MOTOR/GENERATOR FAULT PROGNOSIS USING VIBRATION SIGNATURE AND FORECASTING TECHNIQUES

Kuanysh Orynbek, BTech in Electric Power Engineering

Submitted in fulfilment of the requirements
for the degree of Master of Science
in Electrical and Computer Engineering

School of Engineering and Digital Sciences
Department of Electrical and Computer Engineering
Nazarbayev University

53, Kabanbay batyr Avenue,
Nur-Sultan, Kazakhstan, 010000

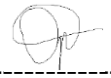
Supervisor: Mehdi Bagheri
Co-supervisor: Amin Zollanvari

April 2022

Declaration

I hereby, declare that this manuscript, entitled "Motor/generator fault prognosis using vibration signature and forecasting techniques", is the result of my own work except for quotations and citations that have been duly acknowledged.

I also declare that, to the best of my knowledge and belief, it has not been previously or concurrently submitted, in whole or in part, for any other degree or diploma at Nazarbayev University or any other national or international institution.



Name: Kuanysh Orynbek

Date: April 2022

Abstract

Over the last few years, the industrial dependency to operate induction motors and generators has been significantly increased. In this instance, it is necessary to monitor the state of induction (asynchronous) machines, as the motor/generators will face with overloading, under-voltage, overvoltage, or even catastrophic failures over the course of their operation. To address this important concern, a fault forecasting architecture using machine learning techniques is studied and developed over the motor/generator vibration signature in this thesis. Initially, mathematical modeling is provided to find the normal and abnormal parameters, and also explore the vibrational frequencies along with the analytical analysis for motor/generators. The Föppl/Jeffcot's rotor modeling system is employed for rotor vibration modeling; however, the transformer core and winding vibration model was considered as a basic theory for the stator core and winding of the motor/generator in an analytical approach. To emulate a faulty condition over an induction motor in this thesis, an experimental setup is designed and developed. The voltage excitation condition for induction motor along with single phasing are considered to be the fault types and examined practically in the laboratory to conduct experiments and collect vibrational data. Afterwards, 1D Convolutional Neural Network (CNN) model is constructed for accurately detecting faults. The MSE of voltage excitation prediction was obtained as 0.000426, whereas the highest fault detection accuracy of single phasing reveled to be 99.58%.

Acknowledgements

Firstly, I would like to express my uttermost gratitude to my supervisors Mehdi Bagheri and Amin Zollanvari. It has been their supervision and direction throughout my studies, which has allowed me to successfully complete this two-year Master's program in Electrical and Computer Engineering. I am appreciative for their supervision, especially in the areas of power systems and machine learning. Furthermore, I would like to thank my family and friends for supporting me from the beginning up to now.

Table of Contents

List of Abbreviations and Symbols

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CSA	Current Signature Analysis
V	Volts
MSE	Mean Squared Error
MAE	Mean Absolute Error
AC	Alternating Current
DBN	Deep Belief Network
RBM	Restrict Boltzmann Machines
SVM	Support Vector Machine
FFT	Fast Fourier Transform
DW	Discrete Wavelet Transform
MP	Matching Pursuit
NN	Neural Network
AI	Artificial Intelligence
TFD	Time Frequency Distribution
KNN	K-Nearest Neighbor
RNN	Recurrent Neural Network
RAE	Relative Absolute Error
DL	Deep Learning
1D CNN	One-dimensional Convolutional Neural Network
2D CNN	Two-dimensional Convolutional Neural Network

List of Tables

List of Figures

Chapter 1 – Introduction

Induction (asynchronous) motors have various applications, and their significant roles in industry, electricity generation, and renewable energy wind power plants are crucial for utility managers and operators. They also have quite well-known benefits such as easy, cost-effective operation, reliability, and efficiency. However, different faults such as overloading, under-voltage or overvoltage, short-circuit, and phase disconnections prevent the proper operation of these devices. Along with the cost, operational limitations and motor/generator availabilities are essential for the industry. Hence, the induction motor/generators are considered a key component in a wide range of industrial processes [1] – [5].

There are two types of techniques - online and offline analyses - that are utilized to evaluate motor states. Lately, the online analysis method is predominantly used because online methods are now an interest of technical societies as they continuously assist in machine monitoring. This method also comprises data from current, voltage, vibration, and temperature. Accordingly, vibration analysis is the most widespread and convenient method compared to others for observing the real-time condition of rotating machines. In order to avoid impeding the proper operation of motor/generators, it is worthy of developing a system that can identify the failure. The necessity of the system primarily relates to industry and manufacture [6] – [11].

1.1.Aims and Objectives

The main aim of this project is to implement a system that can evaluate and analyze a real-time condition of the motor/generator using a signature of vibration signal and machine learning methods. The system is supposed to be able to identify motor excitation voltage, such as under-voltage and overvoltage. Furthermore, to determine the motor's normal state and abnormal state using one-phase disconnection is also planned to be studied. It helps to define device failures early on before they become catastrophic incidents. In the same way, this work focuses on researching and creating accurate prediction vibration signals for asynchronous

motors/generators similar to those found in experiments. Through conducting experimental measurements, several case studies will be implemented to explore the motor's under-voltage and overvoltage situations and phase disconnections. The collected data will be utilized to create predictive models for voltage prognosis. The model with the highest accuracy should be selected as the prediction model. The following objectives are planned to be performed to achieve the main aim:

- to perform a literature review to understand the basics of the vibration analysis methods along with indicating state of the art,
- to create a mathematical model for motor/generator vibration analysis,
- to build an experimental setup and collect vibration signal data,
- to develop a code for forecasting the state of induction motors,
- to make a model selection based on hyperparameters and come up with a desirable solution in motor/generator prognosis.

1.2.Literature review

1.2.1. Motor/generator - asynchronous machines

Asynchronous or induction motors are classified into two groups. Induction motors are available in single and three-phase configurations. A single-phase asynchronous motor can be powered by a single-phase alternating current source. However, the three-phase asynchronous motor is able to be connected to a three-phase AC power source. To achieve certain goals, such as implementing a program for identifying failures, it is essential to acquire relevant data. In this step, it is crucial to identify the motor/generator type and its habit under the operational condition to analyze its data later experimentally and precisely in prognosis. Van and Yang [3] developed a cost-effective vibration measurement analysis over a single-phase induction motor. They proposed that data-driven and model-based techniques help to increase the accuracy of prediction ability. Moreover, in [12], the authors implemented a prognostic system to forecast

the wear condition of asynchronous machines. Single-phase motors are mainly utilized in household applications and small industrial workshops, and their defects and faults can be addressed with their private owners, or even their replacement might be economical. In contrast, three-phase motors have mainly professional and industrial applications, and their failure is vital for utility management or operators. Korkua *et al.* [11] described three-phase induction motor fault detecting using vibration signal analysis. Accordingly, the necessity to identify failures in the three-phase asynchronous motor is critical [11] – [23].

1.2.1.1.Motor fault types

There are some specific fault categories that might occur only in single- or three-phase motors. For instance, overvoltage, under-current, under-voltage, overload, and over-temperature might occur for single- or three-phase motors/generators, while a phase disconnection can be initiated in three-phase induction motors/generators. Nevertheless, the most representative fault types will be discussed in this section. The authors in [4], [13], [15], [24] discussed about the broken rotor bar/fault. In addition, in [6], [18], [25], researchers stated that the vibration spectrum could be used to detect motor/generator failure. It means that during motor operation in a normal state, the vibration signature is aligned and adjusted to a specific vibrational frequency; however, during the fault, the main vibrational frequency might be altered from its original value [7], [21], [26]. Another common fault is an unbalanced load operational condition. According to [14], [16] – [18], unbalanced load conditions will affect the three-phase induction motors negatively. Moreover, it has significant effects on temperature rise and may lead to insulation failure in motors or may substantially decrease motor efficiency [7], [11], [22]. Obviously, it is similar to generators. The authors in [17], [18], [24] investigated the third failure type, which is bearing fault. The bearing fault initiates due to electro erosion and deterioration of spinning elements [20], [21], [26], and it is quite common in motors/generators. The last popular common fault type in induction machines is the stator fault

[3], [12], [24]. A stator fault may occur due to insulation degradation. During insulation degradation, the possibility of short-circuit faults would be considerably increased [4], [6], [26]. Consequently, this degradation may result in a short circuit between the conductor turns. To sum up, this fault implies a flow of a huge amount of current, which can disturb the motor and the system [11], [21], [22].

1.2.1.2.Vibration analysis

To develop a system that is able to perform motor/generator fault prognosis, vibrational signal analysis can be considered as a fast and non-destructive solution. According to [7], vibration elements are particularly essential for evaluating the state of an induced electromagnetic engine system at the frequencies of line, twice line, and twice slip. A fair understanding and proper knowledge of the motor/generator physical factors, which can cause vibration in the device, is essential in identifying induction engine issues, associated with the functioning of electromagnetic systems. In [6], the authors have considered that the analysis can be performed based on two data steps process; the first is collecting the relevant information to fault prognosis, while the second is analyzing the data for diagnosing. Initially, it is essential to know how vibration occurs. It arises when the inherent frequency of the machine is close to the frequency of the force applied to the stator [7], [27], [28]. Therefore, the authors in [11] employed current signature analysis (CSA) based on stator current for fault detection analysis. In [11], vibration detection techniques using accelerometers with low power consumption are highly recommended and concluded as an economical solution. Moreover, a study by [26] explained auto-extract useful information from vibrational signals that describe an induction generator's operational status using a Deep Belief Network (DBN)-based technique. It is intended to monitor the motor automatically and intelligently, which is an integral component of a wide spectrum of manufacturing machinery. The DBN model utilizes an effective learning technique termed greedy layer-wise formation and is structured using layered Restricted

Boltzmann Machines (RBMs). Vibration signals are used to implement the DBN easily, and outputs from activation functions in trained networks are among the qualities necessary for problem detection. Once we compare this technique with other existing methods for induction motor's defect diagnostics, such as wavelet transformation, the proposed approach may deduce features from the vibration signal directly in order to evaluate the motor/generator condition, which is very accurate. The efficacy of the suggested technique for diagnostics of induction motor failures was validated by experiments [7], [26], [28].

1.2.2. Forecasting techniques

Artificial intelligence (AI) is an imitation of human intelligence using machines. Machine learning (ML) is the part of AI that concentrates on using data and algorithms to simulate a human learning. It can be used to predict, classify, cluster and associate objects. Furthermore, ML algorithms can build the model that predicts and makes decisions using the dataset, which is also known as training data. It can contribute to improve, develop and solve many problems. There are numerous types and algorithms of ML. These algorithms can be classified in reinforcement, semi-supervised, unsupervised and supervised types. From ML methods, Artificial Neural Networks (ANN) is an interconnected group of nodes with weights and activation functions and resemble the learning process of a biological brain learning [28]. For instance, using the ANN structure, Rahman *et al.* [22] have employed recording, monitoring, analysis, and classification of motor vibration data in MATLAB. They concluded that the usage of simple features and ANN structure could effectively classify different fault types of motor.

Convolutional Neural Network (CNN) is one of the main deep learning methods for processing data [29]. Furthermore, it is trendy in image processing, as it is suitable and effective to process matrix data [8], [9], [28]. The authors in [15] built a system that could diagnose an induction motor fault based on CNN. In their research, vibration signal data were collected, and

these values were fed into the CNN and employed in the diagnosis of faults. Experimental measurement and simulation were used to verify this method. In addition, Shao *et al.* [29] consolidated that the CNN is considered a powerful method. The proposed system can process multiple signals in a single iteration. The process begins with signal reception and multiple signals are then transformed into a time-frequency distribution (TFD). The proposed architecture was applied to TFD and was used to predict the condition of the induction motor using a fully connected layer. Another study [24] used CNN that shows an improvement in accuracy of 3-10% over the conventional techniques, such as Wavelet Transform, Fast Fourier Transform (FFT) and Hilbert Transforms. Additionally, Liu *et al.* [30] stated that the CNN has achieved good performances in various application domains.

A study by [21] described a new induction motor defect detection method based on machine learning with single and multiple electric and mechanical defects. Two signal processing approaches are proposed for the extraction of features: matching pursuit (MP) and discrete wavelet transforms (DW). They also employed K-Nearest Neighbor (KNN). Consequently, the fine KNN and Gaussian SVM, bagged trees, and weighted and subspace KNN classifiers yielded presumably high percent accuracy classifications for fault diagnosis of motors [18]. Another study [10] used a deep recurrent neural network (RNN) to capture the hidden patterns of vibration time series in order to predict the fault of the transformer in the early stages. The findings of this technique demonstrated a positive prediction of voltage excitation because the relative absolute error (RAE) equals 0.56%. Additionally, inter-turn faults revealed some RNN issues with an RAE of 17.58%.

Chapter 2 - Mathematical Model

The asynchronous motor consists of two main components, a rotor and a stator. Moreover, both will be discussed separately as they are isolated physically. The Föppl/Jeffcot's rotor modeling system [31] is employed for rotor vibration modeling. However, the stator core and winding will be described individually. The transformer core and winding vibration model will be considered as a fundamental theory for stator core and winding.

2.1. Rotor component vibration model

The research by Jeffcott from 1919 contains the earliest reported fundamental concepts of rotor dynamics. Jeffcott validated Föppl's prediction of the existence of a steady supercritical solution by 1895 and expanded Foppl's study by considering external damping [32]. The Föppl/Jeffcot rotor, often known as the Jeffcot rotor, is a simplified rotor arrangement. It is considered that a Jeffcott rotor has a flexible shaft of low mass with a hard disk at the mid-span [33], as shown in Figure 1. The inventors created the basic concept of rotor vibration prediction and attenuation using a more straightforward rotor system. It is frequently used in the real world to analyze increasingly sophisticated rotor-dynamic systems [31].

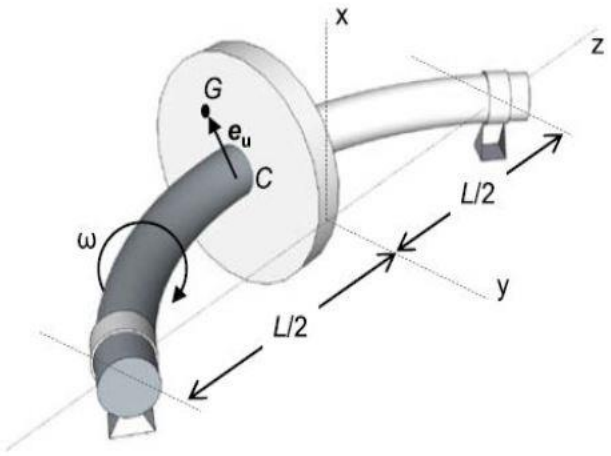


Figure 1: Rotor model of Föppl/Jeffcott, taken from [31]

According to [31], the rotor disk with mass *m* is positioned at the center of the shaft. The shaft is supposed to be massless during the calculation because its mass is ignorable compared to the rotor disk with mass *m* [31]. In this instance, the geometric center of disk *C* is placed at the point (u_{xC}, u_{yC}). However, the disk center mass *G* is positioned at (u_{xG}, u_{yG}). The vector that connects points *C* and *G* is the unbalanced eccentricity e_u. It depicts the rotor disk's imbalance. Furthermore, the disk's rotating speed is represented by ω. At time $t=0$, it is considered that e_u is parallel with the x_{axis}. The shaft cross-section of Föppl/Jeffcott single mass rotor is appeared in Figure 2.

