i = i + 1;
    end
    toc
    Validation_labels = categorical(validation_labels);

end
```

4. CWRU_gather_data_k_fold_CV.mat

```
function [train_data, Train_labels, validation_data, Validation_labels] = CWRU_gather_data_k_fold_CV(, read_channel, k, n)

% Read file names and folder names in data path, create and shuffle an Image Datastore to
% using file names as labels

%Uses readFcn1_1D_ch1 to read first channel
if read_channel == 1

    % Convert labels for training set and testing set into catagorical arrays
    imdsTrain.Labels = categorical(imdsTrain.Labels);
    imdsTrain.ReadFcn = @readFcn1_1D_ch1;
    imdsTest.Labels = categorical(imdsTest.Labels);
    imdsTest.ReadFcn = @readFcn1_1D_ch1;

    % Read filenames from datastores and write samples to unified 4D arrays
    fprintf('Reading data files \n');
    tic
    i = 1;
    train_data = zeros(1200,1,1,length(imdsTrain.Files)-1);
    train_labels = strings(length(imdsTrain.Files)-1,1);
    while i < length(imdsTrain.Labels)
        train_data(:,:,:,i) = readFcn1_1D_ch1(string(imdsTrain.Files(i)));
        train_labels(i) = imdsTrain.Labels(i);
        i = i + 1;
    end
    Train_labels = categorical(train_labels);
    i = 1;
    validation_data = zeros(1200,1,1,length(imdsTest.Files)-1);
    validation_labels = strings(length(imdsTest.Files)-1,1);
    while i < length(imdsTest.Labels)
        validation_data(:,:,:,i) = readFcn1_1D_ch1(string(imdsTest.Files(i)));
        validation_labels(i) = imdsTest.Labels(i);
        i = i + 1;
    end
    toc
    Validation_labels = categorical(validation_labels);

end


%Uses readFcn1_1D_ch2 to read second channel
if read_channel == 2
```

```
    imds = shuffle(imageDatastore(data_path,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames'));

    [imdsTrain,imdsTest] = splitEachLabel(imds,numTrainingFiles, numTestingFiles,'randomize');

    % Convert labels for training set and testing set into catagorical arrays
    imdsTrain.Labels = categorical(imdsTrain.Labels);
    imdsTrain.ReadFcn = @readFcn1_1D_ch2;
    imdsTest.Labels = categorical(imdsTest.Labels);
    imdsTest.ReadFcn = @readFcn1_1D_ch2;

    % Read filenames from datastores and write samples to unified 4D arrays
    fprintf('Reading data files \n');
    tic
    i = 1;
    train_data = zeros(1200,1,1,length(imdsTrain.Files)-1);
    train_labels = strings(length(imdsTrain.Files)-1,1);
    while i < length(imdsTrain.Labels)
        train_data(:,:,:,i) = readFcn1_1D_ch2(string(imdsTrain.Files(i)));
        train_labels(i) = imdsTrain.Labels(i);
        i = i + 1;
    end
    Train_labels = categorical(train_labels);
    i = 1;
    validation_data = zeros(1200,1,1,length(imdsTest.Files)-1);
    validation_labels = strings(length(imdsTest.Files)-1,1);
    while i < length(imdsTest.Labels)
        validation_data(:,:,:,i) = readFcn1_1D_ch2(string(imdsTest.Files(i)));
        validation_labels(i) = imdsTest.Labels(i);
        i = i + 1;
    end
    toc
    Validation_labels = categorical(validation_labels);

end
```

5. CWRU_network_commander.mat

```
Clc

% I have replaced this function with a better one…

% Gather and prepare data for training and testing:

data_path = strcat('D:\CWRU_data\augmented_time_data');
save_network = true;
keep_training_gui_open = false;
network_save_path = 'D:\CWRU_data\cached_cnns\test';
network_save_name = '1D_time_data_1200_1_1_t50';

numTrainingFiles = 50;
numTestingFiles = 50;
read_channel = 1;

%%% Catch console output for saving a .txt info file describing network
%%% parameters
if save_network == true
    diary on
    fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'))
    fprintf('\n\n')
end

% Create sub-sets for training and testing files with specified numbers in each
[train_data, train_labels, validation_data, validation_labels] = CWRU_gather_data(data_path, numTrainingFiles, numTestingFiles, read_channel);

%%% Specify details for each conv. layer of the network. Each column
%%% represents the parameters for a layer, thus each must have the same
%%% number of columns. 1D should be represented as [X, 1], 2D can be
%%% represented as [X1, X2] and 3D can be represented as [X1, X2, X3]

conv_filter = [[ 4, 1 ];[ 8, 1 ]];
conv_stride = [[ 2, 1 ];[ 4, 1 ]];
pool_filter = [[ 2, 1 ];[ 2, 1 ]];
pool_stride = [[ 1, 1 ];[ 1, 1 ]];
```

```
num_filters = [ 16; 16];
batch_norm = [true; true];
relu = [true; true];
output_size = length(countcats(train_labels));

%%% Training options
options = trainingOptions('sgdm', ...
  'InitialLearnRate',0.01, ...
  'Verbose',false, ...
  'Plots','training-progress', ...
  'ValidationData',{validation_data,validation_labels}, ...
  'ValidationFrequency', 60, ...
  'MaxEpochs',7, ...
  'MiniBatchSize',32,...
  'ExecutionEnvironment','gpu');

[trained_net,info] = network_trainer(train_data, train_labels, output_size, batch_norm, relu, conv_filter, conv_stride, num_filters, pool_filter, pool_stride,
options);
fprintf('\n\nFinal training accuracy: %0.2f\nFinal validation accuracy: %0.2f\nInitial learn rate: %0.5f\n', info.TrainingAccuracy(end),
info.ValidationAccuracy(end), info.BaseLearnRate(1))
fprintf('\n\n\n\n')

if save_network == true

  save(strcat(network_save_path,'\',network_save_name,'.mat'), "trained_net", "info");
  diary_filename = strcat(network_save_path,'\',network_save_name,'.txt');
  diary(diary_filename);
  diary off
end

if keep_training_gui_open == false
  delete(findall(0));
end
```

6. CWRU_network_commander_cross_validation.mat

```
function network = CWRU_network_commander_cross_validation(network)

% This function receives custom “network” struct that contains parameters for this function to build a      % CNN, and train it using k-fold cross validation,
save details to a log, and return trained k trained  % networks stored inside the k-dimensional network struct to the calling function. Used for Manuscript 1

  %%% Rename some variables to make it simpler to call on them later
  save_network = network.save_network;
  keep_training_gui_open = network.keep_training_gui_open;
  network_save_path = network.network_save_path;
  network_save_name = network.network_save_name;
  normalization = network.normalization;
  conv_filter = network.conv_filter;
  conv_stride = network.conv_stride;
  pool_filter = network.pool_filter;
  pool_stride = network.pool_stride;
  num_filters = network.num_filters;
  batch_norm = network.batch_norm;
  relu = network.relu;
  dropout = network.dropout;
  k = network.k;
  MaxEpochs = network.MaxEpochs;
  dataset = network.Dataset;
  data_window = network.data_window;
  data_aug_stride = network.data_aug_stride;
  data_ch = network.data_ch;
  MiniBatchSize = network.MiniBatchSize;
  preprocessor = network.preprocessor;
  spectrogram_window = network.spectrogram_window;
  scalogram_bins = network.scalogram_bins;

  %Import, shuffle, and perform augmentation on the dataset
  data = load(dataset);
  data_cats = string(cell2mat(data.unified_data(:,2)));

  for cat_index = 1:length(data_cats)
    %fprintf(char(data_cats(cat_index)))
    current_data = cell2mat(data.unified_data(cat_index,1));
    dimensions = size(current_data);
```

```
    ix = randperm(dimensions(1));
    new_data.(char(data_cats(cat_index))) = current_data(ix,:);
    %call augmentation_funciton
    %data.(char(data_cats(cat_index))) = CWRU_data_augmentation_from_uni(current_data,data_window,data_aug_stride,data_ch);
  end
 data = new_data;

 %n = current iteration, 1 <= n <= k
 % Initialize a bunch of values to store results over the folds
 n = 1;
 validation_accuracy_tally = zeros(k, 1);
 train_accuracy_tally = zeros(k, 1);
 time_tally = zeros(k, 1);
 training_size_tally = zeros(k, 1);
 %network_taly = [];

 while n <= k

    %create empty train and validation data matricies
    cell_train_data = [];
    cell_validation_data = [];

    train_labels = [];
    validation_labels = [];

    %read each category one at a time to perform seperation of
    %different categories between training and validatiaon
    %indepentantly, ensuring that equal numbers of examples from each
    %cat make their way into validation vs training (previous problems
    %included instances where a given cat had no training examples due
    %to fully randomized division)
    for cat_index = 1:length(data_cats)

       %temporarily hang onto data from present category
       current_data = data.(char(data_cats(cat_index)));

       subset_size = floor(length(current_data(:,1))/k);

       current_train_data = current_data(1+((n-1)*subset_size):((n)*subset_size),:);
       %train_labels = unified_data(1+((n-1)*subset_size):((n)*subset_size));

       current_validation_data = current_data;
       %validation_labels = master_labels;
       current_validation_data(1+((n-1)*subset_size):((n)*subset_size),:) = [];
       %validation_labels(((1+(n-1)*subset_size):((n)*subset_size))) = [];

       %[train_data,train_labels] = CWRU_data_augmentation_from_uni(train_data,data_window,data_aug_stride,data_ch);
       %[validation_data,validation_labels] = CWRU_data_augmentation_from_uni(validation_data,data_window,data_aug_stride,data_ch);

       current_train_data = CWRU_data_augmentation_from_uni(current_train_data,data_window,data_aug_stride,data_ch);
       current_validation_data = CWRU_data_augmentation_from_uni(current_validation_data,data_window,data_aug_stride,data_ch);

       current_train_labels = categorical(current_train_data(:,2));
       current_validation_labels = categorical(current_validation_data(:,2));

       current_train_data = current_train_data(:,1);
       current_validation_data = current_validation_data(:,1);

       cell_train_data = [cell_train_data ; current_train_data];
       cell_validation_data = [cell_validation_data ; current_validation_data];

       train_labels = [train_labels ; current_train_labels];
       validation_labels = [validation_labels ; current_validation_labels];
    end

    train_data = [];
    validation_data = [];
    for train_data_row_index = 1:length(cell_train_data)
       train_data(:,:,:,train_data_row_index) = cell_train_data{train_data_row_index};
    end

    for validation_data_row_index = 1:length(cell_validation_data)
       validation_data(:,:,:,validation_data_row_index) = cell_validation_data{validation_data_row_index};
    end
```

```
if ~strcmp(preprocessor,'none')
   train_data = generalized_preprocessor(train_data,preprocessor,spectrogram_window,scalogram_bins);
   validation_data = generalized_preprocessor(validation_data,preprocessor,spectrogram_window,scalogram_bins);
end

train_dim = size(train_data);
idx_train = randperm(train_dim(end));
train_data = train_data(:,:,:,idx_train);
train_labels = train_labels(idx_train);

validation_dim = size(validation_data);
idx_validation = randperm(validation_dim(end));
validation_data = validation_data(:,:,:,idx_validation);
validation_labels = validation_labels(idx_validation);

training_size_tally(n) = length(train_data);
% This parameter determines the size of the output of the final FC
% layer. It has to be defined after the data is imported because it
% determines the required size by reading the number of output
% categories in the label space.
output_size = length(countcats(train_labels));


%%% Training options
options = trainingOptions('adam', ...
'InitialLearnRate',0.001, ...
'Verbose',false, ...
'Plots','training-progress', ...
'ValidationData',{validation_data,validation_labels}, ...
'ValidationFrequency', floor(2*length(validation_labels)/k/MiniBatchSize), ...
'MaxEpochs',MaxEpochs, ...
'MiniBatchSize',MiniBatchSize,...
'ExecutionEnvironment','gpu');

[temp_trained_net,temp_info,temp_time] = network_trainer(train_data, train_labels, output_size, batch_norm, relu, dropout, conv_filter, conv_stride,
num_filters, pool_filter, pool_stride, options);


if ~strcmp(temp_info,'Invalid network')

   network.categories = categories(validation_labels);

   network.true_validation_labels{n} = validation_labels;
   predicted_validation_labels_idx{n} = predict(temp_trained_net,validation_data);
   predicted_validation_labels = strings(size(validation_labels));
   prediction_index = 1;
   while prediction_index <= length(validation_labels)
      [maximum,index] = max(predicted_validation_labels_idx{n}(prediction_index,:));
      predicted_validation_labels(prediction_index) = network.categories{index};
      prediction_index = prediction_index+1;
   end
   network.predicted_validation_labels{n} = categorical(predicted_validation_labels);
   network.confusion_mat_validation{n} = confusionmat(validation_labels,network.predicted_validation_labels{n});


   network.true_train_labels{n} = train_labels;
   predicted_train_labels_idx{n} = predict(temp_trained_net,train_data);
   predicted_train_labels = strings(size(train_labels));
   prediction_index = 1;
   while prediction_index <= length(train_labels)
      [maximum,index] = max(predicted_train_labels_idx{n}(prediction_index,:));
      predicted_train_labels(prediction_index) = network.categories{index};
      prediction_index = prediction_index+1;
   end
   network.predicted_train_labels{n} = categorical(predicted_train_labels);
   network.confusion_mat_train{n} = confusionmat(train_labels,network.predicted_train_labels{n});


   network.trained_network{n} = temp_trained_net;
   network.info{n} = temp_info;
   network.time(n) = temp_time;
   network.training_size_tally = training_size_tally;

   fprintf('\n\nFinal training accuracy: %0.2f\nFinal validation accuracy: %0.2f\nInitial learn rate: %0.5f\n', temp_info.TrainingAccuracy(end),
temp_info.ValidationAccuracy(end), temp_info.BaseLearnRate(1))
```

```
            fprintf('\n\n\n\n')

            validation_accuracy_tally(n) = temp_info.ValidationAccuracy(end);
            train_accuracy_tally(n) = temp_info.TrainingAccuracy(end);
            time_tally(n) = temp_time;

            if keep_training_gui_open == false
               delete(findall(0));
            end
          else
            n = k;
          end
         n = n+1;
        end

        if save_network == true && ~strcmp(temp_info,'Invalid network')
          save(strcat(network_save_path,'\',network_save_name,'.mat'),"network");
          diary_filename = strcat(network_save_path,'\',network_save_name,'_summary.txt');
          diary(diary_filename);
          fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
          fprintf('\n\n')
          temp_trained_net.Layers
          fprintf('\n\n')
          fprintf('Validation accuracy:');
          fprintf('  %f  ', validation_accuracy_tally);
          fprintf('\nValidation mean = %f\n',mean(validation_accuracy_tally))
          fprintf('Validation variance = %f\n\n',var(validation_accuracy_tally))
          fprintf('Train accuracy:');
          fprintf('  %f  ', train_accuracy_tally);
          fprintf('\nTraining mean = %f\n',mean(train_accuracy_tally))
          fprintf('Training ariance = %f\n\n',var(train_accuracy_tally))
          fprintf('Training time:');
          fprintf('  %f  ', time_tally);
          fprintf('\nMean = %f\n',mean(time_tally))
          fprintf('Training samples used:');
          fprintf('  %f  ', training_size_tally);
          fprintf('\nMean = %f\n',mean(training_size_tally))
          diary
        end
        if strcmp(temp_info,'Invalid network')
          save(strcat(network_save_path,'\invalid_',network_save_name,'.mat'),"network");
          diary_filename = strcat(network_save_path,'\invalid_',network_save_name,'_summary.txt');
          diary(diary_filename);
          fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
          fprintf('\n\n')
          temp_trained_net.Layers
          diary
        end
      end
```

7. CWRU_network_queuer.mat

```
      % for k-fold cross validation, specify k
      default_network.k = 4;

      % Commander parameters
      %default_network.Dataset = 'D:\2009PHM_data\labeled_data\gear_data\augmented_raw_time\2009PHM_ch1_t2400.mat';
      %default_network.Dataset = 'D:\CWRU_data\augmented_time_data\master_data_ch1_4D_1200.mat';
      %default_network.Dataset = 'D:\CWRU_data\raw_data\sorted_all_unified_data.mat';
      %default_network.Dataset =
["D:\Paderborn_data\sorted_data_3\testing\testing_data.mat","D:\Paderborn_data\sorted_data_3\training\training_data.mat"];

      default_network.save_network = true;
      default_network.keep_training_gui_open = false;
      %default_network.network_save_path = 'D:\master_paper_2\case_study_1\part_1\ch1_dropout_16epochs_adam\4_1\';
      default_network.network_save_name = '';

      default_network.data_ch = 1;
      %default_network.data_window = 416;

      %%%


      for preprocessor = ["env_spec"]  %defined in generalized_preprocessor.m, values are 'none' 'fourier_spec' 'env_spec' 'spectrogram'
```

```
default_network.preprocessor = preprocessor;


conv_filter_stride = {[2,2] , [1,1];...
                [4,4] , [1,1];... Filter size first column, stride second column.
                [6,6] , [1,1];... Trains a new batch of networks for each row
                [8,8] , [1,1];...
                };

conv_filter_stride = {[4,1] , [2,1];...
                [8,1]  , [2,1];... Filter size first column, stride second column.
                [16,1] , [4,1];... Trains a new batch of networks for each row
                [32,1] , [4,1];...
                [64,1] , [4,1];...
                [128,1], [4,1];...
                [256,1], [4,1]};



filter_index = 1;
while filter_index <= length(conv_filter_stride)

   for data_window = [9984 4992 2496 1248 832 416]%
      default_network.data_window = data_window;
   %%%
      %this data window is implicated in filenames calculated below

      %default_network.preprocessor = 'none';

      general_conv_filter = conv_filter_stride{filter_index,1};
      general_conv_stride = conv_filter_stride{filter_index,2};
      default_network.network_save_path =
strcat('D:\master_paper_2\case_study_2\artificial_damage_only\',default_network.preprocessor,'_v2\',string(general_conv_filter(1)),'_',string(general_conv_filter(
2)));

      %default_network.data_aug_stride = data_window/4*3;
      default_network.data_aug_stride = data_window;
      default_network.normalization = 'zerocenter';
      %In the version of matlab in which this code was written, only
      %'zerocenter' and 'none' are availible as normalization methods. Future
      %versions support 'zscore' which could be better. I may circle back
      %and write my own version of these functions since I believe
      %they're very simple.

      %%% Specify details for each conv. layer of the network. Each column
      %%% represents the parameters for a layer, thus each must have the same
      %%% number of columns. 1D should be represented as [X, 1], 2D can be
      %%% represented as [X1, X2] and 3D can be represented as [X1, X2, X3]
      default_network.conv_filter = [general_conv_filter;general_conv_filter];
      default_network.conv_stride = [general_conv_stride;general_conv_stride];
      default_network.pool_filter = [[ 2, 1 ];[ 2, 1 ]];
      default_network.pool_stride = [[ 2, 1 ];[ 2, 1 ]];
      default_network.num_filters = [ 16; 16];
      default_network.batch_norm = [true; true];
      default_network.relu = [true; true];
      default_network.dropout = [true; true];
      default_network.MaxEpochs = 3;
      default_network.MiniBatchSize = 32;
      default_network.spectrogram_window = 104;
      default_network.scalogram_bins = [1:16];

      mkdir(default_network.network_save_path)

      % To use the above parameters as the default, create new networks and
      % overwrite the variables which are to change in each
      network1 = default_network;
      network1.conv_filter = [general_conv_filter];
      network1.conv_stride = [general_conv_stride];
      network1.pool_filter = [ 2, 2 ];
      network1.pool_stride = [ 1, 1 ];
      network1.num_filters = 16;
      network1.batch_norm = false;
      network1.relu = true;
```

```
network1.dropout = true;
%network1.MaxEpochs = 4;
network1.network_save_name = strcat('t',sprintf('%05d',data_window),'_1x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network2 = default_network;
network2.conv_filter = [general_conv_filter;general_conv_filter];
network2.conv_stride = [general_conv_stride;general_conv_stride];
network2.pool_filter = [[ 2, 1 ];[ 2, 1 ]];
network2.pool_stride = [[ 1, 1 ];[ 1, 1 ]];
network2.num_filters = [ 16; 16];
network2.batch_norm = [false; false];
network2.relu = [true; true];
network2.dropout = [true; true];
%network2.MaxEpochs = 4;
network2.network_save_name = strcat('t',sprintf('%05d',data_window),'_2x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_3e');

network3 = default_network;
network3.conv_filter = [general_conv_filter;general_conv_filter;general_conv_filter];
network3.conv_stride = [general_conv_stride;general_conv_stride;general_conv_stride];
network3.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
network3.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
network3.num_filters = [ 16; 16; 16];
network3.batch_norm = [false; false; false];
network3.relu = [true; true; true];
network3.dropout = [true; true; true];
%network3.MaxEpochs = 4;
network3.network_save_name = strcat('t',sprintf('%05d',data_window),'_3x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network4 = default_network;
network4.conv_filter = [general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
network4.conv_stride = [general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride];
network4.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
network4.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
network4.num_filters = [ 16; 16; 16; 16];
network4.batch_norm = [true; true; true; true];
network4.relu = [true; true; true; true];
network4.MaxEpochs = 4;
network4.network_save_name = strcat('t',sprintf('%05d',data_window),'_4x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network5 = default_network;
network5.conv_filter = [general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
network5.conv_stride = [general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride];
network5.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
network5.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
network5.num_filters = [ 16; 16; 16; 16; 16];
network5.batch_norm = [true; true; true; true; true];
network5.relu = [true; true; true; true; true];
network5.MaxEpochs = 4;
network5.network_save_name = strcat('t',sprintf('%05d',data_window),'_5x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network6 = default_network;
network6.conv_filter = [general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
network6.conv_stride = [general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride];
network6.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
network6.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
network6.num_filters = [ 16; 16; 16; 16; 16; 16];
network6.batch_norm = [true; true; true; true; true; true];
network6.relu = [true; true; true; true; true; true];
network6.MaxEpochs = 4;
network6.network_save_name = strcat('t',sprintf('%05d',data_window),'_6x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network7 = default_network;
network7.conv_filter =
[general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
network7.conv_stride =
[general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride];
network7.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
network7.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
network7.num_filters = [ 16; 16; 16; 16; 16; 16; 16];
network7.batch_norm = [true; true; true; true; true; true; true];
network7.relu = [true; true; true; true; true; true; true];
network7.MaxEpochs = 4;
network7.network_save_name = strcat('t',sprintf('%05d',data_window),'_7x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network8 = default_network;
```

```
            network8.conv_filter =
[general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
            network8.conv_stride =
[general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_s
tride];
            network8.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
            network8.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
            network8.num_filters = [ 16; 16; 16; 16; 16; 16; 16; 16];
            network8.batch_norm = [true; true; true; true; true; true; true; true];
            network8.relu = [true; true; true; true; true; true; true; true];
            network8.MaxEpochs = 4;
            network8.network_save_name = strcat('t',sprintf('%05d',data_window),'_8x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

            network9 = default_network;
            network9.conv_filter = [[4,1];[4,1]];
            network9.conv_stride = [[2,1];[2,1]];
            network9.pool_filter = [[ 2, 1 ];[ 2, 1 ]];
            network9.pool_stride = [[ 2, 1 ];[ 2, 1 ]];
            network9.num_filters = [ 16; 16; 16; 16; 16; 16; 16; 16];
            network9.batch_norm = [true; true; true; true; true; true; true; true];
            network9.relu = [true; true; true; true; true; true; true; true];
            network9.MaxEpochs = 4;
            network9.network_save_name = strcat('t',sprintf('%05d',data_window),'_8x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');


            network_queue = {network2};%,, network3;network1, network4, network5, network6, network7, network8, network9};
            i = 1;
            while i <= length(network_queue)
               temp_network = network_queue{i};
               %network = CWRU_network_commander_cross_validation(temp_network);
               network = paderborn_network_commander_v2(temp_network);
               fprintf('%f of %f network configurations trained\n', i, length(network_queue))
               i = i+1;
            end

        %%%
           end
           filter_index = filter_index + 1;

        end
        %%%

        end
```

8. CWRU_sorted_data_unifier.mat

```
        load('D:\CWRU_data\raw_data\unified_data.mat')

        %drive end ball
        drive_end_ball = unified_data(1:12,:);

        %drive end inner race
        drive_end_inner_race = unified_data(13:24,:);

        %drive end outer race
        drive_end_outer_race = unified_data(25:52,:);

        %fan end ball
        fan_end_ball = unified_data(53:64,:);

        %fan end inner race
        fan_end_inner_race = unified_data(65:76,:);

        %fan end outer
        fan_end_outer_race = unified_data(77:97,:);

        %healthy:
        healthy = unified_data(end-3:end,:);

        %save('D:\CWRU_data\raw_data\sorted_unified_data','drive_end_ball', 'drive_end_inner_race', 'drive_end_outer_race', 'fan_end_ball',
'fan_end_inner_race', 'fan_end_outer_race' , 'healthy')
        save('D:\CWRU_data\raw_data\sorted_fan_end_unified_data','fan_end_ball', 'fan_end_inner_race', 'fan_end_outer_race' , 'healthy')
        save('D:\CWRU_data\raw_data\sorted_drive_end_unified_data','drive_end_ball', 'drive_end_inner_race', 'drive_end_outer_race', 'healthy')
```

9. CWRU_spectrogram_fusion.mat

```
data_path = 'D:\CWRU_data\augmented_data_1\';
fault_list = ["drive_end_ball",...  1
   "drive_end_inner_race",...      2
   "drive_end_outer_race",...      3
   "fan_end_ball",...              4
   "fan_end_inner_race",...        5
   "fan_end_outer_race",...        6
   "healthy"];  %                  7

%%%%    Inputs    %%%%
fault_id = fault_list(4);
noise_id = 'snr_0';
%i = 2000;                      %1 <= i <= ~2000
i = i+1;

clean_data_path = strcat(data_path,'snr_inf\',fault_id,'\');
noisy_data_path = strcat(data_path,noise_id,'\',fault_id,'\');

imds_clean = imageDatastore(clean_data_path,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames');
imds_noisy = imageDatastore(noisy_data_path,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames');

s_clean_ch1 = readFcn3_ch1(string(imds_clean.Files(i)));
s_clean_ch2 = readFcn3_ch2(string(imds_clean.Files(i)));

s_noisy_ch1 = readFcn3_ch1(string(imds_noisy.Files(i)));
s_noisy_ch2 = readFcn3_ch2(string(imds_noisy.Files(i)));

s_clean_root_product = sqrt(s_clean_ch1.*s_clean_ch2);
s_noisy_root_product = sqrt(s_noisy_ch1.*s_noisy_ch2);

subplot(2,3,1)
imagesc(s_clean_ch1)
xlabel({"Raw CH1",strcat('Entropy = ', string(entropy(s_clean_ch1)))})
subplot(2,3,2)
imagesc(s_clean_ch2)
title(regexprep(fault_id, '_', ' '));
xlabel({"Raw CH2",strcat('Entropy = ', string(entropy(s_clean_ch2)))})
subplot(2,3,3)
imagesc(s_clean_root_product)
xlabel({"root-product of CH1 & CH2",strcat('Entropy = ', string(entropy(s_clean_root_product)))})

subplot(2,3,4)
imagesc(s_noisy_ch1)
xlabel({"Raw CH1",strcat('Entropy = ', string(entropy(s_noisy_ch1)))})
subplot(2,3,5)
imagesc(s_noisy_ch2)
title(regexprep(noise_id, '_', ' '));
xlabel({"Raw CH2",strcat('Entropy = ', string(entropy(s_noisy_ch2)))})
subplot(2,3,6)
imagesc(s_noisy_root_product)
xlabel({"root-product of CH1 & CH2",strcat('Entropy = ', string(entropy(s_noisy_root_product)))})
```

10. CWRU_spectrogram_writer.mat

