FPGA IMPLEMENTATION OF 1D CONVOLUTIONAL NEURAL NETWORK FOR EARLY DETECTION OF BEARING FAULTS IN INDUCTION MOTORS

BARIŞ DAL

Master's Thesis

Graduate School
Izmir University of Economics
Izmir
2022

FPGA IMPLEMENTATION OF 1D CONVOLUTIONAL NEURAL NETWORK FOR EARLY DETECTION OF BEARING FAULTS IN INDUCTION MOTORS

BARIŞ DAL

A Thesis Submitted to

The Graduate School of Izmir University of Economics

Master of Science Program in Electrical and Electronics Engineering

Izmir

2022

ABSTRACT

FPGA IMPLEMENTATION OF 1D CONVOLUTIONAL NEURAL NETWORK FOR EARLY DETECTION OF BEARING FAULTS IN INDUCTION MOTORS

Dal, Barış

M.Sc. in Electrical and Electronics Engineering

Advisor: Prof. Dr. Murat Aşkar

January, 2022

Induction motors are the core part of various industrial applications because they provide stability, low cost, and easy maintenance. The breakdown of the induction motors may lead to a slow down the production chain resulting in a serious money loss, or it may be harmful to the environment and people's health. Induction motors include roller bearings between rotating parts which reduce the friction and increase the speed and performance and the faults on these bearings are the most confronted motor failure. The early detection of these failures facilitates repairing or replacement of the motor with considerably less amount of money rather than dealing with tremendous problems that may occur later. The literature presents different approaches to detect the bearing faults, but there has not been a detailed work that establishes Field Programmable Gate Array (FPGA) /Application Specific Integrated Circuit (ASIC) design to solve the problem using 1D Convolutional Neural Network (CNN). In this

thesis, FPGA implementation of 1D CNN for early detection of bearing faults in induction motors is introduced. The proposed model employs a benchmark Case Western Reserve University (CWRU) dataset which provides vibration signals of 4 classes (healthy, ball bearing, inner-race, and outer-race). Parameters of the proposed architecture are extracted from the trained model as 32-bit floating-point numbers. Next, the fixed-point (8-bit) representations of the parameters are determined for mapping to FPGA. Then, the mathematical model of each layer in the proposed model is developed utilizing Verilog and implemented on Xilinx-ZYBO Z7-10 FPGA board using Vivado 19.1 software.

Keywords: FPGA, convolutional neural network, bearing fault, 1D CNN, fixed-point, Verilog HDL

ÖZET

ASENKRON MOTORLARDA RULMAN HATALARININ ERKEN TESPİTİ İÇİN 1B-EVRİŞİMSEL SİNİR AĞININ FPGA ÜZERİNDE UYGULAMASI

Dal, Barış

Elektrik ve Elektronik Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Prof. Dr. Murat Aşkar

Ocak, 2022

Asenkron Motorlar, stabilite, düşük maliyet ve kolay bakım sağladıkları için çeşitli endüstriyel uygulamaların temel parçasını oluşturmaktadır. Asenkron motorların arızalanması, üretim zincirinin yavaşlamasına neden olarak ciddi bir para kaybına neden olabilir veya çevreye ve insan sağlığına zararlı olabilir. Asenkron motorlar, dönen parçalar arasında sürtünmeyi azaltıp, hız ve performansı artıran rulmanlar içerir ve bu rulmanlardaki arızalar en çok karşılaşılan motor arızalarıdır. Bu arızaların erken tespiti, sonradan oluşabilecek büyük problemlerle uğraşmak yerine, motorun çok daha ucuza tamir edilmesine veya değiştirilmesine olanak sağlar. Rulman hatalarının tespiti için literatürde farklı yaklaşımlar bulunmaktadır, ancak 1B-Evrişimsel Sinir Ağı (ESA) kullanarak sorunu çözmek için Alanda Programlanabilir Kapı Dizisi

(FPGA)/Uygulamaya Özgül Tümdevre (ASIC) tasarımını kuran detaylı bir çalışma bulunmamaktadır. Bu yüksek lisans tezinde, asenkron motorlarda rulman hatalarının erken teşhisi için 1B-Evrişimsel Sinir Ağının FPGA üzerinde uygulaması sunulmuştur. Önerilen model, Case Western Reserve Üniversitesi (CWRU) tarafından sağlanan 4 farklı sınıfa ait (sağlıklı, bilye hatası, iç bilezik arızası ve dış bilezik arızası) titreşim sinyalleri veri kümesini kullanır. Önerilen mimarinin parametreleri, eğitilmiş modelden 32 bitlik kayan nokta sayıları olarak çıkarılır. Daha sonra, FPGA üzerinde bu sayıların depolanması için sayıların sabit nokta (8 bit) temsilleri belirlenir. Sonraki adımda, önerilen modeldeki her katmanın matematiksel eşleniği Verilog Donanım Tanımlama Dili (HDL) kullanılarak geliştirilmiş ve Vivado 19.1 yazılımı kullanılarak Xilinx-ZYBO Z7-10 FPGA kartında gerçeklenmiştir.

Anahtar Kelimeler: FPGA, evrişimsel sinir ağı, rulman hatası, 1B-ESA, sabit nokta gösterimi, Verilog HDL

To my family …

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor Prof. Murat Aşkar for inspiring me to choose the field of deep learning and FPGA, for giving me the motivation and encouragement, and for guiding me along this journey.

I would also like to thank Prof. Levent Eren and Prof. Cüneyt Güzeliş for their tremendous contributions and valuable suggestions for my thesis.

My sincere thanks to the Scientific and Technological Council of Turkey (TUBITAK) for the award-winning support "2210-A National Scholarship Program for MSc Students".

Finally, I would like to thank my family for their love, support, and patience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Induction Motors are extensively being used in various fields of Industry due to their price, robustness, and easy maintenance. Hence, it is the core part of innumerable critical applications in which they may be exposed to electrical or mechanical failures. Bearing faults are types of mechanical failures which are the most common failure seen on motor faults (Yeh et al., 2008). Even though it is a challenging task to identify bearing faults, if they are identified sufficiently early, it is possible to repair without costing too much money (Eren and Devaney, 2004). Since early detection of bearing faults could save time and severe expenses, it is a crucial task to detect motor faults in the early stages for repairing or reinstating motors.

Many researchers have shown interest in and been working on the early detection of motor faults. Their studies published in the literature could be grouped into three main approaches which are model-based, signal-based, and knowledge-based. In the model-based approach, mathematical model is employed for representing regular operation of the induction motor. Then, the established model is compared with the induction motor to be tested and this comparison produces a non-zero value when there is any fault. One of the drawbacks of this method is that the disturbances and the model uncertainties are not included in this model. Therefore, the disturbances and the model uncertainties should be included for developing a better model (Soualhi, Clerc and Razik, 2013). In the signal-based model, several signal processing techniques which are time-domain analysis, frequency-domain analysis, enhanced frequency domain analysis, and time-frequency analysis are adopted for monitoring the conditions of the induction motor. These methods are applied to the measurements of different types of signals such as vibrations, power, torque, etc. which are susceptible to faults (Filippetti, Bellini and Capolino, 2013). Before applying these techniques, it is essential to consider complexity of the signal processing technique and the required resources. While more complex signal processing techniques may yield effective results, it enhances the computational cost. The aforementioned methods above need a developed model or some patterns, on the other hand, the knowledge-based method requires a large dataset. The knowledge-based methods are consisted of two different

approaches which are qualitative and quantitative methods. Qualitative approaches such as fault tree, expert system, and signed diagraph utilize symbolic intelligence, as quantitative approaches utilize unsupervised learning systems, supervised learning systems, and reinforcement learning (Dai and Gao, 2013). In some complex cases, the hybrid systems may produce better results in the sense of fault detection. In such a case, principal component analysis (PCA), the fast Fourier transform (FFT), or wavelet transform could be used for feature extraction. The drawback of the knowledge-based method is that it massively depends on training data and which features are selected (Ince et al., 2016).

Machine learning (ML) approaches in the literature have drawn serious attention for motor fault diagnosis and detection, however, it requires selection of some features to feed as input to the classifier of the model. Hence, numerous machine learning techniques have been used in the literature for bearing fault diagnosis and detection. Chow et al. (1991) applied an artificial neural network (ANN) to detect stator winding and bearing faults in induction motors. The model has been built as 3-layer with several different hidden nodes and the dataset consists of 35 training and 70 testing data. The obtained results have satisfying accuracy with more than 95%. PCA method has been proposed for effective feature selection for condition monitoring of the motor in (Malhi and Gao, 2004). In their work, they use features both from PCA-based and non-PCA to prove the effectiveness of their work. PCA-based features significantly reduced the percentage of misclassification compared to non-PCA features. Wang et al. (2013) presented a method utilizing Hilbert-Huang transform and support vector machines (SVM) for intelligent fault diagnosis. In their approach, the wavelet packet threshold method is employed for denoising of acquired signal from the sample engine. HHT is performed on a denoised signal to extract features and then, the SVM model is developed. They have obtained more than 90% of accuracy in their work.

Processing on large datasets requires extensive resources and feature extraction methods could be applied to decrease computational cost. As explained above, the ML algorithms need some features for the classifier input and these features could be selected either manually or automatically. The problem is that manual selected features

may perform well in some cases mentioned previously, but the selection of optimal features for a specific dataset is not known yet (Neupane and Seok, 2020). In addition, the feature extraction process usually takes a lot of time and may require advanced signal processing algorithms which may require lots of resources. These reasons make the feature extraction operation is not a good candidate for real-time applications. To eliminate the feature extraction operation, deep learning which is a subset of machine learning has been employed by many researchers for motor fault diagnosis and detection.

Janssens et al. (2016) presented their work to detect bearing faults by using a convolutional neural network (CNN). This was the first attempt that has inspired many researchers in the field to utilize CNN for bearing faults. They used two accelerometers perpendicular to each other to get two different 1D raw data and then merged them into a 2D vector form like an image to feed the CNN model. The model consists of a single convolutional layer with 32 filters and 200 neurons in fully-connected layer. They have managed to achieve 93.61% of accuracy. In Wen et al. (2017), a method called signal-to-image conversion which takes raw data and converts to an image is proposed for preprocessing of the data. LeNet-5 has been chosen for the deep learning model with appropriate padding. The performances are compared with other methods such as Sparse filter, SVM, ANN, etc. and the CNN model has achieved 99.79% of accuracy best among the other methods. Furthermore, Zhang et al. (2018) suggested a deep, fully convolutional neural network (DFCNN). Spectrogram of the vibration signal is fed as the input to the model which has 4 convolutional layers. Linear SVM achieved 91.43% of accuracy while Particle swarm optimization with linear SVM produces 94.28%. On the other hand, DFCNN performs 99.22% of accuracy and surpasses the other methods.