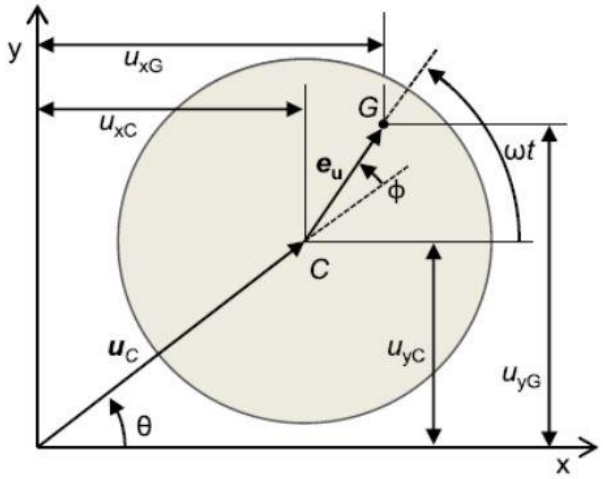


Figure 2: Shaft cross-section, taken from [31]

The Föppl/Jeffcott's lateral bending stiffness is defined by assuming that the rotor disk has no effect on the massless shaft's stiffness:

$$k_s = \frac{48EI}{L^3} \tag{2.1}$$

where *I* denotes the shaft area moment of inertia, *L* denotes the distance between the bearings, *E* is the beam's elastic modulus. Furthermore, the inertia *I* can be represented with *D*, which indicates the diameter of the cylindrical shaft:

$$I = \frac{\pi D^4}{64} \tag{2.2}$$

It was further assumed that at the rotor mid-span, the effective damping influence on the displacement of the disk is minimal. Thus, the constant of damping is c_s. The damping and inertial forces that created by the deformation of the shaft operate on the disk under the assumption that the shaft is massless. The motion's lateral equations along the *x* and *y* axes are found to be:

$$m\ddot{u}_{xG} = -k_S u_{xC} - c_s \dot{u}_{xC} \tag{2.3}$$

$$m\ddot{u}_{yG} = -k_S u_{yC} - c_s \dot{u}_{yC} \tag{2.4}$$

The usage of the rotor angle of rotation ωt and the geometric center *C* can be used to rewrite the coordinates of the disk's mass center:

$$u_{yG} = u_{yC} + e_u \sin(\omega t) \tag{2.5}$$

$$u_{xG} = u_{xC} + e_u \cos(\omega t) \tag{2.6}$$

The motion equations for the Jeffcott rotor was obtained by inserting the second time derivative of (2.5) and (2.6) equations into (2.3) and (2.4)

$$m\ddot{u}_{xG} + k_S u_{xC} + c_s \dot{u}_{xC} = m e_u \omega^2 \cos(\omega t) \tag{2.7}$$

$$m\ddot{u}_{yG} + k_S u_{yC} + c_s \dot{u}_{yC} = m e_u \omega^2 \sin(\omega t) \tag{2.8}$$

The gyroscopic effects operating on the rotor are not included in this model since the bearings are assumed to be endlessly rigid, and the disk of rotor does not tilt.

2.1.1. Damped Free Vibration

The vibration response of the rotor of Jeffcott was explored with a non-zero effective shaft dampening impact on the system. In this case, the motion equation (2.7) and (2.8) becomes:

$$m\ddot{u}_{xG} + k_S u_{xC} + c_s \dot{u}_{xC} = 0 \tag{2.9}$$

$$m\ddot{u}_{yG} + k_S u_{yC} + c_s \dot{u}_{yC} = 0 \tag{2.10}$$

The solutions to the (2.9) and (2.10) systems of second order derivatives are as:

$$u_{xC} = A_x e^{st} \tag{2.11}$$

$$u_{yC} = A_y e^{st} \tag{2.12}$$

The equations (2.13) and (2.14) were obtained by substituting the above mentioned solutions into (2.9) and (2.10)

$$(ms^2 + k_S + c_s)A_x e^{st} = 0 \tag{2.13}$$

$$(ms^2 + k_S + c_s)A_y e^{st} = 0 \tag{2.14}$$

If the equation for the damped characteristic applies, these equations apply to any initial condition:

$$ms^2 + k_S + c_s = 0 \tag{2.15}$$

The system's damped eigenvalues are represented as,

$$s_{1,2} = -\frac{c_s}{2m} \pm j\sqrt{\frac{k_s}{m} - \left(\frac{c_s}{2m}\right)} \tag{2.16}$$

Moreover, the system of rotor is typically undamped, this means:

$$\frac{c_s}{2m} < \frac{k_s}{m} \tag{2.17}$$

The damping ratio ς determines as,

$$\varsigma = \frac{c_s}{2m\omega_n} \tag{2.18}$$

Once the system becomes over damped, the effective damping's ratio c_s to the critical value is represented by this value. Thus, equations (2.13) and (2.14) can be rewritten in a different way

$$s_{1,2} = -\varsigma\omega_n \pm j\omega_n\sqrt{1-\varsigma^2} \tag{2.19}$$

The damped natural frequency is the imaginary component of *s*,

$$\omega_d = \omega_n\sqrt{1-\varsigma^2} \tag{2.20}$$

The damping coefficient of 0.1 is usually considered necessary for the machine's safe operation. The linear combination of the (2.11), (2.12) and (2.16) is discovered to be the final solution to the underdamped free vibration

$$u_{xC} = e^{-\varsigma\omega_n t}\left(A_{x1}e^{j\omega_n t} + A_{x2}e^{-j\omega_n t}\right) =$$

$$e^{-\varsigma\omega_n t}(B_{x1}\cos(\omega_n t) + B_{x2}\sin(\omega_n t)) \tag{2.21}$$

$$u_{yC} = e^{-\varsigma\omega_n t}\left(A_{y1}e^{j\omega_n t} + A_{y2}e^{-j\omega_n t}\right) =$$

$$e^{-\varsigma\omega_n t}\left(B_{y1}\cos(\omega_n t) + B_{y2}\sin(\omega_n t)\right) \tag{2.22}$$

The A_{xi} and A_{yi} values, as well as B_{xi} and B_{yi} values, are dependent on the rotor's starting state.

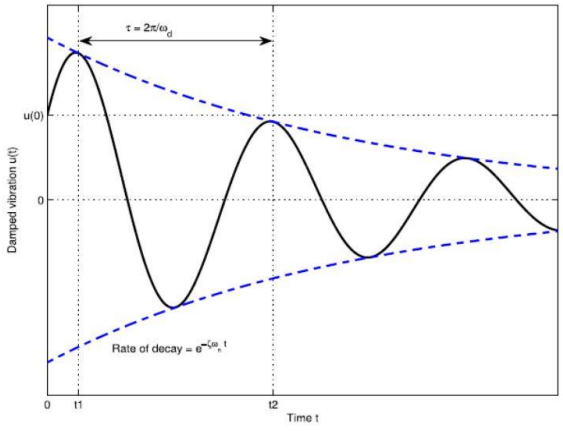


Figure 3: An underdamped system's typical response, taken from [31]

2.1.2. Un-damped Free Vibration

When the $e_u=0$ and $c_s=0$ are negligible, the un-damped free vibration analysis considers rotor vibration. Therefore, the equations in (2.7) and (2.8) are modified to

$$m\ddot{u}_{xG} + k_S u_{xC} = 0 \tag{2.23}$$

$$m\ddot{u}_{yG} + k_S u_{yC} = 0 \tag{2.24}$$

This second order homogeneous system has the following solution:

$$u_{xC} = A_x e^{st} \tag{2.25}$$

$$u_{yC} = A_y e^{st} \tag{2.26}$$

For a number of complicated constant *s*. The initial condition of the rotor disk yields the values of the constants A_x and A_y. Equations (2.27) and (2.28) are obtained by substituting the solution in equations (22.5), (2.26) into equations (2.23) and (2.24)

$$ms^2 A_x e^{st} + k_S A_x e^{st} = (ms^2 + k_S) A_x e^{st} = 0 \tag{2.27}$$

$$ms^2 A_y e^{st} + k_S A_y e^{st} = (ms^2 + k_S) A_y e^{st} = 0 \tag{2.28}$$

If the un-damped characteristic equation holds, for any value of A_x and A_y, the foregoing equations apply,

$$ms^2 + k_S = 0 \tag{2.29}$$

Solving (2.29) equivalency for the complex constant *s*, the equation (2.30) result is derived,

$$s_{1,2} = \pm j\omega_n \tag{2.30}$$

The shaft's natural frequency is represented by ω_n, which is calculated as,

$$\omega_n = \sqrt{\frac{k_s}{m}} = \sqrt{\frac{48EI}{L^3 m}} \tag{2.31}$$

Furthermore, the system's un-damped critical speed is determined as

$$\omega_{cr} = \pm\omega_n \tag{2.32}$$

The un-damped free vibration's final solutions are

$$u_{xC} = A_{x1}e^{j\omega_n t} + A_{x2}e^{-j\omega_n t} = B_{x1}\cos(\omega_n t) + B_{x2}\sin(\omega_n t) \tag{2.33}$$

$$u_{yC} = A_{y1}e^{j\omega_n t} + A_{y2}e^{-j\omega_n t} = B_{y1}\cos(\omega_n t) + B_{y2}\sin(\omega_n t) \tag{2.34}$$

For certain A_x and A_y, or B_x and B_y values that may be determined based on the rotor's beginning circumstances.

2.2. Stator vibration model

2.2.1. Stator core vibration model

Magnetostriction is the primary cause of stator core vibration [34]. It is the term used to describe the length change in a ferromagnetic material after being magnetized [35]. The stator core vibration explosion to the magnetic field is represented by equation (2.35)

$$U_0 \sin(\omega t) = -N_w A_c \frac{dB}{dt}, \tag{2.35}$$

where U_0 is presented the applied voltage to the winding, N_w is the winding turns' number. Moreover, ω is the angular frequency, A_c is the cross-section area, whereas the magnetic induction is provided by B. As a result, the magnetic induction can be examined as follows:

$$B = \frac{-U_0}{N_w A_c}\int \sin(\omega t)dt = \frac{U_0}{N_w A_c \omega}\cos(\omega t) = B_0\cos(\omega t) \tag{2.36}$$

where B_0 is the induction magnitude as the maximum value. B_0 is equal to or less than B_s; that is induction's saturation level

$$B_0 = \frac{U_0}{N_w A_c \omega} \leq B_s \tag{2.37}$$

Furthermore, magnetic induction and field density are linked by magnetic permeability.

$$B = \mu H \tag{2.38}$$

where H is the intensity and μ is the permeability of magnetic. The saturated magnetic intensity is H_c, which is found when H reaches the highest value:

$$B_s = \mu H_c \tag{2.39}$$

Therefore, the magnetic intensity of saturated induction and the field intensity of applied magnetics may be calculated as

$$B = \frac{B_s}{H_c} H \tag{2.40}$$

By substituting (2.36) for (2.40), the field intensity of applied magnetics is acquired,

$$H = \frac{H_c B_0}{B_s} \cos(\omega t) \tag{2.41}$$

Any variations in the core laminate length are triggered by changes in the magnetic field strength. As a result, the maximum movement of the core laminate owing to field intensity variations is provided by

$$x_{core} = \frac{dL}{L} = \frac{\lambda_s}{H_c^2} \int_{-H}^{H} H dH = \frac{2\lambda_s}{H_c^2} \int_0^H |H| dH = \frac{\lambda_s H^2}{H_c^2} = \frac{\lambda_s}{H_c^2} \frac{H_c^2}{B_s^2} B_0^2 \cos^2(\omega t) =$$

$$\frac{\lambda_s}{H_c^2} \frac{H_c^2}{B_s^2} \left(\frac{U_0}{N_w A_c \omega}\right)^2 cos^2(\omega t) = \frac{\lambda_s U_0^2}{B_s^2 N_w^2 A_c^2 \omega^2} cos^2(\omega t) \tag{2.42}$$

Here λ_s is presented as the magnetostriction highest value. As a result, the acceleration of the core laminates is calculated as follow:

$$\ddot{x}_{core} = \frac{d^2 x_{core}}{dt^2} = -\frac{2\lambda_s L U_0^2}{B_s^2 N_w^2 A_c^2} \cos(2\omega t) \quad (2.43)$$

The value of the core vibration of the stator is directly proportional to the voltage square of excitation, as shown by equation (2.43). In addition, the frequency of the vibration of the core is matched to the voltage fundamental frequency's second harmonic order.

2.2.2. Stator winding vibration model

Winding vibration is created by the electromagnetic force induced by the leaking magnetic flux, $B_{leakage}$, passing through the winding and its current. The direction of electromagnetic force can be adjusted, and as a result, the direction of mechanical force can be changed. Furthermore, the winding can be mechanically modeled using springs. As a result, winding vibrations can be studied using the spring-force model. As a result, two distinct outcomes can be assumed.

2.2.2.1. Free vibration without damping factor

In the absence of external limitations, any impulse force strike can cause the stator winding to travel vertically. The stator winding will naturally oscillate since there is an uncontrolled movement. The unit extension of the spring factor, *k*, represents the relationship between the movement and applied force, and the equation is produced as,

$$F = -kx \quad (2.44)$$

where *F* denotes force, whereas *x* denotes displacement. The force equation is achieved as follows when the spring weight is taken into account, and a single impulse is delivered to the spring:

$$ma = F' = W - F = W - (W + kx) \quad (2.45)$$

The winding weight is indicated by *W*, while a is the factor of acceleration. Then, Equation (2.45) can be rewritten as

$$ma = \frac{W}{g}a = \frac{W}{g}\ddot{x} = W - (W + kx) \tag{2.46}$$

The gravity acceleration is denoted by the letter g. The equation for motion of natural winding is found by rearranging (2.46):

$$\frac{W}{g}\ddot{x} + kx = 0, \;\; \ddot{x} + \frac{g}{W}kx = 0 \tag{2.47}$$

The result of modification of (2.47) is

$$\frac{gk}{W} = \alpha^2, \;\; \ddot{x} + \alpha^2 x = 0. \tag{2.48}$$

The differential equation (2.48) has the following solution:

$$x_1 = C_1\cos(\alpha t) + C_2\sin(\alpha t) \tag{2.49}$$

where C_1 and C_2 are the starting condition constants, and α is provided by

$$\alpha = \frac{2\pi}{\tau_n}, \tau_n = 2\pi\sqrt{\frac{\delta_{st}}{g}}, \quad f_n = \frac{1}{2\pi}\sqrt{\frac{g}{\delta_{st}}} \tag{2.50}$$

when the damping component is neglected, f_n signifies the natural oscillation frequency of the spring.