```
read_folder = 'D:\CWRU_data\raw_data';
write_folder = 'D:\CWRU_data\augmented_data_1';

folders = dir(read_folder);
folders(ismember({folders.name}, {'.', '..'})) = [];

augmentation_window = 1200;
augmentation_overlap = 600;

spectrogram_window = 80;
spectrogram_overlap = 60;

n = 1;
i = 1;
while i <= length(folders)
```

```
  read_sub_folder = strcat(read_folder,'\',folders(i).name);
  write_sub_folder = strcat(write_folder,'\',folders(i).name);
  mkdir(write_sub_folder)

  files = dir(read_sub_folder);
  files(ismember({files.name}, {'.', '..'})) = [];


  fprintf('Current folder: %s \n', read_sub_folder)
  j = 1;
  while j <= length(files)

    file = strcat(read_sub_folder,'\',files(j).name);
    fprintf('loading: %s \n',file)
    raw_de_struct = load(file,'*_DE_time');
    raw_fe_struct = load(file,'*_FE_time');

    raw_de_field = fields(raw_de_struct);
    raw_fe_field = fields(raw_fe_struct);

    raw_de = raw_de_struct.(char(raw_de_field));
    raw_fe = raw_fe_struct.(char(raw_fe_field));

    augmentation_start_pos = 1;
    augmentation_end_pos = augmentation_start_pos + augmentation_window;
    while augmentation_end_pos <= length(raw_de)

      temp_de = raw_de(augmentation_start_pos:augmentation_end_pos);
      temp_fe = raw_fe(augmentation_start_pos:augmentation_end_pos);

      s_de = abs(spectrogram(temp_de,spectrogram_window,spectrogram_overlap));
      s_fe = abs(spectrogram(temp_fe,spectrogram_window,spectrogram_overlap));

      s_fused = zeros(length(s_de(:,1)),length(s_de(1,:)),2);

      s_fused(:,:,1) = s_de;
      s_fused(:,:,2) = s_fe;

      save_file = strcat(write_sub_folder,'\',string(n),'.mat');
      save(save_file,"s_fused");
      n = n+1;

      augmentation_start_pos = augmentation_start_pos + augmentation_window - augmentation_overlap;
      augmentation_end_pos = augmentation_start_pos + augmentation_window;
    end
    j = j+1;
  end

  i = i+1;
end
```

11. CWRU_spectrogram_writer_noise.mat

```
read_folder = 'D:\CWRU_data\raw_data';
write_folder = 'D:\CWRU_data\augmented_data_1\snr_0';

snr_log = 0;

folders = dir(read_folder);
folders(ismember({folders.name}, {'.', '..'})) = [];

augmentation_window = 1200;
augmentation_overlap = 600;

spectrogram_window = 80;
spectrogram_overlap = 60;

n = 1;
i = 1;
while i <= length(folders)

  read_sub_folder = strcat(read_folder,'\',folders(i).name);
```

```
    write_sub_folder = strcat(write_folder,'\',folders(i).name);
    mkdir(write_sub_folder)

    files = dir(read_sub_folder);
    files(ismember({files.name}, {'.', '..'})) = [];


    fprintf('Current folder: %s \n', read_sub_folder)
    j = 1;
    while j <= length(files)

        file = strcat(read_sub_folder,'\',files(j).name);
        fprintf('loading: %s \n',file)
        raw_de_struct = load(file,'*_DE_time');
        raw_fe_struct = load(file,'*_FE_time');

        raw_de_field = fields(raw_de_struct);
        raw_fe_field = fields(raw_fe_struct);

        raw_de = raw_de_struct.(char(raw_de_field));
        raw_fe = raw_fe_struct.(char(raw_fe_field));

        augmentation_start_pos = 1;
        augmentation_end_pos = augmentation_start_pos + augmentation_window;
        while augmentation_end_pos <= length(raw_de)

            temp_de = raw_de(augmentation_start_pos:augmentation_end_pos);
            temp_fe = raw_fe(augmentation_start_pos:augmentation_end_pos);

            temp_de = awgn(temp_de,snr_log,'measured');
            temp_fe = awgn(temp_fe,snr_log,'measured');

            s_de = abs(spectrogram(temp_de,spectrogram_window,spectrogram_overlap));
            s_fe = abs(spectrogram(temp_fe,spectrogram_window,spectrogram_overlap));

            s_fused = zeros(length(s_de(:,1)),length(s_de(1,:)),2);

            s_fused(:,:,1) = s_de;
            s_fused(:,:,2) = s_fe;

            save_file = strcat(write_sub_folder,'\',string(n),'.mat');
            save(save_file,"s_fused");
            n = n+1;

            augmentation_start_pos = augmentation_start_pos + augmentation_window - augmentation_overlap;
            augmentation_end_pos = augmentation_start_pos + augmentation_window;
        end
        j = j+1;
    end

    i = i+1;
end
```

12. FEMTO_Spectrogram_test.mat

```
read_path = strcat('D:\FEMTOData\Complete_set\Raw_data\Bearing3_3\acc_00434.csv');
temp = load(read_path);
ch1 = temp(:,5);
ch2 = temp(:,6);

fs = 25600;
window = 50;
%f = 10;
noverlap = 35;

[s1,f1,t1] = spectrogram(ch1,window,noverlap);
[s2,f2,t2] = spectrogram(ch2,window,noverlap);

%[s2,f2,t2] = spectrogram(ch2,window,noverlap,f,fs);

s1 = abs(s1);
%file_name_1 = strcat(char(sprintf('%05d', i)), '_ch1.mat');
```

```
%sub_output_folder = strcat(output_folder,'\',string(10*round(10*((length_files - i)/length_files),1)));
%mkdir(sub_output_folder);
%write_path_1 = strcat(sub_output_folder,'\',bearing_id,'_',file_name_1);
%save(write_path_1,'s1');

 s2 = abs(s2);
% file_name_2 = strcat(char(sprintf('%05d', i)), '_ch2.mat');
% write_path_2 = strcat(output_folder,'\',bearing_id,'_',file_name_2);

%noise_amp_2 = FEMTO_awgn_test_v2(0);
%[ch1n2,ch2n2] = add_noise_2d(ch1,ch2,noise_amp_2);
ch1n2 = awgn(ch1,0,1.3373);

[s1n2,f1,t1] = spectrogram(ch1n2,window,noverlap);
%[s2n2,f2,t2] = spectrogram(ch2n2,window,noverlap);
s1n2 = abs(s1n2);

%noise_amp_3 = FEMTO_awgn_test_v2(-10);
%[ch1n3,ch2n3] = add_noise_2d(ch1,ch2,noise_amp_3);
 ch1n3 = awgn(ch1,-5,1.3373);

[s1n3,f1,t1] = spectrogram(ch1n3,window,noverlap);
%[s2n3,f2,t2] = spectrogram(ch2n3,window,noverlap);
s1n3 = abs(s1n3);

climit = [0,max(s1(:))];

subplot(1,3,1)
set(gca, 'FontName', 'Times New Roman')
imagesc(s1,climit);
title("Clean signal");
subplot(1,3,2)
set(gca, 'FontName', 'Times New Roman')
imagesc(s1n2,climit);
title('SNR_d_B = 0');
subplot(1,3,3)
set(gca, 'FontName', 'Times New Roman')
imagesc(s1n3,climit);
title('SNR_d_B = -5');

%imagesc([s1;s2]);

%s1 = s2;
%save(write_path_2,'s1');


subplot(1,3,1)
set(gca, 'FontName', 'Times New Roman')
subplot(1,3,2)
set(gca, 'FontName', 'Times New Roman')
subplot(1,3,3)
set(gca, 'FontName', 'Times New Roman')
```

13. FEMTO_mse_calculator.mat

```
save_figures = false;
%snr_list = ["snr_inf","snr_20","snr_10","snr_5","snr_0","snr_-5"];

snr_list = ["snr_inf"];

load net3D_reg_v09.mat
regnet_3d = net3D_reg_v09;

load net2D_reg_v09_ch1.mat
regnet_ch1 = net2D_reg_v09_ch1;

load net2D_reg_v09_ch2.mat
regnet_ch2 = net2D_reg_v09_ch2;

list_bearing_id = ["bearing1_1","bearing1_2","bearing1_3",...
   "bearing1_4","bearing1_5","bearing1_6","bearing1_7",...
   "bearing2_1","bearing2_2","bearing2_3","bearing2_4",...
   "bearing2_5","bearing2_6","bearing2_7","bearing3_1",...
```

```
    "bearing3_2","bearing3_3"];

%list_bearing_id = ["bearing1_1"];

%predict_rate determines how many spectrograms to skip before reading and
%making another prediction. must be >= 1
predict_rate = 5;

for snr_label = snr_list
    fprintf('%s \n',snr_label);
    n = 1;
    prediction_table = zeros(3,17);
    for bearing_id = list_bearing_id

        %fprintf('%s , %s : ',snr_label, bearing_id);

        test_spectrogram_folder = strcat('D:\FEMTOData\Test_010\',snr_label,'\', bearing_id);
        test_images = imageDatastore(test_spectrogram_folder,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames');

        % Convert labels to categoricals
        test_images.Labels = categorical(test_images.Labels);
        test_images.ReadFcn = @readFcn1;

        %sort sequence of test images. It gets confused due to subfolder locations
        names = test_images.Files;
        numbers=cellfun(@extract_number,names);
        [~,order]=sort(numbers);
        names=names(order);

        tallied_predictions3d = [];
        tallied_predictionsch1 = [];
        tallied_predictionsch2 = [];

        i = 1;
        while i <= length(names)

            prediction3d = predict(regnet_3d, readFcn2(string(names(i))));
            tallied_predictions3d = vector_concatonator(tallied_predictions3d,prediction3d);

            predictionch1 = predict(regnet_ch1, readFcn1_ch1(string(names(i))));
            tallied_predictionsch1 = vector_concatonator(tallied_predictionsch1,predictionch1);

            predictionch2 = predict(regnet_ch2, readFcn1_ch2(string(names(i))));
            tallied_predictionsch2 = vector_concatonator(tallied_predictionsch2,predictionch2);

            %fprintf('%05d / %05d \n' , i, length(names));
            i = i+predict_rate;

        end

        expected_result = linspace(1,0,length(tallied_predictionsch2));
        expected_result = transpose(expected_result);
        mse_3d = sqrt(immse(tallied_predictions3d,expected_result));
        mse_2d_ch1 = sqrt(immse(tallied_predictionsch1,expected_result));
        mse_2d_ch2 = sqrt(immse(tallied_predictionsch2,expected_result));

        prediction_table(1,n) = mse_3d;
        prediction_table(2,n) = mse_2d_ch1;
        prediction_table(3,n) = mse_2d_ch2;

        fprintf('%f    %f    %f \n',mse_3d,mse_2d_ch1,mse_2d_ch2);
        n = n+1;
    end
end
```

14. FEMTO_noise_reg_comp.mat

```
snr_label_list = ["snr_20","snr_10","snr_5","snr_0"];
snr_val = [20,10,5,0];

%figure_file_destination = strcat('D:\FEMTOData\Test_010\net_reg_v03b\noise_comp\');
%mkdir(figure_file_destination)

load net3D_reg_v09.mat
```

```
regnet_3d = net3D_reg_v09;

load net2D_reg_v09_ch1.mat
regnet_ch1 = net2D_reg_v09_ch1;

load net2D_reg_v09_ch2.mat
regnet_ch2 = net2D_reg_v09_ch2;

bearing_id = "bearing3_3";

%list_bearing_id = ["bearing1_7"];

%predict_rate determines how many spectrograms to skip before reading and
%making another prediction. must be >= 1
predict_rate = 2;

smoothing_constant = 500;

clf
for snr_label = snr_label_list
   test_spectrogram_folder = strcat('D:\FEMTOData\Test_010\',snr_label,'\', bearing_id);
   test_images = imageDatastore(test_spectrogram_folder,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames');

   % Convert labels to categoricals
   test_images.Labels = categorical(test_images.Labels);
   test_images.ReadFcn = @readFcn1;

   %sort sequence of test images. It gets confused due to subfolder locations
   names = test_images.Files;
   numbers=cellfun(@extract_number,names);
   [~,order]=sort(numbers);
   names=names(order);

   tallied_predictions3d = [];
   tallied_predictionsch1 = [];
   tallied_predictionsch2 = [];

   i = 1;
   while i <= length(names)

      prediction3d = predict(regnet_3d, readFcn2(string(names(i))));
      tallied_predictions3d = vector_concatonator(tallied_predictions3d,prediction3d);

      predictionch1 = predict(regnet_ch1, readFcn1_ch1(string(names(i))));
      tallied_predictionsch1 = vector_concatonator(tallied_predictionsch1,predictionch1);

      predictionch2 = predict(regnet_ch2, readFcn1_ch2(string(names(i))));
      tallied_predictionsch2 = vector_concatonator(tallied_predictionsch2,predictionch2);

      fprintf('%05d / %05d \n' , i, length(names));
      i = i+predict_rate;

   end

   subplot(1,3,1)
   xlabel('Time (s/10)');
   ylabel('Health indicator')
   set(gca, 'FontName', 'Times New Roman')
   hold on
   smoothed_predictions3d = smooth(tallied_predictions3d,'moving',[round(smoothing_constant/predict_rate),0]);
   plot(smoothed_predictions3d);
   ylim([0,1]);
   xlim([0, length(tallied_predictions3d)]);
   xlabel('Time (s/10)');


   subplot(1,3,2)
   xlabel('Time (s/10)');
   set(gca, 'FontName', 'Times New Roman')
   hold on
   smoothed_predictionsch1 = smooth(tallied_predictionsch1,'moving',[round(smoothing_constant/predict_rate),0]);
   plot(smoothed_predictionsch1);
   ylim([0,1]);
   xlim([0, length(tallied_predictions3d)]);
```

```
    subplot(1,3,3)
    xlabel('Time (s/10)');
    hold on
    set(gca, 'FontName', 'Times New Roman')
    smoothed_predictionsch2 = smooth(tallied_predictionsch2,'moving',[round(smoothing_constant/predict_rate),0]);
    plot(smoothed_predictionsch2);
    ylim([0,1]);
    xlim([0, length(tallied_predictions3d)]);

end
legend('20_d_B','10_d_B','5_d_B','0_d_B');
```

15. FEMTO_spectrogram_progression_comparisson.mat

```
snr_label = 'snr_0';

save_figure = true;

list_bearing_id = ["bearing1_1","bearing1_2","bearing1_3",...
    "bearing1_4","bearing1_5","bearing1_6","bearing1_7",...
    "bearing2_1","bearing2_2","bearing2_3","bearing2_4",...
    "bearing2_5","bearing2_6","bearing2_7","bearing3_1",...
    "bearing3_2","bearing3_3"];

for bearing_id = list_bearing_id

    figure_file_destination = strcat('D:\FEMTOData\Test_010\spectrogram_progression\',snr_label,'\');
    mkdir(figure_file_destination)

    n = 1;
    i = 1;
    test_spectrogram_folder = strcat('D:\FEMTOData\Test_010\',snr_label,'\', bearing_id);

    test_images = imageDatastore(test_spectrogram_folder,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames');
    test_images.ReadFcn = @readFcn1_ch1;

    names = test_images.Files;
    numbers=cellfun(@extract_number,names);
    [~,order]=sort(numbers);
    names=names(order);

    step = round((length(names)-7)/7);

    spectro_max = readFcn2(string(names(1+7*step)));
    climit = [0,max(spectro_max(:))];


    while n <= 8

        spectro = readFcn2(string(names(i)));

        subplot(2,4,n)
        imagesc(spectro(:,:,1));
        title(strcat(bearing_id, ":   ", string(i), " / ", string(length(names))));

        i = i + step;
        n = n+1;
    end

    if save_figure == true
        saveas(gcf,strcat(figure_file_destination,bearing_id,'.png'))
    end

end
```

16. FEMTO_spectrogram_write_3d_regression.mat

```
%%% Inputs:

%target_snr = 8;
```

```
for target_snr = [-5]
  for bearing_id =
["bearing1_1","bearing1_2","bearing1_3","bearing1_4","bearing1_5","bearing1_6","bearing1_7","bearing2_1","bearing2_2","bearing2_3","bearing2_4","bearing2_
5","bearing2_6","bearing2_7","bearing3_1","bearing3_2","bearing3_3"]

    input_folder = strcat('D:\FEMTOData\Complete_set\Raw_data\', bearing_id);
    output_folder = strcat('D:\FEMTOData\Test_010\','snr_',string(target_snr),'\', bearing_id);
    mkdir(output_folder);

    w = warning('query','last');
    id = w.identifier;
    warning('off',id)

    %%%% Code:

    files = dir(fullfile(input_folder, 'acc*.csv'));
    file_names = strings(1,length(files));

    fs = 25600;
    window = 50;
    noverlap = 35;

    i = 1;
    length_files = length(files);

    while i <= length_files

      file_names(i) = files(i).name;

      i = i+1;
    end

    snr_net = zeros(length_files,1);

    i = 1;
    while i <= length_files

      read_path = strcat(files(i).folder,'\', files(i).name);
      temp = load(read_path);
      ch1 = temp(:,5);
      ch2 = temp(:,6);

      [ch1,ch2] = add_noise_2d_v2(ch1,ch2,target_snr);

      [s1,f1,t1] = spectrogram(ch1,window,noverlap);
      [s2,f2,t2] = spectrogram(ch2,window,noverlap);

      s1 = abs(s1);
      s2 = abs(s2);

      s = zeros(129,168,2);

      s(:,:,1) = s1;
      s(:,:,2) = s2;

      file_name_1 = strcat(char(sprintf('%05d', i)), '.mat');
      sub_output_folder = strcat(output_folder,'\',string(10*round(10*((length_files - i)/length_files),1)));
      %write_path_1 = strcat(sub_output_folder,'\',bearing_id,'_',file_name_1);
      %mkdir(sub_output_folder);
      write_path_1 = strcat(sub_output_folder,'\',bearing_id,'_',file_name_1);
      mkdir(sub_output_folder);
      save(write_path_1,'s');

      %fprintf('%05d / %05d \n' , i, length_files);

      i = i+1;
    end
  end
end
```

17. FEMTO_variance_calculator.mat

```
save_figures = false;
```

```
%snr_list = ["snr_inf","snr_20","snr_10","snr_5","snr_0","snr_-5"];

snr_list = ["snr_inf"];

load net3D_reg_v03.mat
regnet_3d = net3D_reg_v03;

load net2D_reg_v03_ch1.mat
regnet_ch1 = net2D_reg_v03_ch1;

load net2D_reg_v03_ch2.mat
regnet_ch2 = net2D_reg_v03_ch2;

list_bearing_id = ["bearing1_1","bearing1_2","bearing1_3",...
   "bearing1_4","bearing1_5","bearing1_6","bearing1_7",...
   "bearing2_1","bearing2_2","bearing2_3","bearing2_4",...
   "bearing2_5","bearing2_6","bearing2_7","bearing3_1",...
   "bearing3_2","bearing3_3"];

%list_bearing_id = ["bearing1_1"];

%predict_rate determines how many spectrograms to skip before reading and
%making another prediction. must be >= 1
predict_rate = 5;

for snr_label = snr_list
   fprintf('%s \n',snr_label);
   n = 1;
   prediction_table = zeros(3,17);
   for bearing_id = list_bearing_id

      %fprintf('%s , %s : ',snr_label, bearing_id);

      test_spectrogram_folder = strcat('D:\FEMTOData\Test_010\',snr_label,'\', bearing_id);
      test_images = imageDatastore(test_spectrogram_folder,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames');

      % Convert labels to categoricals
      test_images.Labels = categorical(test_images.Labels);
      test_images.ReadFcn = @readFcn1;

      %sort sequence of test images. It gets confused due to subfolder locations
      names = test_images.Files;
      numbers=cellfun(@extract_number,names);
      [~,order]=sort(numbers);
      names=names(order);

      tallied_predictions3d = [];
      tallied_predictionsch1 = [];
      tallied_predictionsch2 = [];

      i = 1;
      while i <= length(names)

         prediction3d = predict(regnet_3d, readFcn2(string(names(i))));
         tallied_predictions3d = vector_concatonator(tallied_predictions3d,prediction3d);

         predictionch1 = predict(regnet_ch1, readFcn1_ch1(string(names(i))));
         tallied_predictionsch1 = vector_concatonator(tallied_predictionsch1,predictionch1);

         predictionch2 = predict(regnet_ch2, readFcn1_ch2(string(names(i))));
         tallied_predictionsch2 = vector_concatonator(tallied_predictionsch2,predictionch2);

         %fprintf('%05d / %05d \n' , i, length(names));
         i = i+predict_rate;

      end

      expected_result = linspace(1,0,length(tallied_predictionsch2));
      expected_result = transpose(expected_result);
      var_3d = var(tallied_predictions3d,expected_result);
      var_2d_ch1 = var(tallied_predictionsch1,expected_result);
      var_2d_ch2 = var(tallied_predictionsch2,expected_result);

      fprintf('%f    %f    %f \n',var_3d,var_2d_ch1,var_2d_ch2);
      n = n+1;
```

```
    end
end
```

18. P3_CWRU_network_commander.mat

```
function network = P3_CWRU_network_commander(network)

    %%% Rename some variables to make it simpler to call on them later
    save_network = network.save_network;
    keep_training_gui_open = network.keep_training_gui_open;
    network_save_path = network.network_save_path;
    network_save_name = network.network_save_name;
    normalization = network.normalization;
    conv_filter = network.conv_filter;
    conv_stride = network.conv_stride;
    pool_filter = network.pool_filter;
    pool_stride = network.pool_stride;
    num_filters = network.num_filters;
    batch_norm = network.batch_norm;
    relu = network.relu;
    dropout = network.dropout;
    MaxEpochs = network.MaxEpochs;
    training_dataset = network.training_dataset;
    validation_dataset = network.validation_dataset;
    data_window = network.data_window;
    data_aug_stride = network.data_aug_stride;
    data_ch = network.data_ch;
    MiniBatchSize = network.MiniBatchSize;
    preprocessor = network.preprocessor;
    spectrogram_window = network.spectrogram_window;
    ValidationFrequency = network.ValidationFrequency;
    use_custom_architecture = network.use_custom_architecture;
    custom_architecture = network.custom_architecture;
    type = network.type;
    data_aug_stride_validation = network.data_aug_stride_validation;

    %Import, shuffle, and perform augmentation on the dataset
    train_data = load(training_dataset);
    train_data = train_data.unified_data;
    test_data = load(validation_dataset);
    test_data = test_data.unified_data;
    %data_cats = categories(categorical(train_data(:,2)));

    train_data = CWRU_data_augmentation_from_uni(train_data,data_window,data_aug_stride,data_ch);
    test_data = CWRU_data_augmentation_from_uni(test_data,data_window,data_aug_stride_validation,data_ch);

    train_labels = categorical(train_data(:,2));
    test_labels = categorical(test_data(:,2));

    if data_ch == 1 || data_ch == 2
        depth = 1;
        train_data = train_data(:,data_ch);
        test_data = test_data(:,data_ch);
    else
        depth = 2;
    end



    new_train_data = zeros(length(cell2mat(train_data(1))),1,depth,length(train_data));
    new_test_data = zeros(length(cell2mat(test_data(1))),1,depth,length(test_data));

    row_index = 1;
    while row_index <= length(train_data)
      new_train_data(:,:,:,row_index) = cell2mat(train_data(row_index));
      row_index = row_index + 1;
    end
    train_data = new_train_data;

    row_index = 1;
    while row_index <= length(test_data)
      new_test_data(:,:,:,row_index) = cell2mat(test_data(row_index));
      row_index = row_index + 1;
    end
```

```
    test_data = new_test_data;

    %validation_accuracy_tally = zeros(k, 1);
    %train_accuracy_tally = zeros(k, 1);
    %time_tally = zeros(k, 1);
    %training_size_tally = zeros(k, 1);
    %network_taly = [];

    if ~strcmp(preprocessor,'none')
       train_data = generalized_preprocessor(train_data,preprocessor,spectrogram_window);
       test_data = generalized_preprocessor(test_data,preprocessor,spectrogram_window);
    end

    train_dim = size(train_data);
    idx_train = randperm(train_dim(end));
    train_data = train_data(:,:,:,idx_train);
    train_labels = train_labels(idx_train);

    test_dim = size(test_data);
    idx_test = randperm(test_dim(end));
    test_data = test_data(:,:,:,idx_test);
    test_labels = test_labels(idx_test);

    if type == "SVM"
       train_data_size = size(train_data);
       new_train_data = zeros(train_data_size(1),train_data_size(end));
       test_data_size = size(test_data);
       new_test_data = zeros(test_data_size(1),test_data_size(end));
       train_data = transpose(new_train_data);
       test_data = transpose(new_test_data);

    end

    training_size = length(train_data);
    % This parameter determines the size of the output of the final FC
    % layer. It has to be defined after the data is imported because it
    % determines the required size by reading the number of output
    % categories in the label space.
    output_size = length(countcats(train_labels));


    %%% Training options
    options = trainingOptions('adam', ...
    'InitialLearnRate',0.0005, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.92, ...
    'LearnRateDropPeriod',2, ...
    'Verbose',false, ...
    'Plots','training-progress', ...
    'ValidationData',{test_data,test_labels}, ...
    'ValidationFrequency', ValidationFrequency, ...
    'MaxEpochs',MaxEpochs, ...
    'MiniBatchSize',MiniBatchSize,...
    'ExecutionEnvironment','gpu');

    [temp_trained_net,temp_info,temp_time] = network_trainer(train_data, train_labels, output_size, batch_norm, relu, dropout, conv_filter, conv_stride,
num_filters, pool_filter, pool_stride, options,use_custom_architecture,custom_architecture, type);

    if type == "SVM"
        predicted_labels = predict(temp_trained_net,test_data);
        network.confusion_mat_validation = confusionmat(test_labels,categorical(predicted_labels))

    elseif ~strcmp(temp_info,'Invalid network')

    network.categories = categories(test_labels);

    network.true_test_labels = test_labels;
    predicted_test_labels_idx = predict(temp_trained_net,test_data);
    predicted_test_labels = strings(size(test_labels));
    prediction_index = 1;
    while prediction_index <= length(test_labels)
       [maximum,index] = max(predicted_test_labels_idx(prediction_index,:));
       predicted_test_labels(prediction_index) = network.categories{index};
       prediction_index = prediction_index+1;
    end
```