The aforementioned methods achieve high accuracy by using deep CNN architectures usually in 2D. Since 2D CNNs are computationally complex, it depends on dedicated resources for both training and real-time applications where the power, memory, and processing resources may be limited such as in mobile applications. To overcome this situation, Ince et al. (2016) introduced a real-time motor fault detection system that utilizes an adaptive 1D CNN, which is less computationally complex and

faster compared to the deep 2D CNNs. Since 1D CNN is faster and does not require any hand-crafted features makes them a good candidate for real-time applications.

None of the methods discussed above concentrates on implementing models on special hardware such as ASIC/FPGA or microcontroller. There has been some research on FPGA implementation for fault detection in induction motors. Contreras-Hernandez et al. (2019) presented Quaternion Signal Analysis (QSA) for motor fault detection. The accelerometer and current signals are used as quaternions for detecting faults in motors by using the decision tree algorithm. They achieved around 99% of accuracy with just using 200 samples while other algorithms use in the order of thousands. Electrical signature analysis is employed in real-time to identify whether the motor is healthy or not (Karim, Memon and Hussain, 2019). The current signal is fed through the analog to digital converter (ADC) of Spartan 3E FPGA. FFT is applied to the acquired signal in real-time and results send through a USB cable to a laptop to show the spectrum. This work lacks a decision-making algorithm for a stand-alone system. Lizzaraga-Morales et al. (2017) introduced a new method for broken bar detection (BRB) and implemented it on FPGA for real-time applications. Their result shows 99.7% of accuracy for identifying BRB in earl stages.

In the literature, there have not been many works that implement 1D CNN on FPGA. Hence, this thesis is one of the early works that attempt to implement 1D CNN on FPGA for bearing faults detection in induction motors. Initially, vibration data from Case Western Reserve University (CWRU) was employed for the training part carried out on using Python. 1D CNN model has been established using the PyTorch framework on PC. Later, filter coefficients, weights, and biases were extracted from the trained model as 32 bits floating numbers. These numbers were converted into fixed-point representations using MATLAB for efficient mapping to the FPGA. Lastly, the feedforward model was implemented on FPGA with Verilog HDL for real-time detection of faults. The algorithm takes 4.232 μs which makes it a suitable design for real-time applications.

The outline of the thesis is as follows; Chapter 2 gives an overview of motor faults, neural networks, and the FPGA. Chapter 3 explains the design and implementation methodology of the work. It presents CWRU Dataset and is followed by the training phase, number representation, and FPGA implementation, respectively. Chapter 4 discusses the results of the algorithm and chapter 5 gives conclusions and propose future works.

CHAPTER 2: OVERVIEW

2.1. Induction Motor Faults

Induction motors have been taking place in plenty of different applications such as industrial and manufacturing operations for many reasons. Hence, if a fault occurs in the motor, its effects may be severe to the people's health, the environment, and profits. To prevent those severe effects, real-time motor health condition is necessary for diagnosing the faults shown in Figure 1. Motor faults can be divided into two categories such as mechanical and electrical faults. The highest probability of occurrence is bearing faults, while the stator faults are the most common in the electrical faults (Yeh et al., 2008).

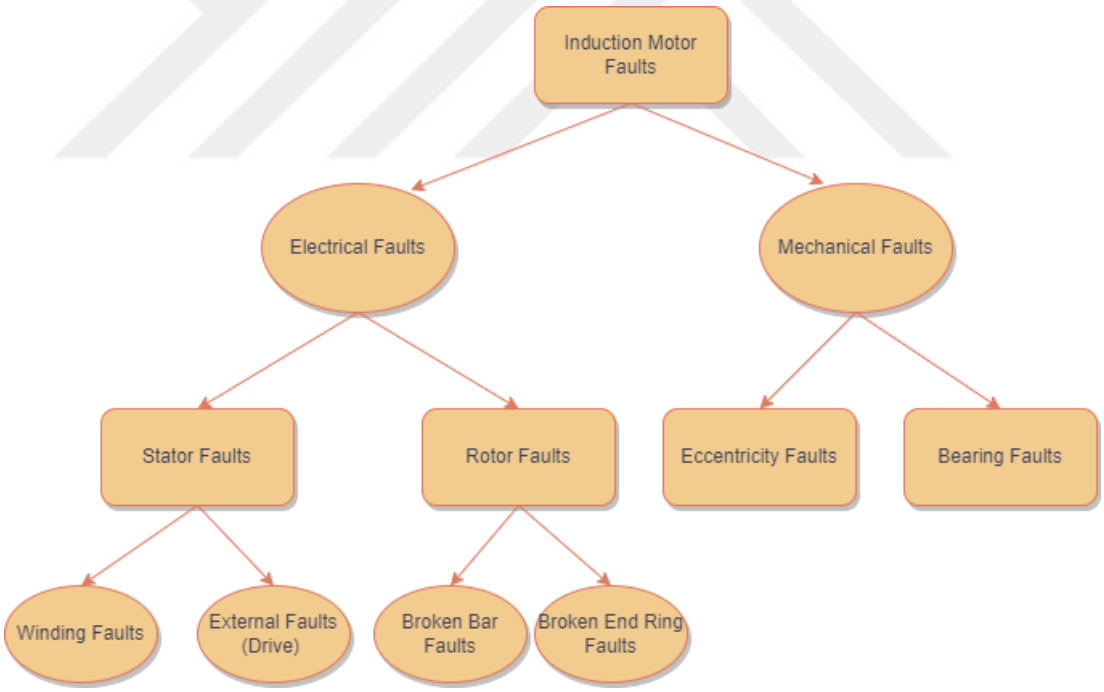


Figure 1. Types of induction motor faults (Yeh et al., 2008)

Figure 2 shows that bearing faults have a 44% of possibility of occurrence in induction motor faults and the other most common failures are stator faults, rotor

faults, and others, respectively. A one-third of the total motor faults have been identified as electrical faults which are stator and rotor-related faults. Electrical faults could be diagnosed by acquiring the necessary signals such as vibration or the current. It has utmost importance to detect electrical faults early because when it occurs, it is not possible to recover it back. More information can be found in the work of Yeh et al. (2008), since the detailed discussion is not in the scope of this thesis.

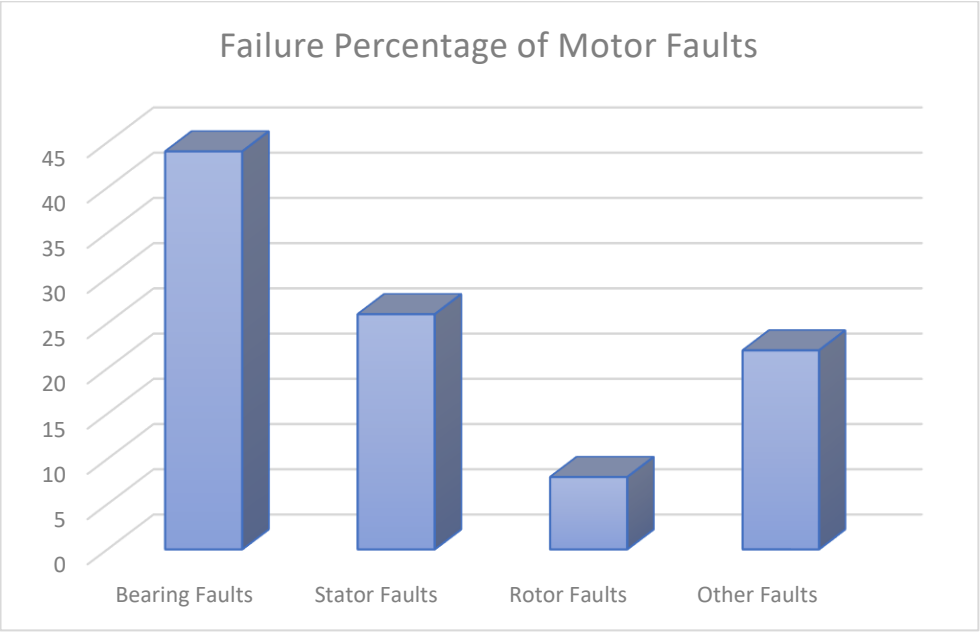


Figure 2. Probability of the most common failures in induction motors

2.1.1. Bearing Faults

Bearing is part of an induction motor that takes place in between the two parts rotating in opposite directions. It reduces the friction between rotating parts that enables higher revolutions with less power consumption. Since it is the most common failure, it should be constantly monitored and analyzed for reducing the probability of occurrence of bearing faults.

Bad environmental conditions, improper usage, and not having regular monitoring/maintenance may cause bearing faults which can be seen on Figure 3. It illustrates outer ring fault, inner ring fault, and ball fault which are the three most common bearing faults identified in the literature (Waziralilah et al., 2018).

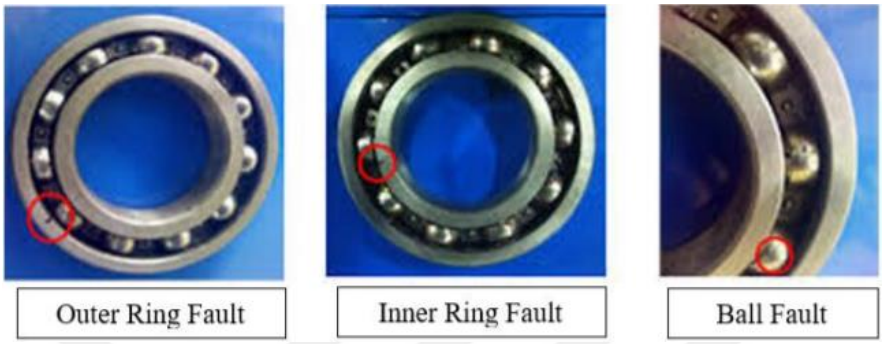


Figure 3. Three different fault types in bearings (Waziralilah et al., 2018)

Bearing faults could be identified through a vibration signal from an accelerometer because they produce vibrations at certain frequencies in the presence of faults. By utilizing bearing geometry illustrated in Figure 4 and the shaft speed of the motor, it is possible to determine frequencies associated with faults (Ince et al., 2016). Wowk et al. (1991) clarified the determination of those frequencies. Instead of determining these fault frequencies, the vibration signal is directly fed it to the 1D-CNN. Hence, it does not require any hand-crafted features which may use lots of resources for implementing a real-time application.