2.2.2.2.Free vibration with damping factor

Natural oscillation of the winding is prevented when the damping factor for the motion of winding owing to the impulsive force is considered. The damping factor may be able to avoid the winding's fast oscillation. In this instance, the winding motion equation of winding is:

$$\frac{W}{g}\ddot{x} = W - (W + kx) - c\dot{x}. \tag{2.51}$$

The motion of winding with damping factor is derived by rearranging (2.51):

$$\frac{W}{g}\ddot{x} + c\dot{x} + kx = 0 \tag{2.52}$$

where *c* is the constant damping factor provided by the starting condition. Modification of (2.52) yields

$$\ddot{x} + \frac{gc}{W}\dot{x} + \frac{gk}{W}x = 0, \tag{2.53}$$

$$\ddot{x} + 2\beta\dot{x} + \alpha^2 x = 0,\ \frac{gc}{W} = 2\beta. \tag{2.54}$$

The linear differential equation (2.54) can be solved in one way:

$$x = e^{\mu t} \tag{2.55}$$

Here *t* denotes the current time. Thus, μ is derived by (2.56)

$$\mu^2 + 2\beta\mu + \alpha^2 = 0, \tag{2.56}$$

From which

$$\mu = -\beta \pm \sqrt{\beta^2 - \alpha^2}. \tag{2.57}$$

Because the square root value in (2.57) becomes negative when $\alpha_2 > \beta_2$. Thus, $\alpha_2 - \beta_2$ has a positive value (2.58)

$$\alpha_1^2 = \alpha^2 - \beta^2 \tag{2.58}$$

The roots of equation (2.56) are derived in this format as

$$\mu_1 = -\beta + j\alpha_1,\ \mu_2 = -\beta - j\alpha_1 \tag{2.59}$$

As a result, two distinct solutions for (2.55) can be calculated:

$$x_1 = \frac{C_1}{2}(e^{\mu_1 t} + e^{\mu_2 t}) = C_1 e^{-\beta t}\cos(\alpha_1 t),$$

$$x_2 = \frac{C_2}{2j}(e^{\mu_1 t} - e^{\mu_2 t}) = C_2 e^{-\beta t}\sin(\alpha_1 t), \quad (2.60)$$

Finally, in the second case, the winding displacement is provided as

$$x = e^{-\beta t}(C_1\cos(\alpha_1 t) + C_2\sin(\alpha_1 t)) \quad (2.61)$$

The bases in equation (2.57) will become real and negative when $\alpha_2<\beta_2$. Furthermore, the displacement is derived as

$$x = C_1 e^{\mu_1 t} + C_2 e^{\mu_2 t} \quad (2.62)$$

In winding movement, there is no such thing as a periodic term for $\alpha_2<\beta_2$, as shown by equation (2.62). As a result, the winding won't oscillate.

The excitation voltage square and current square values are proportional to the magnitudes of the stator core and winding vibrations, according to vibration modeling of the stator of a motor/generator. Furthermore, both the core and the winding vibrate at a fundamental frequency of 2ω, where the current and voltage signal fundamental frequencies are specified as ω. As a result, once the motor/generator is in service, it is theoretically expected that it will be possible to acquire a sinusoidal time series with a frequency of 2ω for core and winding vibrations.

Chapter 3 – Methodology and Experimental Study

It is critical to investigate electrical faults because electrical disruptions are practical rather than permanent mechanical faults. Electrical disruptions have almost no effect on rotor insulation due to the presence of a squirrel-cage rotor [36]; hence, experimental measurements are only performed over the motor stator. This demonstrates the rotor's durability and capacity to remain unaffected by electrical problems. Consequently, rotor vibration is only dependent on the shaft's natural frequency, according to Jeffcott's rotor model. As a result, the stator winding insulation and core vibrations have the highest impact on motor performance.

Voltage excitation is one of the most common electrical issues. Likewise, unbalanced supply voltage due to overvoltage and under-voltage will make three-phase electrical appliances riskier [37]. Under-voltage and overvoltage are types of voltage excitation that are studied in this research work. During the under-voltage condition, once the voltage decreases, the current would be increased. Alternatively, the current will take a lower value when the voltage takes a higher value. It means that when an under-voltage fault develops, the induction motor will consume a large current. Consequently, the coils will be damaged due to overheating [38]. On the contrary, if an overvoltage is developed, insulation failure is the risk as the insulation can tolerate its nominal and maximum breakdown voltage levels. The overvoltage also causes a high current value to flow from the primary or stator coil of an induction motor and causes winding failure [39].

The following fault type that is studied in this thesis is single-phase disconnection (two-phase operation). For single-phase disconnection in the supply system of the induction motor, the asynchronous motor will continue to operate with the other two phases. This condition is known as single phasing. In this instance, the unbalanced current in the stator winding causes torque fluctuation and irregular vibration in the motor. Moreover, overheating can cause severe damage to motor insulation [40]. Taking everything into account, two cases have been

considered in this thesis. There are voltage excitation, which is under-voltage and overvoltage of motor/generator with and without load, and single phasing with and without load, respectively.

3.1. Experimental setup

To collect vibration signal data, a practical setup was assembled and demonstrated in Figure 4, where different components were utilized. There is a three-phase squirrel-cage asynchronous motor, model M4-EV, Kistler vibration sensor, and DEWE 43 DAQ (data acquisition) system. Furthermore, the power supply, transformer, and two switches were utilized. A detailed description of each piece of equipment and its connections is provided in the further part.

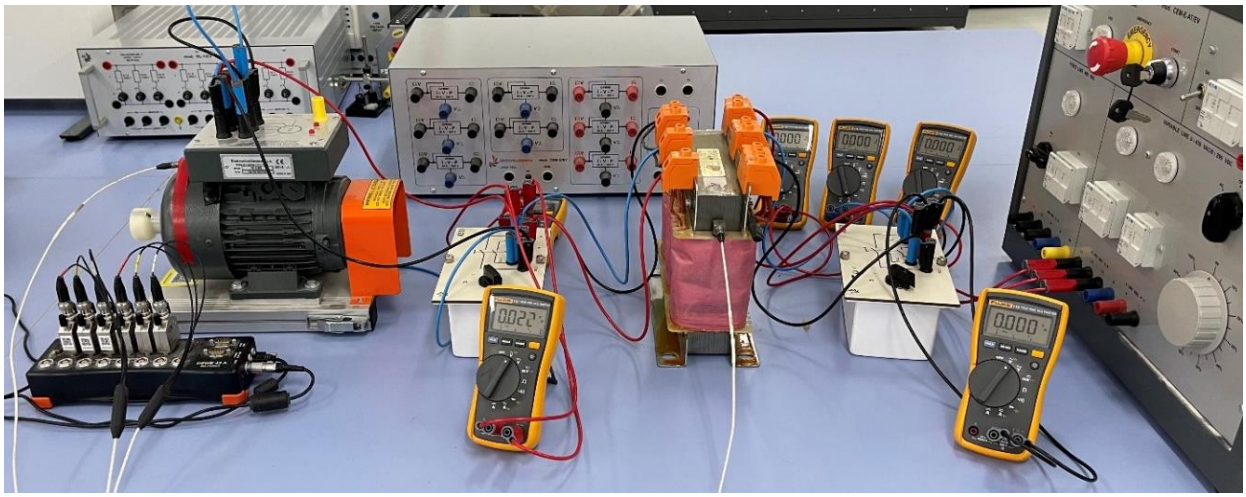


Figure 4: Experimental setup

The rated power of the model M4-EV induction motor is P = 500 W. Additionally, the rated voltage is V = 400/230 V, whereas frequency f = 50 Hz, as shown in Fig A.1 (see Appendix A). The transformer was used to imitate a grid model. In addition, the switches were employed to facilitate connection and disconnection of the circuit along with emulating the single phasing occurrence. However, the main measuring device is the DEWE 43 DAQ. It is connected to the three-dimensional vibration sensor Kistler. It is attached to the surface of the motor stator, as shown in Figure A.2 (see Appendix A). After the assembling of all elements,

the acquisition of data was started. The data collection was under three cases with load and no load conditions. There are overvoltage, under-voltage, and phase disconnections. The different values of voltage will be applied and obtained the vibration signal in the time domain. In the beginning, overvoltage and under-voltage scenarios were examined. For the under-voltage, the following voltage values were employed: 320V, 340V, 360V, 380V, and 400V. It means 80%, 85%, 90%, 95% and 100% of the rated voltage. However, for the overvoltage, 420V and 440V were performed, respectively. During data collection for overvoltage, it was crucial to conduct it for only 2-3 seconds, due to overstress condition on insulation media for the test object [39]. Finally, phase disconnections were considered as the last scenario. In this case, a single-phase disconnection was examined. The switches were used to emulate this condition over the motor. Indeed, one phase was disconnected during motor operation, and the data was collected and compared with the normal operational condition. Furthermore, another motor with the same characteristics was connected to the motor paly as a generator or to emulate a load for our motor under test. The previous procedure was repeated to acquire data with load conditions.

3.2 Data preparation

The first case study is the motor in under and over excitations with a load and with no load conditions. In this instance, the input of the 1D CNN model is the vibration signals of the motor stator, whereas the output is the value of the voltage applied to the motor. The acquired data were segmented by 80 samples per segment. It means that 1 second consists of 50 segments data. The data were randomly shuffled, and divided into 60 percent training, 20 percent validation, and 20 percent test sets in order to train, evaluate and test the model performance, respectively.

The second case study is a disconnection of a single phase supply from the motor with a load and without load conditions. The input contains vibration signals from the motor stator. However, the output is represented as “0” and “1”. In this case, “0” means a normal state, and

"1" is called the abnormal state, when one phase is disconnected during the motor operation. 40 samples of data were considered per segment and 1 second consisted of 100 segment of data. The data were shuffled at random and divided into three sets: 60% for training, 20% for validation, and 20% for test, respectively. All observations were normalized by dividing the maximum values of input and output. Hence, the data were scaled between -1 and 1.

3.3. Convolutional Neural Networks

CNN is the architecture that is included in the class of ANN for processing data, which has a grid-like topology, such as time series data or images [41] – [43]. The convolutional layers are the fundamental building blocks of CNN architecture [43], [44]. The kernel is a multidimensional array of parameters learned by the backpropagation learning algorithm. Each CNN layer uses a kernel to convolve the multidimensional array of input. These multidimensional arrays are referred to as tensors in the context of DL.

Let *X* and *Y* denote the three dimensional (3-D) input tensor and 3-D output tensor, respectively, whereas K represents the 4-D kernel tensor. Convolving *K* over *X* is obtained by computing the output feature map *Y* [45], [46]. This is given as:

$$Y_{l,m,n} = \sum_{i,j,k} X_{i,m+j-1,n+k-1} K_{l,i,j,k} \tag{3.1}$$

where the indices are elements of the multidimensional arrays *X*, *K*, and *Y*, and the summation is over all valid indices. When a network contains multiple convolutional layers, the output feature map of one layer is used as the input to the next layer [46].

3.3.1. Architecture of 1D CNN

The CNN involves layers such as input, convolutional, pooling, flatten, fully connected, and output (Figure 5) [47]. The convolution layer receives the input features. A filter is applied to an input feature in the convolution layer to create a feature map. The results are activated using an activation function. The convolution layer's output is fed into the pooling layer, which reduces the size of the feature map. The data from the pooled feature map is fed into a flattened

layer, which converts it to a one-dimensional array that can be fed into the next layer. The fully connected layer receives the flattened layer output. The weights are used to process the data in the fully connected layer. The output of the fully connected layer is fed into the output layer [47], [48].

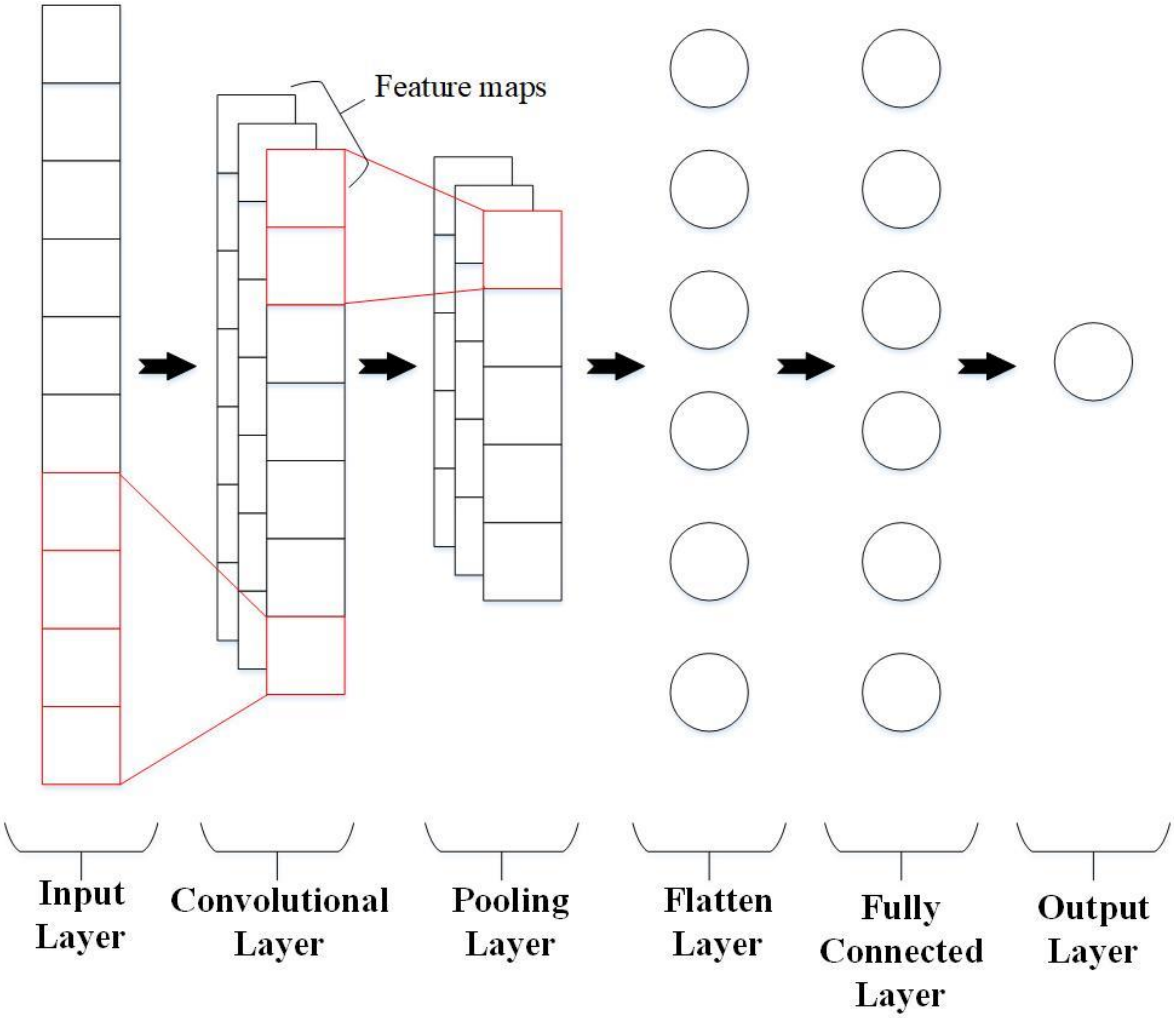


Figure 5: One-dimensional CNN architecture, taken from [47]

After the convolutional operation, activation functions such as the *tanh* function, *sigmoid* function, or *Rectified Linear Unit* (*ReLU*) transform the output value nonlinearly [49]-[52]. The equations of activation functions [49] can be defined as:

$$sigmoid = \phi(x) = \frac{e^x}{1+e^x} \tag{3.2}$$

$$tanh = \phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.3}$$

$$ReLU = \phi(x) = \max(x, 0) \tag{3.4}$$

More complex activation function, such as Leaky ReLU [53], has the same characteristics as ReLU. It has fast training and low computational cost. So, the *ReLU* leakage function [53] can be determined by (3.5),

$$Leaky\ ReLU = \phi(x) = \begin{cases} \alpha x \rightarrow & x \leq 0 \\ x \rightarrow & x > 0 \end{cases} \quad \phi'(x) = \begin{cases} x \rightarrow & x \leq 0 \\ 1 \rightarrow & x > 0 \end{cases} \tag{3.5}$$

The *Leaky ReLU* is used as the activation function in the convolution layer. The activation function is not applied to the remaining layers.