```
        network.predicted_test_labels = categorical(predicted_test_labels);
        network.confusion_mat_validation = confusionmat(test_labels,network.predicted_test_labels);

        network.true_train_labels = train_labels;
        predicted_train_labels_idx = predict(temp_trained_net,train_data);
        predicted_train_labels = strings(size(train_labels));
        prediction_index = 1;
        while prediction_index <= length(train_labels)
          [maximum,index] = max(predicted_train_labels_idx(prediction_index,:));
          predicted_train_labels(prediction_index) = network.categories{index};
          prediction_index = prediction_index+1;
        end
        network.predicted_train_labels = categorical(predicted_train_labels);
        network.confusion_mat_train = confusionmat(train_labels,network.predicted_train_labels);

        network.trained_network = temp_trained_net;
        network.info = temp_info;
        network.time = temp_time;
        network.training_size = training_size;

        fprintf('\n\nFinal training accuracy: %0.2f\nFinal validation accuracy: %0.2f\nInitial learn rate: %0.5f\n', temp_info.TrainingAccuracy(end),
temp_info.ValidationAccuracy(end), temp_info.BaseLearnRate(1))
        fprintf('\n\n\n\n')

        validation_accuracy = temp_info.ValidationAccuracy(end);
        train_accuracy = temp_info.TrainingAccuracy(end);
        time = temp_time;

        if keep_training_gui_open == false
          delete(findall(0));
        end
        else
          fprintf("Training failure.")
      end

      if save_network == true && ~strcmp(temp_info,'Invalid network')
        save(strcat(network_save_path,'\',network_save_name,'.mat'),"network");
        diary_filename = strcat(network_save_path,'\',network_save_name,'_summary.txt');
        diary(diary_filename);
        fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
        fprintf('\n\n')
        temp_trained_net.Layers
        fprintf('\n\n')
        fprintf('Validation accuracy:');
        fprintf('  %f  ', validation_accuracy);
        fprintf('\nValidation mean = %f\n',mean(validation_accuracy))
        fprintf('Validation variance = %f\n\n',var(validation_accuracy))
        fprintf('Train accuracy:');
        fprintf('  %f  ', train_accuracy);
        fprintf('\nTraining mean = %f\n',mean(train_accuracy))
        fprintf('Training ariance = %f\n\n',var(train_accuracy))
        fprintf('Training time:');
        fprintf('  %f  ', time);
        fprintf('\nMean = %f\n',mean(time))
        fprintf('Training samples used:');
        fprintf('  %f  ', training_size);
        fprintf('\nMean = %f\n',mean(training_size))
        diary
      end

      if strcmp(temp_info,'Invalid network')
        save(strcat(network_save_path,'\invalid_',network_save_name,'.mat'),"network");
        diary_filename = strcat(network_save_path,'\invalid_',network_save_name,'_summary.txt');
        diary(diary_filename);
        fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
        fprintf('\n\n')
        temp_trained_net.Layers
        diary
      end
      end
```

19. P3_CWRU_network_commander_cross_validation.mat

```
      function network = P3_CWRU_network_commander_cross_validation(network)
```

```
%%% Rename some variables to make it simpler to call on them later
save_network = network.save_network;
keep_training_gui_open = network.keep_training_gui_open;
network_save_path = network.network_save_path;
network_save_name = network.network_save_name;
normalization = network.normalization;
conv_filter = network.conv_filter;
conv_stride = network.conv_stride;
pool_filter = network.pool_filter;
pool_stride = network.pool_stride;
num_filters = network.num_filters;
batch_norm = network.batch_norm;
relu = network.relu;
dropout = network.dropout;
k = network.k;
MaxEpochs = network.MaxEpochs;
dataset = network.Dataset;
data_window = network.data_window;
data_aug_stride = network.data_aug_stride;
data_ch = network.data_ch;
MiniBatchSize = network.MiniBatchSize;
preprocessor = network.preprocessor;
spectrogram_window = network.spectrogram_window;
scalogram_bins = network.scalogram_bins;

%Import, shuffle, and perform augmentation on the dataset
data = load(dataset);
data_cats = string(cell2mat(data.unified_data(:,2)));

for cat_index = 1:length(data_cats)
   %fprintf(char(data_cats(cat_index)))
   current_data = cell2mat(data.unified_data(cat_index,1));
   dimensions = size(current_data);
   ix = randperm(dimensions(1));
   new_data.(char(data_cats(cat_index))) = current_data(ix,:);
   %call augmentation_funciton
   %data.(char(data_cats(cat_index))) = CWRU_data_augmentation_from_uni(current_data,data_window,data_aug_stride,data_ch);
end
data = new_data;

%n = current iteration, 1 <= n <= k
% Initialize a bunch of values to store results over the folds
n = 1;
validation_accuracy_tally = zeros(k, 1);
train_accuracy_tally = zeros(k, 1);
time_tally = zeros(k, 1);
training_size_tally = zeros(k, 1);
%network_taly = [];

while n <= k

   %create empty train and validation data matricies
   cell_train_data = [];
   cell_validation_data = [];

   train_labels = [];
   validation_labels = [];

   %read each category one at a time to perform seperation of
   %different categories between training and validatiaon
   %indepentantly, ensuring that equal numbers of examples from each
   %cat make their way into validation vs training (previous problems
   %included instances where a given cat had no training examples due
   %to fully randomized division)
   for cat_index = 1:length(data_cats)

      %temporarily hang onto data from present category
      current_data = data.(char(data_cats(cat_index)));

      subset_size = floor(length(current_data(:,1))/k);

      current_train_data = current_data(1+((n-1)*subset_size):((n)*subset_size),:);
      %train_labels = unified_data(1+((n-1)*subset_size):((n)*subset_size));
```

```
    current_validation_data = current_data;
    %validation_labels = master_labels;
    current_validation_data(1+((n-1)*subset_size):((n)*subset_size),:) = [];
    %validation_labels(((1+(n-1)*subset_size):((n)*subset_size))) = [];

    %[train_data,train_labels] = CWRU_data_augmentation_from_uni(train_data,data_window,data_aug_stride,data_ch);
    %[validation_data,validation_labels] = CWRU_data_augmentation_from_uni(validation_data,data_window,data_aug_stride,data_ch);

    current_train_data = CWRU_data_augmentation_from_uni(current_train_data,data_window,data_aug_stride,data_ch);
    current_validation_data = CWRU_data_augmentation_from_uni(current_validation_data,data_window,data_aug_stride,data_ch);

    current_train_labels = categorical(current_train_data(:,2));
    current_validation_labels = categorical(current_validation_data(:,2));

    current_train_data = current_train_data(:,1);
    current_validation_data = current_validation_data(:,1);

    cell_train_data = [cell_train_data ; current_train_data];
    cell_validation_data = [cell_validation_data ; current_validation_data];

    train_labels = [train_labels ; current_train_labels];
    validation_labels = [validation_labels ; current_validation_labels];
  end

  train_data = [];
  validation_data = [];
  for train_data_row_index = 1:length(cell_train_data)
    train_data(:,:,:,train_data_row_index) = cell_train_data{train_data_row_index};
  end

  for validation_data_row_index = 1:length(cell_validation_data)
    validation_data(:,:,:,validation_data_row_index) = cell_validation_data{validation_data_row_index};
  end

  if ~strcmp(preprocessor,'none')
    train_data = generalized_preprocessor(train_data,preprocessor,spectrogram_window,scalogram_bins);
    validation_data = generalized_preprocessor(validation_data,preprocessor,spectrogram_window,scalogram_bins);
  end

  train_dim = size(train_data);
  idx_train = randperm(train_dim(end));
  train_data = train_data(:,:,:,idx_train);
  train_labels = train_labels(idx_train);

  validation_dim = size(validation_data);
  idx_validation = randperm(validation_dim(end));
  validation_data = validation_data(:,:,:,idx_validation);
  validation_labels = validation_labels(idx_validation);

  training_size_tally(n) = length(train_data);
  % This parameter determines the size of the output of the final FC
  % layer. It has to be defined after the data is imported because it
  % determines the required size by reading the number of output
  % categories in the label space.
  output_size = length(countcats(train_labels));


  %%% Training options
  options = trainingOptions('adam', ...
  'InitialLearnRate',0.001, ...
  'Verbose',false, ...
  'Plots','training-progress', ...
  'ValidationData',{validation_data,validation_labels}, ...
  'ValidationFrequency', floor(2*length(validation_labels)/k/MiniBatchSize), ...
  'MaxEpochs',MaxEpochs, ...
  'MiniBatchSize',MiniBatchSize,...
  'ExecutionEnvironment','gpu');

  [temp_trained_net,temp_info,temp_time] = network_trainer(train_data, train_labels, output_size, batch_norm, relu, dropout, conv_filter, conv_stride,
num_filters, pool_filter, pool_stride, options);


  if ~strcmp(temp_info,'Invalid network')

    network.categories = categories(validation_labels);
```

```
            network.true_validation_labels{n} = validation_labels;
            predicted_validation_labels_idx{n} = predict(temp_trained_net,validation_data);
            predicted_validation_labels = strings(size(validation_labels));
            prediction_index = 1;
            while prediction_index <= length(validation_labels)
               [maximum,index] = max(predicted_validation_labels_idx{n}(prediction_index,:));
               predicted_validation_labels(prediction_index) = network.categories{index};
               prediction_index = prediction_index+1;
            end
            network.predicted_validation_labels{n} = categorical(predicted_validation_labels);
            network.confusion_mat_validation{n} = confusionmat(validation_labels,network.predicted_validation_labels{n});


            network.true_train_labels{n} = train_labels;
            predicted_train_labels_idx{n} = predict(temp_trained_net,train_data);
            predicted_train_labels = strings(size(train_labels));
            prediction_index = 1;
            while prediction_index <= length(train_labels)
               [maximum,index] = max(predicted_train_labels_idx{n}(prediction_index,:));
               predicted_train_labels(prediction_index) = network.categories{index};
               prediction_index = prediction_index+1;
            end
            network.predicted_train_labels{n} = categorical(predicted_train_labels);
            network.confusion_mat_train{n} = confusionmat(train_labels,network.predicted_train_labels{n});


            network.trained_network{n} = temp_trained_net;
            network.info{n} = temp_info;
            network.time(n) = temp_time;
            network.training_size_tally = training_size_tally;

            fprintf('\n\nFinal training accuracy: %0.2f\nFinal validation accuracy: %0.2f\nInitial learn rate: %0.5f\n', temp_info.TrainingAccuracy(end),
temp_info.ValidationAccuracy(end), temp_info.BaseLearnRate(1))
            fprintf('\n\n\n\n')

            validation_accuracy_tally(n) = temp_info.ValidationAccuracy(end);
            train_accuracy_tally(n) = temp_info.TrainingAccuracy(end);
            time_tally(n) = temp_time;

            if keep_training_gui_open == false
               delete(findall(0));
            end
         else
            n = k;
         end
        n = n+1;
      end

      if save_network == true && ~strcmp(temp_info,'Invalid network')
         save(strcat(network_save_path,'\',network_save_name,'.mat'),"network");
         diary_filename = strcat(network_save_path,'\',network_save_name,'_summary.txt');
         diary(diary_filename);
         fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
         fprintf('\n\n')
         temp_trained_net.Layers
         fprintf('\n\n')
         fprintf('Validation accuracy:');
         fprintf('  %f  ', validation_accuracy_tally);
         fprintf('\nValidation mean = %f\n',mean(validation_accuracy_tally))
         fprintf('Validation variance = %f\n\n',var(validation_accuracy_tally))
         fprintf('Train accuracy:');
         fprintf('  %f  ', train_accuracy_tally);
         fprintf('\nTraining mean = %f\n',mean(train_accuracy_tally))
         fprintf('Training ariance = %f\n\n',var(train_accuracy_tally))
         fprintf('Training time:');
         fprintf('  %f  ', time_tally);
         fprintf('\nMean = %f\n',mean(time_tally))
         fprintf('Training samples used:');
         fprintf('  %f  ', training_size_tally);
         fprintf('\nMean = %f\n',mean(training_size_tally))
         diary
      end
      if strcmp(temp_info,'Invalid network')
         save(strcat(network_save_path,'\invalid_',network_save_name,'.mat'),"network");
```

```
    diary_filename = strcat(network_save_path,'\invalid_',network_save_name,'_summary.txt');
    diary(diary_filename);
    fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
    fprintf('\n\n')
    temp_trained_net.Layers
    diary
  end
end
```

20. P3_CWRU_network_queuer.mat

```
% Commander parameters
%test_case_number = "HP_2_1";
for k_iteration = [1]

%for test_case_number = ["HP_1_2", "HP_1_3", "HP_2_1", "HP_2_3", "HP_3_1", "HP_3_2"]
%for test_case_number = ["D_E", "D_F", "E_D", "E_F", "F_D", "F_E"]
for test_case_number = "Padderborn"

  if test_case_number == "1"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\Test_case_1\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\Test_case_1\Validation\unified_data.mat';
  elseif test_case_number == "2"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\Test_case_2\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\Test_case_2\Validation\unified_data.mat';
  elseif test_case_number == "3"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\Test_case_3\Training\first_half_unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\Test_case_3\Validation\second_half_unified_data.mat';
  elseif test_case_number == "Padderborn"
    default_network.training_dataset = 'D:\Paderborn_data\sorted_data_2\training\training_data_3.mat';
    default_network.validation_dataset = 'D:\Paderborn_data\sorted_data_2\testing\testing_data_3.mat';
  elseif test_case_number == "HP_1_2"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\HP_1_2\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\HP_1_2\Validation\unified_data.mat';
  elseif test_case_number == "HP_1_3"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\HP_1_3\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\HP_1_3\Validation\unified_data.mat';
  elseif test_case_number == "HP_2_1"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\HP_2_1\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\HP_2_1\Validation\unified_data.mat';
  elseif test_case_number == "HP_2_3"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\HP_2_3\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\HP_2_3\Validation\unified_data.mat';
  elseif test_case_number == "HP_3_1"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\HP_3_1\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\HP_3_1\Validation\unified_data.mat';
  elseif test_case_number == "HP_3_2"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\HP_3_2\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\HP_3_2\Validation\unified_data.mat';

  elseif test_case_number == "D_E"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\D_E\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\D_E\Validation\unified_data.mat';
  elseif test_case_number == "D_F"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\D_F\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\D_F\Validation\unified_data.mat';
  elseif test_case_number == "E_D"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\E_D\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\E_D\Validation\unified_data.mat';
  elseif test_case_number == "E_F"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\E_F\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\E_F\Validation\unified_data.mat';
  elseif test_case_number == "F_D"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\F_D\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\F_D\Validation\unified_data.mat';
  elseif test_case_number == "F_E"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\F_E\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\F_E\Validation\unified_data.mat';

  elseif test_case_number == "EF_D"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\EF_D\Training\unified_data.mat';
    default_network.validation_dataset = 'D:\CWRU_data\Paper_3\EF_D\Validation\unified_data.mat';
  elseif test_case_number == "ED_F"
    default_network.training_dataset = 'D:\CWRU_data\Paper_3\ED_F\Training\unified_data.mat';
```

```
   default_network.validation_dataset = 'D:\CWRU_data\Paper_3\ED_F\Validation\unified_data.mat';
elseif test_case_number == "EF_DD"
   default_network.training_dataset = 'D:\CWRU_data\Paper_3\EF_D\Training\unified_data.mat';
   default_network.validation_dataset = 'D:\CWRU_data\Paper_3\EF_D\Validation\unified_data.mat';
end

default_network.save_network = true;
default_network.keep_training_gui_open = false;
default_network.network_save_path = strcat('D:\CWRU_data\Paper_3\Test_case_',string(test_case_number),'\results');

default_network.network_save_name = 'ADCN_v3.mat';

default_network.data_ch = 1;

default_network.preprocessor = "fourier_spec";   %defined in generalized_preprocessor.m, values are 'none' 'fourier_spec' 'env_spec' 'spectrogram'
default_network.data_window = 5118*2;
stride_factor = 0.8;
default_network.data_aug_stride = default_network.data_window*stride_factor;
network4.data_aug_stride_validation = default_network.data_aug_stride;
default_network.normalization = 'zerocenter';

%%% Specify details for each conv. layer of the network. Each column
%%% represents the parameters for a layer, thus each must have the same
%%% number of columns. 1D should be represented as [X, 1], 2D can be
%%% represented as [X1, X2] and 3D can be represented as [X1, X2, X3]
default_network.conv_filter = [[ 3, 1 ];[ 3, 1 ]];
default_network.conv_stride = [[ 1, 1 ];[ 1, 1 ]];
default_network.pool_filter = [[ 2, 1 ];[ 2, 1 ]];
default_network.pool_stride = [[ 2, 1 ];[ 2, 1 ]];
default_network.num_filters = [ 16; 16];
default_network.batch_norm = [true; true];
default_network.relu = [true; true];
default_network.dropout = [false; false];
default_network.MaxEpochs = 50;
default_network.ValidationFrequency = 1000;
default_network.MiniBatchSize = 32;
default_network.spectrogram_window = 104;
default_network.use_custom_architecture = false;
default_network.custom_architecture = [];
default_network.type = "CNN";

% To use the above parameters as the default, create new networks and
% overwrite the variables which are to change in each
network1 = default_network;
%network1.conv_filter = [general_conv_filter];
%network1.conv_stride = [general_conv_stride];
%network1.pool_filter = [ 2, 2 ];
%network1.pool_stride = [ 1, 1 ];
%network1.num_filters = 16;
%network1.batch_norm = true;
%network1.relu = true;
%network1.dropout = true;

network2 = default_network;
network2.use_custom_architecture = true;
network2.MaxEpochs = 3;
network2.data_window = 5118*2;
network2.preprocessor = "fourier_spec";
stride_factor = 0.03;
network2.data_aug_stride = network2.data_window*stride_factor;
network2.data_aug_stride_validation = network2.data_window;
network2.custom_architecture = 'D:\CWRU_data\Paper_3\custom_architectures\ACDIN_v3_7out.mat';
network2.network_save_path = strcat('D:\CWRU_data\Paper_3\ACDIN_v2_results\fourier');
mkdir(network2.network_save_path);
network2.network_save_name = strcat('ACDIN_',network2.preprocessor,"_",test_case_number,"_",string(k_iteration),".mat");
%save('D:\CWRU_data\Paper_3\custom_architectures\ACDIN_v3_3out.mat',"lgraph_4")

network3 = default_network;
network3.type = "SVM";
network3.preprocessor =  "none";

network4 = default_network;
network4.use_custom_architecture = true;
network4.data_window = 2048*2;
network4.preprocessor = "fourier_spec";
```

```
    network4.MaxEpochs = 4;
    network4.ValidationFrequency = 1000;
    network4.custom_architecture = 'D:\CWRU_data\Paper_3\custom_architectures\WDCNN_v1_7out.mat';
    network4.network_save_path = strcat('D:\CWRU_data\Paper_3\WDCNN_v3_results\fourier');
    stride_factor = 0.03;
    network4.data_aug_stride = network4.data_window*stride_factor;
    network4.data_aug_stride_validation = network4.data_window;
    mkdir(network4.network_save_path);
    network4.network_save_name = strcat('WDCNN_',network4.preprocessor,"_",test_case_number,"_",string(k_iteration),".mat");
    %save('D:\CWRU_data\Paper_3\custom_architectures\WDCNN_v1_7out.mat',"layers_3")


    network5 = default_network;
    network5.MiniBatchSize = 8;
    network5.use_custom_architecture = true;
    network5.data_window = 11500;
    network5.preprocessor = "2D_spectrogram";
    network5.MaxEpochs = 4;
    network5.ValidationFrequency = 1000;
    network5.custom_architecture = 'D:\CWRU_data\Paper_3\custom_architectures\alexnet_v1_7out.mat';
    network5.network_save_path = strcat('D:\CWRU_data\Paper_3\Alexnet_v1_\spectrogram');
    stride_factor = 0.03;
    network5.data_aug_stride = network5.data_window*stride_factor;
    network5.data_aug_stride_validation = network5.data_window;
    mkdir(network5.network_save_path);
    network5.network_save_name = strcat('alexnet_',network5.preprocessor,"_",test_case_number,"_",string(k_iteration),".mat");
    network5.data_ch = 3;
    %save('D:\CWRU_data\Paper_3\custom_architectures\alexnet_v1_7out.mat',"layers_1")

    network6 = default_network;
    network6.MiniBatchSize = 6;
    network6.use_custom_architecture = true;
    network6.data_window = 11500;
    network6.preprocessor = "2D_spectrogram_v2";
    network6.MaxEpochs = 4;
    network6.ValidationFrequency = 1000;
    network6.custom_architecture = 'D:\CWRU_data\Paper_3\custom_architectures\resnet_v1_7out.mat';
    network6.network_save_path = strcat('D:\CWRU_data\Paper_3\padderborn\resnet_v2\spectrogram');
    stride_factor = 0.03;
    if test_case_number == "Padderborn"
       stride_factor = 0.5;
    end
    network6.data_aug_stride = network6.data_window*stride_factor;
    network6.data_aug_stride_validation = network6.data_window;
    mkdir(network6.network_save_path);
    network6.network_save_name = strcat('alexnet_',network6.preprocessor,"_",test_case_number,"_",string(k_iteration),".mat");
    network6.data_ch = 3;
    %save('D:\CWRU_data\Paper_3\custom_architectures\resnet_v1_7out.mat',"lgraph_1")

    mkdir(default_network.network_save_path)

    network_queue = {network6};

    i = 1;
    while i <= length(network_queue)
       temp_network = network_queue{i};
       network = P3_CWRU_network_commander(temp_network);
       fprintf('%f of %f network configurations trained\n', i, length(network_queue))
       i = i+1;
    end

end

end
```

21. PHM2009_data_augmentation_from_uni.mat

```
function augmented_data = PHM2009_data_augmentation_from_uni(unified_data,data_window,data_aug_stride,data_ch)

augmented_data = cell(1,2);

unified_row_index = 1;

while unified_row_index <= length(unified_data(:,1))
```

```
    %augmented_row = cell(1,2);
    current_label = char(unified_data{unified_row_index,2});
    raw_row_signal = cell2mat(unified_data(unified_row_index,1));
    raw_duration = length(raw_row_signal);

    starting_index = 1;
    while starting_index < raw_duration - data_window


        if data_ch == 1 || data_ch == 2
            current_segment = raw_row_signal(starting_index:starting_index+data_window-1,:,data_ch);
        else
            current_segment = raw_row_signal(starting_index:starting_index+data_window-1,:,:);
        end


        new_row = [{current_segment},{current_label}];

        augmented_data = [augmented_data ; new_row];

         starting_index = starting_index + data_aug_stride;
    end
    unified_row_index = unified_row_index + 1;
end

augmented_data = augmented_data(2:end,:);
%{
if data_ch == 1 || data_ch == 2
    %master_data = zeros(length(augmented_data{1,1}),1,1,length(augmented_data));
    %master_labels = strings(1,length(augmented_data));
    j = 1;

    while j <= length(augmented_data)

        augmented_data(j,1) = {augmented_data{j,1}(:,:,data_ch)};

        %both_channels_data = augmented_data{j,1};
        %master_data(:,:,:,j) = both_channels_data(:,:,data_ch,:);
        %master_labels(j) = augmented_data{j,2};

        j = j+1;
    end
end

%{
if data_ch == 3

    %master_data = zeros(length(augmented_data{1,1}),1,2,length(augmented_data));
    %master_labels = strings(1,length(augmented_data));
    j = 1;

    while j <= length(augmented_data)

        master_data(:,:,:,j) = augmented_data{j,1};
        master_labels(j) = augmented_data{j,2};

        j = j+1;
    end
    %}
end

%master_labels = categorical(master_labels);

%i = 1;
%while i <= length(labels)

%    new_labels(i) = labels{i};
%    i = i+1;
    %}
%end
end
```

22. PHM2009_data_unifier.mat

```
% Specify the folder where the files.
myFolder = 'D:\2009PHM_data\labeled_data\gear_data';

unified_data = {};

% Get a list of all files in the folder with the desired file name pattern.
filePattern = fullfile(myFolder, '*.mat');
theFiles = dir(filePattern);

master_data = cell(length(theFiles),2);
master_labels = cell(length(theFiles),1);
master_index = 1;

for k = 1 : length(theFiles)
   baseFileName = theFiles(k).name;
   fullFileName = fullfile(myFolder, baseFileName);
   %fprintf(1, 'Now reading %s\n', fullFileName);
   %theFolders(k) = get_folder_name(fullFileName);

   raw_measurement = load(fullFileName);
   raw_struct = load(fullFileName,'*');
   raw_field = fields(raw_struct);
   raw_data = raw_struct.(char(raw_field));

   %drop the tachometer reading, add a 1-tall height dimension
   new_raw_data = zeros(length(raw_data(:,1)),1,2);
   new_raw_data(:,1,:) = raw_data(:,1:2);

   raw_data = new_raw_data;

   master_data(k,1) = {raw_data};
   master_data(k,2) = {baseFileName(1:5)};

end

unified_data = [unified_data; master_data];
%sub_folder_index = sub_folder_index + 1;

save('D:\2009PHM_data\labeled_data\gear_data\unified_data.mat','unified_data')

function folder_name = get_folder_name(fullFileName)
   index_1 = 1;
   index_2 = 1;
   j = 1;
   while j <= length(fullFileName)-1
      temp_char = fullFileName(length(fullFileName)-j);
      if temp_char == '\'
         if index_1 == 1
            index_1 = j;
         elseif (index_1 ~= 1) && (index_2 == 1)
               index_2 = j;
         end
      end
      j = j + 1;
   end
   folder_name = fullFileName(length(fullFileName)-index_2+1:length(fullFileName)-index_1-1);
end
```

23. PHM2009_network_commander_cross_validation.mat

```
function network = CWRU_network_commander_cross_validation(network)

   %%% Rename some variables to make it simpler to call on them later
   save_network = network.save_network;
   keep_training_gui_open = network.keep_training_gui_open;
   network_save_path = network.network_save_path;
   network_save_name = network.network_save_name;
   normalization = network.normalization;
   conv_filter = network.conv_filter;
   conv_stride = network.conv_stride;
   pool_filter = network.pool_filter;
   pool_stride = network.pool_stride;
   num_filters = network.num_filters;
   batch_norm = network.batch_norm;
```

```
relu = network.relu;
dropout = network.dropout;
k = network.k;
MaxEpochs = network.MaxEpochs;
dataset = network.Dataset;
data_window = network.data_window;
data_aug_stride = network.data_aug_stride;
data_ch = network.data_ch;
MiniBatchSize = network.MiniBatchSize;
preprocessor = network.preprocessor;
spectrogram_window = network.spectrogram_window;
scalogram_bins = network.scalogram_bins;

%Import, shuffle, and perform augmentation on the dataset
data = load(dataset);
data_cats = string(cell2mat(data.unified_data(:,2)));
%{
for cat_index = 1:length(data_cats)
   %fprintf(char(data_cats(cat_index)))
   current_data = cell2mat(data.unified_data(cat_index,1));
   dimensions = size(current_data);
   ix = randperm(dimensions(1));
   new_data.(char(data_cats(cat_index))) = current_data(ix,:);
   %call augmentation_funciton
   %data.(char(data_cats(cat_index))) = CWRU_data_augmentation_from_uni(current_data,data_window,data_aug_stride,data_ch);
end
data = new_data;
%}

%n = current iteration, 1 <= n <= k
% Initialize a bunch of values to store results over the folds

n = 1;
validation_accuracy_tally = zeros(k, 1);
train_accuracy_tally = zeros(k, 1);
time_tally = zeros(k, 1);
training_size_tally = zeros(k, 1);
%network_taly = [];

while n <= k

   %create empty train and validation data matricies
   cell_train_data = [];
   cell_validation_data = [];

   train_labels = [];
   validation_labels = [];

   %read each category one at a time to perform seperation of
   %different categories between training and validatiaon
   %indepentantly, ensuring that equal numbers of examples from each
   %cat make their way into validation vs training (previous problems
   %included instances where a given cat had no training examples due
   %to fully randomized division)
   for cat_index = 1:length(data_cats)

      %temporarily hang onto data from present category
      current_data = data.unified_data(cat_index,:);
      current_data = PHM2009_data_augmentation_from_uni(current_data,data_window,data_aug_stride,data_ch);

      subset_size = floor(length(current_data(:,1))/k);

      current_train_data = current_data(1+((n-1)*subset_size):((n)*subset_size),:);
      %train_labels = unified_data(1+((n-1)*subset_size):((n)*subset_size));

      current_validation_data = current_data;
      %validation_labels = master_labels;
      current_validation_data(1+((n-1)*subset_size):((n)*subset_size),:) = [];
      %validation_labels(((1+(n-1)*subset_size):((n)*subset_size))) = [];

      %[train_data,train_labels] = CWRU_data_augmentation_from_uni(train_data,data_window,data_aug_stride,data_ch);
      %[validation_data,validation_labels] = CWRU_data_augmentation_from_uni(validation_data,data_window,data_aug_stride,data_ch);

      current_train_labels = categorical(current_train_data(:,2));
      current_validation_labels = categorical(current_validation_data(:,2));
```