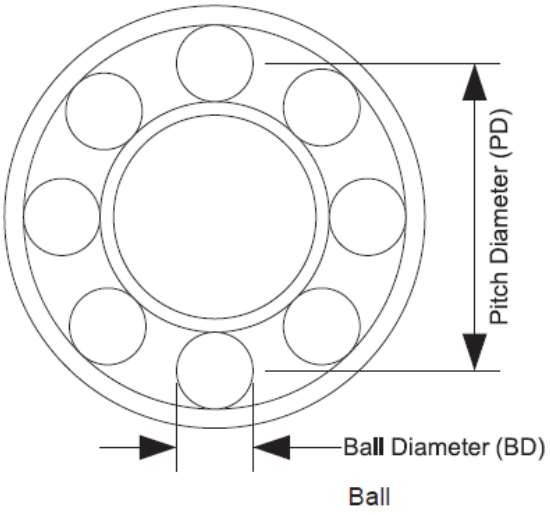
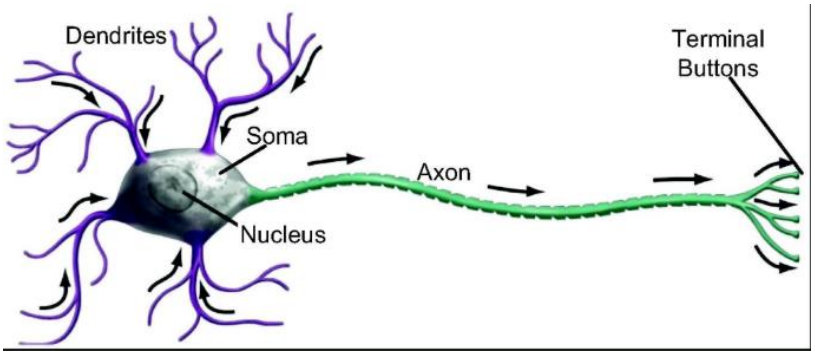


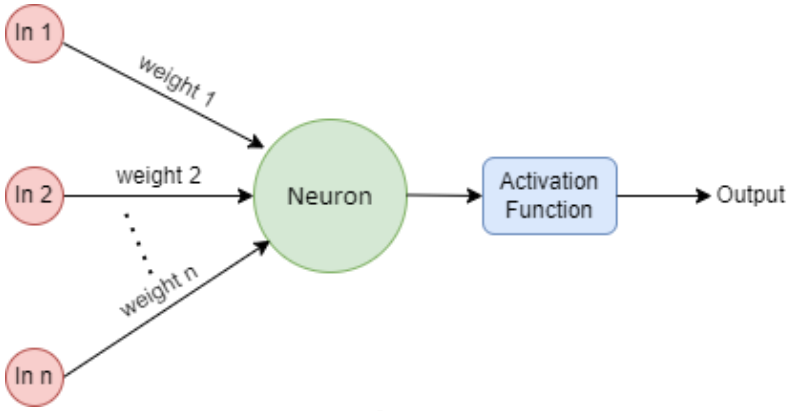
Figure 4. Ball and Pitch diameters of a bearing (Ince et al., 2016)

2.2. Artificial Neural Networks

Artificial Neural Network (ANN) is a mathematical method that learns through mimicking the biological nervous system. The ANN includes artificial neurons that resemble neurons in the human brain as shown in Figure 5a. The biological neurons consist of dendrites, the nucleus, and the axon. Dendrites acquire electrical signals through biological synapses and the signal is processed in the nucleus. If the processed signal is above a threshold, it creates action potentials (Kiranyaz et al., 2021). A similar procedure occurs in artificial neurons illustrated in Figure 5b that is analogous to biological neurons.



(a)



(b)

Figure 5. (a) Signal flow in a biological neuron (Kiranyaz et al., 2021) (b) An artificial neuron

Equation 1 shows the mathematical representation of an artificial neuron in which upcoming input data is multiplied with corresponding weight. All the multiplications in a neuron are summed, and it passes through an activation function.

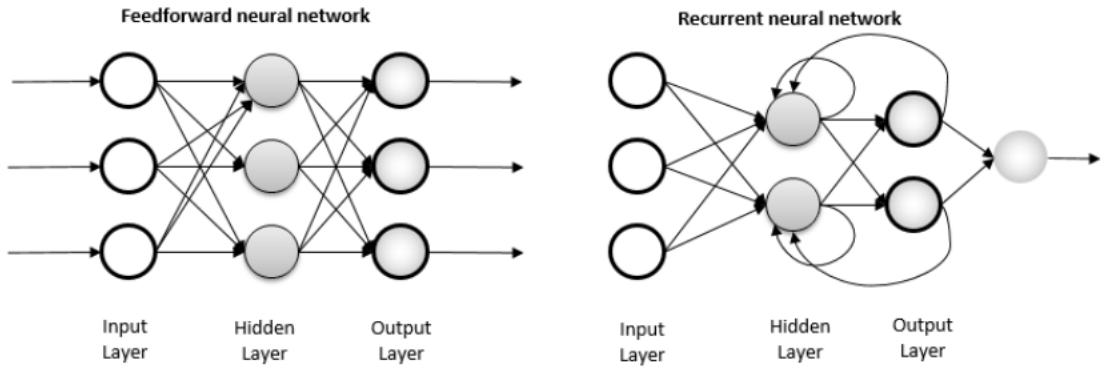
$$y = f(b + \sum_i^N x_i * w_i) \quad (1)$$

where x_i is the input and w_i is the weight, activation function is shown as f, and the b corresponds to bias. In the literature, there has been significant research on employing the ANN in various applications such as medical, financial, advertising, automotive, etc. Not depending on advanced statistical training, accessibility to various training algorithms, and capability of the finding of complicated relations in a network are the reasons why the ANN is found in many published works. On the other hand, it has several drawbacks such as the black-box nature of the model, complex computations, and susceptible to overfitting the model (Tu, 1996).

Feedforward and recurrent (feedback) are the two main concepts of the ANN (Figure 6). In feedforward networks, the information flows in only one direction from input to output. There is no loop that takes the information from the output and brings

it back to the input. On the other hand, recurrent networks contain a loop, so that the data can flow in both directions from input to output (forward) and backward. Therefore, recurrent networks are robust, but they can be a complex architecture due to dynamic change of the state until equilibrium point is reached. By using these two networks, numerous designs that employ feedforward or recurrent are established for different types of applications.

The ANN can learn to do certain tasks, but it requires training which is simply a trial and error. There are several approaches to perform training such as supervised, unsupervised, and reinforcement learnings. In supervised learning, inputs and the corresponding outputs are available in the training dataset. In the training, the input is fed to the network, and it produces an output.



(a) (b)

Figure 6. (a) Feedforward network and (b) Recurrent networks (Pekel and Kara, 2017)

The error between the produced output and the actual output is calculated for updating the weights. This procedure is carried on until the actual output is reached. On the contrary, there is no information about the output in unsupervised learning. Since there is no correct answer for the output, it is convenient for finding a hidden pattern in the network. The last approach is reinforcement learning. Observation is the key element in this type of network. The choice of the model is established through observation of

its surroundings. If a negative observation occurs, the network's weights are altered to generate a new decision for the later observations.

2.3. Convolutional Neural Networks

Inspiration from the visual cortex of animal has contributed to the establishment of Convolutional Neural Networks (CNNs) that is enhanced version of feedforward ANN. Although it has been used for visual tasks such as image and video recognition systems for more than 40 years, its popularity has raised since 2012 with the help of advancement in computing power, availability of large datasets, and boosted algorithms (Rawat and Wang, 2017). The advantage of CNNs is that the feature extraction and classification parts are combined into a single block which takes the raw data without requiring any hand-crafted features as opposed to the ANN. In addition, unlike Multi-layer Perceptron (MLP) network, CNN is capable of processing large datasets with high performance due to the sparse connection of neurons with coefficients. Moreover, CNNs can easily overcome the problems such as noisy inputs and inputs of different sizes.

CNNs usually have several convolutional layers which simply perform a convolution operation on the input. For this reason, CNNs are widely used in image processing applications. Since the images are 2 dimensional, their kernels or weights consist of a 2D vector. The sample CNN model which takes a grayscale image of a size of 24 x 24 is shown in Figure 7. The input image is categorized into two classes by using the CNN model that contains 2 consecutive convolution-pooling layers followed by a fully-connected and output layers. The convolutional layers with (K_x, K_y) as the kernel size, have 4 and 6 neurons, respectively. Since the padding is not employed, the dimension of the feature map is decreased after the convolution operation. Subsampling layers, (S_x, S_y), which further reduce the dimension of the feature map, are chosen as 3 for the first pooling layer and 4 for the next one. After the second pooling layer, the 2D feature map is converted into a scalar which creates the fully-connected layer. Then, it flows into the output layer that provides the category of the input image (Kiranyaz et al., 2021).

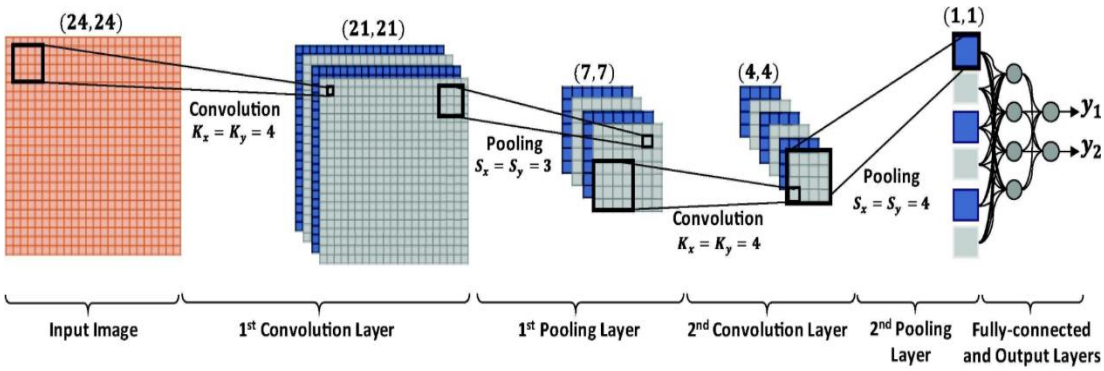


Figure 7. The simple representation of CNN architecture (Kiranyaz et al., 2021)

The architecture explained above is the feedforward operation of the sample CNN model which requires the determination of convolution kernels and coefficients. To obtain them, the CNN model should be trained by using supervised learning such as the backpropagation (BP) algorithm. In each iteration, weights are updated through their gradients of magnitude for each input and target output until the error is in an acceptable range. More information about BP in 2D CNNs could be obtained in Kiranyaz et al. (2016).