3.4 Tuning of hyperparameters

Estimating the appropriate value of hyperparameters for a learning algorithm is an essential stage and is known as a hyperparameter tuning. The number of filters, learning rate, and kernel size hyperparameters are selected for tuning [54].

The following assumptions are used to define the CNN hyperparameter space: the batch size of 32, epochs from 1 to 100 without early stopping, no more than 3 convolutional layers, the learning rates of 0.001 and 0.0001 with Adam optimizer [55], the number of filters 32, 64,128 for an increasing filter pattern, and 256, 128, 64 for a decreasing filter pattern. Moreover, the size of kernels is considered to be 2 to 5. The *Leaky ReLU* activation function, the convolutional operation, and the max-pooling operation are all present in each layer (except for the last convolutional layer). The above assumptions set the cardinality of the space of hyperparameters to 4800: 100 (epoch size) × 6 (filters) × 4 (kernel size) × 2 (learning rate). 1D CNN models were trained using this hyperparameter space and the model with the best performance on validation set was then identified. For the CNN models both MSE and MAE were recorded, although MSE was used for model selection.

3.5. Implementation

All computations were done on a standard PC with an NVIDIA GeForce GTX 1660Ti graphical processing unit. Furthermore, this PC has an AMD Ryzen 5 4600H processor with a RAM capacity of 16.0 GB.

3.6. Evaluation Metrics

3.6.1. Mean Absolute Error (MAE)

The measure of the absolute difference between two continuous variables can be measured using MAE and is defined as

$$MAE = \frac{1}{N}\sum_{i=1}^{N}(av_i - pv_i)^2 \tag{3.9}$$

where *N* is the number of test data sets, av_i and pv_i are actual and predicted values, respectively.

3.6.2. Mean Squared Error (MSE)

A mean squared error is defined as the squared difference between the actual and predicted values.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}|av_i - pv_i| \tag{3.10}$$

where N, av_i and pv_i are the number of test data sets, actual value and predicted value, respectively.

Chapter 4 – Testing and Results

In this thesis, two different case studies were examined for the collection data. There are motor voltage excitation and single phasing faults with load and no load conditions, respectively.

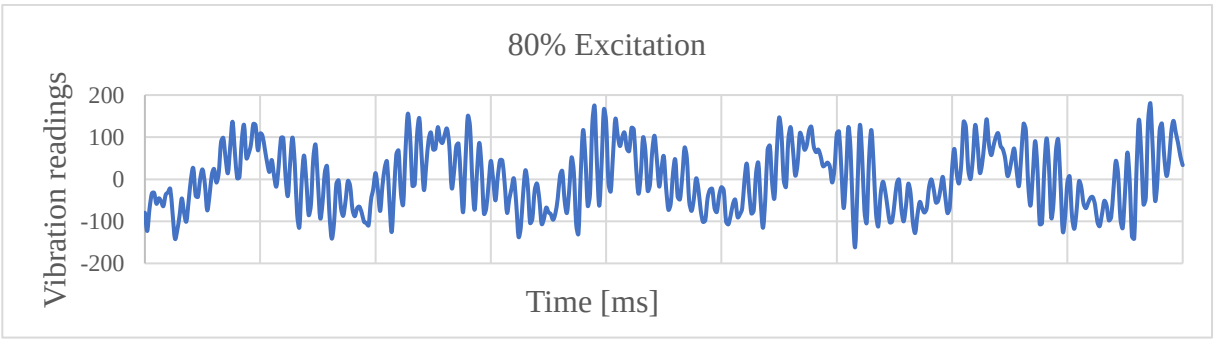
4.1. Experimental measurements

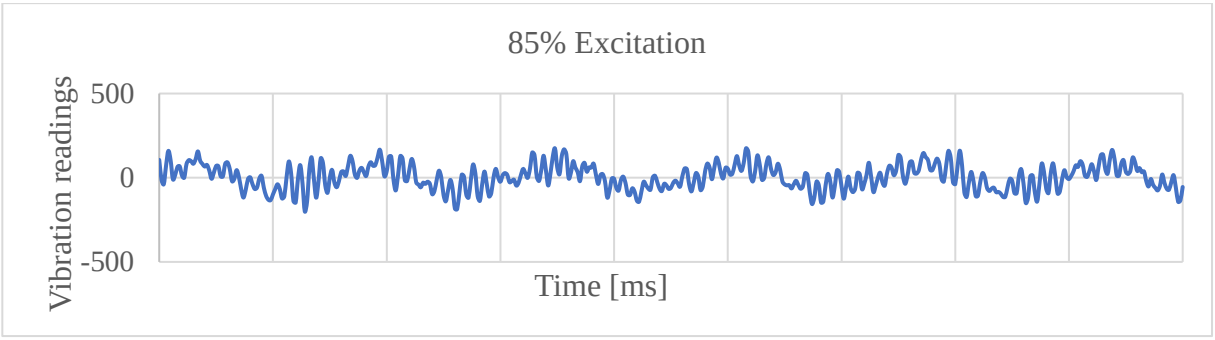
4.1.1. Motor under and over excitations with load and no load condition

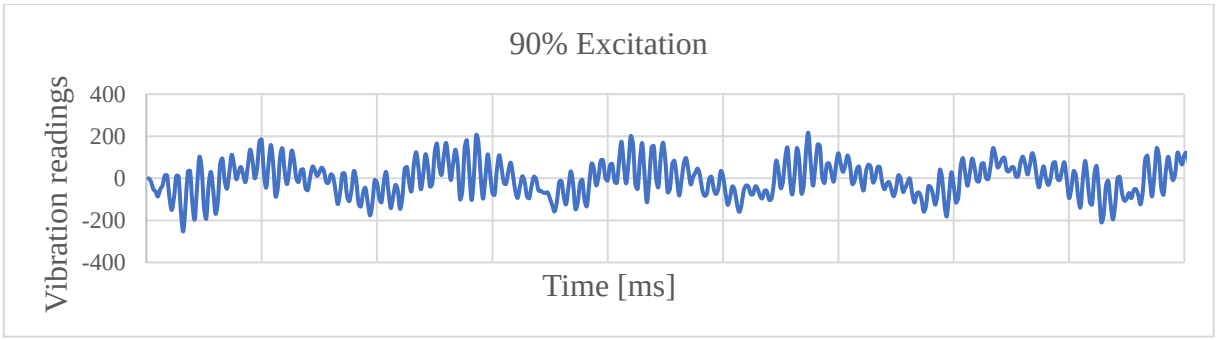
The supply voltage was changed from 80% to 110% with a 5% step shift to find how the vibration signal would be altered when the excitation voltage is changed; see Table 1. The vibration signals were gathered using a three-dimensional Kistler vibration sensor. The stator core and winding vibrations accelerate primarily in one dimension. Therefore, vibration readings of only one dimension were chosen. Data were acquired at a frequency of 20 kHz. Consequently, while exporting data from DewesoftX software to ease signal processing, the sample rate was decreased to 4 kHz. It means 4000 data samples per second were derived. The vibration readings for no load conditions are illustrated in Figure 6.

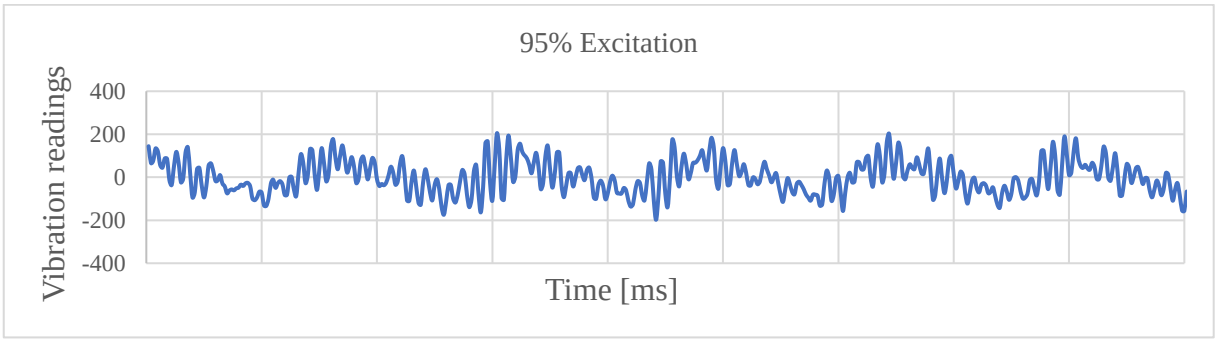
Table 1: Excitation voltages of experimental motor

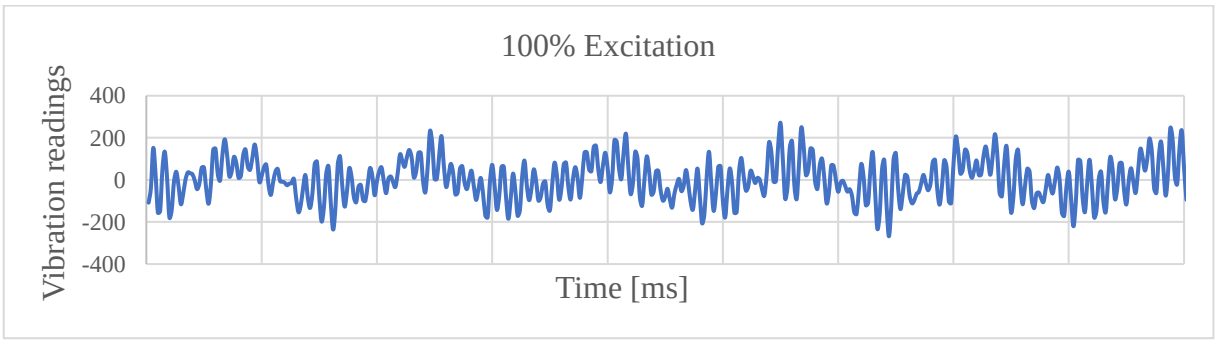
Excitation Voltage %	80	85	90	95	100	105	110
Injected Voltage [V]	320	340	360	380	400	420	440

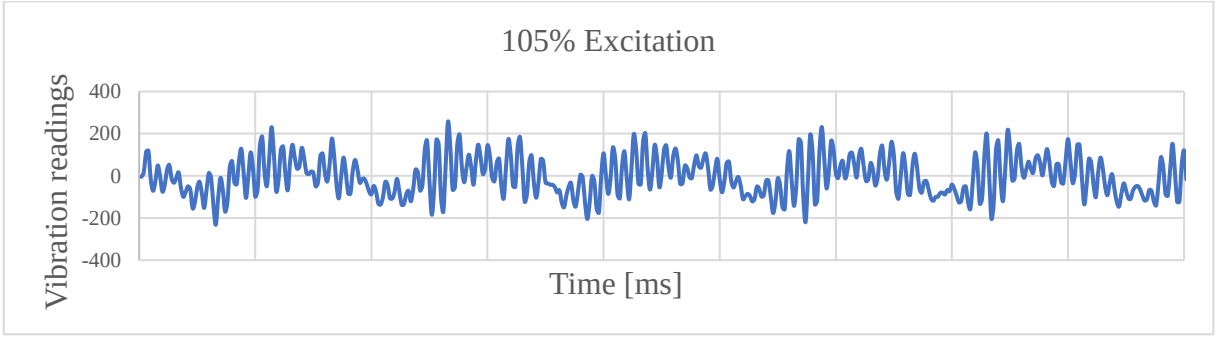



85% Excitation
Vibration readings
500
0
-500
Time [ms]


90% Excitation
Vibration readings
400
200
0
-200
-400
Time [ms]


95% Excitation
Vibration readings
400
200
0
-200
-400
Time [ms]


100% Excitation
Vibration readings
400
200
0
-200
-400
Time [ms]


105% Excitation
Vibration readings
400
200
0
-200
-400
Time [ms]

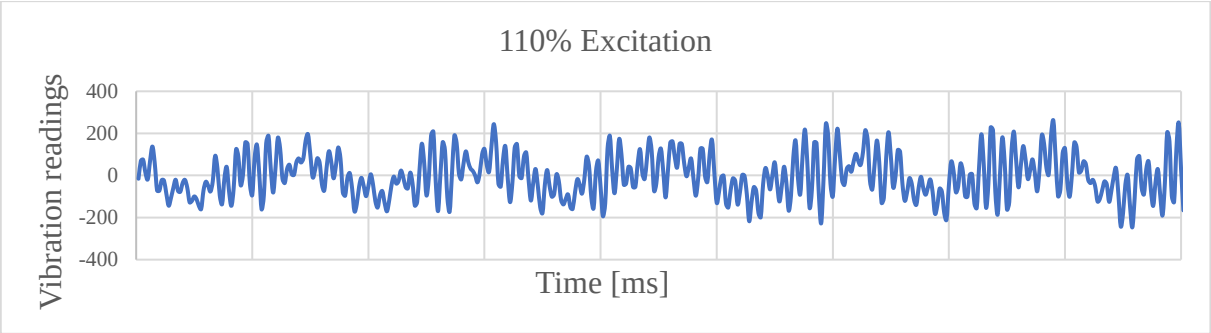
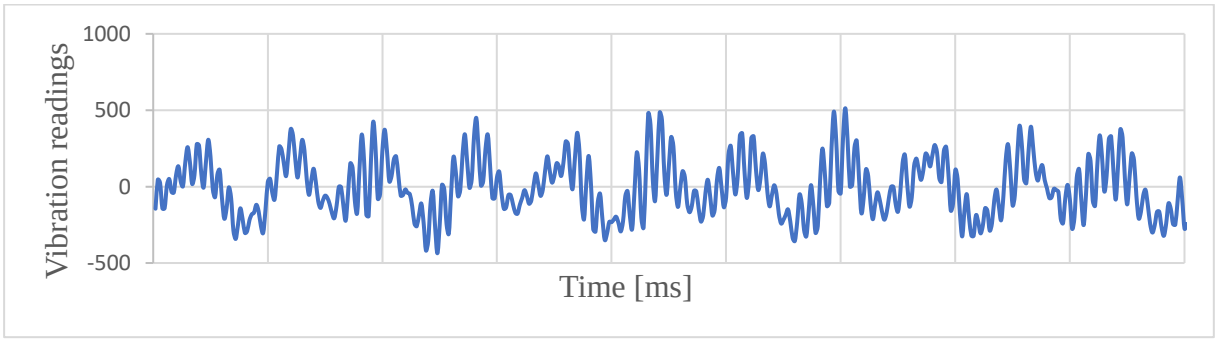