```
    current_train_data = current_train_data(:,1);
    current_validation_data = current_validation_data(:,1);

    cell_train_data = [cell_train_data ; current_train_data];
    cell_validation_data = [cell_validation_data ; current_validation_data];

    train_labels = [train_labels ; current_train_labels];
    validation_labels = [validation_labels ; current_validation_labels];
  end

  train_data = [];
  validation_data = [];
  for train_data_row_index = 1:length(cell_train_data)
    train_data(:,:,:,train_data_row_index) = cell_train_data{train_data_row_index};
  end

  for validation_data_row_index = 1:length(cell_validation_data)
    validation_data(:,:,:,validation_data_row_index) = cell_validation_data{validation_data_row_index};
  end

  if ~strcmp(preprocessor,'none')
    train_data = generalized_preprocessor(train_data,preprocessor,spectrogram_window,scalogram_bins);
    validation_data = generalized_preprocessor(validation_data,preprocessor,spectrogram_window,scalogram_bins);
  end

  train_dim = size(train_data);
  idx_train = randperm(train_dim(end));
  train_data = train_data(:,:,:,idx_train);
  train_labels = train_labels(idx_train);

  validation_dim = size(validation_data);
  idx_validation = randperm(validation_dim(end));
  validation_data = validation_data(:,:,:,idx_validation);
  validation_labels = validation_labels(idx_validation);

  training_size_tally(n) = length(train_data);
  % This parameter determines the size of the output of the final FC
  % layer. It has to be defined after the data is imported because it
  % determines the required size by reading the number of output
  % categories in the label space.
  output_size = length(countcats(train_labels));


  %%% Training options
  options = trainingOptions('adam', ...
  'InitialLearnRate',0.001, ...
  'Verbose',false, ...
  'Plots','training-progress', ...
  'ValidationData',{validation_data,validation_labels}, ...
  'ValidationFrequency', floor(2*length(validation_labels)/k/MiniBatchSize), ...
  'MaxEpochs',MaxEpochs, ...
  'MiniBatchSize',MiniBatchSize,...
  'ExecutionEnvironment','gpu');

  [temp_trained_net,temp_info,temp_time] = network_trainer(train_data, train_labels, output_size, batch_norm, relu, dropout, conv_filter, conv_stride,
num_filters, pool_filter, pool_stride, options);


  if ~strcmp(temp_info,'Invalid network')

    network.categories = categories(validation_labels);

    network.true_validation_labels{n} = validation_labels;
    predicted_validation_labels_idx{n} = predict(temp_trained_net,validation_data);
    predicted_validation_labels = strings(size(validation_labels));
    prediction_index = 1;
    while prediction_index <= length(validation_labels)
      [maximum,index] = max(predicted_validation_labels_idx{n}(prediction_index,:));
      predicted_validation_labels(prediction_index) = network.categories{index};
      prediction_index = prediction_index+1;
    end
    network.predicted_validation_labels{n} = categorical(predicted_validation_labels);
    network.confusion_mat_validation{n} = confusionmat(validation_labels,network.predicted_validation_labels{n});
```

```
            network.true_train_labels{n} = train_labels;
            predicted_train_labels_idx{n} = predict(temp_trained_net,train_data);
            predicted_train_labels = strings(size(train_labels));
            prediction_index = 1;
            while prediction_index <= length(train_labels)
               [maximum,index] = max(predicted_train_labels_idx{n}(prediction_index,:));
               predicted_train_labels(prediction_index) = network.categories{index};
               prediction_index = prediction_index+1;
            end
            network.predicted_train_labels{n} = categorical(predicted_train_labels);
            network.confusion_mat_train{n} = confusionmat(train_labels,network.predicted_train_labels{n});


            network.trained_network{n} = temp_trained_net;
            network.info{n} = temp_info;
            network.time(n) = temp_time;
            network.training_size_tally = training_size_tally;

            fprintf('\n\nFinal training accuracy: %0.2f\nFinal validation accuracy: %0.2f\nInitial learn rate: %0.5f\n', temp_info.TrainingAccuracy(end),
temp_info.ValidationAccuracy(end), temp_info.BaseLearnRate(1))
            fprintf('\n\n\n\n')

            validation_accuracy_tally(n) = temp_info.ValidationAccuracy(end);
            train_accuracy_tally(n) = temp_info.TrainingAccuracy(end);
            time_tally(n) = temp_time;

            if keep_training_gui_open == false
               delete(findall(0));
            end
         else
            n = k;
         end
         n = n+1;
      end

      if save_network == true && ~strcmp(temp_info,'Invalid network')
         save(strcat(network_save_path,'\',network_save_name,'.mat'),"network");
         diary_filename = strcat(network_save_path,'\',network_save_name,'_summary.txt');
         diary(diary_filename);
         fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
         fprintf('\n\n')
         temp_trained_net.Layers
         fprintf('\n\n')
         fprintf('Validation accuracy:');
         fprintf('  %f  ', validation_accuracy_tally);
         fprintf('\nValidation mean = %f\n',mean(validation_accuracy_tally))
         fprintf('Validation variance = %f\n\n',var(validation_accuracy_tally))
         fprintf('Train accuracy:');
         fprintf('  %f  ', train_accuracy_tally);
         fprintf('\nTraining mean = %f\n',mean(train_accuracy_tally))
         fprintf('Training ariance = %f\n\n',var(train_accuracy_tally))
         fprintf('Training time:');
         fprintf('  %f  ', time_tally);
         fprintf('\nMean = %f\n',mean(time_tally))
         fprintf('Training samples used:');
         fprintf('  %f  ', training_size_tally);
         fprintf('\nMean = %f\n',mean(training_size_tally))
         diary
      end
      if strcmp(temp_info,'Invalid network')
         save(strcat(network_save_path,'\invalid_',network_save_name,'.mat'),"network");
         diary_filename = strcat(network_save_path,'\invalid_',network_save_name,'_summary.txt');
         diary(diary_filename);
         fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
         fprintf('\n\n')
         temp_trained_net.Layers
         diary
      end
   end
```

24. PHM2009_network_queuer.mat

```
   % for k-fold cross validation, specify k
```

```
default_network.k = 3;

% Commander parameters
%default_network.Dataset = 'D:\2009PHM_data\labeled_data\gear_data\augmented_raw_time\2009PHM_ch1_t2400.mat';
%default_network.Dataset = 'D:\CWRU_data\augmented_time_data\master_data_ch1_4D_1200.mat';
%default_network.Dataset = 'D:\CWRU_data\raw_data\sorted_all_unified_data.mat';
%default_network.Dataset =
["D:\Paderborn_data\sorted_data_3\testing\testing_data.mat","D:\Paderborn_data\sorted_data_3\training\training_data.mat"];
default_network.Dataset = 'D:\2009PHM_data\labeled_data\gear_data\unified_data.mat';

default_network.save_network = true;
default_network.keep_training_gui_open = false;
%default_network.network_save_path = 'D:\master_paper_2\case_study_1\part_1\ch1_dropout_16epochs_adam\4_1\';
default_network.network_save_name = '';

default_network.data_ch = 1;
%default_network.data_window = 416;

%%%


for preprocessor = ["env_spec"]  %defined in generalized_preprocessor.m, values are 'none' 'fourier_spec' 'env_spec' 'spectrogram'

default_network.preprocessor = preprocessor;


conv_filter_stride = {[2,2] , [1,1];...
            [4,4] , [1,1];... Filter size first column, stride second column.
            [6,6] , [1,1];... Trains a new batch of networks for each row
            [8,8] , [1,1];...
            };

conv_filter_stride = {[4,1] , [2,1];...
            [8,1]  , [2,1];... Filter size first column, stride second column.
            [16,1] , [4,1];... Trains a new batch of networks for each row
            [32,1] , [4,1];...
            [64,1] , [4,1];...
            [128,1], [4,1];...
            [256,1], [4,1]};



filter_index = 1;
while filter_index <= length(conv_filter_stride)

  for data_window = [9982 4992 2496 1248 832 416]
    default_network.data_window = data_window;
  %%%
    %this data window is implicated in filenames calculated below

    %default_network.preprocessor = 'none';

    general_conv_filter = conv_filter_stride{filter_index,1};
    general_conv_stride = conv_filter_stride{filter_index,2};
    default_network.network_save_path =
strcat('D:\master_paper_2\case_study_3\all_cases\',default_network.preprocessor,'v3\',string(general_conv_filter(1)),'_',string(general_conv_filter(2)));

    default_network.data_aug_stride = data_window/4;
    %default_network.data_aug_stride = data_window;
    default_network.normalization = 'zerocenter';
    %In the version of matlab in which this code was written, only
    %'zerocenter' and 'none' are availible as normalization methods. Future
    %versions support 'zscore' which could be better. I may circle back
    %and write my own version of these functions since I believe
    %they're very simple.

    %%% Specify details for each conv. layer of the network. Each column
    %%% represents the parameters for a layer, thus each must have the same
    %%% number of columns. 1D should be represented as [X, 1], 2D can be
    %%% represented as [X1, X2] and 3D can be represented as [X1, X2, X3]
    default_network.conv_filter = [general_conv_filter;general_conv_filter];
    default_network.conv_stride = [general_conv_stride;general_conv_stride];
    default_network.pool_filter = [[ 2, 1 ];[ 2, 1 ]];
    default_network.pool_stride = [[ 2, 1 ];[ 2, 1 ]];
    default_network.num_filters = [ 16; 16];
```

```
default_network.batch_norm = [true; true];
default_network.relu = [true; true];
default_network.dropout = [true; true];
default_network.MaxEpochs = 40;
default_network.MiniBatchSize = 16;
default_network.spectrogram_window = 104;
default_network.scalogram_bins = [1:16];

mkdir(default_network.network_save_path)

% To use the above parameters as the default, create new networks and
% overwrite the variables which are to change in each
network1 = default_network;
network1.conv_filter = [general_conv_filter];
network1.conv_stride = [general_conv_stride];
network1.pool_filter = [ 2, 2 ];
network1.pool_stride = [ 1, 1 ];
network1.num_filters = 16;
network1.batch_norm = false;
network1.relu = true;
network1.dropout = true;
%network1.MaxEpochs = 4;
network1.network_save_name = strcat('t',sprintf('%05d',data_window),'_1x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network2 = default_network;
network2.conv_filter = [general_conv_filter;general_conv_filter];
network2.conv_stride = [general_conv_stride;general_conv_stride];
network2.pool_filter = [[ 2, 1 ];[ 2, 1 ]];
network2.pool_stride = [[ 1, 1 ];[ 1, 1 ]];
network2.num_filters = [ 16; 16];
network2.batch_norm = [false; false];
network2.relu = [true; true];
network2.dropout = [true; true];
%network2.MaxEpochs = 4;
network2.network_save_name = strcat('t',sprintf('%05d',data_window),'_2x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_3e');

network3 = default_network;
network3.conv_filter = [general_conv_filter;general_conv_filter;general_conv_filter];
network3.conv_stride = [general_conv_stride;general_conv_stride;general_conv_stride];
network3.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
network3.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
network3.num_filters = [ 16; 16; 16];
network3.batch_norm = [false; false; false];
network3.relu = [true; true; true];
network3.dropout = [true; true; true];
%network3.MaxEpochs = 4;
network3.network_save_name = strcat('t',sprintf('%05d',data_window),'_3x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network4 = default_network;
network4.conv_filter = [general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
network4.conv_stride = [general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride];
network4.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
network4.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
network4.num_filters = [ 16; 16; 16; 16];
network4.batch_norm = [true; true; true; true];
network4.relu = [true; true; true; true];
network4.MaxEpochs = 4;
network4.network_save_name = strcat('t',sprintf('%05d',data_window),'_4x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network5 = default_network;
network5.conv_filter = [general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
network5.conv_stride = [general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride];
network5.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
network5.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
network5.num_filters = [ 16; 16; 16; 16; 16];
network5.batch_norm = [true; true; true; true; true];
network5.relu = [true; true; true; true; true];
network5.MaxEpochs = 4;
network5.network_save_name = strcat('t',sprintf('%05d',data_window),'_5x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

network6 = default_network;
network6.conv_filter = [general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
network6.conv_stride = [general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride];
network6.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
network6.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
```

```
            network6.num_filters = [ 16; 16; 16; 16; 16; 16];
            network6.batch_norm = [true; true; true; true; true; true];
            network6.relu = [true; true; true; true; true; true];
            network6.MaxEpochs = 4;
            network6.network_save_name = strcat('t',sprintf('%05d',data_window),'_6x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

            network7 = default_network;
            network7.conv_filter =
[general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
            network7.conv_stride =
[general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride];
            network7.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
            network7.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
            network7.num_filters = [ 16; 16; 16; 16; 16; 16; 16];
            network7.batch_norm = [true; true; true; true; true; true; true];
            network7.relu = [true; true; true; true; true; true; true];
            network7.MaxEpochs = 4;
            network7.network_save_name = strcat('t',sprintf('%05d',data_window),'_7x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

            network8 = default_network;
            network8.conv_filter =
[general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter;general_conv_filter];
            network8.conv_stride =
[general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_stride;general_conv_s
tride];
            network8.pool_filter = [[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ];[ 2, 1 ]];
            network8.pool_stride = [[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ];[ 1, 1 ]];
            network8.num_filters = [ 16; 16; 16; 16; 16; 16; 16; 16];
            network8.batch_norm = [true; true; true; true; true; true; true; true];
            network8.relu = [true; true; true; true; true; true; true; true];
            network8.MaxEpochs = 4;
            network8.network_save_name = strcat('t',sprintf('%05d',data_window),'_8x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');

            network9 = default_network;
            network9.conv_filter = [[4,1];[4,1]];
            network9.conv_stride = [[2,1];[2,1]];
            network9.pool_filter = [[ 2, 1 ];[ 2, 1 ]];
            network9.pool_stride = [[ 2, 1 ];[ 2, 1 ]];
            network9.num_filters = [ 16; 16; 16; 16; 16; 16; 16; 16];
            network9.batch_norm = [true; true; true; true; true; true; true; true];
            network9.relu = [true; true; true; true; true; true; true; true];
            network9.MaxEpochs = 4;
            network9.network_save_name = strcat('t',sprintf('%05d',data_window),'_8x[',string(general_conv_filter(1)),',',string(general_conv_filter(2)),']_4e');


            network_queue = {network2};%,, network3;network1, network4, network5, network6, network7, network8, network9};
            i = 1;
            while i <= length(network_queue)
              temp_network = network_queue{i};
              %network = CWRU_network_commander_cross_validation(temp_network);
              network = PHM2009_network_commander_cross_validation(temp_network);
              fprintf('%f of %f network configurations trained\n', i, length(network_queue))
              i = i+1;
            end

        %%%
          end
          filter_index = filter_index + 1;

        end
        %%%

        end
```

25. Paderborn_data_unifier.mat

```
        % Specify the folder where the labeled sub-folders live.
        root_folder = 'D:\Paderborn_data\sorted_data_2\training';

        % Get a list of all files and folders in this folder.
        files = dir(root_folder);
        % Get a logical vector that tells which is a directory.
        dirFlags = [files.isdir];
        % Extract only those that are directories.
```

```
subFolders = files(dirFlags);
sub_folder_index = 3;
unified_data = {};

while sub_folder_index <= length(subFolders)

   myFolder = subFolders(sub_folder_index).name;
   myFolder = strcat(root_folder,'\',myFolder,'\');

   s = myFolder;     % initial subdirectory
   [~,d] = dos(['dir /s /b ' fullfile(s,'*.mat')]);  % OS dir command
   d = textscan(d,'%s','delimiter','\n'); d=d{:};   % convert string to cellstr array

   master_data = cell(length(d),2);
   %master_labels = cell(length(d),1);
   master_index = 1;

   for k = 1 : length(d)

      fullFileName = cell2mat(d(k));

      raw_measurement = load(fullFileName);

      raw_t_field = cell2mat(fields(raw_measurement));

      raw_t = raw_measurement.(char(raw_t_field));
      Y_data = raw_t.Y(7).Data;
      raw_data = zeros(length(Y_data),1,1);
      raw_data(:,1,1) = Y_data;

      master_data(k,1) = {raw_data};
      master_data(k,2) = {subFolders(sub_folder_index).name};

      master_index = master_index + 1;

   end

   unified_data = [unified_data; master_data];
   sub_folder_index = sub_folder_index + 1;

end

save('D:\Paderborn_data\sorted_data_2\training\training_data_3.mat','unified_data')
```

26. Paderborn_network_queuer.mat

```
% for k-fold cross validation, specify k
default_network.k = 4;

% Commander parameters
default_network.testing_data = 'D:\Paderborn_data\sorted_data_2\augmented_raw_time\pdbn_testing_t1200.mat';
default_network.training_data = 'D:\Paderborn_data\sorted_data_2\augmented_raw_time\pdbn_training_t1200.mat';
default_network.save_network = true;
default_network.keep_training_gui_open = false;
default_network.network_save_path = 'D:\2009PHM_data\cached_cnns\raw_time\t2400\ch1';
default_network.network_save_name = 'ch1_t1200';

%%% Specify details for each conv. layer of the network. Each column
%%% represents the parameters for a layer, thus each must have the same
%%% number of columns. 1D should be represented as [X, 1], 2D can be
%%% represented as [X1, X2] and 3D can be represented as [X1, X2, X3]
default_network.conv_filter = [[ 4, 1 ];[ 3, 1 ]];
default_network.conv_stride = [[ 2, 1 ];[ 4, 1 ]];
default_network.pool_filter = [[ 2, 1 ];[ 2, 1 ]];
default_network.pool_stride = [[ 1, 1 ];[ 1, 1 ]];
default_network.num_filters = [ 16; 8];
default_network.batch_norm = [true; true];
default_network.relu = [true; true];
default_network.MaxEpochs = 4;

mkdir(default_network.network_save_path)

% Touse the above parameters as the default, create new networks and
% overwrite the variables which are to change in each
```

```
network1 = default_network;
network1.conv_filter = [[ 4, 1 ];[ 4, 1 ]];
network1.network_save_name = '1D_time_data_1200_1_1[ 4, 1 ];[ 4, 1 ]';

network2 = default_network;
network2.conv_filter = [[ 4, 1 ];[ 6, 1 ]];
network2.network_save_name = '1D_time_data_1200_1_1[ 4, 1 ];[ 6, 1 ]';

network3 = default_network;
network3.conv_filter = [[ 4, 1 ];[ 8, 1 ]];
network3.network_save_name = '1D_time_data_1200_1_1[ 4, 1 ];[ 8, 1 ]';

network4 = default_network;
network4.conv_filter = [[ 6, 1 ];[ 4, 1 ]];
network4.network_save_name = '1D_time_data_1200_1_1[ 6, 1 ];[ 4, 1 ]';

network5 = default_network;
network5.conv_filter = [[ 6, 1 ];[ 6, 1 ]];
network5.network_save_name = '1D_time_data_1200_1_1[ 6, 1 ];[ 6, 1 ]';

network6 = default_network;
network6.conv_filter = [[ 6, 1 ];[ 8, 1 ]];
network6.network_save_name = '1D_time_data_1200_1_1[ 6, 1 ][ 8, 1 ]';

network7 = default_network;
network7.conv_filter = [[ 8, 1 ];[ 4, 1 ]];
network7.network_save_name = '1D_time_data_1200_1_1[ 8, 1 ];[ 4, 1 ]';

network8 = default_network;
network8.conv_filter = [[ 8, 1 ];[ 6, 1 ]];
network8.network_save_name = '1D_time_data_1200_1_1[ 8, 1 ];[ 6, 1 ]';

network9 = default_network;
network9.conv_filter = [[ 8, 1 ];[ 8, 1 ]];
network9.network_save_name = '1D_time_data_1200_1_1[ 8, 1 ];[ 8, 1 ]';

network_queue = {network1, network2, network3, network4, network5, network6, network7, network8, network9};
i = 1;
while i <= length(network_queue)
   temp_network = network_queue{i};
   network = paderborn_network_commander_cross_validation(temp_network);
   fprintf('%f of %f network configurations trained\n', i, length(network_queue))
   i = i+1;
end
```

27. confusion_mat_summation.mat

```
%{
i = 2;
confusion_mat = network.confusion_mat_validation{1};
while i <= 4

   confusion_mat = confusion_mat + network.confusion_mat_validation{i};

   i = i+1;


end
cats = network.categories;
new_cats = strings(1,7);
i = 1;
%}
confusion_mat = network.confusion_mat_validation;
cats = network.categories;
new_cats = string(cats);
while i <= length(cats)
   if strcmp(cats{i},'drive_end_ball')
      new_cats(i) = "Drive end ball";
   elseif strcmp(cats{i},'drive_end_inner_race')
      new_cats(i) = "Drive end inner race";
   elseif strcmp(cats{i},'drive_end_outer_race')
      new_cats(i) = "Drive end outer race";
   elseif strcmp(cats{i},'fan_end_ball')
      new_cats(i) = "Fan end ball";
```

```
    elseif strcmp(cats{i},'fan_end_inner_race')
        new_cats(i) = "Fan end inner race";
    elseif strcmp(cats{i},'fan_end_outer_race')
        new_cats(i) = "Fan end outer race";
    elseif strcmp(cats{i},'healthy')
        new_cats(i) = "Healthy";
    end
    i = i+1;

end

confusionchart(confusion_mat,new_cats)
```

28. femto_autoencoder_plots.mat

```
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\training_data_fourier_spec.mat")
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\testing_data_fourier_spec.mat")
load("D:\master_paper_3\saved_network_architectures_2\autoencoders_v04.mat")

current_data = processed_testing_bearings.Bearing1_7(:,1,end);
%subplot(2,1,1)
%plot(processed_testing_bearings.Bearing3_3(:,1,1))
%subplot(2,1,2)
%plot(processed_testing_bearings.Bearing3_3(:,1,end))

%{
encoded_test_data_1 = encode(autoenc1,current_data);
decoded_test_data = decode(autoenc1,encoded_test_data_1);
%}

encoded_test_data_1 = encode(autoenc1,current_data);
encoded_test_data_2 = encode(autoenc2,encoded_test_data_1);
encoded_test_data_3 = encode(autoenc3,encoded_test_data_2);
encoded_test_data_4 = encode(autoenc4,encoded_test_data_3);
encoded_test_data_5 = encode(autoenc5,encoded_test_data_4);
encoded_test_data_6 = encode(autoenc6,encoded_test_data_5);

decoded_test_data_5 = decode(autoenc6,encoded_test_data_6);
decoded_test_data_4 = decode(autoenc5,decoded_test_data_5);
decoded_test_data_3 = decode(autoenc4,decoded_test_data_4);
decoded_test_data_2 = decode(autoenc3,encoded_test_data_3);
decoded_test_data_1 = decode(autoenc2,decoded_test_data_2);
decoded_test_data = decode(autoenc1,decoded_test_data_1);
%}


subplot(2,1,1)
plot(current_data)
title("Original Spectrum")
subplot(2,1,2)
plot(decoded_test_data)
title("Encoded - Decoded Spectrum")


%encoded_test_data_6 = encoded_test_data_6./std(encoded_test_data_6);
%encoded_test_data_6 = encoded_test_data_6-mean(encoded_test_data_6);
```

29. femto_autoencoder_trainer.mat

```
load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\training_data_fourier_spec.mat")
load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\testing_data_fourier_spec.mat")

vibration_ch = 1;

testing_bearings = string(cell2mat(fields(processed_testing_bearings)));
training_bearings = string(cell2mat(fields(processed_training_bearings)));

merged_testing_bearings = [];
bearing_index = 1;

while bearing_index <= length(testing_bearings)
```

```
   bearing = testing_bearings(bearing_index);
   merged_testing_bearings = [merged_testing_bearings, squeeze(processed_testing_bearings.(bearing)(:,vibration_ch,:))];

   bearing_index = bearing_index + 1;
end

merged_training_bearings = [];
bearing_index = 1;
while bearing_index <= length(training_bearings)
   bearing = training_bearings(bearing_index);
   merged_training_bearings = [merged_training_bearings, squeeze(processed_training_bearings.(bearing)(:,vibration_ch,:))];

   bearing_index = bearing_index + 1;
end

[m,n] = size(merged_testing_bearings);
idx = randperm(n) ;
merged_testing_bearings = merged_testing_bearings(:,idx);

[m,n] = size(merged_training_bearings);
idx = randperm(n) ;
merged_training_bearings = merged_training_bearings(:,idx);

autoenc1 = trainAutoencoder(merged_training_bearings,640,'MaxEpochs',200);
encoded_train_data_1 = encode(autoenc1,merged_training_bearings);
decoded_train_data = decode(autoenc1,encoded_train_data_1);

autoenc2 = trainAutoencoder(encoded_train_data_1,320,'MaxEpochs',240);
encoded_train_data_2 = encode(autoenc2,encoded_train_data_1);

autoenc3 = trainAutoencoder(encoded_train_data_2,160,'MaxEpochs',288);
encoded_train_data_3 = encode(autoenc3,encoded_train_data_2);

autoenc4 = trainAutoencoder(encoded_train_data_3,80,'MaxEpochs',346);
encoded_train_data_4 = encode(autoenc4,encoded_train_data_3);

autoenc5 = trainAutoencoder(encoded_train_data_4,40,'MaxEpochs',414);
encoded_train_data_5 = encode(autoenc5,encoded_train_data_4);

autoenc6 = trainAutoencoder(encoded_train_data_5,20,'MaxEpochs',414);
encoded_train_data_6 = encode(autoenc6,encoded_train_data_5);

stackednet = stack(autoenc1,autoenc2,autoenc3, autoenc4, autoenc5, autoenc6);
encoded_train_data_stacked = encode(stackednet,merged_training_bearings);


decoded_train_data_5 = decode(autoenc6,encoded_train_data_6);
decoded_train_data_4 = decode(autoenc5,decoded_train_data_5);
decoded_train_data_3 = decode(autoenc4,decoded_train_data_4);
decoded_train_data_2 = decode(autoenc3,decoded_train_data_3);
decoded_train_data_1 = decode(autoenc2,decoded_train_data_2);
decoded_train_data = decode(autoenc1,decoded_train_data_1);

encoded_test_data_1 = encode(autoenc1,merged_testing_bearings);
encoded_test_data_2 = encode(autoenc2,encoded_test_data_1);
encoded_test_data_3 = encode(autoenc3,encoded_test_data_2);
encoded_test_data_4 = encode(autoenc4,encoded_test_data_3);
encoded_test_data_5 = encode(autoenc5,encoded_test_data_4);
encoded_test_data_6 = encode(autoenc6,encoded_test_data_5);

decoded_test_data_5 = decode(autoenc6,encoded_test_data_6);
decoded_test_data_4 = decode(autoenc5,decoded_test_data_5);
decoded_test_data_3 = decode(autoenc4,decoded_test_data_4);
decoded_test_data_2 = decode(autoenc3,decoded_test_data_3);
decoded_test_data_1 = decode(autoenc2,decoded_test_data_2);
decoded_test_data = decode(autoenc1,decoded_test_data_1);
%}

%encoded_test_data_1 = encode(autoenc1,merged_testing_bearings);
%decoded_test_data = decode(autoenc1,encoded_test_data_1);

training_data_RMSE = calc_RMSE(merged_training_bearings,decoded_train_data);
testing_data_RMSE = calc_RMSE(merged_testing_bearings,decoded_test_data);
fprintf("training RMSE, testing RMSE \n")
display([training_data_RMSE,testing_data_RMSE])
```

```
save("D:\master_paper_3\saved_network_architectures_2\autoencoders_v04.mat","autoenc1","autoenc2","autoenc3","autoenc4","autoenc5","autoenc6")

function rmse=calc_RMSE(a,b)
rmse=sqrt(mean((a(:)-b(:)).^2));
end
```

30. femto_cnn_network_trainer.mat

```
% use the line below with the output from deepNetworkDesigner to save
% untrained network architectures...
%save("D:\master_paper_4\femto_data\network_architectures\classical_cnn_v02.mat","layers_1")

clear all % Useful to do to free up RAM if other functions have been run

%% Beginning of user input section...