2.3.1. 1D Convolutional Neural Networks

The sample CNN model above works with 2-dimensional input e.g., image. Therefore, the model is called 2D CNN. On the other hand, 1D CNN utilizes 1-dimensional kernels, input, and output instead of 2D matrices. Furthermore, 1D CNN has drawn significant attention for 1-dimensional signals, and it outperforms 2D CNN models because of the following reasons (Kiranyaz et al., 2016):

- 1D CNN significantly reduces the computational complexity due to decrease in dimensionality compared to the 2D CNN.

- Usually, compact 1D CNN network consists of 1 or 2 hidden layers requiring less than 10K parameters. On the other hand, deep 2D CNN models require

more than 1M parameters which makes it difficult to train and implement compared to the compact 1D CNN architecture.

- Sophisticated 2D CNN architecture utilizes certain hardware resources for training whereas 1D CNN can be trained on any CPU and it is comparably fast.
- Since 1D CNN is less computationally complex, it is more convenient for a real-time application.

In case of scarce data and the acquisition of highly varied signals such as electrocardiograph (ECG), mechanical structures, or motors, 1D CNN shows better performance. Unlike in the 2D CNN, the filters and features maps are 1D arrays in 1D CNN. A 1D raw input signal is fed to a sample 1D CNN model as sown in Figure 8. Convolution layers and dense layers are introduced in the model. 1-dimensional convolution operations along with activation function and pooling operation appear in CNN layers, and the dense layer is like the MLP. Hyper-parameters consist of the number of hidden layers, kernel size, subsampling factor, and preferred pooling and activation functions. In Figure 8, the input size is 1000 samples, and it is directly fed into the convolution layer without obtaining hand-crafted features whereas the size of the output of the first convolution is 960 samples due to having a filter size of 41 (input size – filter size + 1). Since the subsampling factor is chosen as 4, 960 samples are further reduced to 960/4 = 240. Applying two more consecutive convolution layers with subsampling decreases the size to 10. It passes through two MLP and output layers to produce a class of the input either 1 or 2 in this case.

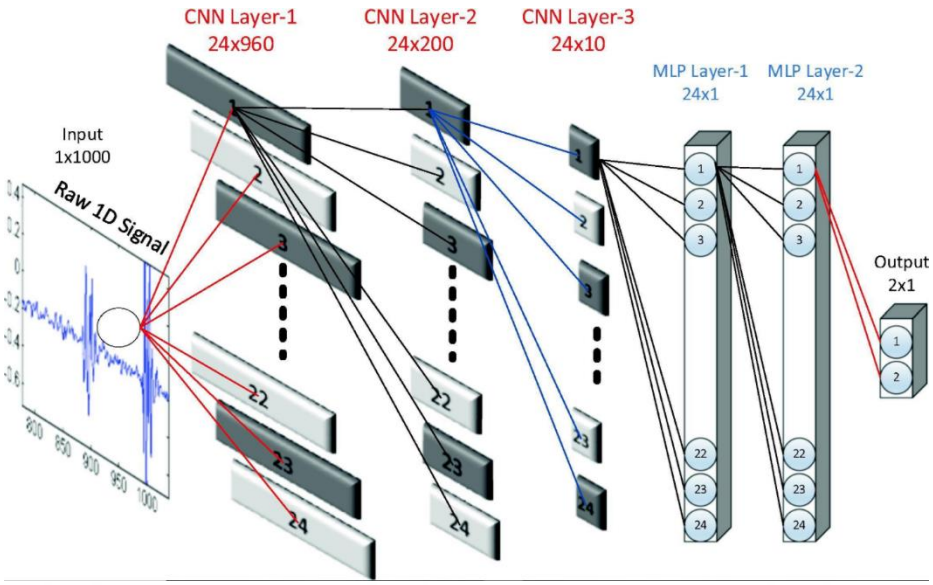


Figure 8. 3 CNN and 2 MLP layers are formed into a small 1D CNN network (Kiranyaz et al., 2021)

2.4. Hardware Implementations of Neural Network Algorithms

Deep Learning (DL) algorithms have become favorable for a variety of different applications such as classification, recognition, analysis, translation, etc. Progress in DL research results in deeper and complex architectures which provide high performance on various tasks, e.g., a deep 2D CNN. While deeper network models with complex transformations are being developed for performance improvement, existing CPU and GPU processors do not produce high performance and energy efficiency for implementing DL algorithms. To overcome this problem, customized hardware structures are developed such as special GPUs, FPGA, and ASIC (Tao et al., 2017).

NVIDIA DGX-1 is one of the most popular specialize GPU that combines software and hardware system to provide high performance with less power consumption for deep learning applications. It accelerates the training by 96 times compared to the dual Xeon Platinum 8180 CPU (Nvidia, 2022). Farabet et al. (2009) presented an FPGA-based processor for convolutional networks. It only employs

FPGA alongside an external memory module without using any other components. A face detection algorithm was designed and tested for 10 frames per second. The proposed design is effective for small robotic applications. Zhang et al. (2015) also proposed an FPGA based accelerator for neural networks. They implemented their design on VC707 FPGA board and achieved 61.62 GFLOPS with 100 MHz frequency. There have been also ASIC implementations of neural networks in the literature such as the DianNao (Chen et al., 2014), and EIE (Han et al., 2016). They both achieved higher performance, less power consumption, and smaller area compared to the equivalent models on the CPU and GPU counterparts.

2.5. Field Programmable Gate Array (FPGA)

Realization of digital circuits can be accomplished through an Application Specific Integrated Circuit (ASIC) or a Field Programmable Gate Array (FPGA) design. ASIC designs have several superiorities over an FPGA such as circuit area, speed, and power consumption, but it requires a tremendous amount of time and money to fabricate (Kiranyaz, Ince and Gabbouj, 2016). In addition, the number of transistors on a chip is multiplied each year as Moore's law states. That would cause more advanced technology requiring further investment to fabricate a chip. On the contrary, FPGA provides a faster and cheaper solution for implementing a digital circuit due to its nature of re-programmability.

FPGAs shown in Figure 9 are devices made of silicon which adopted for building customized digital circuits. It involves customizable logic blocks (CLBs) such as look-up tables (LUTs), flip-flops, and multiplexers with interconnections that enable the programming of these blocks. In addition, FPGA is connected to the outer world through input/output (I/O) blocks shown in Figure 9 which encloses CLBs' arrays. Furthermore, complex operations can be implemented on FPGA through programming languages such as Verilog or VHDL. Despite the similarity between FPGAs and microprocessors in the sense of programmability, FPGA is much faster, more reliable, and consumes less power compared to microprocessors.

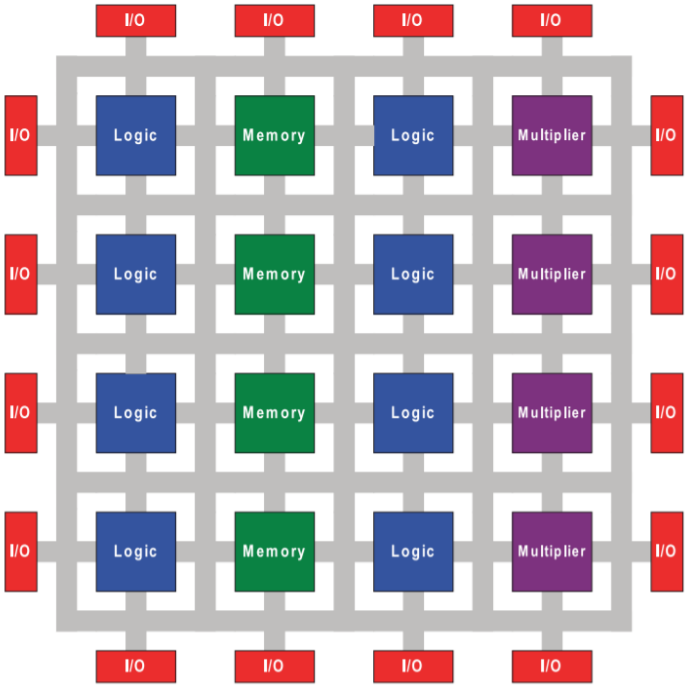


Figure 9. The simple architecture of FPGA (Kuon et al., 2008)

2.5.1. FPGA Design Flow

The development of a model on the FPGA requires specific procedures to be followed as illustrated in Figure 10. It consists of two parts namely design flow and simulation flow. Behavioral, functional, and timing blocks are formed as the simulation flow while the rest of the blocks in Figure 10 constitutes the design flow.

The design starts with the design entry in which the system requirements are translated to build necessary functions. There are several options available that can be employed for the description of the functions such as high-level synthesis compiler or hardware description languages (Verilog or VHDL). When the design entry is completed, Register Transfer Level (RTL) or behavioral simulation is applied to test the Verilog code. The second phase is the synthesis part where the circuit elements (flip-flops, mux, etc.) are constructed by using the Verilog code. The synthesis must also be simulated for verifying the performance of the model. The layout of the design

is constructed in the implementation part which is the third step in the design flow. Simply the design is mapped to the FPGA with certain constraints (pin locations) and timing requirements such as delays, and the clock period. To verify the implementation, a timing analysis is performed through a simulation. It performs comprehensive simulation by considering all the aspects in the design such as the actual FPGA board, delays in logic blocks, wirings, and the clock frequency. Since it is longer than the other simulations, it produces a detailed description of the whole design. Then, the bit-stream file is generated and deployed into the FPGA for programming. Finally, the deployed design on FPGA is tested for the problem requirements.