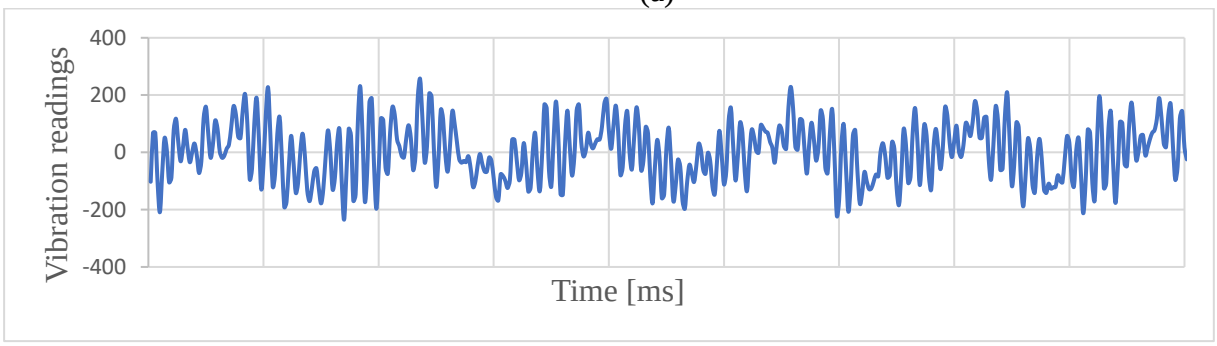
Figure 6: Vibration signals for different excitations

4.1.2. Single phasing of Motor with load and no load condition

The second case study used a switch to examine vibration data and signal processing to recognize the motor's single phasing. The motor's vibration under load and no load conditions were utilized for training a model throughout this procedure while the motor was in service. The generated model was used to predict the state of the motor; precisely, whether it is in a normal state or an abnormal state. An abnormal state means when one phase is disconnected. Similar to the previous case, the sampling rate of the observations was 20 kHz. The vibration sampling rate was decreased to 4 kHz and exported from the DewesoftX software. The vibration signals of normal and abnormal states are shown in Figure 7.



(a)



(b)

Figure 7: Time-series vibration signals, (a) abnormal state, (b) normal state

4.2. 1D CNN model outputs

One-dimensional CNN models were developed for four different cases: voltage excitation with load and no load conditions, and single phasing with load and no load conditions. The scripts for each model are shown in Appendix B.

4.2.1. Voltage excitation with load condition

As mentioned before (Section 3.4), the best 1D CNN model was selected on the lowest MSE achieved on validation set. Table 2 shows the top 10 outcomes of the model selection stage. The complete list of models presented in Table 2 is shown in Table C.1. (see Appendix C). In Table 2, F[f_{i1},.., f_{in}] represents the structure of the CNN classifier with f_i denoting the number of filters in layer *i*, and K[h] identifying the kernel size [height] used in all layers. Table 2 provides that the best model for voltage excitation prognosis has the following structure: F[256, 128, 64], K[5] (the model highlighted in bold in the table). According to Table 2, the MSE and MAE of voltage excitation prediction with load condition equal to 0.000426 and 0.014925, respectively.

Table 2: The selected CNN architecture based on the lowest MSE of voltage excitation prediction with load condition

CNN architectures	Learning rate	MSE	MAE
F[256, 128, 64], K[5]	**0.001**	**0.000426**	**0.014925**
F[32, 64, 128], K[5]	0.001	0.000454	0.015796
F[256, 128, 64], K[4]	0.001	0.000459	0.016186
F[32, 64], K[5]	0.001	0.000468	0.016406
F[256, 128], K[5]	0.001	0.000468	0.01635
F[32, 64, 128], K[3]	0.001	0.00048	0.01617
F[256, 128, 64], K[3]	0.001	0.000513	0.017176
F[32, 64, 128], K[4]	0.001	0.00052	0.016868
F[256, 128], K[3]	0.001	0.000569	0.017961
F[32, 64, 128], K[2]	0.001	0.000602	0.018408

More detailed information about the structure of the best 1D CNN model can be found in Table 3. The architecture has 215839 parameters.

Table 3: The CNN model used for the analysis of load condition model F [256, 128, 64], K(5)

Index	Layer type	Output shape	Number of parameters
1	Conv1D	[86, 256]	1536
2	MaxPooling1D	[43, 256]	0
3	Conv1D	[39, 128]	163968
4	MaxPooling1D	[19, 128]	0
5	Conv1D	[15, 64]	41024
6	MaxPooling1D	[7, 64]	0
7	Flatten	[448]	0
8	Dense	[20]	8980
9	Dense	[15]	315
10	Dense	[1]	16

4.2.2. Voltage excitation with no load condition

According to Table 4, it can be seen that the model that shows the lowest MSE value for voltage excitation with no load condition has the following structure: F[256, 128], K[5] (the model highlighted in bold in the table). The MSE and MAE of the constructed model equal to 0.001637 and 0.028586, respectively. The full list of models shown in Table 4 is illustrated in Table C.2. (see Appendix C).

Table 4: The selected CNN architecture based on the lowest MSE of voltage excitation prediction with no load condition

CNN architectures	Learning rate	MSE	MAE
F[256, 128], K[5]	**0.001**	**0.001637**	**0.028586**
F[256, 128], K[4]	0.001	0.001748	0.029672
F[32, 64], K[5]	0.001	0.001754	0.030453
F[32, 64], K[4]	0.001	0.001808	0.031603
F[256], K[5]	0.001	0.001829	0.03071
F[32, 64, 128], K[5]	0.001	0.001844	0.031011
F[256], K[4]	0.001	0.001892	0.032975
F[256, 128], K[3]	0.001	0.001895	0.031669
F[32, 64], K[3]	0.001	0.001902	0.032656
F[256, 128, 64], K[3]	0.001	0.001936	0.033202

Table 5 shows more detailed information about the structure of this constructed 1D CNN model. It has 2 convolutional layers with the max-pooling operation, and the output is flattened. There are 215839 parameters in this model.

Table 5: The CNN model used for the analysis of no load condition model F [256, 128], K(5)

Index	Layer type	Output shape	Number of parameters
1	Conv1D	[86, 256]	1536
2	MaxPooling1D	[43, 256]	0
3	Conv1D	[39, 128]	163968
4	MaxPooling1D	[19, 128]	0
5	Flatten	[2432]	0
6	Dense	[20]	48660
7	Dense	[15]	315
8	Dense	[1]	16

4.2.3. Single phasing with load condition

The result of the model selection stage is shown in Table 6. The final selected CNN structure of single phasing of the motor is chosen based on the obtained highest accuracy. Furthermore, Table 6 presents that the best model has the following structure: F[256, 128], K[5] (the model highlighted in bold in the table). The highest accuracy is equal to 99.58%. The comprehensive list of models represented in Table 6 is demonstrated in Table C.3. (see Appendix C).

Table 6: The selected CNN architecture based on the highest accuracy of single phasing with load condition

CNN architectures	Learning rate	MSE	Accuracy
F[256, 128], K[5]	**0.001**	**0.003807**	**0.99589**
F[32, 64, 128], K[5]	0.001	0.00415	0.995205
F[256, 128, 64], K[4]	0.001	0.004002	0.993151
F[256, 128, 64], K[5]	0.001	0.004501	0.993151
F[32, 64], K[3]	0.001	0.010146	0.992466
F[32, 64], K[4]	0.001	0.008963	0.992466
F[32], K[5]	0.001	0.011745	0.992466
F[256], K[5]	0.001	0.008244	0.992466
F[32, 64], K[5]	0.001	0.006903	0.991781
F[32, 64, 128], K[3]	0.001	0.007522	0.991096

Furthermore, Table 7 displays detailed information about the CNN model's structure. In this table, Conv1D, MaxPooling1D and Leaky ReLU activation are used to describe convolutional operations. The number of total parameters is 183775.

Table 7: The CNN model used for the analysis of load condition model F [256, 128], K(5)

Index	Layer type	Output shape	Number of parameters
1	Conv1D	[36, 256]	1536
2	MaxPooling1D	[18, 256]	0
3	Conv1D	[14, 128]	163968
4	MaxPooling1D	[7, 128]	0
5	Flatten	[896]	0
6	Dense	[20]	17940
7	Dense	[15]	315
8	Dense	[1]	16

4.2.4. Single phasing with no load condition

Table 8 shows the results of the model selection strategy for one phase disconnection with no load condition. Additionally, Table 8 tabulates that the best model has the following structure: F[256, 128], K[4] (the model highlighted in bold in the table). The highest accuracy of the constructed model is equal to 96.85%. The complete list of models provided in Table 8 is elucidated in Table C.4. (See Appendix C). Moreover, the models with a similar number of filters and the kernel size (F[256, 128, 64], K[5]), but with the different learning rate show the same accuracy, which is equal to 96.3% (the third and fourth model in Table 8).

Table 8: The selected CNN architecture based on the highest accuracy of single phasing with no load condition

CNN architectures	Learning rate	MSE	Accuracy
F[256, 128], K[4]	**0.001**	**0.029235397**	**0.96851852**
F[256, 128, 64], K[4]	0.001	0.026971309	0.96851852
F[256, 128, 64], K[5]	0.001	0.031876623	0.96388889
F[256, 128, 64], K[5]	0.0001	0.031008006	0.96388889
F[32, 64, 128], K[3]	0.001	0.030320587	0.96296296
F[32, 64], K[4]	0.001	0.033742246	0.96296296
F[32], K[5]	0.001	0.031903708	0.96296296
F[32], K[4]	0.001	0.031805294	0.96203704
F[32, 64], K[5]	0.001	0.031429155	0.96203704
F[256, 128], K[3]	0.001	0.033781285	0.96111111

According to Table 8, we observe that the 5 top models have at least 2 convolutional layers and the minimum kernel size is 3. Moreover, Table 9 contains detailed information about the CNN model's structure. The number of total parameters is 201951.

Table 9: The CNN model used for the analysis of no load condition model F [256, 128], K(4)

Index	Layer type	Output shape	Number of parameters
1	Conv1D	[117, 256]	1280
2	MaxPooling1D	[58, 256]	0
3	Conv1D	[55, 128]	131200
4	MaxPooling1D	[27, 128]	0
5	Flatten	[3456]	0
6	Dense	[20]	69140
7	Dense	[15]	315
8	Dense	[1]	16

The average fault detection accuracy of single phasing with and without load were achieved as 95.53% and 98.24%. It shows that the applied data were collected very precisely. Chen *et al.* [51] developed several ML models for rolling bearing fault diagnosis. There are the 1D CNN model, Long-Short Term Memory (LSTM) model, KNN model, Multilayer Perceptron (MLP) model, SVM model and Random Forest model. Additionally, the data were collected by similar experimental grid system using motor with different characteristics. In [51], researchers used the *tanh* function as an activation function in 1D CNN model. The average accuracy of the developed approach was achieved by 99.2%. It shows that in the future the usage of this activation function might give results that are more accurate. Nevertheless, the average accuracies of LSTM, MLP, Random Forest, KNN and SVM are 86.79%, 78.59%, 68.38%, 33.26% and 71.05%, respectively. Overall, in this research 1D CNN model was constructed for accurately detecting faults. The average accuracy of the proposed model is higher by 11.45%, 19.65%, 29.86%, 64.98% and 27.19% than those 5 models, which were described earlier. These outcomes prove the effectiveness of the proposed approach.

Chapter 5 – Conclusion and Future Work

This thesis aimed to study and implement a computational pipeline that can evaluate and analyze a real-time condition assessment for motors/generators using vibrational signal signature and machine learning techniques. Vibration analysis, along with the other available methods for motor/generator assessments was discussed. The problems with the rotational machines' failure were provided, and a wide literature review was conducted to clarify the state of the art in this technology. Mathematical modeling of motor/generator vibrations was analyzed and technically reviewed, and different methodologies to evaluate vibrational signals were explained. It was also explained and discussed that the rotor and stator would oscillate based on the fundamental frequency of the injected signal (voltage). Having this mathematical model, we are able to examine the accuracy of the measurement results and validate the practical data and sensors' performance in motor/generator vibration analysis. In addition, voltage excitation faults such as under-voltage and overvoltage fault types along with single phasing of motors were practically examined for motor fault emulation and prognosis. To achieve the main goal of this research thesis, a single-dimensional CNN was selected as the model of choice for motor vibrational signal evaluation. The reason to choose a single-dimensional CNN model was the flexibility and minimal data preprocessing. In this regard, after conducting an exhaustive search over a prespecified hyperparameter space, a single CNN predictor was created for each fault type. To summarise the results, the MSE of voltage excitation prediction with load, and no load conditions were 0.000426 and 0.001637, respectively. Furthermore, the fault detection accuracy of single phasing with load and no load conditions were achieved as 99.58 and 96.85%. The concept of building a system that can diagnose motor faults is worthwhile, as it would be able to address many industrial requests worldwide.

The future direction of the research will focus on considering bearing fault types, such as inner and outer race and cage faults. Moreover, devices from the Internet of Things (IoT) can be examined to monitor the state of the motor/generator remotely.

Bibliography

[1] D. Mori and T. Ishikawa, “Force and vibration analysis of induction motors,” IEEE Transactions on Magnetics, vol. 41, no. 5, pp. 1948– 1951, 2005.

[2] Z. Fengge, T. Ningze, and W. Fengxiang, “Analysis of vibration modes for large induction motor,” in 2005 International Conference on Electrical Machines and Systems, vol. 1, pp. 64–67 Vol. 1, 2005.

[3] D. K. Soother, J. Daudpoto, and A. Shaikh, “Vibration measurement system for the low power induction motor,” Engineering Science and Technology International Research Journal, vol. 2, no. 4, pp. 53–57, 2018.

[4] A. M. Alturas, “Electrical faults and their effects on induction machines,” vol. 15, pp. 36–46, 12 2019.

[5] T. Van Tung and B.-S. Yang, “Machine fault diagnosis and prognosis: The state of the art,” International Journal of Fluid Machinery and Systems, vol. 2, no. 1, pp. 61–71, 2009.

[6] M. Tsypkin, “Induction motor condition monitoring: Vibration analysis technique - a practical implementation,” in 2011 IEEE International Electric Machines Drives Conference (IEMDC), pp. 406–411, 2011.