% select cnn type from options:
cnn_type = "classical"; % or: resnet

% IF NOT USING BIG LOOP define augmentation type and parameters:
% augmentation.type = "pitch_shift"; % or: none , gaussian_noise ,
% amplitude_transform
% region_dropout , pitch_shift , random_dropout

% big old loop begins here:
for type = [ "region_dropout","region_dropout","region_dropout","region_dropout" ]
augmentation.type = type;

augmentation.factor = 12; % the size of the augmented dataset will be the original size multiplied by the augmentaiton factor
augmentation.noise_scaling_factor = 0.3;
augmentation.dropout_region_size = 1280 * 0.1;
augmentation.dropout_region_number = 2;
augmentation.dropout_random_fraction = 0.2;
augmentation.pitch_shift_semitones = [-0.5, 0, 0.5, 1]; %length of this array must be equal to augmentation.factor!
augmentation.amplitude_transform_exponents = [0.5 , 1 , 1.5, 2];

% auto-generate save folder, modify as desired
net_save_folder = strcat("D:\master_paper_4\femto_data\trained_nets\",augmentation.type,"_X",string(augmentation.factor), "_v02\");
net_save_prefix = "stcnn_v01";

% dataset paths to choose from... modify if desired
non_aug_spectrogram_dataset_path = "D:\master_paper_4\femto_data\training_data_spectrogram_fpt_trimmed.mat";
non_aug_spectrum_dataset_path = "D:\master_paper_4\femto_data\training_data_fourier_spec_norm_fpt_trimmed.mat";
non_aug_raw_time_dataset_path = "D:\master_paper_4\femto_data\training_data_fpt_trimmed.mat";
% cnn paths to choose from... modify if desired
classical_cnn_path = "D:\master_paper_4\femto_data\network_architectures\classical_cnn_v02.mat";
resnet_path = "";

if cnn_type == "classical"
net_path = classical_cnn_path;
elseif cnn_type == "resnet"
   net_path = resnet_path;
else
   warning("Variable cnn_type did not resolve to one of the acceptable values. Check code comments for acceptable values")
end

% Select dataset from path options...
dataset_path = non_aug_spectrum_dataset_path;
% select training duration using # epochs
max_epochs = 10;


%% End of input section...

network = load(net_path);
network_fields = string(fields(network));
network = network.(network_fields);

% if using pitch shift, we need to begin with raw time, pitch shifting and
% conversion to frequency domain is done in augmentation function
if augmentation.type == "pitch_shift"
   fprintf("Loading dataset: %s\n", non_aug_raw_time_dataset_path)
   processed_training_bearings = load(non_aug_raw_time_dataset_path);
```

```
    processed_training_bearings = processed_training_bearings.training_data;
    merge_idx = 3;
else
    fprintf("Loading dataset: %s\n", dataset_path)
    load(dataset_path)
    merge_idx = 4;
end

mkdir(net_save_folder)
fileID = fopen(strcat(net_save_folder,"log.txt"),'a');
fprintf(fileID,"\n\n\n");
fprintf(fileID,"Network: %s\nDataset: %s\nMax epochs: %s\n",net_path, dataset_path, string(max_epochs));

field_names = string(fields(augmentation));
field_index = 1;
while field_index <= length(field_names)

    fprintf(fileID,"%s : %s \n",field_names(field_index), string(augmentation.(field_names(field_index))));

    field_index = field_index + 1;
end

all_bearings = sort(string(cell2mat(fields(processed_training_bearings))));

%%% start point should = 1 to begin with bearing1-1
i = 1;

while i <= length(all_bearings)

    test_bearing = all_bearings(i);

    % The dataset is reloaded for each testing bearing... keeping a
    % seperate variable allocated for undivided and unshuffled training data means
    % keeping a lot of RAM occupied (in some instances)
    if i == 1
      fprintf("Dataset already loaded\n" )
    elseif augmentation.type == "pitch_shift"
      fprintf("Loading dataset: %s\n", non_aug_raw_time_dataset_path)
      processed_training_bearings = load(non_aug_raw_time_dataset_path);
      processed_training_bearings = processed_training_bearings.training_data;
      merge_idx = 3;
    else
      fprintf("Loading dataset: %s\n", dataset_path)
      load(dataset_path)
    end

    % Assigns linear labeling, 1 for beginning of experiment (or alarm time), 0 for end.
    training_labels = getLabels(processed_training_bearings);

    fprintf("Assigning %s as validation data.\n",test_bearing)
    processed_testing_bearings = processed_training_bearings.(test_bearing);
    testing_labels = training_labels.(test_bearing);

    % If using pitch shift, all data started in
    if augmentation.type == "pitch_shift"
      %original_testing_size = size(processed_testing_bearings);
      %new_testing_size = [0.5*original_testing_size(1), 2, 1, original_testing_size(end)];
      %new_testing_data = zeros(new_testing_size);
      %i = 1;
      %while i <=
      testing_augmentation = augmentation;
      testing_augmentation.pitch_shift_semitones = 0;
      testing_augmentation.factor = 1;
      [processed_testing_bearings, testing_labels] = femto_data_aug(processed_testing_bearings, testing_labels, testing_augmentation);
    end

    processed_training_bearings = rmfield(processed_training_bearings,test_bearing);
    training_labels =  rmfield(training_labels,test_bearing);

    training_bearings = string(cell2mat(fields(processed_training_bearings)));

    % Take data from individual bearing structs and combine them into 4D
    % numerical matricies
    merged_training_bearings = [];
    merged_training_labels = [];
```

```
    bearing_index = 1;
    while bearing_index <= length(training_bearings)
        bearing = training_bearings(bearing_index);
        merged_training_bearings = cat(merge_idx,merged_training_bearings, processed_training_bearings.(bearing));
        merged_training_labels = [merged_training_labels, training_labels.(bearing)];

        processed_training_bearings = rmfield(processed_training_bearings,bearing);

        bearing_index = bearing_index + 1;
    end

    % Run the data through data augmentation
    if augmentation.type ~= "none"
        [merged_training_bearings, merged_training_labels] = femto_data_aug(merged_training_bearings, merged_training_labels, augmentation);
    end

    % Shuffle the order of training and testing datasets (important that
    % labels and samples still correspond with eachother)

    dim = size(processed_testing_bearings);
    idx = randperm(dim(end)) ;
    processed_testing_bearings = processed_testing_bearings(:,:,:,idx);
    testing_labels = testing_labels(idx);

    dim = size(merged_training_bearings);
    idx = randperm(dim(end));
    merged_training_bearings = merged_training_bearings(:,:,:,idx);
    merged_training_labels = merged_training_labels(idx);

    % Need to meet MATLAB's expectations for row / column vector with label
    % space
    testing_labels = transpose(testing_labels);
    merged_training_labels = transpose(merged_training_labels);

    options = trainingOptions('adam', ...
        'InitialLearnRate', 0.001, ...
        'LearnRateSchedule','piecewise', ...
        'LearnRateDropFactor',0.2, ...
        'LearnRateDropPeriod',2, ...
        'MaxEpochs',max_epochs, ...
        'MiniBatchSize',32, ...
        'Plots','training-progress', ...
        'ValidationData', {processed_testing_bearings,testing_labels}, ...
        'ValidationFrequency', 50);

    [net,info] = trainNetwork(merged_training_bearings,merged_training_labels,network,options);

    fprintf(fileID,"%s    Validation RMSE  %f \n",test_bearing, info.ValidationRMSE(end));

    save(strcat(net_save_folder,net_save_prefix,test_bearing,".mat"),"net", "info")

    i = i + 1;
end

fprintf(fileID,"\n\n\n");

end

function labels = getLabels(bearings)
    field_names = sort(string(cell2mat(fields(bearings))));
    i = 1;
    while i <= length(field_names)
        dim = size(bearings.(field_names(i)));
        labels.(field_names(i)) = [1:dim(end)]./dim(end);
        labels.(field_names(i)) = -labels.(field_names(i))+1;
        i = i + 1;
    end
end
```

31. femto_data_aug.mat

```
function [processed_training_bearings,new_training_labels] = femto_data_aug(processed_training_bearings, training_labels, augmentation)
```

```
%%%% random noise, standard normal / gaussian distribution
if augmentation.type == "gaussian_noise"

   new_size = size(processed_training_bearings) .* [1 1 1 augmentation.factor];

   new_processed_training_bearings = processed_training_bearings;
   new_training_labels = training_labels;
   i = 2;
   while i <= augmentation.factor
      new_processed_training_bearings = cat(4, new_processed_training_bearings, processed_training_bearings);
      new_training_labels = cat(2, new_training_labels, training_labels);
     i = i + 1;
   end
   noise = augmentation.noise_scaling_factor * randn(new_size);
   processed_training_bearings = new_processed_training_bearings + noise;

   % reset standard of deviation to 1;
   processed_training_bearings = processed_training_bearings/mean(mean(std(processed_training_bearings)));

elseif augmentation.type == "region_dropout"

   new_size = size(processed_training_bearings) .* [1 1 1 augmentation.factor];

   new_processed_training_bearings = processed_training_bearings;
   new_training_labels = training_labels;
   i = 2;
   while i <= augmentation.factor
      new_processed_training_bearings = cat(4, new_processed_training_bearings, processed_training_bearings);
      new_training_labels = cat(2, new_training_labels, training_labels);
     i = i + 1;
   end

   dropout_map = ones(new_size);

   i = 1;
   while i <= new_size(end)

      j = 1;
      while j <= augmentation.dropout_region_number
         for ch = 1:new_size(2)
            region_start = randi([1,new_size(1)-augmentation.dropout_region_size],1);
            dropout_map(region_start:region_start + augmentation.dropout_region_size, ch, 1, i) = 0;
         end
         j = j + 1;
      end
      i = i + 1;
   end

   processed_training_bearings = new_processed_training_bearings.*dropout_map;

elseif augmentation.type == "random_dropout"

   new_size = size(processed_training_bearings) .* [1 1 1 augmentation.factor];

   new_processed_training_bearings = processed_training_bearings;
   new_training_labels = training_labels;
   i = 2;
   while i <= augmentation.factor
      new_processed_training_bearings = cat(4, new_processed_training_bearings, processed_training_bearings);
      new_training_labels = cat(2, new_training_labels, training_labels);
     i = i + 1;
   end
   dropout_map = ones(new_size);
   dropout_samples = round(augmentation.dropout_random_fraction*new_size(1));
   i = 1;
   while i <= new_size(end)
      for ch = 1:new_size(2)

         dropout_map(1:dropout_samples,ch,i) = zeros([dropout_samples,1]);
         idx = randperm(new_size(1));
         dropout_map(:,ch,i) = dropout_map(idx,ch,i);

      end
```

```
      i = i + 1;
    end
    processed_training_bearings = new_processed_training_bearings.*dropout_map;

elseif augmentation.type == "pitch_shift"

    original_size = size(processed_training_bearings);
    i = 1;
    while i <= original_size(end)
      resized_processed_training_bearings(:,:,1,i) = processed_training_bearings(:,:,i);
      i = i + 1;
    end
    processed_training_bearings = resized_processed_training_bearings;

    new_size = size(processed_training_bearings) .* [1 1 1 augmentation.factor];
    new_size_spectrum = size(processed_training_bearings) .* [0.5 1 1 augmentation.factor];

    new_processed_training_bearings = zeros(new_size_spectrum);
    new_training_labels = [];

    new_index = 1;
    for pitch_shift_semitone = augmentation.pitch_shift_semitones
      original_index = 1;
      new_training_labels = [new_training_labels , training_labels];

      while original_index <= original_size(end)
        for ch = 1:new_size(2)

          whole_spec = singleSidedSpec(shiftPitch(processed_training_bearings(:,ch,original_index),pitch_shift_semitone));
          new_processed_training_bearings(:,ch,new_index) = whole_spec(1:1280,:,:);

        end
        new_index = new_index + 1;
        original_index = original_index + 1;
      end

    end
    processed_training_bearings = new_processed_training_bearings;
    processed_training_bearings = processed_training_bearings/mean(mean(std(processed_training_bearings)));
    processed_training_bearings = processed_training_bearings-mean(mean(mean(processed_training_bearings)));

elseif augmentation.type == "amplitude_transform"

    original_size = size(processed_training_bearings);
    i = 1;

    new_size = size(processed_training_bearings) .* [1 1 1 augmentation.factor];

    new_processed_training_bearings = zeros(new_size);
    new_training_labels = [];

    new_index = 1;
    for exponent = augmentation.amplitude_transform_exponents
      original_index = 1;
      new_training_labels = [new_training_labels , training_labels];

      while original_index <= original_size(end)
        for ch = 1:new_size(2)

          new_processed_training_bearings(:,ch,new_index) = abs( processed_training_bearings(:,ch,original_index)) .^ exponent;

        end
        new_index = new_index + 1;
        original_index = original_index + 1;
      end
    end

    processed_training_bearings = new_processed_training_bearings;

    processed_training_bearings = processed_training_bearings/mean(mean(std(processed_training_bearings)));
    processed_training_bearings = processed_training_bearings-mean(mean(mean(processed_training_bearings)));

end

end
```

```
function spectrum = singleSidedSpec(signal)
y = fft(signal);
L = length(signal);
P2 = abs(y/L);
spectrum = P2(1:L/2+1);
spectrum(2:end-1) = 2*spectrum(2:end-1);
end
```

32. femto_data_fpt_trimmer.mat

```
load('D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\training_data.mat')

vibration_ch = 1;

training_bearings = string(cell2mat(fields(training_data)));

%TRAINING DATA PREP
bearing_index = 1;
while bearing_index <= length(training_bearings)
   bearing = training_bearings(bearing_index);
   current_data = training_data.(bearing);
   alarm_time = getAlarmTime(current_data);

   alarm_time = alarm_time - 5;

   current_data = current_data(:,:,alarm_time:end);
   training_data.(bearing) = current_data;

   bearing_index = bearing_index + 1;
end
load handel
sound(y,Fs)

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\training_data_fpt_trimmed.mat","training_data",'-v7.3')

function alarm_time = getAlarmTime(current_data)

   current_data = squeeze(current_data(:,1,:));

   raw_kurt = kurtosis(current_data);
   kurt_hist = histogram(raw_kurt,10000);
   k = 1;
   current_sum = 0;
   while current_sum <= length(raw_kurt)*0.89
      current_sum = current_sum + kurt_hist.Values(k);
      k = k+1;
   end
   alarm = kurt_hist.BinEdges(k);
   alarm_idx = find(raw_kurt>alarm);

   alarm_idx_new = diff(alarm_idx);
   m = 3;
   while m <= length(alarm_idx_new)
      if alarm_idx_new(m) == 1 && alarm_idx_new(m-1) == 1 && alarm_idx_new(m-2) == 1
         break
      end
      %fprintf("%.0f  %.0f  %.0f  \n",alarm_idx_new(m), alarm_idx_new(m-1),alarm_idx_new(m-2))
      m = m+1;
   end

   alarm_time = alarm_idx(m);

end
```

33. femto_data_preprocessor.mat

```
preprocessor = "spectrum";% autoenc_v01  feature_extractor_1


if preprocessor == "feature_extractor_1"
```

```
    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data.mat")
    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data.mat")

    testing_bearings = string(cell2mat(fields(testing_data)));
    training_bearings = string(cell2mat(fields(training_data)));

    [processed_testing_bearings,processed_training_bearings] = feature_extractor_1(testing_data,training_data,testing_bearings,training_bearings);

end


if preprocessor == "spectrum"

    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data.mat")
    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data.mat")

    testing_bearings = string(cell2mat(fields(testing_data)));
    training_bearings = string(cell2mat(fields(training_data)));

    [processed_testing_bearings,processed_training_bearings] = fourier_preprocessor(testing_data,training_data,testing_bearings,training_bearings);

elseif preprocessor == "autoenc_v01"

    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data_fourier_spec.mat")
    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data_fourier_spec.mat")
    load("D:\master_paper_3\saved_network_architectures\autoencoders_v01.mat");

    testing_bearings = string(cell2mat(fields(processed_testing_bearings)));
    training_bearings = string(cell2mat(fields(processed_training_bearings)));

    [processed_testing_bearings,processed_training_bearings] = autoenc_v01(processed_testing_bearings, processed_training_bearings,
testing_bearings,training_bearings,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);

    save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data_autoenc_v01.mat","processed_training_bearings")
    save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data_autoenc_v01.mat","processed_testing_bearings")

end


function [processed_testing_bearings,processed_training_bearings] = feature_extractor_1(testing_data,training_data,testing_bearings,training_bearings);

bearing_index = 1;
while bearing_index <= length(testing_bearings)

    bearing = testing_bearings(bearing_index);

    current_data = testing_data.(bearing);
    size_current_data = size(current_data);
    size_feature = size(feature_extractor(current_data(:,:,1)));
    new_data = zeros([size_feature(1),1,size_current_data(end)]);

    sample_index = 1;
    while sample_index <= size_current_data(end)
        current_sample = current_data(:,:,sample_index);

        new_sample = feature_extractor(current_sample);

        new_data(:,:,sample_index) = new_sample;

        sample_index = sample_index + 1;
    end
    new_data = new_data-mean(mean(new_data));
    new_data = new_data/mean(std(new_data));
    processed_testing_bearings.(bearing) = new_data;
    bearing_index = bearing_index + 1;
end

bearing_index = 1;
while bearing_index <= length(training_bearings)

    bearing = training_bearings(bearing_index);

    current_data = training_data.(bearing);
    size_current_data = size(current_data);
```

```
    size_feature = size(feature_extractor(current_data(:,:,1)));
    new_data = zeros([size_feature(1),1,size_current_data(end)]);

    sample_index = 1;
    while sample_index <= size_current_data(end)
        current_sample = current_data(:,:,sample_index);

        new_sample = feature_extractor(current_sample);

        new_data(:,:,sample_index) = new_sample;

        sample_index = sample_index + 1;
    end
    new_data = new_data-mean(mean(new_data));
    new_data = new_data/mean(std(new_data));
    processed_training_bearings.(bearing) = new_data;
    bearing_index = bearing_index + 1;
end

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data_manual_features_2.mat","processed_training_bearings")
save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data_manual_features_2.mat","processed_testing_bearings")

function new_sample = feature_extractor(current_sample)

    spec_ch1 = singleSidedSpec(current_sample(:,1));
    spec_ch2 = singleSidedSpec(current_sample(:,2));

    %%time domain features
    std_ch1 = std(current_sample(:,1));
    std_ch2 = std(current_sample(:,2));
    rms_ch1 = rms(current_sample(:,1));
    rms_ch2 = rms(current_sample(:,2));
    kurt_ch1 = kurtosis(current_sample(:,1));
    kurt_ch2 = kurtosis(current_sample(:,2));
    crstfac_ch1 = peak2rms(current_sample(:,1));
    crstfac_ch2 = peak2rms(current_sample(:,2));
    shpfac_ch1 = rms_ch1/mean(abs(current_sample(:,1)));
    shpfac_ch2 = rms_ch2/mean(abs(current_sample(:,2)));

    %%frequency domain features
    mean_spec_ch1 = mean(spec_ch1);
    mean_spec_ch2 = mean(spec_ch2);
    %cent_freq_ch1 = center_frequency(spec_ch1);
    %cent_freq_ch2 = center_frequency(spec_ch2);
    rms_spec_ch1 = rms(spec_ch1);
    rms_spec_ch2 = rms(spec_ch2);
    std_spec_ch1 = std(spec_ch1);
    std_spec_ch2 = std(spec_ch2);

    new_sample = [std_ch1; std_ch2; rms_ch1; rms_ch2; kurt_ch1; kurt_ch2; crstfac_ch1; crstfac_ch2; shpfac_ch1; shpfac_ch2;...
            mean_spec_ch1; mean_spec_ch2; rms_spec_ch1; rms_spec_ch2; std_spec_ch1; std_spec_ch2];

end

function cent_freq_ch1 = center_frequency(spec_ch1)
    cent_freq_ch1 = 0;
    k = 1;
    while k <= length(spec_ch1)
        cent_freq_ch1 = cent_freq_ch1 + spec_ch1(k)*k;
        k = k + 1;
    end
    cent_freq_ch1 = cent_freq_ch1 / mean([1:k]);

end

function spectrum = singleSidedSpec(signal)
y = fft(signal);
L = length(signal);
P2 = abs(y/L);
spectrum = P2(1:L/2+1);
spectrum(2:end-1) = 2*spectrum(2:end-1);
end
```

```
    end

    function [encoded_testing_bearings,encoded_training_bearings] = autoenc_v01(testing_data, training_data,
testing_bearings,training_bearings,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);

      bearing_index = 1;
      while bearing_index <= length(testing_bearings)
        bearing = testing_bearings(bearing_index);

        current_data = testing_data.(bearing);
        new_data_size = size(current_data);
        new_data_size(:,1,1) = 20;
        new_data = zeros(new_data_size);

        ch1_data = squeeze(current_data(:,1,:));
        ch2_data = squeeze(current_data(:,2,:));

        ch1_data_encoded = autoenc_v01(ch1_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);
        ch2_data_encoded = autoenc_v01(ch2_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);

        new_data(:,1,:) = ch1_data_encoded;
        new_data(:,2,:) = ch2_data_encoded;

        encoded_testing_bearings.(bearing) = new_data;
        bearing_index = bearing_index + 1;
      end

      bearing_index = 1;
      while bearing_index <= length(training_bearings)
        bearing = training_bearings(bearing_index);

        current_data = training_data.(bearing);
        new_data_size = size(current_data);
        new_data_size(:,1,1) = 20;
        new_data = zeros(new_data_size);

        ch1_data = squeeze(current_data(:,1,:));
        ch2_data = squeeze(current_data(:,2,:));

        ch1_data_encoded = autoenc_v01(ch1_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);
        ch2_data_encoded = autoenc_v01(ch2_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);

        new_data(:,1,:) = ch1_data_encoded;
        new_data(:,2,:) = ch2_data_encoded;

        encoded_training_bearings.(bearing) = new_data;
        bearing_index = bearing_index + 1;
      end

      function encoded_data = autoenc_v01(original_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6)

        encoded_test_data_1 = encode(autoenc1,original_data);
        encoded_test_data_2 = encode(autoenc2,encoded_test_data_1);
        encoded_test_data_3 = encode(autoenc3,encoded_test_data_2);
        encoded_test_data_4 = encode(autoenc4,encoded_test_data_3);
        encoded_test_data_5 = encode(autoenc5,encoded_test_data_4);
        encoded_data = encode(autoenc6,encoded_test_data_5);

      end

    end

    function [processed_testing_bearings,processed_training_bearings] = fourier_preprocessor(testing_data,training_data,testing_bearings,training_bearings)

    bearing_index = 1;
    while bearing_index <= length(testing_bearings)
      bearing = testing_bearings(bearing_index);

      current_data = testing_data.(bearing);
      new_data_size = size(current_data).*[0.5,1,1];
      new_data = zeros(new_data_size);

      sample_index = 1;
      while sample_index <= new_data_size(end)
        current_sample = current_data(:,:,sample_index);
```

```
        new_data(:,1,sample_index) = singleSidedSpec(current_sample(:,1),new_data_size);
        new_data(:,2,sample_index) = singleSidedSpec(current_sample(:,2),new_data_size);
        sample_index = sample_index + 1;
    end
    new_data = new_data-mean(mean(new_data));
    new_data = new_data./mean(std(new_data));
    processed_testing_bearings.(bearing) = new_data;
    bearing_index = bearing_index + 1;
end

bearing_index = 1;
while bearing_index <= length(training_bearings)
    bearing = training_bearings(bearing_index);

    current_data = training_data.(bearing);
    new_data_size = size(current_data).*[0.5,1,1];
    new_data = zeros(new_data_size);

    sample_index = 1;
    while sample_index <= new_data_size(end)
        current_sample = current_data(:,:,sample_index);
        new_data(:,1,sample_index) = singleSidedSpec(current_sample(:,1),new_data_size);
        new_data(:,2,sample_index) = singleSidedSpec(current_sample(:,2),new_data_size);
        sample_index = sample_index + 1;
    end
    new_data = new_data-mean(mean(new_data));
    new_data = new_data./mean(std(new_data));
    processed_training_bearings.(bearing) = new_data;
    bearing_index = bearing_index + 1;
end

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data_fourier_spec_norm.mat","processed_training_bearings")
save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data_fourier_spec_norm.mat","processed_testing_bearings")

function spectrum = singleSidedSpec(signal,new_data_size)
y = fft(signal);
L = length(signal);
P2 = abs(y/L);
spectrum = P2(1:L/2+1);
spectrum(2:end-1) = 2*spectrum(2:end-1);
spectrum = spectrum(1:new_data_size(1));
end

end
```

34. femto_data_preprocessor_v2.mat

```
preprocessor = "spectrum";% autoenc_v01  feature_extractor_1 spectrum spectrogram

if preprocessor == "spectrogram"

    load("D:\master_paper_4\femto_data\training_data.mat")

    training_bearings = string(cell2mat(fields(training_data)));
    processed_training_bearings = get_spectrogram(training_data,training_bearings);

    save("D:\master_paper_4\femto_data\training_data_spectrum.mat","processed_training_bearings",'-v7.3')

elseif preprocessor == "feature_extractor_1"

    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data.mat")

    testing_bearings = string(cell2mat(fields(testing_data)));
    training_bearings = string(cell2mat(fields(training_data)));

    [processed_testing_bearings,processed_training_bearings] = feature_extractor_1(testing_data,testing_bearings);


elseif preprocessor == "spectrum"

    load("D:\master_paper_4\femto_data\training_data.mat")
```

```
    training_bearings = string(cell2mat(fields(training_data)));
    processed_training_bearings = fourier_preprocessor(training_data, training_bearings);

    save("D:\master_paper_4\femto_data\training_data_fourier_spec_norm.mat","processed_training_bearings",'-v7.3')


elseif preprocessor == "autoenc_v01"

    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data_fourier_spec.mat")
    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data_fourier_spec.mat")
    load("D:\master_paper_3\saved_network_architectures\autoencoders_v01.mat");

    testing_bearings = string(cell2mat(fields(processed_testing_bearings)));
    training_bearings = string(cell2mat(fields(processed_training_bearings)));

    [processed_testing_bearings,processed_training_bearings] = autoenc_v01(processed_testing_bearings, processed_training_bearings,
testing_bearings,training_bearings,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);

    save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data_autoenc_v01.mat","processed_training_bearings")
    save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data_autoenc_v01.mat","processed_testing_bearings")

end


function processed_training_bearings = get_spectrogram(training_data,training_bearings)

    bearing_index = 1;
    while bearing_index <= length(training_bearings)

        bearing = training_bearings(bearing_index);

        current_data = training_data.(bearing);
        current_data_size = size(current_data);
        %data_size = size(spectrogram(current_data(:,1,1),22,11,447));
        %data_size(1) = 128;
        data_size = [224,224];

        new_data_size = [data_size,2,current_data_size(end)];
        new_data = zeros(new_data_size);

        sample_index = 1;
        while sample_index <= new_data_size(end)
            current_sample = current_data(:,:,sample_index);
            spec_ch1 = abs(spectrogram(current_sample(:,1),22,11,447));
            spec_ch2 = abs(spectrogram(current_sample(:,2),22,11,447));

            spec_ch1 = spec_ch1 - mean(mean(spec_ch1))+1;
            spec_ch2 = spec_ch2 - mean(mean(spec_ch2))+1;
            spec_ch1 = spec_ch1/mean(std(spec_ch1));
            spec_ch2 = spec_ch2/mean(std(spec_ch2));

            new_data(:,:,1,sample_index) = spec_ch1(1:224,1:224);
            new_data(:,:,2,sample_index) = spec_ch2(1:224,1:224);
            sample_index = sample_index + 1;
        end
        processed_training_bearings.(bearing) = new_data;
        bearing_index = bearing_index + 1;
    end



end

function [processed_testing_bearings,processed_training_bearings] = feature_extractor_1(testing_data,training_data,testing_bearings,training_bearings);

bearing_index = 1;
while bearing_index <= length(testing_bearings)

    bearing = testing_bearings(bearing_index);

    current_data = testing_data.(bearing);
    size_current_data = size(current_data);
    size_feature = size(feature_extractor(current_data(:,:,1)));
    new_data = zeros([size_feature(1),1,size_current_data(end)]);
```