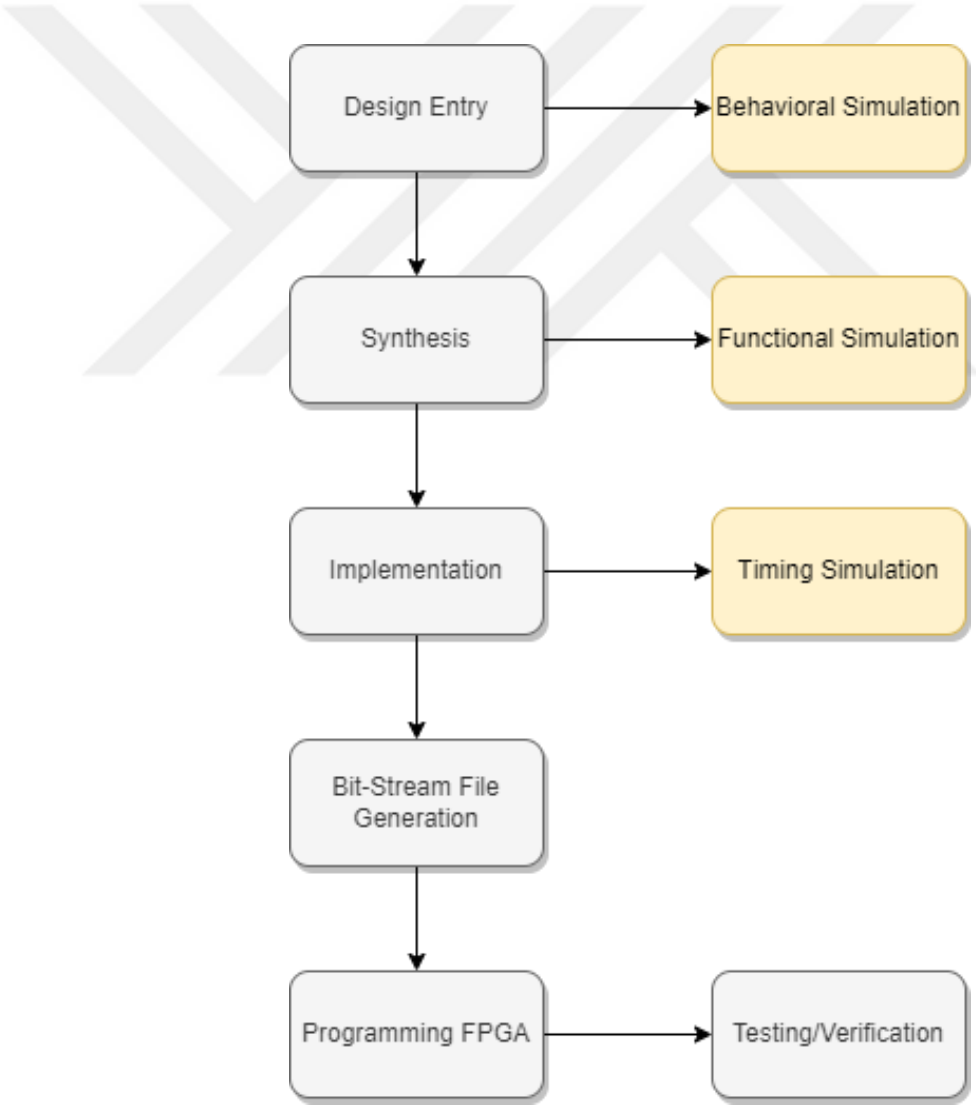


Figure 10. Simplified design and simulation flow on FPGA

2.5.2. Xilinx ZYBO FPGA

Xilinx Zybo Z7-10 ARM&FPGA System on Chip (SoC) board, as illustrated in Figure 11, is chosen for the development and implementation of the thesis. It includes the Xilinx Zynq-7000 family which combines a dual-core ARM Cortex-A9 processor and Xilinx 7-series FPGA logic. Integration of ARM and FPGA makes this board a preferable choice for developing a complex system, however, system design on SoC would not be a generic design that applies to other FPGAs. Hence, only the FPGA (programmable logic) part of the SoC is used for a generic design. Verilog Hardware Description Language (HDL) is adopted for the modeling of the system and Vivado Design Suite Environment is chosen for the design, synthesis, simulation, and programming of the FPGA. It has the following important features:

- Clock speed of 125 MHz
- LUTs (Look up tables) → 17600
- Flip-flops → 35200
- Block RAM → 240 KB
- 80 DSP (Digital Signal Processing) slices
- 6 buttons, 4 switches, and 5 LEDs
- 6 Pmod (peripheral module interface) Ports includes 40 I/O and 4 analog inputs in XADC (12-bits, 1 MSPS)

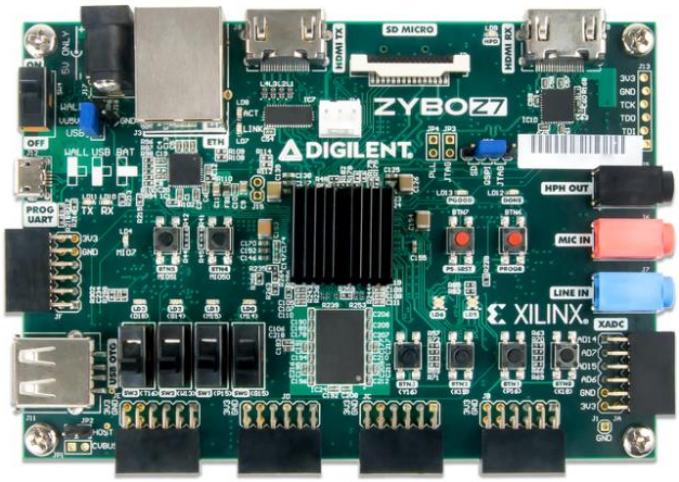


Figure 11. Overview of the Xilinx Zybo Z7 SoC (Source: Digilent, 2021)

CHAPTER 3: DESIGN AND IMPLEMENTATION METHODOLOGY

This chapter presents the methodology followed in the thesis. Initially, a detailed explanation of the Case Western Reserve University (CWRU) vibration dataset is given. The dataset is used for training the 1D CNN model for the detection of bearing faults. Secondly, the proposed model construction in the PyTorch framework using Python is offered. In the next part, parameter representation on the FPGA and the fixed-point conversion using MATLAB are discussed. Lastly, the implementation of the feedforward 1D CNN model on FPGA is presented.

3.1. Dataset

Data is the crucial part of developing a satisfactory architecture such as ML (machine learning) and DL (deep learning). If those methods are trained using an immense volume of data, a better result would be obtained. Especially in DL algorithms, the output of the model could be more accurate with the availability of more data. Generally, a dataset is divided into two classes based on how complex the dataset is: simple and complex datasets (Neupane and Seok, 2020). In general, a simple dataset, also known as a good dataset, can be easily utilized, and modified for useful statistical analysis. Moreover, a good dataset has a solid configuration which means having well-labeled, balanced, and none of corrupted data. A complex dataset, on the other hand, can be described as a huge dataset with a considerable number of records and extensive diversity. It is a tough task to classify real-world data due to bias in the dataset. Standard algorithms may not be sufficient for the classification of such datasets. In addition, the complexity of a dataset raises when the presence of a collection of data from several sources is due to disorganization and discrepancy between data. Moreover, the size of the dataset is related to the complexity. Since standard methods presume balanced data cases, they cannot accurately classify all the aspects of imbalanced data. Therefore, they are inclined to predict the class in which the most data is present in the dataset.

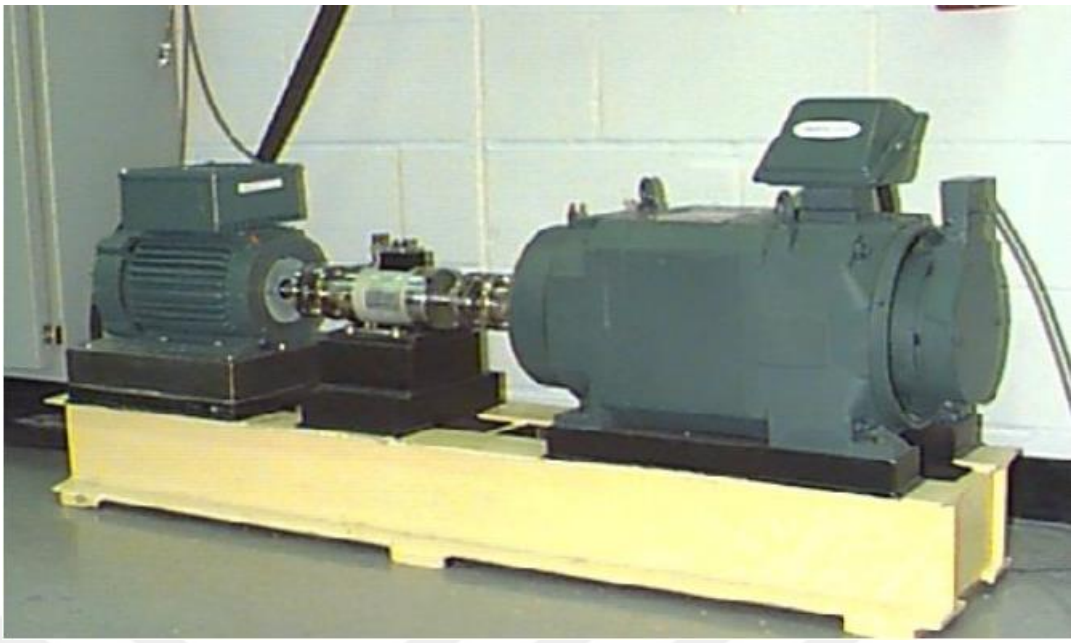
The bearing fault detection in induction machines is one of the essential problems dealt with by large number of researchers in the field. One of the reasons is probably numerous bearing datasets available in public. Some of the datasets used in literature as follows:

- CWRU Dataset (CWRU, 2004)
- FEMTO (Nectoux et al., 2012)
- XJTU-SY (Wang et al., 2020)
- Paderborn University Dataset (Lessmeier et al., 2016)
- IMS Dataset (Lee et al., 2007)
- MFPT (MPFT, 2013)

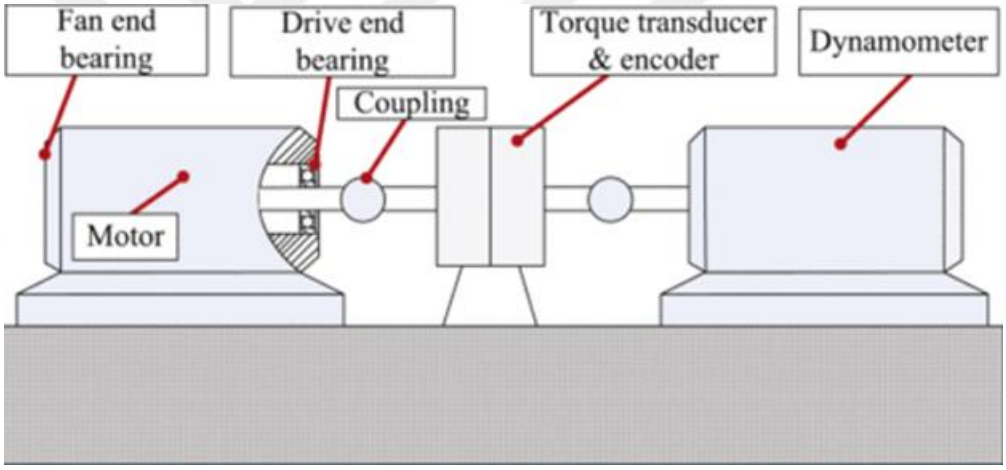
3.1.1. CWRU Vibration Dataset

CWRU has made accessible one of the most popular vibration datasets presenting healthy and faulty bearings test data adopted in the literature for evaluation and validation of numerous ML and DL methods. It contains three different classes which are healthy bearings, drive-end, and fan-end faults. Hence, this thesis utilizes the CWRU dataset for generating a 1D CNN model for the early detection of bearing faults.