[7] M. Tsypkin, “Induction motor condition monitoring: Vibration analysis technique — diagnosis of electromagnetic anomalies,” in 2017 IEEE AUTOTESTCON, pp. 1–7, 2017.

[8] E. G. Strangas, “4 - fault diagnosis and failure prognosis of electrical drives,” in Fault Diagnosis and Prognosis Techniques for Complex Engineering Systems (H. Karimi, ed.), pp. 127–180, Academic Press, 2021.

[9] M. Bagheri, V. Nurmanova, A. Zollanvari, S. Nezhivenko, and B. T. Phung, “Iot application in transformer fault prognosis using vibration signal,” in 2018 IEEE International Conference on High Voltage Engi- neering and Application (ICHVE), pp. 1–5, 2018.

[10] A. Zollanvari, K. Kunanbayev, S. Akhavan Bitaghsir, and M. Bagheri, “Transformer fault prognosis using deep recurrent neural network over vibration signals,” IEEE Transactions on Instrumentation and Measurement, vol. 70, pp. 1–11, 2021.

[11] S. Korkua, H. Jain, W.J. Lee, and C. Kwan, “Wireless health monitoring system for vibration detection of induction motors,” in 2010 IEEE Indus- trial and Commercial Power Systems Technical Conference-Conference Record, pp. 1–6, IEEE, 2010.

[12] M. Rocchi, F. Mosciaro, F. Grottesi, M. Scortichini, A. Giantomassi, M. Pirro, M. Grisostomi, and G. Ippoliti, “Fault prognosis for rotating electrical machines monitoring using recursive least square,” in 2014 6th European Embedded Design in Education and Research Conference (EDERC), pp. 269–273, 2014.

[13] I. Preethi, S. Suryaprakash, and M. Mathankumar, “A state-of-art approach on fault detection in three phase induction motor using ai techniques,” in 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), pp. 567–573, 2021.

[14] D. A. Papathanasopoulos, E. D. Mitronikas, K. N. Giannousakis, andE. S. Dermatas, “An alternative approach for condition monitoring of brushless dc motor drives,” in 2020 International Conference on Electrical Machines (ICEM), vol. 1, pp. 1280–1286, 2020.

[15] J.-H. Lee, J.-H. Pack, and I.-S. Lee, “Fault diagnosis of induction motor using convolutional neural network,” Applied Sciences, vol. 9, no. 15, p. 2950, 2019.

[16] A. Dineva, A. Mosavi, M. Gyimesi, I. Vajda, N. Nabipour, and T. Rabczuk, “Fault diagnosis of rotating electrical machines using multi- label classification,” Applied

Sciences, vol. 9, no. 23, p. 5086, 2019.
[17] T. Plante, A. Nejadpak, and C. Xia Yang, “Faults detection and failures prediction using vibration analysis,” in 2015 IEEE AUTOTESTCON, pp. 227–231, 2015.
[18] M. Z. Ali, M. N. S. K. Shabbir, X. Liang, Y. Zhang, and T. Hu, “Experimental investigation of machine learning based fault diagnosis for induction motors,” in 2018 IEEE Industry Applications Society Annual Meeting (IAS), pp. 1–14, 2018.
[19] M. Iorgulescu and R. Beloiu, “Study of dc motor diagnosis based on the vibration spectrum and current analysis,” in 2012 International Conference on Applied and Theoretical Electricity (ICATE), pp. 1–4, 2012.
[20] K. Kim and A. G. Parlos, “Induction motor fault diagnosis based on neu- ropredictors and wavelet signal processing,” IEEE/ASME Transactions on mechatronics, vol. 7, no. 2, pp. 201–219, 2002.
[21] M. Z. Ali, M. N. S. K. Shabbir, X. Liang, Y. Zhang, and T. Hu, “Machine learning-based fault diagnosis for single- and multi-faults in induction motors using measured stator currents and vibration signals,” IEEE Transactions on Industry Applications, vol. 55, no. 3, pp. 2378– 2391, 2019.
[22] M. Rahman, T. Azam, and S. K. Saha, “Motor fault detection using vibration patterns,” in International Conference on Electrical Computer Engineering (ICECE 2010), pp. 486–489, 2010.
[23] V. S, K. Ram, P. Alagumariappan, and B. B., “Condition monitoring of industrial motors using machine learning classifiers,” SSRN Electronic Journal, 01 2021.
[24] P. Chattopadhyay, N. Saha, C. Delpha, and J. Sil, “Deep learning in fault diagnosis of induction motor drives,” in 2018 Prognostics and System Health Management Conference (PHM-Chongqing), pp. 1068– 1073, 2018.
[25] Z. Fan, H. Yi, J. Xu, K. Xie, Y. Qi, S. Ren, and H. Wang, “Performance study and optimization design of high-speed amorphous alloy induction motor,” Energies, vol. 14, no. 9, 2021.
[26] S. Shao, W. Sun, P. Wang, R. X. Gao, and R. Yan, “Learning features from vibration signals for induction motor fault diagnosis,” in 2016 International Symposium on Flexible Automation (ISFA), pp. 71–76, 2016.
[27] A. Villarroel, G. Zurita, and R. Velarde, “Development of a low-cost vibration measurement system for industrial applications,” Machines, vol. 7, no. 1, 2019.
[28] K. Kudelina, T. Vaimann, B. Asad, A. Rasso˜lkin, A. Kallaste, and G. Demidova, “Trends and challenges in intelligent condition monitor- ing of electrical machines using machine learning,” Applied Sciences, vol. 11, no. 6, 2021.
[29] S. Shao, R. Yan, Y. Lu, P. Wang, and R. X. Gao, “Dcnn-based multi-signal induction motor fault diagnosis,” IEEE Transactions on Instrumentation and Measurement, vol. 69, no. 6, pp. 2658–2669, 2020.
[30] C. Liu, W. Hsaio and Y. Tu, "Time Series Classification With Multivariate Convolutional Neural Network," in IEEE Transactions on Industrial Electronics, vol. 66, no. 6, pp. 4788-4797, June 2019, doi: 10.1109/TIE.2018.2864702.
[31] R. Surovec, J. Bocko and J. Šarloši, "Lateral Rotor Vibration Analysis Model", American Journal of Mechanical Engineering, vol. 2, no. 7, pp. 282-285, 2014. Available: 10.12691/ajme-2-7-23.
[32] R. Tiwari, Rotor Systems: analysis and identification. CRC press, 2017.
[33] C.-W. Lee. Vibration analysis of rotors, vol. 21. Springer Science & Business Media, 1993
[34] M. Bagheri and B. T. Phung, "Frequency response and vibration analysis in transformer winding turn-to-turn fault recognition," 2016 International Conference on Smart Green Technology in Electrical and Information Systems (ICSGTEIS), 2016, pp. 10-15, doi:

10.1109/ICSGTEIS.2016.7885758.

[35] M. Bagheri, A. Zollanvari and S. Nezhivenko, "Transformer Fault Condition Prognosis Using Vibration Signals Over Cloud Environment," in IEEE Access, vol. 6, pp. 9862-9874, 2018, doi: 10.1109/ACCESS.2018.2809436.

[36] M. Drif, N. Benouzza, B. Kraloua, A. Bendiabdellah and J. A. Dente, "Squirrel cage rotor faults detection in induction motor utilizing stator power spectrum approach," 2002 International Conference on Power Electronics, Machines and Drives (Conf. Publ. No. 487), 2002, pp. 133-138, doi: 10.1049/cp:20020102.

[37] S. Chauhan and S. B. Singh, "Effects of Voltage Unbalance and Harmonics on 3-Phase Induction Motor During the Condition of Undervoltage and Overvoltage," 2019 6th International Conference on Signal Processing and Integrated Networks (SPIN), 2019, pp. 1141-1146, doi: 10.1109/SPIN.2019.8711753.

[38] M. Proctor and T. Smith, "Application of Undervoltage Protection to Critical Motors," 2019 IEEE IAS Pulp, Paper and Forest Industries Conference (PPFIC), 2019, pp. 1-5, doi: 10.1109/PPFIC43189.2019.9052381.

[39] H. D. Do, A. Anuchin, D. Shpak, A. Zharkov and A. Rusakov, "Overvoltage protection for interior permanent magnet synchronous motor testbench," 2018 25th International Workshop on Electric Drives: Optimization in Control of Electric Drives (IWED), 2018, pp. 1-4, doi: 10.1109/IWED.2018.8321396.

[40] W. Kersting, "Causes and Effects of Single-Phasing Induction Motors", *IEEE Transactions on Industry Applications*, vol. 41, no. 6, pp. 1499-1505, 2005. Available: 10.1109/tia.2005.857467.

[41] R. Ribeiro Junior, F. de Almeida and G. Gomes, "Fault classification in three-phase motors based on vibration signal analysis and artificial neural networks", *Neural Computing and Applications*, vol. 32, no. 18, pp. 15171-15189, 2020. Available: 10.1007/s00521-020-04868-w.

[42] T. Ince, S. Kiranyaz, L. Eren, M. Askar and M. Gabbouj, "Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks", *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 7067-7075, 2016. Available: 10.1109/tie.2016.2582729.

[43] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks", *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 2017. Available: 10.1145/3065386.

[44] L. Wen, X. Li, L. Gao and Y. Zhang, "A New Convolutional Neural Network-Based Data-Driven Fault Diagnosis Method", *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5990-5998, 2018. Available: 10.1109/tie.2017.2774777.

[45] A. Serikbay, M. Bagheri, A. Zollanvari and B. T. Phung, "Accurate Surface Condition Classification of High Voltage Insulators based on Deep Convolutional Neural Networks," in IEEE Transactions on Dielectrics and Electrical Insulation, vol. 28, no. 6, pp. 2126-2133, December 2021, doi: 10.1109/TDEI.2021.009648.

[46] B. Abibullaev and A. Zollanvari, "A Systematic Deep Learning Model Selection for P300-Based Brain–Computer Interfaces," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 52, no. 5, pp. 2744-2756, May 2022, doi: 10.1109/TSMC.2021.3051136.

[47] E. Chaerun Nisa and Y. Kuan, "Comparative Assessment to Predict and Forecast Water-Cooled Chiller Power Consumption Using Machine Learning and Deep Learning Algorithms", *Sustainability*, vol. 13, no. 2, p. 744, 2021. Available: 10.3390/su13020744.

[48] L. Eren, "Bearing Fault Detection by One-Dimensional Convolutional Neural Networks", *Mathematical Problems in Engineering*, vol. 2017, pp. 1-9, 2017. Available: 10.1155/2017/8617315.

[49] J. Jiao, M. Zhao, J. Lin and K. Liang, "A comprehensive review on convolutional neural network in machine fault diagnosis", Neurocomputing, vol. 417, pp. 36-63, 2020. Available: 10.1016/j.neucom.2020.07.088.
[50] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj and D. Inman, "1D convolutional neural networks and applications: A survey", *Mechanical Systems and Signal Processing*, vol. 151, p. 107398, 2021. Available: 10.1016/j.ymssp.2020.107398.
[51] C. Chen, Z. Liu, G. Yang, C. Wu and Q. Ye, "An Improved Fault Diagnosis Using 1D-Convolutional Neural Network Model", *Electronics*, vol. 10, no. 1, p. 59, 2020. Available: 10.3390/electronics10010059.
[52] R. Mukhopadhyay, P. S. Panigrahy, G. Misra and P. Chattopadhyay, "Quasi 1D CNN-based Fault Diagnosis of Induction Motor Drives," 2018 5th International Conference on Electric Power and Energy Conversion Systems (EPECS), 2018, pp. 1-5, doi: 10.1109/EPECS.2018.8443552.
[53] R. Junior, I. Areias, M. Campos, C. Teixeira, L. da Silva and G. Gomes, "Fault detection and diagnosis in electric motors using 1d convolutional neural networks with multi-channel vibration signals", *Measurement*, vol. 190, p. 110759, 2022. Available: 10.1016/j.measurement.2022.110759.
[54] F. Chollet, *Deep learning with Python*.
[55] D. Kingma and J. Ba, "Adam: A method for stochastic optimization", 2014, arXiv preprint arXiv: 1412.6980.