```
    sample_index = 1;
    while sample_index <= size_current_data(end)
        current_sample = current_data(:,:,sample_index);

        new_sample = feature_extractor(current_sample);

        new_data(:,:,sample_index) = new_sample;

        sample_index = sample_index + 1;
    end
    new_data = new_data-mean(mean(new_data));
    new_data = new_data/mean(std(new_data));
    processed_testing_bearings.(bearing) = new_data;
    bearing_index = bearing_index + 1;
end

bearing_index = 1;
while bearing_index <= length(training_bearings)

    bearing = training_bearings(bearing_index);

    current_data = training_data.(bearing);
    size_current_data = size(current_data);
    size_feature = size(feature_extractor(current_data(:,:,1)));
    new_data = zeros([size_feature(1),1,size_current_data(end)]);

    sample_index = 1;
    while sample_index <= size_current_data(end)
        current_sample = current_data(:,:,sample_index);

        new_sample = feature_extractor(current_sample);

        new_data(:,:,sample_index) = new_sample;

        sample_index = sample_index + 1;
    end
    new_data = new_data-mean(mean(new_data));
    new_data = new_data/mean(std(new_data));
    processed_training_bearings.(bearing) = new_data;
    bearing_index = bearing_index + 1;
end

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data_manual_features_2.mat","processed_training_bearings")
save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data_manual_features_2.mat","processed_testing_bearings")

function new_sample = feature_extractor(current_sample)

    spec_ch1 = singleSidedSpec(current_sample(:,1));
    spec_ch2 = singleSidedSpec(current_sample(:,2));

    %%time domain features
    std_ch1 = std(current_sample(:,1));
    std_ch2 = std(current_sample(:,2));
    rms_ch1 = rms(current_sample(:,1));
    rms_ch2 = rms(current_sample(:,2));
    kurt_ch1 = kurtosis(current_sample(:,1));
    kurt_ch2 = kurtosis(current_sample(:,2));
    crstfac_ch1 = peak2rms(current_sample(:,1));
    crstfac_ch2 = peak2rms(current_sample(:,2));
    shpfac_ch1 = rms_ch1/mean(abs(current_sample(:,1)));
    shpfac_ch2 = rms_ch2/mean(abs(current_sample(:,2)));

    %%frequency domain features
    mean_spec_ch1 = mean(spec_ch1);
    mean_spec_ch2 = mean(spec_ch2);
    %cent_freq_ch1 = center_frequency(spec_ch1);
    %cent_freq_ch2 = center_frequency(spec_ch2);
    rms_spec_ch1 = rms(spec_ch1);
    rms_spec_ch2 = rms(spec_ch2);
    std_spec_ch1 = std(spec_ch1);
    std_spec_ch2 = std(spec_ch2);

    new_sample = [std_ch1; std_ch2; rms_ch1; rms_ch2; kurt_ch1; kurt_ch2; crstfac_ch1; crstfac_ch2; shpfac_ch1; shpfac_ch2;...
            mean_spec_ch1; mean_spec_ch2; rms_spec_ch1; rms_spec_ch2; std_spec_ch1; std_spec_ch2];
```

```
end

function cent_freq_ch1 = center_frequency(spec_ch1)
  cent_freq_ch1 = 0;
  k = 1;
  while k <= length(spec_ch1)
    cent_freq_ch1 = cent_freq_ch1 + spec_ch1(k)*k;
    k = k + 1;
  end
  cent_freq_ch1 = cent_freq_ch1 / mean([1:k]);

end

function spectrum = singleSidedSpec(signal)
y = fft(signal);
L = length(signal);
P2 = abs(y/L);
spectrum = P2(1:L/2+1);
spectrum(2:end-1) = 2*spectrum(2:end-1);
end



end

function [encoded_testing_bearings,encoded_training_bearings] = autoenc_v01(testing_data, training_data,
testing_bearings,training_bearings,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);

  bearing_index = 1;
  while bearing_index <= length(testing_bearings)
    bearing = testing_bearings(bearing_index);

    current_data = testing_data.(bearing);
    new_data_size = size(current_data);
    new_data_size(:,1,1) = 20;
    new_data = zeros(new_data_size);

    ch1_data = squeeze(current_data(:,1,:));
    ch2_data = squeeze(current_data(:,2,:));

    ch1_data_encoded = autoenc_v01(ch1_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);
    ch2_data_encoded = autoenc_v01(ch2_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);

    new_data(:,1,:) = ch1_data_encoded;
    new_data(:,2,:) = ch2_data_encoded;

    encoded_testing_bearings.(bearing) = new_data;
    bearing_index = bearing_index + 1;
  end

  bearing_index = 1;
  while bearing_index <= length(training_bearings)
    bearing = training_bearings(bearing_index);

    current_data = training_data.(bearing);
    new_data_size = size(current_data);
    new_data_size(:,1,1) = 20;
    new_data = zeros(new_data_size);

    ch1_data = squeeze(current_data(:,1,:));
    ch2_data = squeeze(current_data(:,2,:));

    ch1_data_encoded = autoenc_v01(ch1_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);
    ch2_data_encoded = autoenc_v01(ch2_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6);

    new_data(:,1,:) = ch1_data_encoded;
    new_data(:,2,:) = ch2_data_encoded;

    encoded_training_bearings.(bearing) = new_data;
    bearing_index = bearing_index + 1;
  end

  function encoded_data = autoenc_v01(original_data,autoenc1,autoenc2,autoenc3,autoenc4,autoenc5,autoenc6)

    encoded_test_data_1 = encode(autoenc1,original_data);
```

```
    encoded_test_data_2 = encode(autoenc2,encoded_test_data_1);
    encoded_test_data_3 = encode(autoenc3,encoded_test_data_2);
    encoded_test_data_4 = encode(autoenc4,encoded_test_data_3);
    encoded_test_data_5 = encode(autoenc5,encoded_test_data_4);
    encoded_data = encode(autoenc6,encoded_test_data_5);

  end

end

function [processed_training_bearings] = fourier_preprocessor(training_data, training_bearings)

bearing_index = 1;
while bearing_index <= length(training_bearings)
  bearing = training_bearings(bearing_index);

  current_data = training_data.(bearing);
  new_data_size = size(current_data).*[0.5,1,1];
  new_data_size = [new_data_size(1),new_data_size(2),1,new_data_size(end)];
  new_data = zeros(new_data_size);

  sample_index = 1;
  while sample_index <= new_data_size(end)
    current_sample = current_data(:,:,sample_index);
    new_data(:,1,1,sample_index) = singleSidedSpec(current_sample(:,1),new_data_size);
    new_data(:,2,1,sample_index) = singleSidedSpec(current_sample(:,2),new_data_size);
    sample_index = sample_index + 1;
  end
  new_data = new_data-mean(mean(new_data));
  new_data = new_data./mean(std(new_data));
  processed_training_bearings.(bearing) = new_data;
  bearing_index = bearing_index + 1;
end


function spectrum = singleSidedSpec(signal,new_data_size)
y = fft(signal);
L = length(signal);
P2 = abs(y/L);
spectrum = P2(1:L/2+1);
spectrum(2:end-1) = 2*spectrum(2:end-1);
spectrum = spectrum(1:new_data_size(1));
end

end
```

35. femto_data_series_constructor.mat

```
load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data_fourier_spec_norm.mat")
load("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data_fourier_spec_norm.mat")

current_point_stride_testing = 20;
current_point_stride_training = 1;
backwards_offset = 5;

vibration_ch = 1;

testing_bearings = string(cell2mat(fields(processed_testing_bearings)));
training_bearings = string(cell2mat(fields(processed_training_bearings)));

%TESTING DATA PREP
bearing_index = 1;
while bearing_index <= length(testing_bearings)
  bearing = testing_bearings(bearing_index);
  current_data = processed_testing_bearings.(bearing);
  current_data = current_data(:,vibration_ch,:);
  size_current_data = size(current_data);

  testing_series.(bearing) = {};
  testing_labels.(bearing) = [];

  current_point = 1;
  while current_point <= size_current_data(end)
    current_label = current_point/size_current_data(end);
```

```
    current_sample = current_data(:,:,current_point);
    backwards_point = current_point - backwards_offset;
    while backwards_point > 1
      current_sample = [current_sample, current_data(:,:,backwards_point)];
      %current_label = [current_label, backwards_point/size_current_data(end)];
      backwards_point = backwards_point-backwards_offset;
    end
    current_sample = current_sample-mean(mean(current_sample));
    current_sample = current_sample/mean(std(current_sample));
    testing_series.(bearing) = [current_sample; testing_series.(bearing)];
    testing_labels.(bearing) = [current_label; testing_labels.(bearing)];

    current_point = current_point + current_point_stride_testing;
  end

  bearing_index = bearing_index + 1;
end

%TRAINING DATA PREP
bearing_index = 1;
while bearing_index <= length(training_bearings)
  bearing = training_bearings(bearing_index);
  current_data = processed_training_bearings.(bearing);
  current_data = current_data(:,vibration_ch,:);
  size_current_data = size(current_data);

  training_series.(bearing) = {};
  training_labels.(bearing) = [];

  current_point = 1;
  while current_point <= size_current_data(end)
    current_label = current_point/size_current_data(end);
    current_sample = current_data(:,:,current_point);
    backwards_point = current_point - backwards_offset;
    while backwards_point > 1
      current_sample = [current_sample, current_data(:,:,backwards_point)];
      %current_label = [current_label, backwards_point/size_current_data(end)];
      backwards_point = backwards_point-backwards_offset;
    end
    current_sample = current_sample-mean(mean(current_sample));
    current_sample = current_sample/mean(std(current_sample));
    training_series.(bearing) = [current_sample; training_series.(bearing)];
    training_labels.(bearing) = [current_label; training_labels.(bearing)];

    current_point = current_point + current_point_stride_training;
  end

  bearing_index = bearing_index + 1;
end
load handel
sound(y,Fs)

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\testing_data_fourier_spec_norm.mat","testing_series", "testing_labels",'-v7.3')
save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data_fourier_spec_norm.mat","training_series", "training_labels",'-v7.3')
```

36. femto_data_series_constructor_2spectrum.mat

```
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_E\training_data_spectrogram_norm.mat")
%load('D:\FEMTOData\Complete_set\Raw_data\unified_data_E\training_data.mat')
%load('D:\FEMTOData\Complete_set\Raw_data\unified_data_C\training_data_fourier_spec_norm.mat')

current_point_stride_training = 2;
backwards_offset = 2;
num_history = 5; %series length

vibration_ch = 1;

training_bearings = string(cell2mat(fields(processed_training_bearings)));

%TRAINING DATA PREP
bearing_index = 1;
while bearing_index <= length(training_bearings)
  bearing = training_bearings(bearing_index);
  current_data = processed_training_bearings.(bearing);
```

```
    size_current_data = size(current_data);

    alarm_time = getAlarmTime(training_data.(bearing));

    training_series.(bearing) = {};
    training_labels.(bearing) = [];

    current_point = backwards_offset*num_history+1;
    while current_point <= size_current_data(end)

        %if current_point <= alarm_time
        %    current_label = 1;
        %else
        %    current_label = 1 - (current_point-alarm_time)/(size_current_data(end)-alarm_time);
        %end
        current_label = 1-current_point/size_current_data(end);

        size_series = size_current_data;
        size_series(1) = 640;
        size_series(3) = 1;
        size_series(4) = num_history;
        current_sample = zeros(size_series);
        current_sample(:,:,:,1) = current_data(1:640,:,current_point);
        for backwards_point_idx = 2:num_history
            backwards_point = current_point - (backwards_point_idx*backwards_offset);
            current_sample(:,:,:,backwards_point_idx) = current_data(1:640,:,backwards_point);
        end
        %removed top half of frequencies!
        %current_sample = current_sample-mean(mean(current_sample));
        %current_sample = current_sample/mean(std(current_sample));
        training_series.(bearing) = [training_series.(bearing); current_sample];
        training_labels.(bearing) = [training_labels.(bearing); current_label];

        current_point = current_point + current_point_stride_training;
    end

    bearing_index = bearing_index + 1;
end
load handel
sound(y,Fs)

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_E\training_data_2d_spectrum_series.mat","training_series", "training_labels",'-v7.3')

function alarm_time = getAlarmTime(current_data)

    current_data = squeeze(current_data(:,1,:));

    raw_kurt = kurtosis(current_data);
    kurt_hist = histogram(raw_kurt,10000);
    k = 1;
    current_sum = 0;
    while current_sum <= length(raw_kurt)*0.89
        current_sum = current_sum + kurt_hist.Values(k);
        k = k+1;
    end
    alarm = kurt_hist.BinEdges(k);
    alarm_idx = find(raw_kurt>alarm);

    alarm_idx_new = diff(alarm_idx);
    m = 3;
    while m <= length(alarm_idx_new)
        if alarm_idx_new(m) == 1 && alarm_idx_new(m-1) == 1 && alarm_idx_new(m-2) == 1
            break
        end
        %fprintf("%.0f  %.0f  %.0f  \n",alarm_idx_new(m), alarm_idx_new(m-1),alarm_idx_new(m-2))
        m = m+1;
    end

    alarm_time = alarm_idx(m);

end
```

37. femto_data_series_constructor_spectrogram.mat

```
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_E\training_data_spectrogram_norm.mat")
%load('D:\FEMTOData\Complete_set\Raw_data\unified_data_E\training_data.mat')

current_point_stride_training = 2;
backwards_offset = 2;
num_history = 5; %series length

vibration_ch = 1;

training_bearings = string(cell2mat(fields(processed_training_bearings)));

%TRAINING DATA PREP
bearing_index = 1;
while bearing_index <= length(training_bearings)
   bearing = training_bearings(bearing_index);
   current_data = processed_training_bearings.(bearing);
   size_current_data = size(current_data);

   alarm_time = getAlarmTime(training_data.(bearing));

   training_series.(bearing) = {};
   training_labels.(bearing) = [];

   current_point = backwards_offset*num_history+1;
   while current_point <= size_current_data(end)

      if current_point <= alarm_time
         current_label = 1;
      else
         current_label = 1 - (current_point-alarm_time)/(size_current_data(end)-alarm_time);
      end
      %current_label = current_point/size_current_data(end);
      size_series = size_current_data;
      size_series(4) = num_history;
      current_sample = zeros(size_series);
      current_sample(:,:,:,1) = current_data(:,:,:,current_point);
      for backwards_point_idx = [2:num_history]
         backwards_point = current_point - (backwards_point_idx*backwards_offset);
         current_sample(:,:,:,backwards_point_idx) = current_data(:,:,:,backwards_point);
      end
      %current_sample = current_sample-mean(mean(current_sample));
      %current_sample = current_sample/mean(std(current_sample));
      training_series.(bearing) = [training_series.(bearing); current_sample];
      training_labels.(bearing) = [training_labels.(bearing); current_label];

      current_point = current_point + current_point_stride_training;
   end

   bearing_index = bearing_index + 1;
end
load handel
sound(y,Fs)

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_E\training_data_spectrogram_series_v2.mat","training_series", "training_labels",'-v7.3')

function alarm_time = getAlarmTime(current_data)

   current_data = squeeze(current_data(:,1,:));

   raw_kurt = kurtosis(current_data);
   kurt_hist = histogram(raw_kurt,10000);
   k = 1;
   current_sum = 0;
   while current_sum <= length(raw_kurt)*0.89
      current_sum = current_sum + kurt_hist.Values(k);
      k = k+1;
   end
   alarm = kurt_hist.BinEdges(k);
   alarm_idx = find(raw_kurt>alarm);

   alarm_idx_new = diff(alarm_idx);
   m = 3;
   while m <= length(alarm_idx_new)
      if alarm_idx_new(m) == 1 && alarm_idx_new(m-1) == 1 && alarm_idx_new(m-2) == 1
         break
```

```
        end
        %fprintf("%.0f  %.0f  %.0f  \n",alarm_idx_new(m), alarm_idx_new(m-1),alarm_idx_new(m-2))
        m = m+1;
      end

      alarm_time = alarm_idx(m);

    end
```

38. femto_data_series_constructor_spectrogram_fpt.mat

```
    load("D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\training_data_spectrogram_norm_fpt_trimmed.mat")
    %load('D:\FEMTOData\Complete_set\Raw_data\unified_data_E\training_data.mat')

    current_point_stride_training = 1;
    backwards_offset = 1;
    num_history = 5; %series length
    start_point = backwards_offset*num_history+1;

    vibration_ch = 1;

    training_bearings = string(cell2mat(fields(processed_training_bearings)));

    %TRAINING DATA PREP
    bearing_index = 1;
    while bearing_index <= length(training_bearings)
      bearing = training_bearings(bearing_index);
      current_data = processed_training_bearings.(bearing);
      size_current_data = size(current_data);

      training_series.(bearing) = {};
      training_labels.(bearing) = [];

      current_point = start_point;
      while current_point <= size_current_data(end)

        current_label = 1 - (current_point-start_point)/(size_current_data(end)-start_point);

        %current_label = current_point/size_current_data(end);
        size_series = size_current_data;
        size_series(4) = num_history;
        current_sample = zeros(size_series);
        current_sample(:,:,:,1) = current_data(:,:,:,current_point);
        for backwards_point_idx = [2:num_history]
          backwards_point = current_point - (backwards_point_idx*backwards_offset);
          current_sample(:,:,:,backwards_point_idx) = current_data(:,:,:,backwards_point);
        end
        %current_sample = current_sample-mean(mean(current_sample));
        %current_sample = current_sample/mean(std(current_sample));
        training_series.(bearing) = [training_series.(bearing); current_sample];
        training_labels.(bearing) = [training_labels.(bearing); current_label];

        current_point = current_point + current_point_stride_training;
      end

      bearing_index = bearing_index + 1;
    end
    load handel
    sound(y,Fs)

    save("D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\training_data_spectrogram_fpt_trimmed_series.mat","training_series", "training_labels",'-
v7.3')
```

39. femto_data_series_constructor_v2.mat

```
    %load("D:\FEMTOData\Complete_set\Raw_data\unified_data_C\training_data_fourier_spec_norm.mat")
    %load('D:\FEMTOData\Complete_set\Raw_data\unified_data_C\training_data.mat')

    current_point_stride_training = 2;
    backwards_offset = 11;

    vibration_ch = 1;
```

```
training_bearings = string(cell2mat(fields(processed_training_bearings)));

%TRAINING DATA PREP
bearing_index = 1;
while bearing_index <= length(training_bearings)
   bearing = training_bearings(bearing_index);
   current_data = processed_training_bearings.(bearing);
   current_data = current_data(:,vibration_ch,:);
   size_current_data = size(current_data);

   alarm_time = getAlarmTime(training_data.(bearing));

   training_series.(bearing) = {};
   training_labels.(bearing) = [];

   current_point = 1;
   while current_point <= size_current_data(end)

      if current_point <= alarm_time
         current_label = 1;
      else
         current_label = 1 - (current_point-alarm_time)/(size_current_data(end)-alarm_time);
      end
      %current_label = current_point/size_current_data(end);
      current_sample = current_data(:,:,current_point);
      backwards_point = current_point - backwards_offset;
      while backwards_point > 1
         current_sample = [current_sample, current_data(:,:,backwards_point)];
         %current_label = [current_label, backwards_point/size_current_data(end)];
         backwards_point = backwards_point-backwards_offset;
      end
      current_sample = current_sample-mean(mean(current_sample));
      current_sample = current_sample/mean(std(current_sample));
      training_series.(bearing) = [training_series.(bearing); current_sample];
      training_labels.(bearing) = [training_labels.(bearing); current_label];

      current_point = current_point + current_point_stride_training;
   end

   bearing_index = bearing_index + 1;
end
load handel
sound(y,Fs)

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_D\training_data_fourier_spec_norm_series.mat","training_series", "training_labels",'-v7.3')

function alarm_time = getAlarmTime(current_data)

   current_data = squeeze(current_data(:,1,:));

   raw_kurt = kurtosis(current_data);
   kurt_hist = histogram(raw_kurt,10000);
   k = 1;
   current_sum = 0;
   while current_sum <= length(raw_kurt)*0.89
      current_sum = current_sum + kurt_hist.Values(k);
      k = k+1;
   end
   alarm = kurt_hist.BinEdges(k);
   alarm_idx = find(raw_kurt>alarm);

   alarm_idx_new = diff(alarm_idx);
   m = 3;
   while m <= length(alarm_idx_new)
      if alarm_idx_new(m) == 1 && alarm_idx_new(m-1) == 1 && alarm_idx_new(m-2) == 1
         break
      end
      %fprintf("%.0f  %.0f  %.0f  \n",alarm_idx_new(m), alarm_idx_new(m-1),alarm_idx_new(m-2))
      m = m+1;
   end

   alarm_time = alarm_idx(m);

end
```

40. femto_data_series_constructor_v3.mat

```
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_C\training_data_fourier_spec_norm.mat")
%load('D:\FEMTOData\Complete_set\Raw_data\unified_data_C\training_data.mat')

current_point_stride_training = 2;
backwards_offset = 2;
num_history = 5; %series length

vibration_ch = 1;

training_bearings = string(cell2mat(fields(processed_training_bearings)));

%TRAINING DATA PREP
bearing_index = 1;
while bearing_index <= length(training_bearings)
   bearing = training_bearings(bearing_index);
   current_data = processed_training_bearings.(bearing);
   current_data = current_data(:,vibration_ch,:);
   size_current_data = size(current_data);

   alarm_time = getAlarmTime(training_data.(bearing));

   training_series.(bearing) = {};
   training_labels.(bearing) = [];

   current_point = backwards_offset*num_history;
   while current_point <= size_current_data(end)

      if current_point <= alarm_time
         current_label = 1;
      else
         current_label = 1 - (current_point-alarm_time)/(size_current_data(end)-alarm_time);
      end
      %current_label = current_point/size_current_data(end);
      current_sample = current_data(:,:,current_point);
      for backwards_point_idx = [1:num_history-1]
         backwards_point = current_point - (backwards_point_idx*backwards_offset);
         current_sample = [current_sample, current_data(:,:,backwards_point)];
      end
      current_sample = current_sample-mean(mean(current_sample));
      current_sample = current_sample/mean(std(current_sample));
      training_series.(bearing) = [training_series.(bearing); current_sample];
      training_labels.(bearing) = [training_labels.(bearing); current_label];

      current_point = current_point + current_point_stride_training;
   end

   bearing_index = bearing_index + 1;
end
load handel
sound(y,Fs)

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_E\training_data_fourier_spec_norm_series.mat","training_series", "training_labels",'-v7.3')

function alarm_time = getAlarmTime(current_data)

   current_data = squeeze(current_data(:,1,:));

   raw_kurt = kurtosis(current_data);
   kurt_hist = histogram(raw_kurt,10000);
   k = 1;
   current_sum = 0;
   while current_sum <= length(raw_kurt)*0.89
      current_sum = current_sum + kurt_hist.Values(k);
      k = k+1;
   end
   alarm = kurt_hist.BinEdges(k);
   alarm_idx = find(raw_kurt>alarm);

   alarm_idx_new = diff(alarm_idx);
   m = 3;
   while m <= length(alarm_idx_new)
      if alarm_idx_new(m) == 1 && alarm_idx_new(m-1) == 1 && alarm_idx_new(m-2) == 1
```

```
            break
        end
        %fprintf("%.0f  %.0f  %.0f  \n",alarm_idx_new(m), alarm_idx_new(m-1),alarm_idx_new(m-2))
        m = m+1;
    end

    alarm_time = alarm_idx(m);

end
```

41. femto_data_series_constructor_v3_fpt_trimmed.mat

```
load("D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\training_data_fourier_spec_norm_fpt_trimmed.mat")
%load('D:\FEMTOData\Complete_set\Raw_data\unified_data_C\training_data.mat')

current_point_stride_training = 1;
backwards_offset = 1;
num_history = 5; %series length
start_point = backwards_offset*num_history;

vibration_ch = 1;

training_bearings = string(cell2mat(fields(processed_training_bearings)));

%TRAINING DATA PREP
bearing_index = 1;
while bearing_index <= length(training_bearings)
    bearing = training_bearings(bearing_index);
    current_data = processed_training_bearings.(bearing);
    current_data = current_data(:,vibration_ch,:);
    size_current_data = size(current_data);

    training_series.(bearing) = {};
    training_labels.(bearing) = [];

    current_point = start_point;
    while current_point <= size_current_data(end)

        current_label = 1 - (current_point-start_point)/(size_current_data(end)-start_point);

        %current_label = current_point/size_current_data(end);
        current_sample = current_data(:,:,current_point);
        for backwards_point_idx = [1:num_history-1]
            backwards_point = current_point - (backwards_point_idx*backwards_offset);
            current_sample = [current_sample, current_data(:,:,backwards_point)];
        end
        current_sample = current_sample-mean(mean(current_sample));
        current_sample = current_sample/mean(std(current_sample));
        training_series.(bearing) = [training_series.(bearing); current_sample];
        training_labels.(bearing) = [training_labels.(bearing); current_label];

        current_point = current_point + current_point_stride_training;
    end

    bearing_index = bearing_index + 1;
end
load handel
sound(y,Fs)

save("D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\training_data_fourier_spec_norm_fpt_trimmed_series.mat","training_series",
"training_labels",'-v7.3')
```

42. femto_data_unifier.mat

```
root_folder = "D:\FEMTOData\Complete_set\Raw_data";
training_subfolders = [];
training_subfolders = ["Bearing1_1","Bearing1_2","Bearing1_3","Bearing1_4","Bearing1_5","Bearing1_6","Bearing1_7",...
                "Bearing2_1","Bearing2_2","Bearing2_3","Bearing2_4","Bearing2_5","Bearing2_6","Bearing2_7",...
                "Bearing3_1","Bearing3_2","Bearing3_3"];

for subfolder = training_subfolders
```

```
    path = strcat(root_folder,"\",subfolder,"\");
    fileList = dir(fullfile(path, 'acc*.csv'));
    length_recording = length(load(strcat(path,fileList(1).name)));
    training_data.(subfolder) = zeros(length_recording,2,length(fileList));
    index = 1;
        while index <= length(fileList)
            current_recording = load(strcat(path,fileList(index).name));
            current_recording = current_recording(:,5:6);
            training_data.(subfolder)(:,:,index) = current_recording;
            index = index + 1;
        end
end

for subfolder = testing_subfolders

    path = strcat(root_folder,"\",subfolder,"\");
    fileList = dir(fullfile(path, 'acc*.csv'));
    length_recording = length(load(strcat(path,fileList(1).name)));
    testing_data.(subfolder) = zeros(length_recording,2,length(fileList));
    index = 1;
        while index <= length(fileList)
            current_recording = load(strcat(path,fileList(index).name));
            current_recording = current_recording(:,5:6);
            testing_data.(subfolder)(:,:,index) = current_recording;
            index = index + 1;
        end
end

%save("D:\FEMTOData\Complete_set\Raw_data\unified_data_3\training_data.mat","training_data")
save("D:\FEMTOData\Complete_set\Raw_data\unified_data_C\training_data.mat","training_data")
```

43. femto_rul_plotter.mat

```
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\testing_fourier_series_v7.mat")
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\training_fourier_series_v7.mat")
%load("D:\master_paper_3\saved_network_architectures_3\trained_lstm_v33_specta_v1.mat","net")