The vibration dataset is established through the experimental setup shown in Figure 12(a). It is constructed using a 2 hp electric motor, a torque transducer, and a dynamometer as illustrated in Figure 12(b). A dynamometer and an electronic system for controlling are utilized for applying torque to the shaft. The faults are artificially created on the test bearings through electro-discharge machining. Each test bearing has only single fault with one of the following diameters: 7 mils, 14 mils, 21 mils, 28 mils, and 40 mils (1 mil is equivalent to 0.001 inches). Bearings from two different manufacturers, namely SKF and NTN, are used. Faults of 7, 14, and 21 mils are created on SKF while the faults of 28 and 40 mils are on NTN. In addition, the fault depth is 11 mils in SKF bearings whereas inner and outer raceways have 50 mils of fault depth and ball bearing has 150 mill of fault depth on the drive end of NTN bearings (CWRU, 2021).



(a)



(b)

Figure 12. (a) Experimental setup of the data collection in CWRU (b) block diagram of the setup (Li et al., 2020)

Vibration data is collected from three different accelerometers placed on the drive-end bearing, fan-end bearing, and supporting base plate. 16 channel DAT recorder was employed for data collection which is processed on MATLAB, and .mat file was created for storing the records. One or more accelerations are combined in each file with sampling frequency of either 12 kHz or 48 kHz. The drive-end vibration

data is gathered at 12k and 48k samples per second while 12k samples per second is collected in the fan-end vibration data. On the other hand, 48k samples per second is used for collecting vibration on base plate (Li et al., 2020). The acceleration data for loads of 0 to 3 horsepower is collected. In the meantime, speed was measured with an encoder and filed by hand between 1720 to 1797 revolutions per minute. Moreover, four categories: 48k normal-baseline, 48k drive-end fault, 12k drive-end fault, and 12k fan-end fault are established through 161 records. Ball bearing, inner-race, and outer-race faults are found in each category. Moreover, the position of the outer-race faults with respect to the load region influences vibration response due to outer-race faults being stationary faults. Hence, outer-race faults are placed in three different locations at 3 o'clock, 6 o'clock, and 12 o'clock for the assessment of the issue.

Data files have certain naming conventions. It starts with the fault type corresponding to the initial letter(s) of the name and is followed by the fault diameter corresponding to the following three numbers. Finally, the last number corresponds to load in horsepower. For example, IR014_1 represents an inner-race fault with 14 mils of diameter and 1 horsepower of load (Li et al., 2020). Furthermore, Table 1 gives valuable information about the physical dimensions (all in inches) of the bearings in the CWRU dataset while Table 2 introduces the frequencies (all in Hz.) associated with the bearings. Information presented in these tables may be useful for the feature extraction part, but it will not be used in this thesis. Instead, raw data will be directly fed into the model for diagnosis purposes.

Table 1. Physical dimensions (inches) of the bearings (CWRU, 2021)

Bearing Faults	Inside Diam.	Outside Diam.	Thickness	Ball Diam.	Pitch Diam.
6205-2RS JEM SKF, Drive-end	0.9843	2.0472	0.5906	0.3126	1.537
6203-2RS JEM SKF, Fan-end	0.6693	1.5748	0.4724	0.2656	1.122

Table 2. Bearing faults associated with the frequency (in Hz) (CWRU, 2021)

Bearing Faults	Inner Ring	Outer Ring	Cage Train	Rolling Element
6205-2RS JEM SKF, Drive-end	5.4152	3.5848	0.3982	4.7135
6203-2RS JEM SKF, Fan-end	4.9469	3.0530	0.3817	3.9874

Table 3. CWRU Vibration Dataset length information (Neupane and Seok, 2020)

Dataset	Maximum	Minimum	Standard Deviation	Mean
12k Drive-end	IR007_3: 122917	B028_0: 120801	442.59	121895.96
48k Drive-end	IR021_2: 491446	IR014_0: 63788	131280	411861.71
12k Fan-end	B014_2: 122269	OR021@3_1: 120617	354.77	121306.51
Normal baseline	Normal_3: 485643	Normal_0: 243938	120468	424,636.75

The length information of the dataset which verifies that the CWRU dataset is complex, deep, and diverse is given in Table 3. The dataset is divided into samples containing 500 data in each as the input for the neural network model. 1D CNN model is constructed in python using the CWRU dataset. It classifies the input data as healthy, ball bearing fault, inner-raceway fault, and outer-raceway fault as illustrated in Figure 13.

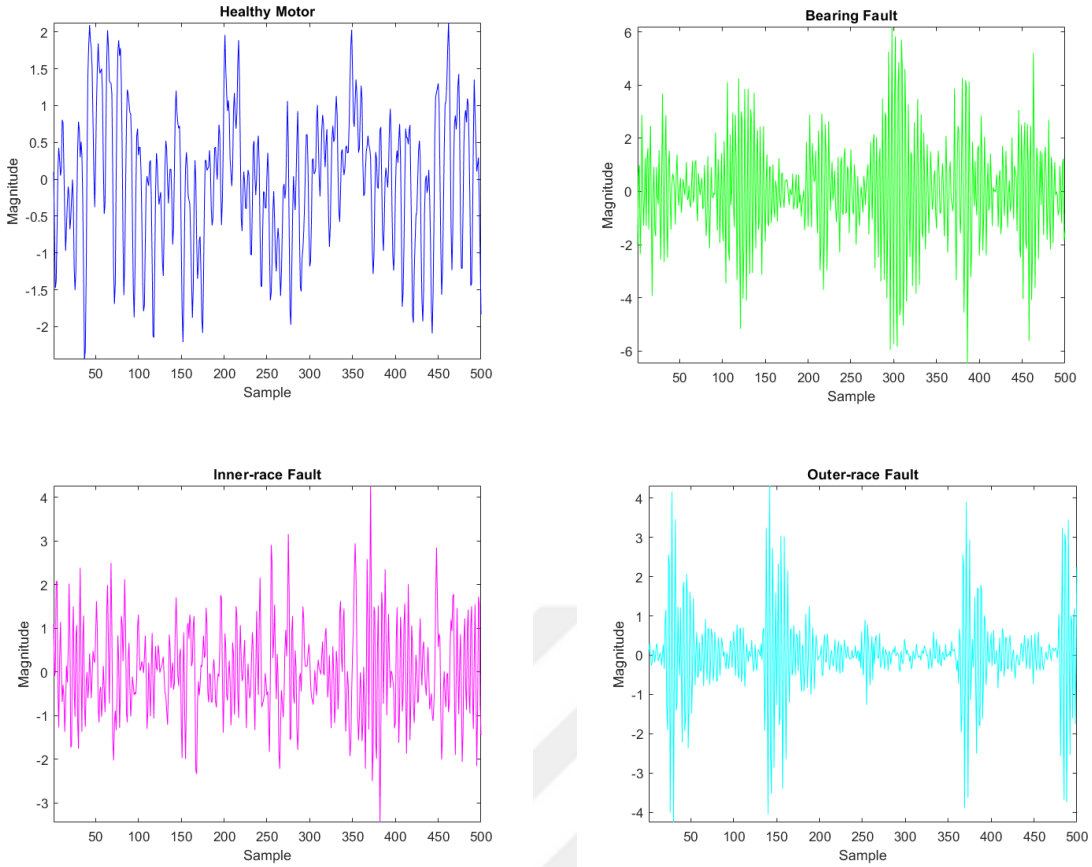


Figure 13. Sample vibration signals for each class from CWRU dataset

3.2. 1D CNN Implementation Using Python

1D CNN classifier architecture, as illustrated in Figure 14, is proposed for the early detection of bearing faults. In this model, raw vibration data from CWRU is directly employed as the input, so no pre-processing algorithms are applied. Therefore, it reduces the complexity of the design for a real-time implementation where the source is scarce. The total length of the dataset is around 18k, and the size of each data is selected as 500 samples (17987, 500). Then, the dataset is divided into train and validation sets which are 80% and 20%, respectively. While dividing the dataset, shuffle is applied to the dataset to reduce the bias and overfitting. Later, the 1D CNN model is formed using the PyTorch framework in Jupyter Notebook, which is an interactive, web-based platform.

To facilitate the FPGA implementation, certain constraints are considered such as:

- The number of convolution filters are set to 8 and 4, respectively.
- No padding and bias are used in the convolution layers.
- ReLu is chosen as the activation function to utilize less logic resources of the FPGA.