Appendix

Appendix A

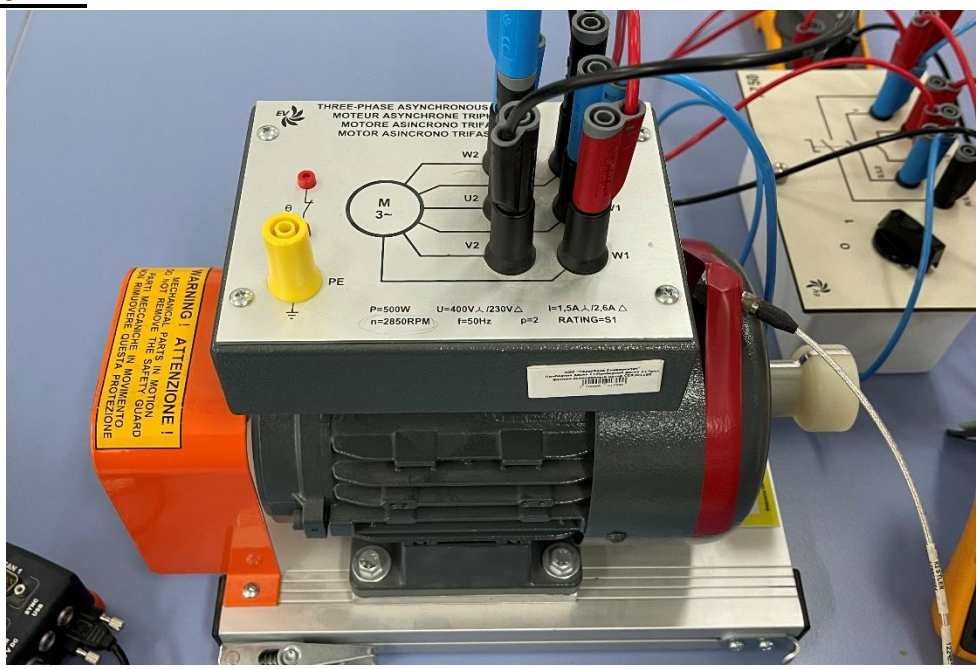


Figure A.1 The three phase induction motor

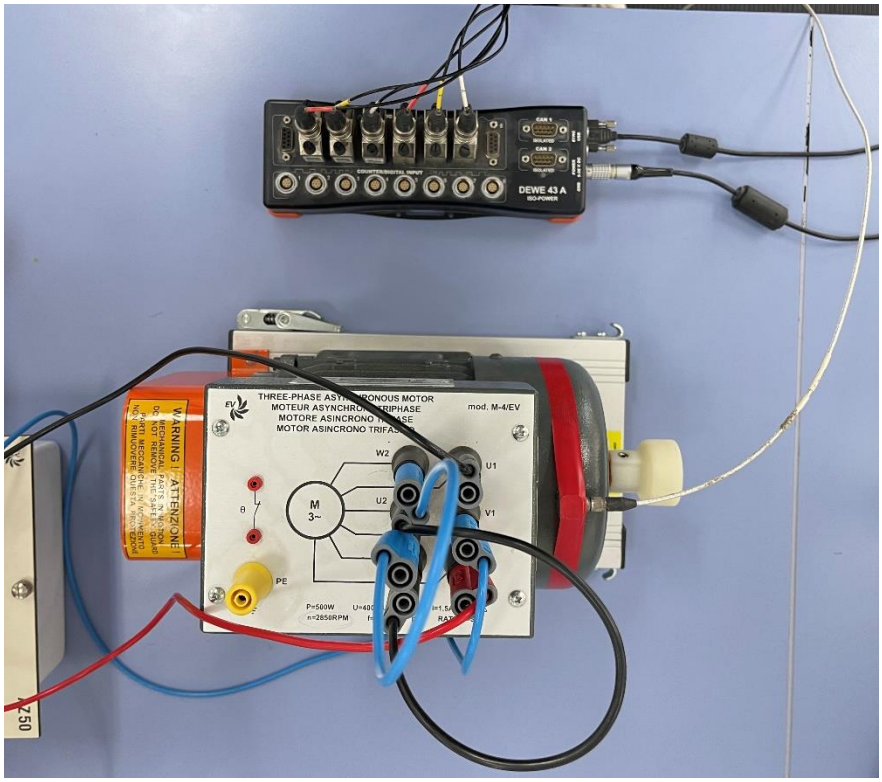


Figure A.2 The connection of vibration sensor

Appendix B. Code scripts

Voltage excitation with load condition

```
import pandas as pd
from numpy import array
import numpy as np
from tensorflow import keras
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten
from tensorflow.keras.metrics import RootMeanSquaredError
from random import shuffle
from sklearn.metrics import mean_absolute_error, mean_squared_error
ind_file = ['80','85','90','95','100','105','110']
max_nl = 0
dataset = []
for ind in ind_file:
    f_nl = pd.read_csv(ind+'_L.csv')
    volt_nl = float(ind)*np.ones((len(f_nl['AI 7']),1))
    if max_nl>=np.amax(np.array(f_nl[['AI 7']])):
      max_nl = max_nl
    else:
      max_nl = np.amax(np.array(f_nl[['AI 7']]))
    f_nl = np.concatenate((f_nl[['AI 7']], volt_nl),axis=1)
    dataset.append(f_nl)
dataset = np.concatenate(dataset,axis =0)/np.array([max_nl,110])
print(dataset)
print(dataset.shape)
from random import shuffle,seed
def split_sequences(sequences, steps):
    X, Y = list(), list()
    data = list()
    for i in range(len(sequences)):
        head = i*steps
        end = head + steps
        if end > len(sequences):
            break
        sequence_x, sequence_y = sequences[head:end, :-1], sequences[end-1, -1]
        data.append([sequence_x,sequence_y])
    seed(42)
    shuffle(data)
    for x, y in data:
        X.append(x)
        Y.append(y)
    return np.array(X), np.array(Y)

steps = 80
X, y = split_sequences(dataset, steps)
features = 1
import tensorflow as tf
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
```

```
result = list()
def get_model(X_train, X_test, y_train, y_test):
   adams = [0.001, 0.0001]
   num_filters = [[32],[32,64],[32,64,128],
            [256],[256,128],[256,128,64]]
   kernel_sizes = [2, 3, 4, 5]
   for adam in adams:
      print('adam: ',adam)
      for kernel_size in kernel_sizes:
         print('kernel_size:', kernel_size)
         for num_filter in num_filters:
            print('filter,', num_filter)
            file = 'File_cnn'
            for fil in num_filter:
               file = file + '_' + str(fil)
            file = file + str(adam) + '_kernel_size_' + str(kernel_size)
            model_name = file + '.h5'
            model = keras.Sequential() #initialization
            if len(num_filter) == 1:
               model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1),input_shape=(steps, features)))
               model.add(MaxPooling1D())
            elif len(num_filter) == 2:
               model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
input_shape=(steps, features), activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
               model.add(MaxPooling1D())
               model.add(Conv1D(filters=num_filter[1], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
               model.add(MaxPooling1D())

            elif len(num_filter) == 3:
               model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
input_shape=(steps, features), activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
               model.add(MaxPooling1D())
               model.add(Conv1D(filters=num_filter[1], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
               model.add(MaxPooling1D())
               model.add(Conv1D(filters=num_filter[2], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
               model.add(MaxPooling1D())
               model.add(Flatten())
               model.add(Dense(units=20, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
               model.add(Dense(units=15, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
               model.add(Dense(units=1, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
               model.compile(loss='mse',
            optimizer=tf.keras.optimizers.Adam(learning_rate=adam),
            metrics=['accuracy']) #compilation
            history = model.fit(X_train, y_train, batch_size=32, epochs=100, validation_split =
0.2)
           yhat = model.predict(X_test)
```

```
            mse = mean_squared_error(y_test, yhat)
            mae = mean_absolute_error(y_test, yhat)
            print('MSE: ', mse)
            print('MAE:', mae)
            result.append([file, mse, mae])
df = pd.DataFrame(result, columns = ['model','MSE', 'MAE'])
df.sort_values(by='MSE')
df.to_excel('CNN_Load_model.xlsx')
get_model(X_train, X_test, y_train, y_test)
```

Voltage excitation with no load condition

```
import pandas as pd
from numpy import array
import numpy as np
from tensorflow import keras
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten
from tensorflow.keras.metrics import RootMeanSquaredError
from random import shuffle
from sklearn.metrics import mean_absolute_error, mean_squared_error
ind_file = ['80','85','90','95','100','105','110']
max_nl = 0
dataset = []
for ind in ind_file:
   f_nl = pd.read_csv(ind+'_nL.csv')
   volt_nl = float(ind)*np.ones((len(f_nl['AI 6']),1))
   if max_nl>=np.amax(np.array(f_nl[['AI 6']])):
     max_nl = max_nl
   else:
     max_nl = np.amax(np.array(f_nl[['AI 6']]))
   f_nl = np.concatenate((f_nl[['AI 6']], volt_nl),axis=1)
   dataset.append(f_nl)
   #len_nl = len(volt_nl)
dataset = np.concatenate(dataset,axis =0)/np.array([max_nl,110])
#print(f_nl.shape)
print(dataset)
print(dataset.shape)
from random import shuffle,seed
def split_sequences(sequences, steps):
   X, Y = list(), list()
   data = list()
   for i in range(len(sequences)):
      head = i*steps
      end = head + steps
      if end > len(sequences):
         break
      sequence_x, sequence_y = sequences[head:end, :-1], sequences[end-1, -1]
      data.append([sequence_x,sequence_y])
   seed(42)
   shuffle(data)
   for x, y in data:
```

```
        X.append(x)
        Y.append(y)
    return np.array(X), np.array(Y)

steps = 80
X, y = split_sequences(dataset, steps)
features = 1
# summarize the data
import tensorflow as tf
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
result = list()
def get_model(X_train, X_test, y_train, y_test):
    adams = [0.001, 0.0001]
    num_filters = [[32],[32,64],[32,64,128],
                [256],[256,128],[256,128,64]]
    kernel_sizes = [2, 3, 4, 5]
    for adam in adams:
        print('adam: ',adam)
        for kernel_size in kernel_sizes:
            print('kernel_size:', kernel_size)
            for num_filter in num_filters:
                print('filter,', num_filter)
                file = 'File_cnn'
                for fil in num_filter:
                    file = file + '_' + str(fil)
                file = file + str(adam) + '_kernel_size_' + str(kernel_size)
                model_name = file + '.h5'
                model = keras.Sequential() #initialization
                if len(num_filter) == 1:
                    model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1),input_shape=(steps, features)))
                    model.add(MaxPooling1D())
                elif len(num_filter) == 2:
                    model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
input_shape=(steps, features), activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                    model.add(MaxPooling1D())
                    model.add(Conv1D(filters=num_filter[1], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                    model.add(MaxPooling1D())
                elif len(num_filter) == 3:
                    model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
input_shape=(steps, features), activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                    model.add(MaxPooling1D())
                    model.add(Conv1D(filters=num_filter[1], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                    model.add(MaxPooling1D())
                    model.add(Conv1D(filters=num_filter[2], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                    model.add(MaxPooling1D())
```

```
                model.add(Flatten())
                model.add(Dense(units=20, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(Dense(units=15, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(Dense(units=1, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.compile(loss='mse',
             optimizer=tf.keras.optimizers.Adam(learning_rate=adam),
             metrics=['accuracy']) #compilation
             history = model.fit(X_train, y_train, batch_size=32, epochs=100, validation_split =
0.2)
            yhat = model.predict(X_test)
            mse = mean_squared_error(y_test, yhat)
            mae = mean_absolute_error(y_test, yhat)
            print('MSE: ', mse)
            print('MAE:', mae)
            result.append([file, mse, mae])
df = pd.DataFrame(result, columns = ['model','MSE', 'MAE'])
df.sort_values(by='MSE')
df.to_excel('CNN_noLoad_model.xlsx')
get_model(X_train, X_test, y_train, y_test)
```

Single phasing with load condition

```
import pandas as pd
from numpy import array
import numpy as np
from tensorflow import keras
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten
from tensorflow.keras.metrics import RootMeanSquaredError
from random import shuffle
from sklearn.metrics import mean_absolute_error, mean_squared_error
ind_file = ['0','1']
max_nl = 0
dataset = []
for ind in ind_file:
    f_nl = pd.read_csv(ind+'_L.csv')
    volt_nl = float(ind)*np.ones((len(f_nl['AI 6']),1))
    if max_nl>=np.amax(np.array(f_nl[['AI 6']])):
      max_nl = max_nl
    else:
      max_nl = np.amax(np.array(f_nl[['AI 6']]))
    f_nl = np.concatenate((f_nl[['AI 6']], volt_nl),axis=1)
    dataset.append(f_nl)
    #len_nl = len(volt_nl)
dataset = np.concatenate(dataset,axis =0)/np.array([max_nl,1])
#print(f_nl.shape)
print(dataset)
from random import shuffle,seed
def split_sequences(sequences, steps):
    X, Y = list(), list()
    data = list()
    for i in range(len(sequences)):
```

```
        head = i*steps
        end = head + steps
        if end > len(sequences):
            break
        sequence_x, sequence_y = sequences[head:end, :-1], sequences[end-1, -1]
        data.append([sequence_x,sequence_y])
    seed(42)
    shuffle(data)
    for x, y in data:
        X.append(x)
        Y.append(y)
    return np.array(X), np.array(Y)

steps = 40
X, y = split_sequences(dataset, steps)
features = 1
# summarize the data
import tensorflow as tf
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
result = list()
def get_model(X_train, X_test, y_train, y_test):
    adams = [0.001, 0.0001]
    num_filters = [[32],[32,64],[32,64,128],
                [256],[256,128],[256,128,64]]
    kernel_sizes = [2, 3, 4, 5]
    for adam in adams:
        print('adam: ',adam)
        for kernel_size in kernel_sizes:
            print('kernel_size:', kernel_size)
            for num_filter in num_filters:
                print('filter,', num_filter)
                file = 'File_cnn'
                for fil in num_filter:
                    file = file + '_' + str(fil)
                file = file + str(adam) + '_kernel_size_' + str(kernel_size)
                model_name = file + '.h5'
                model = keras.Sequential() #initialization
                if len(num_filter) == 1:
                    model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1),input_shape=(steps, features)))
                    model.add(MaxPooling1D())
                elif len(num_filter) == 2:
                    model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
input_shape=(steps, features), activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                    model.add(MaxPooling1D())
                    model.add(Conv1D(filters=num_filter[1], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                    model.add(MaxPooling1D())
                elif len(num_filter) == 3:
```

```
                model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
input_shape=(steps, features), activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(MaxPooling1D())
                model.add(Conv1D(filters=num_filter[1], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(MaxPooling1D())
                model.add(Conv1D(filters=num_filter[2], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(MaxPooling1D())
                model.add(Flatten())
                model.add(Dense(units=20, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(Dense(units=15, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(Dense(units=1, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.compile(loss='mse',
             optimizer=tf.keras.optimizers.Adam(learning_rate=adam),
             metrics=['accuracy']) #compilation
             history = model.fit(X_train, y_train, batch_size=32, epochs=100, validation_split =
0.2)
            yhat = model.predict(X_test)
            mse = mean_squared_error(y_test, yhat)
            n=0
            for i in range(len(yhat)):
                 yhat[i]=round(float(yhat[i]))
                 if yhat[i] == y_test[i]:
                    n=n+1
            accuracy = n/len(yhat)
            print('MSE: ', mse)
            print('accuracy: ', accuracy)
            result.append([file, mse, accuracy])
df = pd.DataFrame(result, columns = ['model','MSE', 'Accuracy'])
df.sort_values(by='MSE')
df.to_excel('CNN_Load_1phase_model.xlsx')
get_model(X_train, X_test, y_train, y_test)
```

Single phasing with no load condition

```
import pandas as pd
from numpy import array
import numpy as np
from tensorflow import keras
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten
from tensorflow.keras.metrics import RootMeanSquaredError
from random import shuffle
from sklearn.metrics import mean_absolute_error, mean_squared_error
ind_file = ['0','1']
max_nl = 0
dataset = []
for ind in ind_file:
   f_nl = pd.read_csv(ind+'_nL.csv')
   volt_nl = float(ind)*np.ones((len(f_nl['AI 6']),1))
   if max_nl>=np.amax(np.array(f_nl[['AI 6']])):
```

```
      max_nl = max_nl
    else:
      max_nl = np.amax(np.array(f_nl[['AI 6']]))
    f_nl = np.concatenate((f_nl[['AI 6']], volt_nl),axis=1)
    dataset.append(f_nl)
dataset = np.concatenate(dataset,axis =0)/np.array([max_nl,1])
from random import shuffle,seed
def split_sequences(sequences, steps):
    X, Y = list(), list()
    data = list()
    for i in range(len(sequences)):
        head = i*steps
        end = head + steps
        if end > len(sequences):
            break
        sequence_x, sequence_y = sequences[head:end, :-1], sequences[end-1, -1]
        data.append([sequence_x,sequence_y])
    seed(42)
    shuffle(data)
    for x, y in data:
        X.append(x)
        Y.append(y)
    return np.array(X), np.array(Y)

steps = 40
X, y = split_sequences(dataset, steps)
features = 1
import tensorflow as tf
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8)
result = list()
def get_model(X_train, X_test, y_train, y_test):
    adams = [0.001, 0.0001]
    num_filters = [[32],[32,64],[32,64,128],
                [256],[256,128],[256,128,64]]
    kernel_sizes = [2, 3, 4, 5]
    for adam in adams:
        print('adam: ',adam)
        for kernel_size in kernel_sizes:
            print('kernel_size:', kernel_size)
            for num_filter in num_filters:
                print('filter,', num_filter)
                file = 'File_cnn'
                for fil in num_filter:
                    file = file + '_' + str(fil)
                file = file + str(adam) + '_kernel_size_' + str(kernel_size)
                model_name = file + '.h5'
                model = keras.Sequential() #initialization
                if len(num_filter) == 1:
```

```
                model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1),input_shape=(steps, features)))
                model.add(MaxPooling1D())
            elif len(num_filter) == 2:
                model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
input_shape=(steps, features), activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(MaxPooling1D())
                model.add(Conv1D(filters=num_filter[1], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(MaxPooling1D())
            elif len(num_filter) == 3:
                model.add(Conv1D(filters=num_filter[0], kernel_size=kernel_size,
input_shape=(steps, features), activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(MaxPooling1D())
                model.add(Conv1D(filters=num_filter[1], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(MaxPooling1D())
                model.add(Conv1D(filters=num_filter[2], kernel_size=kernel_size,
activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(MaxPooling1D())
                model.add(Flatten())
                model.add(Dense(units=20, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(Dense(units=15, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.add(Dense(units=1, activation=tf.keras.layers.LeakyReLU(alpha=0.1)))
                model.compile(loss='mse',
            optimizer=tf.keras.optimizers.Adam(learning_rate=adam),
            metrics=['accuracy']) #compilation
            history = model.fit(X_train, y_train, batch_size=32, epochs=100, validation_split =
0.2)
            yhat = model.predict(X_test)
            mse = mean_squared_error(y_test, yhat)
            n=0
            for i in range(len(yhat)):
                yhat[i]=round(float(yhat[i]))
                if yhat[i] == y_test[i]:
                    n=n+1
            accuracy = n/len(yhat)
            print('MSE: ', mse)
            print('accuracy: ', accuracy)
            result.append([file, mse, accuracy])
df = pd.DataFrame(result, columns = ['model','MSE', 'Accuracy'])
df.sort_values(by='MSE')
df.to_excel('CNN_noLoad_1phase_model.xlsx')
get_model(X_train, X_test, y_train, y_test)
```