%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\testing_manual_features_1_series_v1.mat","testing_series", "testing_labels")
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\training_manual_features_1_series_v1.mat","training_series", "training_labels")
%network = load("D:\master_paper_3\saved_network_architectures_2\lstm_16_v2.mat");

testing_bearings = string(cell2mat(fields(testing_series)));
%training_bearings = string(cell2mat(fields(training_series)));

data_ch = 1;

bearing_index = 1;
    while bearing_index <= length(testing_bearings)
        bearing = testing_bearings(bearing_index);

        current_data = testing_series.(bearing);
        current_labels = testing_labels.(bearing);

        predictions = predict(net,current_data);

        plot(transpose(cell2mat(current_data(end))))

        %{
        predictions = zeros(size(current_labels));
        end_point = 10;
        while end_point <= length(current_labels)
            predictions(end_point-9:end_point) = predict(net,current_data(end_point-9:end_point));
            end_point = end_point + 10;
        end
        if end_point-10 < length(current_labels)
        predictions(end_point-10:end) = predict(net,current_data(end_point-10:end));
        end
        %}
        plot(current_labels, predictions)
        ylim([0,1])
        display(bearing_index)

        bearing_index = bearing_index + 1;
```

```
    end
```

44. femto_rul_plotter_composite.mat

```
%load('D:\master_paper_4\femto_data\training_data.mat')
%load('D:\master_paper_4\femto_data\training_data_fourier_spec_norm.mat')

save_location = "C:\Users\jakeh\Documents\Masters\Paper 4\MATLAB_figs\composite_no_smoothing\";


bearing_list = sort(string(fields(training_data)));
j = 1;
net_save_prefix = "stcnn_v01";
alarm_time_list = [2112,708,2024,1079,2398,1627,1961,29,748,261,333,398,662,219,494,1429,421];



while j <= length(bearing_list)

bearing = bearing_list(j);

ch_data = squeeze(training_data.(bearing)(:,1,:));

size_ch_data = size(ch_data);

current_data = processed_training_bearings.(bearing);
net_subfolders_list = ["none_v01", "gaussian_noise_v01" , "region_dropout_v02" , "random_dropout_v02" , "pitch_shift_v02"];
alarm_time = alarm_time_list(j);

for net_subfolder = net_subfolders_list

   net_folder = strcat("D:\master_paper_4\femto_data\trained_nets\", net_subfolder, "\");
   load(strcat(net_folder,net_save_prefix,bearing,".mat"));

   predictions = predict(net,current_data);
   hold on
   %plot(smoothdata(predictions,'loess',[20,0]))
   plot(predictions)

end

size_current_data = size(current_data);
true_rul = ones(size_current_data(end),1);
true_rul(alarm_time:end) = transpose(linspace(1,0,size_current_data(end)-alarm_time+1));
plot(true_rul)

xlim([alarm_time,length(true_rul)])

%xlim([1,length(training_labels.(bearing))])
%yline(0,'HandleVisibility','off')

j = j + 1;

title(bearing)
x0 = 550;
y0 = 250;
width = 675;
height= 550;

legend(["None", "GN" , "RegD" , "RanD" , "PS" , "True HI" ], 'Location','northwest')
set(gcf,'position',[x0,y0,width,height])
xlabel("Measurement")
ylabel("Health indicator")
saveas(gcf,strcat(save_location,bearing),"png")
set(gca,'fontname','times')

clf

end
```

45. femto_rul_plotter_v2.mat

```
%load('D:\master_paper_4\femto_data\training_data.mat')
%load('D:\master_paper_4\femto_data\training_data_fourier_spec_norm.mat')

save_location = "D:\master_paper_4\femto_data\save_plots\none_v01";

bearing_list = sort(string(fields(training_data)));
j = 1;
net_save_prefix = "stcnn_v01";

while j <= length(bearing_list)

bearing = bearing_list(j);

ch_data = squeeze(training_data.(bearing)(:,1,:));

size_ch_data = size(ch_data);

subplot(2,1,1)

i = 1;
plot_start = 1;
hold on
while i <= size_ch_data(end)
   data = ch_data(:,i);
   plot([plot_start:plot_start+length(data)-1],data, 'b')
   plot_start = plot_start+length(data);
   i = i +1;
end
xlim([1,plot_start])
title_text = char(bearing);
title_text(9) = ' ';
title(title_text)

subplot(2,1,2)
net_folder = "D:\master_paper_4\femto_data\trained_nets\none_v01\";
load(strcat(net_folder,net_save_prefix,bearing,".mat"));

current_data = processed_training_bearings.(bearing);
predictions = predict(net,current_data);
plot(predictions)
hold on
plot(smoothdata(predictions,'loess',[10,0]))
%plot(training_labels.(bearing))
legend(["Raw estimation","Smoothed Estimation","Alarm threshold"])
%xlim([1,length(training_labels.(bearing))])
yline(0)
j = j + 1;

saveas(gcf,strcat(save_location,bearing,'.png'))

clf

end
```

46. femto_rul_plotter_v3.mat

```
%load('D:\master_paper_4\femto_data\training_data.mat')
%load('D:\master_paper_4\femto_data\training_data_fourier_spec_norm.mat')

save_location = "C:\Users\jakeh\Documents\Masters\Paper 4\MATLAB_figs\full_lifetime_ps\";

bearing_list = sort(string(fields(training_data)));
j = 1;
net_save_prefix = "stcnn_v01";

while j <= length(bearing_list)

bearing = bearing_list(j);

ch_data = squeeze(training_data.(bearing)(:,1,:));

size_ch_data = size(ch_data);

net_folder = "D:\master_paper_4\femto_data\trained_nets\random_dropout_v02\";
```

```
load(strcat(net_folder,net_save_prefix,bearing,".mat"));

current_data = processed_training_bearings.(bearing);
predictions = predict(net,current_data);
plot(predictions)
hold on
plot(smoothdata(predictions,'loess',[100,0]))
%plot(training_labels.(bearing))
legend(["Raw estimation","Smoothed Estimation","Alarm threshold"])
%xlim([1,length(training_labels.(bearing))])
yline(0,'HandleVisibility','off')
j = j + 1;

x0 = 550;
y0 = 250;
width = 675;
height= 550;

set(gcf,'position',[x0,y0,width,height])

saveas(gcf,strcat(save_location,bearing,'.png'))



clf

end
```

47. femto_series_network_trainer.mat

```
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_5\testing_fourier_series_v1.mat")
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_5\training_fourier_series_v1.mat")
network = load("D:\master_paper_3\saved_network_architectures_2\lstm_v34.mat");

%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\testing_manual_features_1_series_v1.mat","testing_series", "testing_labels")
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_2\training_manual_features_1_series_v1.mat","training_series", "training_labels")
%network = load("D:\master_paper_3\saved_network_architectures_2\lstm_16_v3.mat");

network_fields = string(fields(network));
network = network.(network_fields);
%%%
%network = network.Layers;
%%%
testing_bearings = string(cell2mat(fields(testing_series)));
training_bearings = string(cell2mat(fields(training_series)));

merged_testing_series = {};
merged_testing_labels = [];
bearing_index = 1;
while bearing_index <= length(testing_bearings)
    bearing = testing_bearings(bearing_index);
    merged_testing_series = [merged_testing_series; testing_series.(bearing)];
    merged_testing_labels = [merged_testing_labels; testing_labels.(bearing)];

    bearing_index = bearing_index + 1;
end

merged_training_series = {};
merged_training_labels = [];
bearing_index = 1;
while bearing_index <= length(training_bearings)
    bearing = training_bearings(bearing_index);
    merged_training_series = [merged_training_series; training_series.(bearing)];
    merged_training_labels = [merged_training_labels; training_labels.(bearing)];

    bearing_index = bearing_index + 1;
end

[m,n] = size(merged_testing_series);
idx = randperm(m) ;
merged_testing_series = merged_testing_series(idx);
merged_testing_labels = merged_testing_labels(idx);

[m,n] = size(merged_training_series);
```

```
idx = randperm(m) ;
merged_training_series = merged_training_series(idx);
merged_training_labels = merged_training_labels(idx);

options = trainingOptions('adam', ...
   'LearnRateSchedule','piecewise', ...
   'LearnRateDropFactor',0.2, ...
   'LearnRateDropPeriod',2, ...
   'MaxEpochs',20, ...
   'MiniBatchSize',32, ...
   'Plots','training-progress', ...
   'ValidationData', {merged_testing_series,merged_testing_labels}, ...
   'ValidationFrequency', 50);

net = trainNetwork(merged_training_series,merged_training_labels,network,options);

%save("D:\master_paper_3\saved_network_architectures_4\trained_lstm_manual_features_1_v3.mat","net")
save("D:\master_paper_3\saved_network_architectures_4\trained_lstm_fourier_v5.mat","net")
%save("D:\master_paper_3\saved_network_architectures_2\lstm_v35.mat","layers_5")
```

48. femto_series_network_trainer_v2.mat

```
%load("D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\training_data_fourier_spec_norm_fpt_trimmed_series.mat")
load("D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\training_data_spectrogram_fpt_trimmed_series.mat")
%save("D:\master_paper_3\saved_network_architectures_fpt_trimmed\lstm_spectrum_v3.mat","layers_1")
network = load('D:\master_paper_3\saved_network_architectures_fpt_trimmed\lstm_spectrogram_v2.mat');
%network = load("D:\master_paper_3\saved_network_architectures_fpt_trimmed\lstm_spectrum_v3.mat");

network_fields = string(fields(network));
network = network.(network_fields);

all_bearings = sort(string(cell2mat(fields(training_series))));
i = 1;
while i <= length(all_bearings)

   test_bearing = all_bearings(i);

   testing_series.(test_bearing) = training_series.(test_bearing);
   testing_labels.(test_bearing) = training_labels.(test_bearing);

   training_series = rmfield(training_series,test_bearing);
   training_labels =  rmfield(training_labels,test_bearing);

   testing_bearings = string(cell2mat(fields(testing_series)));
   training_bearings = string(cell2mat(fields(training_series)));

   merged_testing_series = {};
   merged_testing_labels = [];
   bearing_index = 1;

   while bearing_index <= length(testing_bearings)
      bearing = testing_bearings(bearing_index);
      merged_testing_series = [merged_testing_series; testing_series.(bearing)];
      merged_testing_labels = [merged_testing_labels; testing_labels.(bearing)];

      bearing_index = bearing_index + 1;
   end

   merged_training_series = {};
   merged_training_labels = [];
   bearing_index = 1;
   while bearing_index <= length(training_bearings)
      bearing = training_bearings(bearing_index);
      merged_training_series = [merged_training_series; training_series.(bearing)];
      merged_training_labels = [merged_training_labels; training_labels.(bearing)];

      bearing_index = bearing_index + 1;
   end

   [m,n] = size(merged_testing_series);
   idx = randperm(m) ;
   merged_testing_series = merged_testing_series(idx);
   merged_testing_labels = merged_testing_labels(idx);
```

```
    [m,n] = size(merged_training_series);
    idx = randperm(m) ;
    merged_training_series = merged_training_series(idx);
    merged_training_labels = merged_training_labels(idx);

    options = trainingOptions('adam', ...
      'InitialLearnRate', 0.001, ...
      'LearnRateSchedule','piecewise', ...
      'LearnRateDropFactor',0.2, ...
      'LearnRateDropPeriod',2, ...
      'MaxEpochs',10, ...
      'MiniBatchSize',32, ...
      'Plots','training-progress', ...
      'ValidationData', {merged_testing_series,merged_testing_labels}, ...
      'ValidationFrequency', 50);

    [net,info] = trainNetwork(merged_training_series,merged_training_labels,network,options);

    fileID = fopen("D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\c\10_epochs.txt",'a');
    fprintf(fileID,"%s    Validation RMSE  %f \n",test_bearing, info.ValidationRMSE(end));


save(strcat("D:\FEMTOData\Complete_set\Raw_data\unified_data_fpt\spectrum_fpt_trimmed_better_labeling\trained_lstm_spectrum_",test_bearing,".mat"),"net", "info")

    training_series.(test_bearing) = testing_series.(test_bearing);
    training_labels.(test_bearing) = testing_labels.(test_bearing);

    clear("testing_series","testing_labels")

    i = i + 1;
end
```

49. femto_timeline_plotter.mat

```
%load net_v05.mat

bearing_id = 'Bearing1_1';

test_spectrogram_folder = strcat('G:\FEMTOData\Test_002\0_noise\', bearing_id);
test_timeseries_folder = strcat('G:\FEMTOData\Complete_set\Raw_data\', bearing_id);

test_images = imageDatastore(test_spectrogram_folder,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames');

% Convert labels to categoricals
test_images.Labels = categorical(test_images.Labels);
test_images.ReadFcn = @readFcn1;

%Sometimes this will not run on GPU, sometimes it can. idk
test_classes = classify(net_v06, test_images);
%,'ExecutionEnvironment','cpu'

files = dir(fullfile(test_timeseries_folder, 'acc*.csv'));
file_names = strings(1,length(files));

i = 1;
length_files = length(files);

while i <= length_files

  file_names(i) = files(i).name;

 i = i+1;
end

i = 1;
while i <= length_files

  read_path = strcat(files(i).folder,'\', files(i).name);
  temp = load(read_path);
  ch1 = temp(:,5);
  ch2 = temp(:,6);

  if strcmp(string(test_classes(i*2-1)), '0-50')
```

```
        colour1 = 'g';
        pause(0.001)
    elseif strcmp(string(test_classes(i*2-1)), '50-80')
        colour1 = 'y';
        pause(0.001)
    elseif strcmp(string(test_classes(i*2-1)), '80-100')
            colour1 = 'r';
            pause(0.001)
    end

    if strcmp(string(test_classes(i*2)), '0-50')
        colour2 = 'g';
        pause(0.001)
    elseif strcmp(string(test_classes(i*2)), '50-80')
        colour2 = 'y';
        pause(0.001)
    elseif strcmp(string(test_classes(i*2)), '80-100')
            colour2 = 'r';
            pause(0.001)
    end

    subplot(2,1,1)
    hold on
    plot((1+(i-1)*2560:i*2560),ch1,colour1)
    pause(0.0001);
    title(strcat(bearing_id, ' Channel 1 Classification Results'));

    subplot(2,1,2)
    hold on
    plot((1+(i-1)*2560:i*2560),ch2,colour2)
    pause(0.001)
    title(strcat(bearing_id, ' Channel 2 Classification Results'));

    fprintf('%05d / %05d \n' , i, length_files);
    i = i+5;

end
```

50. femto_timeline_regression_2D_3D.mat

```
save_figures = false;
snr_label = "snr_inf";
snr_val = 0;
figure_file_destination = strcat('D:\FEMTOData\Test_010\net_reg_v09\',snr_label,'\');
mkdir(figure_file_destination)

load net3D_reg_v09.mat
regnet_3d = net3D_reg_v09;

load net2D_reg_v09_ch1.mat
regnet_ch1 = net2D_reg_v09_ch1;

load net2D_reg_v09_ch2.mat
regnet_ch2 = net2D_reg_v09_ch2;

list_bearing_id = ["bearing1_1","bearing1_2","bearing1_3",...
    "bearing1_4","bearing1_5","bearing1_6","bearing1_7",...
    "bearing2_1","bearing2_2","bearing2_3","bearing2_4",...
    "bearing2_5","bearing2_6","bearing2_7","bearing3_1",...
    "bearing3_2","bearing3_3"];

list_bearing_id = ["bearing1_7"];

%predict_rate determines how many spectrograms to skip before reading and
%making another prediction. must be >= 1
predict_rate = 10;

for bearing_id = list_bearing_id

    test_spectrogram_folder = strcat('D:\FEMTOData\Test_010\',snr_label,'\', bearing_id);
    test_images = imageDatastore(test_spectrogram_folder,'FileExtensions', '.mat','IncludeSubfolders', true, 'LabelSource', 'foldernames');

    % Convert labels to categoricals
```

```
test_images.Labels = categorical(test_images.Labels);
test_images.ReadFcn = @readFcn1;

%sort sequence of test images. It gets confused due to subfolder locations
names = test_images.Files;
numbers=cellfun(@extract_number,names);
[~,order]=sort(numbers);
names=names(order);

tallied_predictions3d = [];
tallied_predictionsch1 = [];
tallied_predictionsch2 = [];

%close
%figure('units','normalized','outerposition',[0 0 1 1])
clf

subplot(4,1,1)
set(gca, 'FontName', 'Times New Roman')
hold on
%title(strcat(bearing_id," ",snr_label, ' Raw Vibration Signals (downsampled 10x)'));

if strcmp(snr_label,'snr_inf')
   load(strcat('D:\FEMTOData\Complete_set\downsampled_concat_raw_data\',bearing_id,'_ch2.mat'))
   plot(downsampled_ch2,'-b')
   hold on
   load(strcat('D:\FEMTOData\Complete_set\downsampled_concat_raw_data\',bearing_id,'_ch1.mat'))
   plot(downsampled_ch1,'-r')
   legend("CH2","CH1");
   legend('Location','northwest')
   xlim([0, length(downsampled_ch1)]);
else
   load(strcat('D:\FEMTOData\Complete_set\downsampled_concat_raw_data\',bearing_id,'_ch1.mat'))
   load(strcat('D:\FEMTOData\Complete_set\downsampled_concat_raw_data\',bearing_id,'_ch2.mat'))
   downsampled_ch1_noisy = awgn(downsampled_ch1,snr_val);
   downsampled_ch2_noisy = awgn(downsampled_ch2,snr_val);
   plot(downsampled_ch1_noisy,'-b')
   hold on
   plot(downsampled_ch2_noisy,'-r')
   plot(downsampled_ch1,'-k')
   plot(downsampled_ch2,'-k')
   legend("Noisy CH1","Noisy CH2","Original CH1","Original CH2");
   legend('Location','northwest')
   xlim([0, length(downsampled_ch1)]);
end

subplot(4,1,2)
set(gca, 'FontName', 'Times New Roman')
hold on
%title(strcat(bearing_id, ' CH1&CH2 Combined Regression Results'));
ylim([-0.25,1.25]);
xlim([0, length(names)]);
yline(1,'--g')
yline(0,'--r')

subplot(4,1,3)
set(gca, 'FontName', 'Times New Roman')
hold on
%title(strcat(bearing_id, ' CH1 Only Regression Results'));
ylim([-0.25,1.25]);
xlim([0, length(names)]);
yline(1,'--g')
yline(0,'--r')

subplot(4,1,4)
set(gca, 'FontName', 'Times New Roman')
hold on
%title(strcat(bearing_id, ' CH2 Only Regression Results'));
ylim([-0.25,1.25]);
xlim([0, length(names)]);
yline(1,'--g')
yline(0,'--r')

i = 1;
while i <= length(names)
```

```
    prediction3d = predict(regnet_3d, readFcn2(string(names(i))));
    tallied_predictions3d = vector_concatonator(tallied_predictions3d,prediction3d);

    predictionch1 = predict(regnet_ch1, readFcn1_ch1(string(names(i))));
    tallied_predictionsch1 = vector_concatonator(tallied_predictionsch1,predictionch1);

    predictionch2 = predict(regnet_ch2, readFcn1_ch2(string(names(i))));
    tallied_predictionsch2 = vector_concatonator(tallied_predictionsch2,predictionch2);

    subplot(4,1,2)
    hold on
    %scatter((1+(i-1)*2560),prediction3d,'b')
    scatter(i,prediction3d,'b')
    pause(0.0001)

    subplot(4,1,3)
    hold on
    %scatter((1+(i-1)*2560),prediction3d,'b')
    scatter(i,predictionch1,'b')
    pause(0.0001)

    subplot(4,1,4)
    hold on
    %scatter((1+(i-1)*2560),prediction3d,'b')
    scatter(i,predictionch2,'b')
    pause(0.0001)

    fprintf('%05d / %05d \n' , i, length(names));
    i = i+predict_rate;

  end

  subplot(4,1,2)
  cla();
  hold on
  smoothed_predictions3d = smooth(tallied_predictions3d,'moving',[round(100/predict_rate),0]);
  plot(tallied_predictions3d,'o')
  plot(smoothed_predictions3d,'k');
  ylim([-0.25,1.25]);
  xlim([0, length(tallied_predictions3d)]);
  yline(1,'--g')
  yline(0,'--r')

  subplot(4,1,3)
  cla();
  hold on
  smoothed_predictionsch1 = smooth(tallied_predictionsch1,'moving',[round(100/predict_rate),0]);
  plot(tallied_predictionsch1,'o')
  plot(smoothed_predictionsch1,'k');
  ylim([-0.25,1.25]);
  xlim([0, length(tallied_predictionsch1)]);
  yline(1,'--g')
  yline(0,'--r')

  subplot(4,1,4)
  cla();
  hold on
  smoothed_predictionsch2 = smooth(tallied_predictionsch2,'moving',[round(100/predict_rate),0]);
  plot(tallied_predictionsch2,'o')
  plot(smoothed_predictionsch2,'k');
  ylim([-0.25,1.25]);
  xlim([0, length(tallied_predictionsch2)]);
  yline(1,'--g')
  yline(0,'--r')
  %legend("Instantaneous Prediction", "Smoothed Prediction", "Healthy Threshold", "Failure Threshold");
  %legend('Location','southoutside')

  if save_figures == true
    saveas(gcf,strcat(figure_file_destination,bearing_id,'.png'))
  end

end
```

51. generalized_preprocessor.mat

```
function processed_data = generalized_preprocessor(original_data,preprocessor,spectrogram_window)

if strcmp(preprocessor,'fourier_spec')
   processed_data = fourier_transform(original_data);
elseif strcmp(preprocessor,'std_normalize')
   processed_data = std_normalize(original_data);
elseif strcmp(preprocessor,'env_spec')
   processed_data = env_spec(original_data);
elseif strcmp(preprocessor,'spectrogram')
   processed_data = spectrogramdata(original_data,spectrogram_window);
elseif strcmp(preprocessor,'scalogram')
   processed_data = scalogram(original_data,scalogram_bins);
elseif strcmp(preprocessor,'2D_spectrogram')
   processed_data = spectrogram_2d(original_data,spectrogram_window);
elseif strcmp(preprocessor,'2D_spectrogram_v2')
   processed_data = spectrogram_2d_v2(original_data,spectrogram_window);
end


function processed_data = spectrogram_2d_v2(original_data,spectrogram_window)
original_dim = size(original_data);
processed_dim = size(spectrogram(original_data(:,1,1,2),spectrogram_window,54,452));
processed_data = zeros(processed_dim(1),processed_dim(2),2,original_dim(end));
for j = [1,2]
i = 1;
while i <= original_dim(end)
   for chan = 1:1:original_dim(2)
      %full_spec = abs(fft(original_data(:,chan,1,i)));
      processed_data(:,:,chan,i) = abs(spectrogram(original_data(:,chan,j,i),spectrogram_window,54,452));
      %average = mean(mean(processed_data(:,:,chan,i)));
      %processed_data(:,:,chan,i) = processed_data(:,:,chan,i)-average;
      %standard_dev = std(std(processed_data(:,:,chan,i)));
      %processed_data(:,:,chan,i) = processed_data(:,:,chan,i)/standard_dev;
   end
   i = i+1;
end
end
processed_data = processed_data(1:224,1:224,:,:);

end


function processed_data = spectrogram_2d(original_data,spectrogram_window)
original_dim = size(original_data);
processed_dim = size(spectrogram(original_data(:,1,1,2),spectrogram_window,54,452));
processed_data = zeros(processed_dim(1),processed_dim(2),2,original_dim(end));
for j = [1,2]
i = 1;
while i <= original_dim(end)
   for chan = 1:1:original_dim(2)
      %full_spec = abs(fft(original_data(:,chan,1,i)));
      processed_data(:,:,chan,i) = abs(spectrogram(original_data(:,chan,j,i),spectrogram_window,54,452));
      %average = mean(mean(processed_data(:,:,chan,i)));
      %processed_data(:,:,chan,i) = processed_data(:,:,chan,i)-average;
      %standard_dev = std(std(processed_data(:,:,chan,i)));
      %processed_data(:,:,chan,i) = processed_data(:,:,chan,i)/standard_dev;
   end
   i = i+1;
end
end
processed_data(:,228,:,:) = [];

end

function processed_data = std_normalize(original_data)
   original_dim = size(original_data);
   processed_dim = original_dim;
   processed_data = original_data;
   i = 1;
   while i <= original_dim(end)
      for chan = 1:1:processed_dim(2)
         average = mean(mean(processed_data(:,:,chan,i)));
```

```
        processed_data(:,:,chan,i) = processed_data(:,:,chan,i)-average;
        standard_dev = std(processed_data(:,:,chan,i));
        processed_data(:,:,chan,i) = processed_data(:,:,chan,i)/standard_dev;
      end
      i = i+1;
    end
end

function processed_data = fourier_transform(original_data)
    original_dim = size(original_data);
    processed_dim = [0.5 1 1 1].*original_dim;
    processed_dim(1) = floor(processed_dim(1));
    processed_data = zeros(processed_dim);
    i = 1;
    while i <= original_dim(end)
      for chan = 1:1:processed_dim(2)
        full_spec = abs(fft(original_data(:,chan,1,i)));
        processed_data(:,chan,1,i) = full_spec(1:processed_dim(1));
        average = mean(mean(processed_data(:,:,chan,i)));
        processed_data(:,:,chan,i) = processed_data(:,:,chan,i)-average;
        standard_dev = std(processed_data(:,:,chan,i));
        processed_data(:,:,chan,i) = processed_data(:,:,chan,i)/standard_dev;
      end
      i = i+1;
    end
end

function processed_data = env_spec(original_data)
original_dim = size(original_data);
processed_dim = size(envspectrum(original_data(:,1,1,1),1));
processed_dim(1) = floor(processed_dim(1));
processed_data = zeros(processed_dim);
i = 1;
while i <= original_dim(end)
    for chan = 1:1:original_dim(2)
      %full_spec = abs(fft(original_data(:,chan,1,i)));
      processed_data(:,1,chan,i) = envspectrum(original_data(:,chan,1,i),1);
      average = mean(mean(processed_data(:,:,chan,i)));
      processed_data(:,:,chan,i) = processed_data(:,:,chan,i)-average;
      standard_dev = std(processed_data(:,:,chan,i));
      processed_data(:,:,chan,i) = processed_data(:,:,chan,i)/standard_dev;
    end
    i = i+1;
end

end

function processed_data = spectrogramdata(original_data,window)
original_dim = size(original_data);
processed_dim = size(spectrogram(original_data(:,1,1,1),window));
processed_data = zeros(processed_dim(1),processed_dim(2),1,original_dim(end));
i = 1;
while i <= original_dim(end)
    for chan = 1:1:original_dim(2)
      %full_spec = abs(fft(original_data(:,chan,1,i)));
      processed_data(:,:,chan,i) = abs(spectrogram(original_data(:,chan,1,i),window));
      %average = mean(mean(processed_data(:,:,chan,i)));
      %processed_data(:,:,chan,i) = processed_data(:,:,chan,i)-average;
      %standard_dev = std(std(processed_data(:,:,chan,i)));
      %processed_data(:,:,chan,i) = processed_data(:,:,chan,i)/standard_dev;
    end
    i = i+1;
end

end

function processed_data = scalogram(original_data,scalogram_bins)
original_dim = size(original_data);
processed_dim = size(cwt(original_data(:,1,1,1),scalogram_bins,'cgau4'));
processed_data = zeros(processed_dim(1),processed_dim(2),1,original_dim(end));
i = 1;
while i <= original_dim(end)
    for chan = 1:1:original_dim(2)
      processed_data(:,:,chan,i) = abs(cwt(original_data(:,chan,1,i),scalogram_bins,'cgau4'));
      %average = mean(mean(processed_data(:,:,chan,i)));
```