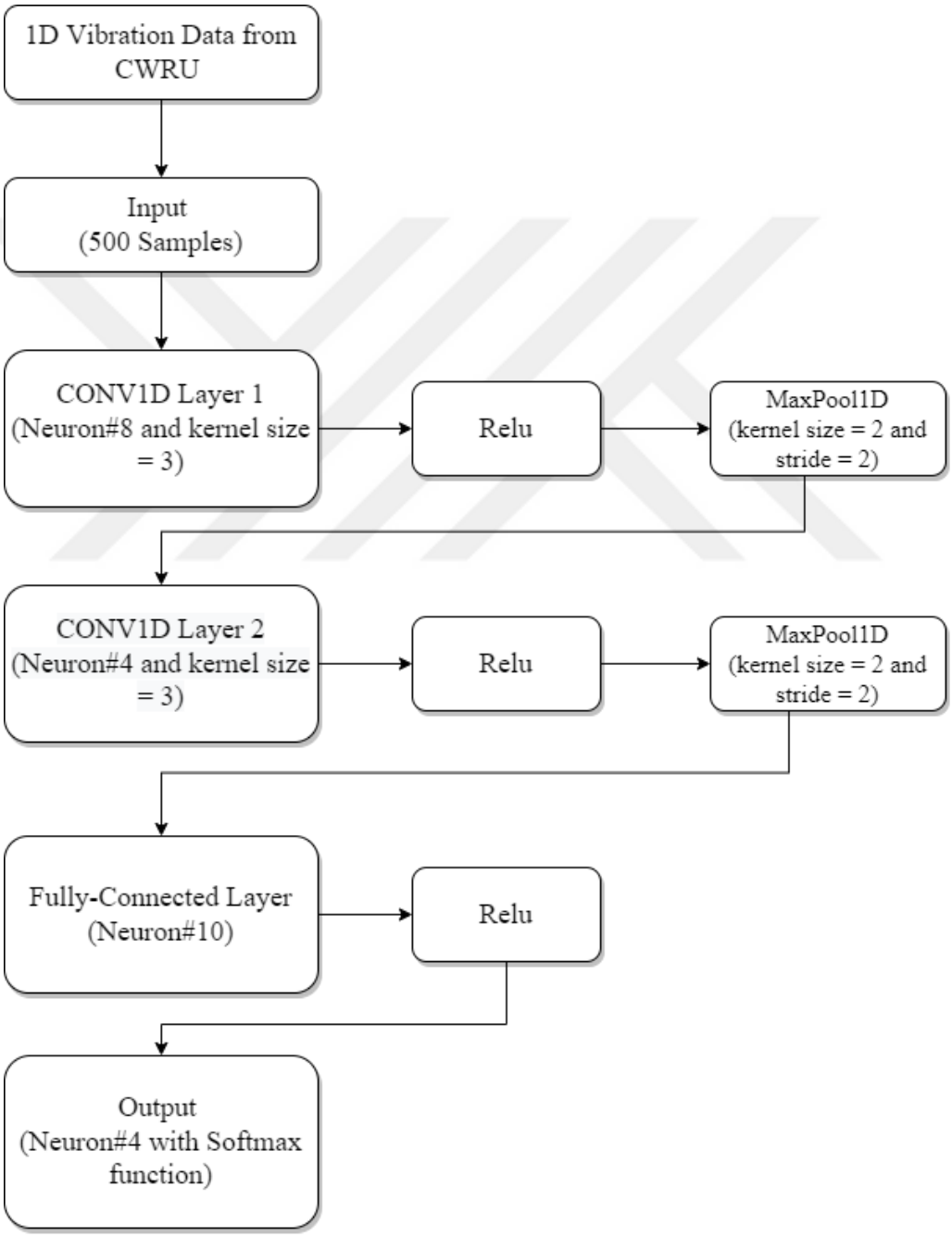


Figure 14. Overview of proposed 1D CNN model for bearing fault detection

The model is established with 2 CNN layers, a fully-connected layer, and the output layer. Each CNN layer involves a rectified linear unit (ReLu) as the activation function and a max-pooling layer. It produces the feature maps that automatically determine the best features. Later, these feature maps are fed into the fully connected layer for the classification part and the output layer provides the class of the input which is one of the following faults: healthy, ball bearing, inner-raceway, or outer-raceway faults.

The model is constructed such that it is not complex for FPGA implementation. Hence, the first convolution layer has only 8 neurons with 3 kernels and the second convolution layer has 4 neurons with 3 kernels without the bias and padding. The fully-connected layer is composed of 10 neurons, and it is followed by a ReLu activation function. Finally, the output layer consists of 4 neurons with the Softmax activation function to classify the input.

Each operation except the activation function reduces the dimension of the input due to not padding with zero. In addition, convolution operations do not include a bias, unlike the fully-connected layer. The Padding and the bias may slightly enhance the model accuracy, but they are not included in the model due to reducing the complexity of the design in FPGA. The size of the output after each convolution can be calculated using equation 2. After the first convolution operation output becomes (8, 498) with no padding, and applying a max-pooling operation (2, 2) reduces the size by half, so it turns into (8, 249).

$$Os = Is + 2 * pd - k + 1 \quad (2)$$

where the Os is the output size, Is is the input size, pd corresponds to padding and k means the kernel size.

The second convolution operation (4, 3) decreases the size to (4, 247) and the following max-pooling layer further reduces the size to (4, 123). When the second convolution layer is completed, the output (4, 123) is converted (4*123) into a single block of (1, 492). This is one of the bottlenecks for deploying the model to the FPGA

due to the connection of each node to earlier and the next node. Hence, it requires immense numbers of weights that need to be stored in the FPGA to calculate each node. Subsequently, the fully-connected (dense) layer yields 10 outputs as the inputs for the output layer. The Softmax activation function is employed after the calculation of the output neurons, the output layer makes the prediction of probability for each class and the neuron having the highest probability among them represents the class of the input corresponding healthy, bearing ball, inner-race, and outer-race.

The training is performed after the model is established to configure the model parameters such as filter coefficients, weights, and biases. Adam optimizer and cross entropy loss methods are employed with learning rate of 0.001, weight decay as e^{-5}, and 40 epochs. It consists of two parts: forward propagation and backpropagation. In forward propagation, the input passes through each layer in the forward direction. Since at the beginning certain parameters are not known, they must be randomly initialized to perform the mathematical operation in each layer. Finally, they produce output. Equation 3 shows the forward propagation of 1D convolution.

$$x(B_j, k) = b_k + \sum_{i=0}^{C-1} w_{k,i} * y_{B_j,i} \quad (3)$$

where x is the output, b_k is the bias, and * is the cross-correlation agent between weight ($w_{k,i}$) and the input ($y_{B_j,i}$). Moreover, B_j is the batch size automatically determined by PyTorch and C stands for the number of channels.

When the forward propagation is completed, the produced output is compared to the expected output by calculating the mean squared error and the formula is given in equation 4.

$$e = \frac{1}{N} \sum_{i=1}^{N} (x_i - y_i)^2 \quad (4)$$

where e is the mean squared error, N is the number of predictions, x_i is the expected network output, and y_i is the predicted network output (Figure 15).

In the backpropagation algorithm, weights and deltas are iteratively updated after each forward propagation. This is done through the computation of the derivative of the mean squared error. This optimization procedure is controlled by the gradient descent method to complete the learning (training) operation. When the training phase is accomplished, parameters are extracted using the 'model.fc1.weight' function for each layer as a 32-bit floating number. In that function, 'model' is the name of the constructed design whereas the 'fc1' is the specific name of the layer whose coefficients will be acquired.

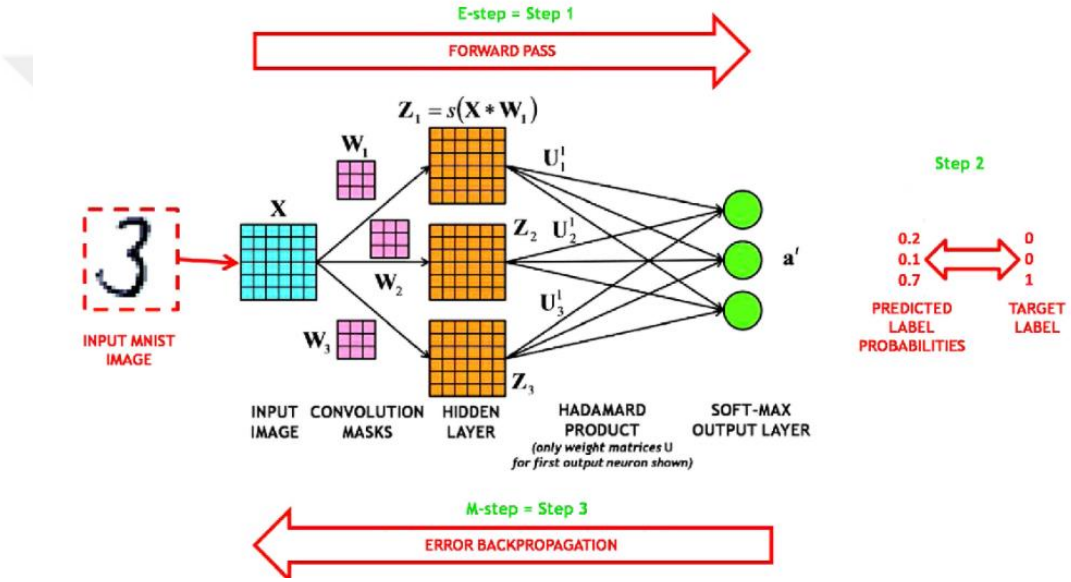


Figure 15. Forward and backpropagation in CNN (Ulloa, 2020)

3.2.1. 1D Convolution

1D convolution is a part of the hidden CNN layer that simply performs 1-dimensional convolution to the input signal, as illustrated in Figure 16, to obtain optimal features. Although it is called the convolution, it is basically a cross-correlation operation because convolution requires the second signal to be flipped by 180^o which does not happen in the so-called convolution operation. The sample input size of 5 convolve with the kernel having a size of 3 is shown in Figure 16. In the

convolution, coefficients in kernels are multiplied with the corresponding input value and summation gives the corresponding output. To calculate the next convolution, the kernel is shifted by one unit in this example due to stride being 1.

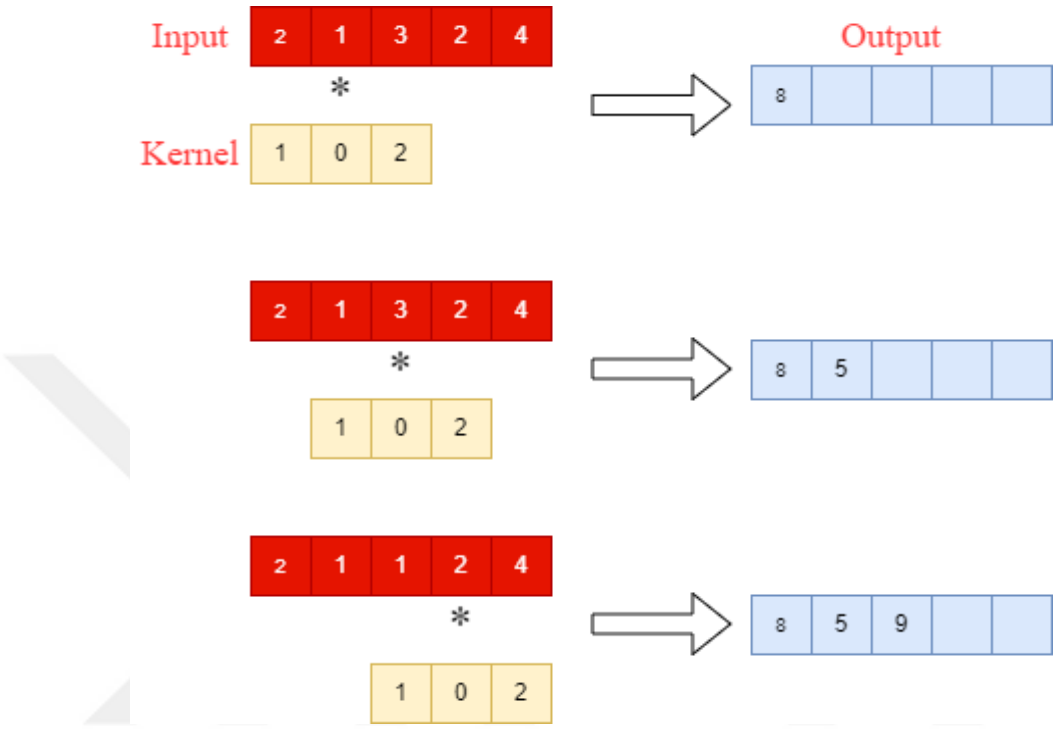


Figure 16. Example of 1D Convolution operation with stride of 1

3.2.2. Activation Functions

The activation function is a controller that activates or deactivates its input to obtain a convenient output. It is an effective operation for a complex network to achieve a better result by applying a nonlinear transformation. In addition, they scale the output to specific ranges and decline the computation cost of the neural network. 1D CNN model mentioned above employs ReLu as the activation function for both convolution layers and the fully-connected layer. If the input is negative, ReLu returns 0, and if the input is positive, ReLu passes the input data to the next layer. Simple formulation is given in equation 5 and the graphical representation is shown in Figure 17.

$$y = f(x) = \max(x, 0) \tag{5}$$

where the x is the input and y, or f(x) is the output of the ReLu. It is certainly straightforward to implement and compute the ReLu function, but if the network contains many negative values, then it may be questionable to use it.