Appendix C

Table C.1: Architecture of voltage unbalances with load condition

CNN architectures	Learning rate	MSE	MAE
F[256, 128, 64], K[5]	**0.001**	**0.000426**	**0.014925**
F[32, 64, 128], K[5]	0.001	0.000454	0.015796
F[256, 128, 64], K[4]	0.001	0.000459	0.016186
F[32, 64], K[5]	0.001	0.000468	0.016406
F[256, 128], K[5]	0.001	0.000468	0.01635
F[32, 64, 128], K[3]	0.001	0.00048	0.01617
F[256, 128, 64], K[3]	0.001	0.000513	0.017176
F[32, 64, 128], K[4]	0.001	0.00052	0.016868
F[256, 128], K[3]	0.001	0.000569	0.017961
F[32, 64, 128], K[2]	0.001	0.000602	0.018408
F[256, 128, 64], K[2]	0.001	0.00061	0.018609
F[32, 64], K[4]	0.001	0.000628	0.019301
F[256, 128, 64], K[5]	0.0001	0.00064	0.019742
F[32, 64, 128], K[5]	0.0001	0.000661	0.019683
F[256], K[5]	0.001	0.000668	0.019541
F[32, 64], K[3]	0.001	0.000687	0.020073
F[256], K[3]	0.001	0.000727	0.020509
F[32, 64, 128], K[4]	0.0001	0.00073	0.021338
F[256, 128], K[2]	0.001	0.000748	0.020566
F[256, 128, 64], K[4]	0.0001	0.000749	0.021432
F[256], K[4]	0.001	0.000806	0.021753
F[256, 128], K[4]	0.001	0.000821	0.022442
F[256, 128], K[4]	0.0001	0.000833	0.022525
F[32, 64, 128], K[3]	0.0001	0.000847	0.022998
F[32, 64], K[2]	0.001	0.000875	0.022292
F[32], K[4]	0.001	0.000921	0.023764
F[256, 128], K[5]	0.0001	0.000933	0.024526
F[256, 128, 64], K[3]	0.0001	0.000937	0.024293
F[32], K[5]	0.001	0.000971	0.023994
F[32, 64], K[5]	0.0001	0.000972	0.024158
F[256, 128, 64], K[2]	0.0001	0.000982	0.024174
F[32], K[3]	0.001	0.000985	0.02409
F[256, 128], K[3]	0.0001	0.001019	0.024991
F[32, 64, 128], K[2]	0.0001	0.001022	0.024709
F[256], K[2]	0.001	0.001043	0.024658
F[256, 128], K[2]	0.0001	0.001186	0.026787
F[32, 64], K[3]	0.0001	0.001228	0.027576
F[32, 64], K[4]	0.0001	0.00141	0.029159
F[256], K[5]	0.0001	0.001493	0.030138
F[32, 64], K[2]	0.0001	0.001596	0.031106
F[32], K[2]	0.001	0.001655	0.031132
F[256], K[4]	0.0001	0.001701	0.032307
F[32], K[5]	0.0001	0.001743	0.032918
F[256], K[3]	0.0001	0.001744	0.032875
F[32], K[2]	0.0001	0.001779	0.033316
F[256], K[2]	0.0001	0.001891	0.034107
F[32], K[3]	0.0001	0.002042	0.035607
F[32], K[4]	0.0001	0.002067	0.035998

Table C.2: Architectures of voltage unbalances with no load condition

CNN architectures	Learning rate	MSE	MAE
F[256, 128], K[5]	**0.001**	**0.001637**	**0.028586**
F[256, 128], K[4]	0.001	0.001748	0.029672
F[32, 64], K[5]	0.001	0.001754	0.030453
F[32, 64], K[4]	0.001	0.001808	0.031603
F[256], K[5]	0.001	0.001829	0.03071
F[32, 64, 128], K[5]	0.001	0.001844	0.031011
F[256], K[4]	0.001	0.001892	0.032975
F[256, 128], K[3]	0.001	0.001895	0.031669
F[32, 64], K[3]	0.001	0.001902	0.032656
F[256, 128, 64], K[3]	0.001	0.001936	0.033202
F[32], K[5]	0.001	0.001952	0.034049
F[256], K[3]	0.001	0.001954	0.032285
F[32], K[3]	0.001	0.00196	0.03356
F[256, 128, 64], K[5]	0.001	0.001978	0.030769
F[32], K[4]	0.001	0.002021	0.034637
F[32, 64, 128], K[3]	0.001	0.002029	0.033056
F[32, 64, 128], K[2]	0.001	0.002068	0.034507
F[32, 64, 128], K[4]	0.001	0.002068	0.032814
F[256, 128], K[4]	0.0001	0.002075	0.035617
F[256, 128, 64], K[4]	0.001	0.002142	0.033801
F[256], K[2]	0.001	0.002146	0.034853
F[256, 128], K[3]	0.0001	0.002177	0.036264
F[256, 128, 64], K[5]	0.0001	0.002199	0.036708
F[32], K[2]	0.001	0.002209	0.036645
F[256, 128, 64], K[4]	0.0001	0.002224	0.036929
F[256, 128, 64], K[2]	0.001	0.002241	0.036106
F[256, 128], K[2]	0.001	0.002298	0.036218
F[32, 64, 128], K[5]	0.0001	0.002299	0.037531
F[256], K[2]	0.0001	0.002327	0.03766
F[256, 128], K[5]	0.0001	0.002333	0.037897
F[256, 128, 64], K[2]	0.0001	0.002335	0.03792
F[256, 128, 64], K[3]	0.0001	0.002498	0.039912
F[32, 64], K[5]	0.0001	0.002527	0.039707
F[32, 64], K[2]	0.0001	0.002529	0.039193
F[32, 64, 128], K[3]	0.0001	0.002591	0.040328
F[32], K[4]	0.0001	0.00261	0.040612
F[32, 64], K[2]	0.001	0.002634	0.039592
F[32], K[2]	0.0001	0.002648	0.040593
F[32, 64, 128], K[4]	0.0001	0.002649	0.040444
F[256], K[4]	0.0001	0.002675	0.040855
F[32, 64], K[3]	0.0001	0.002709	0.041013
F[32, 64, 128], K[2]	0.0001	0.00275	0.041516
F[32, 64], K[4]	0.0001	0.002752	0.041788
F[32], K[5]	0.0001	0.002754	0.041451
F[256], K[5]	0.0001	0.002821	0.042651
F[256, 128], K[2]	0.0001	0.002859	0.042312
F[256], K[3]	0.0001	0.00304	0.043872
F[32], K[3]	0.0001	0.003076	0.043996

Table C.3: Architectures of single phasing with Load condition

CNN architectures	Learning rate	MSE	Accuracy
F[256, 128], K[5]	**0.001**	**0.003807**	**0.99589**
F[32, 64, 128], K[5]	0.001	0.00415	0.995205

F[256, 128, 64], K[4]	0.001	0.004002	0.993151
F[256, 128, 64], K[5]	0.001	0.004501	0.993151
F[32, 64], K[3]	0.001	0.010146	0.992466
F[32, 64], K[4]	0.001	0.008963	0.992466
F[32], K[5]	0.001	0.011745	0.992466
F[256], K[5]	0.001	0.008244	0.992466
F[32, 64], K[5]	0.001	0.006903	0.991781
F[32, 64, 128], K[3]	0.001	0.007522	0.991096
F[256], K[4]	0.001	0.011187	0.991096
F[32, 64, 128], K[4]	0.001	0.00783	0.990411
F[256, 128, 64], K[2]	0.001	0.008302	0.989726
F[256, 128], K[4]	0.001	0.007413	0.989041
F[256, 128, 64], K[5]	0.0001	0.011435	0.989041
F[32], K[2]	0.001	0.015751	0.988356
F[32], K[3]	0.001	0.015457	0.988356
F[32, 64, 128], K[2]	0.001	0.010615	0.987671
F[256, 128], K[2]	0.001	0.011999	0.986986
F[256], K[3]	0.001	0.014716	0.985616
F[256, 128], K[3]	0.001	0.011711	0.985616
F[256, 128, 64], K[4]	0.0001	0.015574	0.985616
F[32, 64], K[2]	0.001	0.01464	0.984932
F[32], K[4]	0.001	0.015923	0.984932
F[256, 128, 64], K[3]	0.0001	0.01551	0.984247
F[256], K[2]	0.001	0.020053	0.980822
F[32, 64, 128], K[2]	0.0001	0.019164	0.980822
F[32, 64, 128], K[4]	0.0001	0.017261	0.980822
F[32, 64, 128], K[5]	0.0001	0.0156	0.980822
F[256, 128], K[5]	0.0001	0.017084	0.980822
F[256, 128, 64], K[2]	0.0001	0.021054	0.979452
F[256, 128], K[4]	0.0001	0.017936	0.979452
F[256, 128, 64], K[3]	0.001	0.017345	0.978767
F[256], K[5]	0.0001	0.019624	0.978767
F[256, 128], K[2]	0.0001	0.019808	0.977397
F[32, 64], K[4]	0.0001	0.020592	0.976712
F[32, 64, 128], K[3]	0.0001	0.021897	0.976027
F[32, 64], K[2]	0.0001	0.027842	0.974658
F[256, 128], K[3]	0.0001	0.021586	0.974658
F[256], K[3]	0.0001	0.023886	0.973288
F[32], K[5]	0.0001	0.028562	0.972603
F[256], K[4]	0.0001	0.025086	0.971233
F[256], K[2]	0.0001	0.025884	0.970548
F[32, 64], K[3]	0.0001	0.028491	0.969863
F[32, 64], K[5]	0.0001	0.026751	0.969863
F[32], K[4]	0.0001	0.034371	0.965753
F[32], K[2]	0.0001	0.032284	0.963699
F[32], K[3]	0.0001	0.037147	0.960959

Table C.4: Architectures of single phasing with no Load condition

CNN architectures	**Learning rate**	**MSE**	**Accuracy**
F[256, 128], K[4]	**0.001**	**0.029235397**	**0.96851852**
F[256, 128, 64], K[4]	0.001	0.026971309	0.96851852
F[256, 128, 64], K[5]	0.001	0.031876623	0.96388889
F[256, 128, 64], K[5]	0.0001	0.031008006	0.96388889
F[32, 64, 128], K[3]	0.001	0.030320587	0.96296296

F[32, 64], K[4]	0.001	0.033742246	0.96296296
F[32], K[5]	0.001	0.031903708	0.96296296
F[32], K[4]	0.001	0.031805294	0.96203704
F[32, 64], K[5]	0.001	0.031429155	0.96203704
F[256, 128], K[3]	0.001	0.033781285	0.96111111
F[256], K[5]	0.001	0.031465539	0.96111111
F[32, 64], K[3]	0.001	0.036054997	0.96018519
F[256, 128], K[3]	0.0001	0.034290784	0.96018519
F[32, 64, 128], K[2]	0.001	0.031211007	0.95925926
F[256, 128, 64], K[4]	0.0001	0.033313749	0.95925926
F[256, 128], K[5]	0.0001	0.03378064	0.95925926
F[32], K[3]	0.001	0.036557604	0.95833333
F[32, 64], K[2]	0.001	0.040139392	0.95740741
F[32, 64, 128], K[5]	0.001	0.032432234	0.95740741
F[256, 128], K[5]	0.001	0.034485459	0.95740741
F[256, 128, 64], K[3]	0.0001	0.035660972	0.95740741
F[32, 64, 128], K[4]	0.0001	0.035453136	0.95740741
F[256], K[4]	0.001	0.032996154	0.95648148
F[256], K[3]	0.001	0.03647002	0.95555556
F[32, 64, 128], K[4]	0.001	0.034756311	0.95555556
F[256, 128], K[4]	0.0001	0.034059055	0.95555556
F[32, 64], K[3]	0.0001	0.041210337	0.95462963
F[32, 64, 128], K[3]	0.0001	0.036421099	0.95462963
F[256, 128, 64], K[2]	0.001	0.039933187	0.9537037
F[256, 128, 64], K[2]	0.0001	0.037606554	0.9537037
F[256], K[3]	0.0001	0.037463399	0.95277778
F[32, 64], K[4]	0.0001	0.04089261	0.95277778
F[32, 64, 128], K[5]	0.0001	0.036444928	0.95185185
F[256, 128], K[2]	0.001	0.043195758	0.95092593
F[256, 128], K[2]	0.0001	0.040004473	0.95092593
F[32], K[2]	0.001	0.041427386	0.95
F[32, 64, 128], K[2]	0.0001	0.039385212	0.95
F[256], K[4]	0.0001	0.039501479	0.95
F[256, 128, 64], K[3]	0.001	0.049036758	0.94814815
F[32, 64], K[2]	0.0001	0.04438209	0.94722222
F[32], K[5]	0.0001	0.041391855	0.94722222
F[32, 64], K[5]	0.0001	0.04071304	0.94722222
F[256], K[5]	0.0001	0.039932136	0.94722222
F[256], K[2]	0.0001	0.041269461	0.9462963
F[32], K[3]	0.0001	0.041731378	0.9462963
F[32], K[4]	0.0001	0.042723607	0.94537037
F[256], K[2]	0.001	0.047482327	0.94444444
F[32], K[2]	0.0001	0.044314036	0.94444444