```
        %processed_data(:,:,chan,i) = processed_data(:,:,chan,i)-average;
        %standard_dev = std(std(processed_data(:,:,chan,i)));
        %processed_data(:,:,chan,i) = processed_data(:,:,chan,i)/standard_dev;
    end
    i = i+1;
end

end

end
```

52. network_trainer.mat

```
 function [network,info,time] = network_trainer(train_data,train_labels, output_size , batch_norm, relu, dropout, conv_filter, conv_stride,num_filters,
pool_filter, pool_stride, options,use_custom_architecture,custom_architecture, type)


if use_custom_architecture
    CNN_type = "custom";
elseif type == "SVM"
    CNN_type = "SVM";
elseif length(conv_filter(1,:)) == 2
    CNN_type = "2D";
elseif length(conv_filter(1,:)) == 3
    CNN_type = "3D";
end

%%% Custom CNN block:
if CNN_type == "custom"

    lgraph = load(custom_architecture);
    field = char(fields(lgraph));
    lgraph = lgraph.(field);

    fprintf('Training network\n');
    tic
    [network,info] = trainNetwork(train_data, train_labels, lgraph, options);
    time = toc;
    fprintf('Training time = %f/n',time)
    fprintf('Network parameters:')
    try
    lgraph.Layers
    catch
    end
    try
    lgraph
    catch
    end
end

%%% SVM block:
if CNN_type == "SVM"
    tic
    t = templateSVM('Standardize',true);
    network = fitcecoc(train_data, train_labels,'Learners',t);
    time = toc;
    info = [];
end


%%% 3D CNN block:
if CNN_type == "3D"

    %Create layer graph and add input layer based on training data size.
    input_size = size(train_data(:,:,:,1));

    lgraph = layerGraph();
    lgraph = addLayers(lgraph, imageInputLayer(input_size,"Name","Input_layer","Normalization",network.normalization));
    previous_layer = "Input_layer";
    %Determine the number of output classes to be used in final layer

    i = 1;
    while i <= length(conv_filter(:,1,1,1))
```

```
            %add conv layer
            layer_name = strcat("Conv_layer_", string(i));
            current_layer = layer_name;
            lgraph = addLayers(lgraph,
convolution3dLayer(conv_filter(i,:,:,:),num_filters(i),"Name",layer_name,"Padding","same","Stride",conv_stride(i,:,:,:),'NumChannels',1));
            lgraph = connectLayers(lgraph,previous_layer,current_layer);
            previous_layer = current_layer;

            %conditionally add batch normalization
            if batch_norm(i)
              layer_name = strcat("Batchnorm_", string(i));
              current_layer = layer_name;
              lgraph = addLayers(lgraph, batchNormalizationLayer("Name",layer_name));
              lgraph = connectLayers(lgraph,previous_layer,current_layer);
              previous_layer = current_layer;
            end

            %conditionally add relu
            if relu(i)
              layer_name = strcat("Relu_", string(i));
              current_layer = layer_name;
              lgraph = addLayers(lgraph, reluLayer("Name",layer_name));
              lgraph = connectLayers(lgraph,previous_layer,current_layer);
              previous_layer = current_layer;
            end

            %Check the values of pool_stride for ith conv. layer, if not all 1,
            %then add pooling layer using values
            do_pool = false;
            for pool_value = pool_filter(i,:,:,:)
              if pool_value ~= 1
                 do_pool = true;
              end
            end

            if do_pool
               layer_name = strcat("Maxpool_", string(i));
               current_layer = layer_name;
               lgraph = addLayers(lgraph, maxPooling3dLayer(pool_filter(i,:,:,:),"Name",layer_name,"Padding","same","Stride",pool_stride(i,:,:,:)));
               lgraph = connectLayers(lgraph,previous_layer,current_layer);
               previous_layer = current_layer;
            end

            i = i + 1;
          end

          lgraph = addLayers(lgraph, fullyConnectedLayer(output_size,"Name","fc"));
          lgraph = connectLayers(lgraph,previous_layer,"fc");
          lgraph = addLayers(lgraph, softmaxLayer("Name","softmax"));
          lgraph = connectLayers(lgraph,"fc","softmax");
          lgraph = addLayers(lgraph, classificationLayer("Name","classoutput"));
          lgraph = connectLayers(lgraph,"softmax","classoutput");

          lgraph.Layers

          fprintf('Training network\n');
          tic
          [network,info] = trainNetwork(train_data, train_labels, lgraph, options);
          time = toc;
          fprintf('Training time = %f/n',time)
          fprintf('Network parameters:')
          lgraph.Layers
        end


        %%% 1D & 2D CNN shared block:
        if CNN_type == "2D"

          %Create layer graph and add input layer based on training data size.
          input_size = size(train_data(:,:,:,1));

          lgraph = layerGraph();
          lgraph = addLayers(lgraph, imageInputLayer(input_size,"Name","Input_layer"));
          previous_layer = "Input_layer";
          %Determine the number of output classes to be used in final layer
```

```
i = 1;
while i <= length(conv_filter(:,1))

   %add conv layer
   layer_name = strcat("Conv_layer_", string(i));
   current_layer = layer_name;
   lgraph = addLayers(lgraph,
convolution2dLayer(conv_filter(i,:),num_filters(i),"Name",layer_name,"Padding","same","Stride",conv_stride(i,:,:,:),'NumChannels',1));
   lgraph = connectLayers(lgraph,previous_layer,current_layer);
   previous_layer = current_layer;

   %conditionally add batch normalization
   if batch_norm(i)
     layer_name = strcat("Batchnorm_", string(i));
     current_layer = layer_name;
     lgraph = addLayers(lgraph, batchNormalizationLayer("Name",layer_name));
     lgraph = connectLayers(lgraph,previous_layer,current_layer);
     previous_layer = current_layer;
   end

   %conditionally add relu
   if relu(i)
     layer_name = strcat("Relu_", string(i));
     current_layer = layer_name;
     lgraph = addLayers(lgraph, reluLayer("Name",layer_name));
     lgraph = connectLayers(lgraph,previous_layer,current_layer);
     previous_layer = current_layer;
   end

   if dropout(i)
     layer_name = strcat("Dropout_", string(i));
     current_layer = layer_name;
     lgraph = addLayers(lgraph, dropoutLayer("Name",layer_name));
     lgraph = connectLayers(lgraph,previous_layer,current_layer);
     previous_layer = current_layer;
   end

   %Check the values of pool_stride for ith conv. layer, if not all 1,
   %then add pooling layer using values
   do_pool = false;
   for pool_value = pool_filter(i,:)
     if pool_value ~= 1
        do_pool = true;
     end
   end

   if do_pool
      layer_name = strcat("Maxpool_", string(i));
      current_layer = layer_name;
      lgraph = addLayers(lgraph, maxPooling2dLayer(pool_filter(i,:),"Name",layer_name,"Padding","same","Stride",pool_stride(i,:,:,:)));
      lgraph = connectLayers(lgraph,previous_layer,current_layer);
      previous_layer = current_layer;
   end

   i = i + 1;
end

lgraph = addLayers(lgraph, fullyConnectedLayer(output_size,"Name","fc"));
lgraph = connectLayers(lgraph,previous_layer,"fc");
lgraph = addLayers(lgraph, softmaxLayer("Name","softmax"));
lgraph = connectLayers(lgraph,"fc","softmax");
lgraph = addLayers(lgraph, classificationLayer("Name","classoutput"));
lgraph = connectLayers(lgraph,"softmax","classoutput");

try
   fprintf('Training network\n');
   tic
   [network,info] = trainNetwork(train_data, train_labels, lgraph, options);
   time = toc;
   fprintf('Training time = %f/n',time)
   fprintf('Network parameters:')
   lgraph.Layers
catch
   info = 'Invalid network';
```

```
      fprintf('Error during raining (possible invlaid network), moving to next iteration.\n')
      network.Layers = lgraph.Layers;
      time = [];
      delete(findall(0));
    end
end

end
```

53. padderborn_raw_data_plotter.mat

```
bearing_list = string(fields(training_data));
j = 1;

while j <= length(bearing_list)

%bearing = bearing_list(j);
bearing = "Bearing1_7";

ch_data = squeeze(training_data.(bearing)(:,1,:));

size_ch_data = size(ch_data);

subplot(2,1,1)

i = 1;
plot_start = 1;
hold on
while i <= size_ch_data(end)
   data = ch_data(:,i);
   plot([plot_start:plot_start+length(data)-1],data)
   plot_start = plot_start+length(data);
   i = i +1;
end
xlim([1,plot_start])

subplot(2,1,2)
raw_kurt = kurtosis(ch_data);
kurt_hist = histogram(raw_kurt,10000);
k = 1;
current_sum = 0;
while current_sum <= length(raw_kurt)*0.89
   current_sum = current_sum + kurt_hist.Values(k);
   k = k+1;
end
alarm = kurt_hist.BinEdges(k);
alarm_idx = find(raw_kurt>alarm);

alarm_idx_new = diff(alarm_idx);
m = 3;
while m <= length(alarm_idx_new)
   if alarm_idx_new(m) == 1 && alarm_idx_new(m-1) == 1 && alarm_idx_new(m-2) == 1
      break
   end
   %fprintf("%.0f  %.0f  %.0f  \n",alarm_idx_new(m), alarm_idx_new(m-1),alarm_idx_new(m-2))
   m = m+1;
end

alarm_time = alarm_idx(m);
plot(raw_kurt)
hold on
xline(alarm_time,'-.r');
xlim([1,length(raw_kurt)])
subplot(2,1,1)
xline(alarm_time*length(data),'-.r');
j = j + 1;
clf
end
```

54. paderborn_network_commander.mat

```
function network = paderborn_network_commander(network)
```

```
%%% Rename some variables to make it simpler to call on them later
save_network = network.save_network;
keep_training_gui_open = network.keep_training_gui_open;
network_save_path = network.network_save_path;
network_save_name = network.network_save_name;
conv_filter = network.conv_filter;
conv_stride = network.conv_stride;
pool_filter = network.pool_filter;
pool_stride = network.pool_stride;
num_filters = network.num_filters;
batch_norm = network.batch_norm;
relu = network.relu;
k = network.k;
MaxEpochs = network.MaxEpochs;

%%% temp code:
network.testing_data = 'D:\Paderborn_data\sorted_data_2\augmented_raw_time\pdbn_testing_t1200.mat';
network.training_data = 'D:\Paderborn_data\sorted_data_2\augmented_raw_time\pdbn_training_t1200.mat';
%%%

traing_dataset = network.training_data;
testing_dataset = network.testing_data;
%Import and shuffle the dataset
load(traing_dataset)

dimensions = size(master_data);
ix = randperm(dimensions(end));
train_data = master_data(:,:,:,ix);
train_labels = master_labels(ix);

load(testing_dataset)

dimensions = size(master_data);
ix = randperm(dimensions(end));
validation_data = master_data(:,:,:,ix);
validation_labels = master_labels(ix);

%n = current iteration, 1 <= n <= k
% Initialize a bunch of values to store results over the folds
%subset_size = floor(length(master_data(1,1,1,:))/k);
%n = 1;
%validation_accuracy_tally = zeros(k, 1);
%train_accuracy_tally = zeros(k, 1);
%time_tally = zeros(k, 1);
%network_taly = [];

%train_data = master_data(:,:,:,1+((n-1)*subset_size):((n)*subset_size));
%train_labels = master_labels(1+((n-1)*subset_size):((n)*subset_size));

%validation_data = master_data;
%validation_labels = master_labels;
%validation_data(:,:,:,1+((n-1)*subset_size):((n)*subset_size)) = [];
%validation_labels(((1+(n-1)*subset_size):((n)*subset_size))) = [];

% This parameter determines the size of the output of the final FC
% layer. It has to be defined after the data is imported because it
% determines the required size by reading the number of output
% categories in the label space.
output_size = length(countcats(train_labels));

%%% Training options
options = trainingOptions('sgdm', ...
'InitialLearnRate',0.01, ...
'Verbose',false, ...
'Plots','training-progress', ...
'ValidationData',{validation_data,validation_labels}, ...
'ValidationFrequency', 4240, ...
'MaxEpochs',MaxEpochs, ...
'MiniBatchSize',32,...
'ExecutionEnvironment','gpu');

[temp_trained_net,temp_info,temp_time] = network_trainer(train_data, train_labels, output_size, batch_norm, relu, conv_filter, conv_stride, num_filters,
pool_filter, pool_stride, options);
network.trained_network{n} = temp_trained_net;
```

```
        network.info{n} = temp_info;
        network.time(n) = temp_time;

        fprintf('\n\nFinal training accuracy: %0.2f\nFinal validation accuracy: %0.2f\nInitial learn rate: %0.5f\n', temp_info.TrainingAccuracy(end),
temp_info.ValidationAccuracy(end), temp_info.BaseLearnRate(1))
        fprintf('\n\n\n\n')

        %validation_accuracy_tally(n) = temp_info.ValidationAccuracy(end);
        %train_accuracy_tally(n) = temp_info.TrainingAccuracy(end);
        %time_tally(n) = temp_time;

        if keep_training_gui_open == false
          delete(findall(0));
        end

        if save_network == true
          save(strcat(network_save_path,'\',network_save_name,'.mat'),"network");
          diary_filename = strcat(network_save_path,'\',network_save_name,'_summary.txt');
          diary(diary_filename);
          fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
          fprintf('\n\n')
          temp_trained_net.Layers
          fprintf('\n\n')
          fprintf('Validation accuracy:');
          fprintf('  %f  ', validation_accuracy_tally);
          fprintf('\nValidation mean = %f\n',mean(validation_accuracy_tally))
          fprintf('Validation variance = %f\n\n',var(validation_accuracy_tally))
          fprintf('Train accuracy:');
          fprintf('  %f  ', train_accuracy_tally);
          fprintf('\nTraining mean = %f\n',mean(train_accuracy_tally))
          fprintf('Training ariance = %f\n\n',var(train_accuracy_tally))
          fprintf('Training time:');
          fprintf('  %f  ', time_tally);
          fprintf('\nMean = %f\n',mean(time_tally))
          diary
        end
      end
```

55. paderborn_network_commander_v2.mat

```
      function network = paderborn_network_commander_v2(network)

        %%% Rename some variables to make it simpler to call on them later
        save_network = network.save_network;
        keep_training_gui_open = network.keep_training_gui_open;
        network_save_path = network.network_save_path;
        network_save_name = network.network_save_name;
        normalization = network.normalization;
        conv_filter = network.conv_filter;
        conv_stride = network.conv_stride;
        pool_filter = network.pool_filter;
        pool_stride = network.pool_stride;
        num_filters = network.num_filters;
        batch_norm = network.batch_norm;
        relu = network.relu;
        dropout = network.dropout;
        k = network.k;
        MaxEpochs = network.MaxEpochs;
        dataset = default_network.dataset;
        data_window = network.data_window;
        data_aug_stride = network.data_aug_stride;
        data_ch = network.data_ch;
        MiniBatchSize = network.MiniBatchSize;
        preprocessor = network.preprocessor;
        spectrogram_window = network.spectrogram_window;
        scalogram_bins = network.scalogram_bins;

        %Import, shuffle, and perform augmentation on the dataset
        train_data = load(dataset(1));
        train_data = train_data.unified_data;
        test_data = load(dataset(2));
        test_data = test_data.unified_data;
        data_cats = categories(categorical(train_data(:,2)));
```

```
train_data = CWRU_data_augmentation_from_uni(train_data,data_window,data_aug_stride,data_ch);
test_data = CWRU_data_augmentation_from_uni(test_data,data_window,data_aug_stride,data_ch);

train_labels = categorical(train_data(:,2));
test_labels = categorical(test_data(:,2));

train_data = train_data(:,1);
test_data = test_data(:,1);

new_train_data = zeros(length(cell2mat(train_data(1))),1,1,length(train_data));
new_test_data = zeros(length(cell2mat(test_data(1))),1,1,length(test_data));

row_index = 1;
while row_index <= length(train_data)
  new_train_data(:,:,:,row_index) = cell2mat(train_data(row_index));
  row_index = row_index + 1;
end
train_data = new_train_data;

row_index = 1;
while row_index <= length(test_data)
  new_test_data(:,:,:,row_index) = cell2mat(test_data(row_index));
  row_index = row_index + 1;
end
test_data = new_test_data;

%validation_accuracy_tally = zeros(k, 1);
%train_accuracy_tally = zeros(k, 1);
%time_tally = zeros(k, 1);
%training_size_tally = zeros(k, 1);
%network_taly = [];

if ~strcmp(preprocessor,'none')
   train_data = generalized_preprocessor(train_data,preprocessor,spectrogram_window,scalogram_bins);
   test_data = generalized_preprocessor(test_data,preprocessor,spectrogram_window,scalogram_bins);
end

train_dim = size(train_data);
idx_train = randperm(train_dim(end));
train_data = train_data(:,:,:,idx_train);
train_labels = train_labels(idx_train);

test_dim = size(test_data);
idx_test = randperm(test_dim(end));
test_data = test_data(:,:,:,idx_test);
test_labels = test_labels(idx_test);

training_size = length(train_data);
% This parameter determines the size of the output of the final FC
% layer. It has to be defined after the data is imported because it
% determines the required size by reading the number of output
% categories in the label space.
output_size = length(countcats(train_labels));


%%% Training options
options = trainingOptions('adam', ...
'InitialLearnRate',0.0001, ...
'Verbose',false, ...
'Plots','training-progress', ...
'ValidationData',{test_data,test_labels}, ...
'ValidationFrequency', floor(2*length(test_labels)/k/MiniBatchSize), ...
'MaxEpochs',MaxEpochs, ...
'MiniBatchSize',MiniBatchSize,...
'ExecutionEnvironment','gpu');

[temp_trained_net,temp_info,temp_time] = network_trainer(train_data, train_labels, output_size, batch_norm, relu, dropout, conv_filter, conv_stride,
num_filters, pool_filter, pool_stride, options);

if ~strcmp(temp_info,'Invalid network')

   network.categories = categories(test_labels);

   network.true_test_labels = test_labels;
   predicted_test_labels_idx = predict(temp_trained_net,test_data);
```

```
            predicted_test_labels = strings(size(test_labels));
            prediction_index = 1;
            while prediction_index <= length(test_labels)
              [maximum,index] = max(predicted_test_labels_idx(prediction_index,:));
              predicted_test_labels(prediction_index) = network.categories{index};
              prediction_index = prediction_index+1;
            end
            network.predicted_test_labels = categorical(predicted_test_labels);
            network.confusion_mat_validation = confusionmat(test_labels,network.predicted_test_labels);


            network.true_train_labels = train_labels;
            predicted_train_labels_idx = predict(temp_trained_net,train_data);
            predicted_train_labels = strings(size(train_labels));
            prediction_index = 1;
            while prediction_index <= length(train_labels)
              [maximum,index] = max(predicted_train_labels_idx(prediction_index,:));
              predicted_train_labels(prediction_index) = network.categories{index};
              prediction_index = prediction_index+1;
            end
            network.predicted_train_labels = categorical(predicted_train_labels);
            network.confusion_mat_train = confusionmat(train_labels,network.predicted_train_labels);

            network.trained_network = temp_trained_net;
            network.info = temp_info;
            network.time = temp_time;
            network.training_size = training_size;

            fprintf('\n\nFinal training accuracy: %0.2f\nFinal validation accuracy: %0.2f\nInitial learn rate: %0.5f\n', temp_info.TrainingAccuracy(end),
temp_info.ValidationAccuracy(end), temp_info.BaseLearnRate(1))
            fprintf('\n\n\n\n')

            validation_accuracy = temp_info.ValidationAccuracy(end);
            train_accuracy = temp_info.TrainingAccuracy(end);
            time = temp_time;

            if keep_training_gui_open == false
              delete(findall(0));
            end
          else
            fprintf("Training failure.")
          end

          if save_network == true && ~strcmp(temp_info,'Invalid network')
            save(strcat(network_save_path,'\',network_save_name,'.mat'),"network");
            diary_filename = strcat(network_save_path,'\',network_save_name,'_summary.txt');
            diary(diary_filename);
            fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
            fprintf('\n\n')
            temp_trained_net.Layers
            fprintf('\n\n')
            fprintf('Validation accuracy:');
            fprintf('  %f  ', validation_accuracy);
            fprintf('\nValidation mean = %f\n',mean(validation_accuracy))
            fprintf('Validation variance = %f\n\n',var(validation_accuracy))
            fprintf('Train accuracy:');
            fprintf('  %f  ', train_accuracy);
            fprintf('\nTraining mean = %f\n',mean(train_accuracy))
            fprintf('Training ariance = %f\n\n',var(train_accuracy))
            fprintf('Training time:');
            fprintf('  %f  ', time);
            fprintf('\nMean = %f\n',mean(time))
            fprintf('Training samples used:');
            fprintf('  %f  ', training_size);
            fprintf('\nMean = %f\n',mean(training_size))
            diary
          end
          if strcmp(temp_info,'Invalid network')
            save(strcat(network_save_path,'\invalid_',network_save_name,'.mat'),"network");
            diary_filename = strcat(network_save_path,'\invalid_',network_save_name,'_summary.txt');
            diary(diary_filename);
            fprintf(datestr(now,'yyyy_mm-dd_HH:MM:SS'));
            fprintf('\n\n')
            temp_trained_net.Layers
            diary
```

```
    end
end
```

56. paper_4_plot_maker.mat

```
plot_index = 1;
clf


%fig 1: raw vibrations measurements from both channels to describe the raw
%dataset
if plot_index == 1
    %load('D:\master_paper_4\femto_data\training_data.mat')

    alarm_time1 = getAlarmTime(training_data.Bearing1_7);
    alarm_time2 = getAlarmTime(training_data.Bearing2_5);
    alarm_time3 = getAlarmTime(training_data.Bearing2_7);
    clf


    ch1_data = bearingMerger( training_data.Bearing1_7(:,1,:) );
    ch2_data = bearingMerger( training_data.Bearing1_7(:,2,:) );

    subplot(3,1,1)
    plot(ch2_data)
    hold on
    plot(ch1_data)
    title("Bearing1-5 raw acceleration data")
    xlabel("Data points")
    ylabel("Amplitude")
    set(gca,'fontname','times')
    xline(alarm_time1*2560,'-', 'Color', '#77AC30', 'LineWidth' , 3)
    legend(["Accelerometer 2", "Accelerometer 1", "FPT"])


    ch1_data = bearingMerger( training_data.Bearing2_5(:,1,:) );
    ch2_data = bearingMerger( training_data.Bearing2_5(:,2,:) );

    subplot(3,1,2)
    plot(ch2_data)
    hold on
    plot(ch1_data)
    title("Bearing2-5 raw acceleration data")
    xlabel("Data points")
    ylabel("Amplitude")
    set(gca,'fontname','times')
    xline(alarm_time2*2560,'-', 'Color', '#77AC30', 'LineWidth' , 3)
    legend(["Accelerometer 2", "Accelerometer 1", "FPT"])


    ch1_data = bearingMerger( training_data.Bearing2_7(:,1,:) );
    ch2_data = bearingMerger( training_data.Bearing2_7(:,2,:) );

    subplot(3,1,3)
    plot(ch1_data)
    hold on
    plot(ch2_data)
    title("Bearing2-7 raw acceleration data")
    xlabel("Data points")
    ylabel("Amplitude")
    set(gca,'fontname','times')
    xline(alarm_time3*2560,'-', 'Color', '#77AC30', 'LineWidth' , 3)
    legend(["Accelerometer 1", "Accelerometer 2", "FPT"])


    x0 = 550;
    y0 = 250;
    width = 675;
    height= 550;

    set(gcf,'position',[x0,y0,width,height])

end
```

```
function merged_signal = bearingMerger(ch_data)

   original_size = size(ch_data);
   new_length = original_size(1)*original_size(end);
   merged_signal = zeros(new_length,1);
   i = 1;
   while i <= original_size(end)
      merged_signal(1 + (i-1)*original_size(1) : i*original_size(1)) = ch_data(:,1,i);

     i = i + 1;
   end
end

function alarm_time = getAlarmTime(current_data)

   current_data = squeeze(current_data(:,1,:));

   raw_kurt = kurtosis(current_data);
   kurt_hist = histogram(raw_kurt,10000);
   k = 1;
   current_sum = 0;
   while current_sum <= length(raw_kurt)*0.89
      current_sum = current_sum + kurt_hist.Values(k);
      k = k+1;
   end
   alarm = kurt_hist.BinEdges(k);
   alarm_idx = find(raw_kurt>alarm);

   alarm_idx_new = diff(alarm_idx);
   m = 3;
   while m <= length(alarm_idx_new)
      if alarm_idx_new(m) == 1 && alarm_idx_new(m-1) == 1 && alarm_idx_new(m-2) == 1
         break
      end
      %fprintf("%.0f  %.0f  %.0f  \n",alarm_idx_new(m), alarm_idx_new(m-1),alarm_idx_new(m-2))
      m = m+1;
   end

   alarm_time = alarm_idx(m);

end
```

57. unifed_data_augmentation.mat

```
load('D:\Paderborn_data\sorted_data_2\testing\testing_data.mat')

save_augmented_data = true;
save_folder = 'D:\Paderborn_data\sorted_data_2\augmented_raw_time\';
save_name = 'pdbn_testing_t1200.mat';
ch = 1; %ch = 3 will keep both channels in a length * height * depth(2) * #training samples

%   Locaitons for unified data:
%      D:\CWRU_data\raw_data\unified_data.mat
%      D:\2009PHM_data\labeled_data\gear_data\unified_data.mat

augmentation_span = 1200;
augmentation_overlap = 350;
augmentation_stride = augmentation_span - augmentation_overlap;

augmented_data = cell(1,2);

unified_row_index = 1;

while unified_row_index <= length(unified_data(:,1))
   augmented_row = cell(1,2);
   current_label = char(unified_data(unified_row_index,2));
   raw_row_signal = cell2mat(unified_data(unified_row_index,1));
   raw_duration = length(raw_row_signal);

   starting_index = 1;
   while starting_index < raw_duration - augmentation_span

      current_segment = raw_row_signal(starting_index:starting_index+augmentation_span-1,:,:);
```

```
        new_row = [{current_segment},{current_label}];

        augmented_data = [augmented_data ; new_row];

        starting_index = starting_index + augmentation_stride;
    end
    unified_row_index = unified_row_index + 1;
end

augmented_data = augmented_data(2:end,:);

if ch == 1 || ch == 2
    master_data = zeros(length(augmented_data{1,1}),1,1,length(augmented_data));
    master_labels = strings(1,length(augmented_data));
    j = 1;

    while j <= length(augmented_data)

        both_channels_data = augmented_data{j,1};
        master_data(:,:,:,j) = both_channels_data(:,:,ch,:);
        master_labels(j) = augmented_data{j,2};

        j = j+1;
    end
end

if ch == 3

    master_data = zeros(length(augmented_data{1,1}),1,2,length(augmented_data));
    master_labels = strings(1,length(augmented_data));
    j = 1;

    while j <= length(augmented_data)

        master_data(:,:,:,j) = augmented_data{j,1};
        master_labels(j) = augmented_data{j,2};

        j = j+1;
    end

end

master_labels = categorical(master_labels);

if save_augmented_data

    save(strcat(save_folder, save_name),'master_data', 'master_labels', '-v7.3');
end
```

58. unified_data_downsampler.mat

```
datasets = ["D:\Paderborn_data\sorted_data_2\training\training_data.mat" , "D:\Paderborn_data\sorted_data_2\testing\testing_data.mat"];

save_locations = [ "D:\Paderborn_data\sorted_data_2\training\training_data_2.mat" , "D:\Paderborn_data\sorted_data_2\testing\testing_data_2.mat"];

fs_original = 64000;
fs_new = 12000;

j = 1;
for dataset = datasets
    load(dataset)
    new_unified_data = unified_data;
    i = 1;
    while i <= length(unified_data)
        new_unified_data{i,1} = resample(unified_data{i,1},fs_new,fs_original);
        i = i+1;
    end

    unified_data = new_unified_data;
    save(save_locations(j),"unified_data");
    j = j+1;
end
```