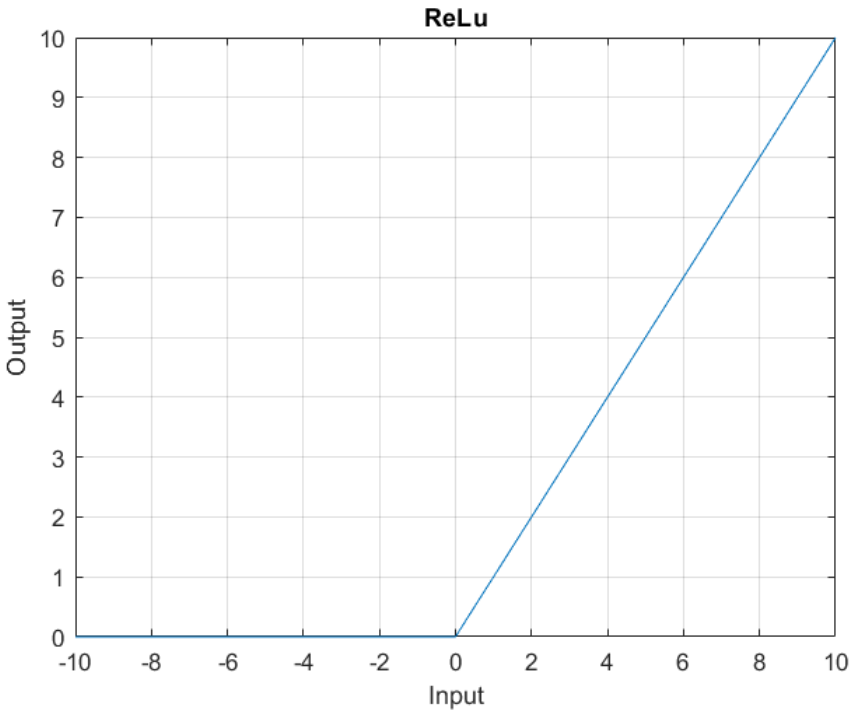


Figure 17. The plot of the Rectified Linear Unit (ReLu)

In the output layer, instead of ReLu, the Softmax activation function is used. Since it computes the probability distribution, it differs from the other activation functions. When it is applied to the neurons in the output layer, there will be a probability for each class and the summation of them yields to 1. Equation 6 gives the formulation of the probability distribution.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{6}$$

where z is the input vector. Each element in the input vector is normalized through the division of the exponential of the input element by the summation of the exponentials of all the elements in the input vector. Hence, the summation of the output vector is assured to be 1. Moreover, the element having the maximum magnitude in the vector will have the maximum probability. For example, if the input array of 2.4, 3.1, 1.7, and 0.6 is fed to the Softmax function, the output will have the following probabilities 0.272, 0.5478, 0.1351, 0.0449, respectively. Since the maximum number in the array is 3.1, it has the maximum probability at the output, and the summation of all the probabilities produces the value of 1 as expected.

3.2.3. Max-pooling Layer

The max-pooling is employed for down-sampling. It decreases the size of the given input vector. It simply takes the maximum of the given set. Since it reduces the size, it further lowers the computation cost and declines the possibility of overfitting. Figure 18 shows the example max-pooling operation with the same configuration (kernel size = 2, stride = 2) used in the 1D CNN model explained above.

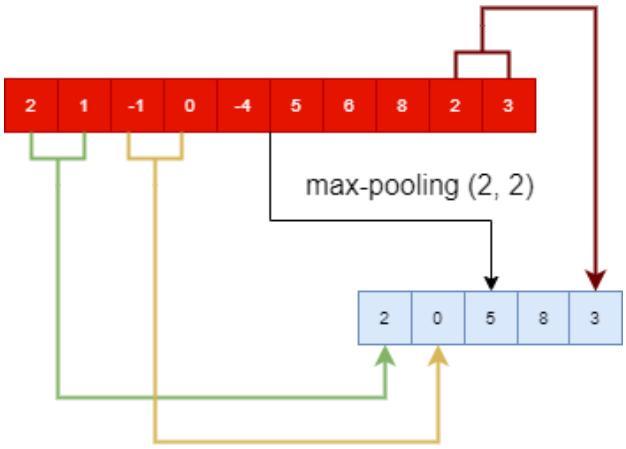


Figure 18. Example demonstration of max-pooling with both kernel size and stride 2

3.2.4. Fully-connected Layer

In the fully-connected layer, each neuron is connected to all previous, and the next neuron as shown in Figure 19. It is used for the classification part.

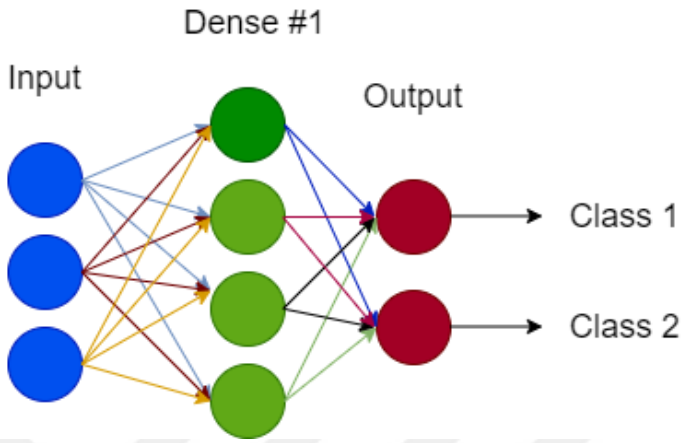


Figure 19. A fully-connected network as an example

The single neuron in the fully-connected layer is called a perceptron. Inputs to perceptron are multiplied with the corresponding weight values, and all the multiplication is summed, and then the bias is added. The ReLu activation function is employed again in this layer. Finally, it passes through the ReLu function which produces the input for the next layer. The mathematical model of a single perceptron is given in equation 7.

$$y = f(b + \sum_{k=1}^{N} x_i * w_i) \quad (7)$$

where x_i is the input, w_i is the corresponding weight, b is the bias, and the f is the activation function.

3.3. Number Representations

In this thesis, the aim is to deploy the 1D CNN classifier architecture to the FPGA. Since the training phase is done, the model can be mapped to the FPGA through

the mathematical representation of each layer and using the weights and biases from the trained model. The convolution and the dense layers can be modeled by just using multiply and add operations while it may be a challenging task to implement activation functions. Hence, the ReLu activation function is chosen in each layer except for the output layer that employs the Softmax function. The implementation of each layer in the FPGA using Verilog will be explained in the next part.

The mathematical modeling is not sufficient for the FPGA implementation because it also requires certain weights and biases to be stored in the memory of the FPGA. After the 1D CNN model is established, filter coefficients, weights, and biases are extracted from the model as a 32-bit floating-point number. These numbers should be represented in the FPGA in binary form. Binary numbers can have a limited sequence of 1's and 0's, so mathematical operations in binary numbers are prone to overflow and underflow. Since the resources in the FPGA are limited, the number of bits required for parameters of the trained model should be carefully decided. There are two options available for the number representation in the FPGA: the first one is the floating-point representation and the second one is fixed-point representation.

3.3.1. Floating-point Representation

FPGAs or any other chips have certain limitations about the memory, so the number representation is a crucial subject that should be meticulously investigated. Unlike fixed-point representation, a floating-point number can achieve high accuracy with the limited number of bits. It can represent very large and very low numbers using scientific notation ($\pm M \times B^E$). For instance, the scientific notation of 10.32 is 1032 x 10^{-2}, where 1032 corresponds to the mantissa (M), 10 is the base (B), and -2 is the exponent (E).

The floating-point number has some specifications called IEEE 754 standards which is the most accepted format presented in 1985. In this standard format, the numbers are represented either in single-precision (32 bits) or double precision (64 bits). In both cases, the most significant bit corresponds to the sign of the number, and

0 is for the positive number whereas 1 is for the negative number. In single precision, 8 bits are employed as exponent parts, and the last 23 bits are used as mantissa, as shown in Figure 20. On the other hand, 11 bits are used as the exponent, and 52 bits are used as mantissa in a double-precision number.

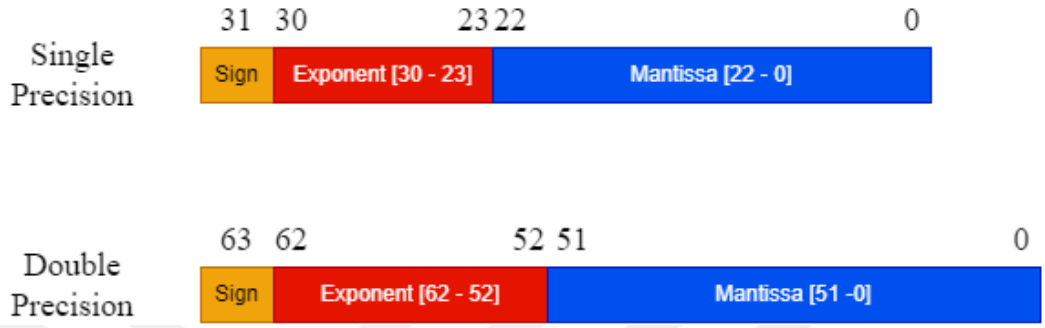


Figure 20. Bit representations of both single and double precision number

There are also some special configurations for numbers such as:

- Zero: all the bits are zero and the sign bit maybe 0 or 1.
- $\pm\infty$: all exponent bits are 1 and all mantissa bits are 0.
- NaN (not a number): all exponent bits are 1 and mantissa is non-zero.

The floating-point number can be calculated by the formula given in equation 8. Bias in that equation corresponds to 127 in single-precision number and 1203 in double-precision number.

$$x = (-1)^S \times (1 + Mantissa) \times 2^{(Exponent - Bias)} \quad (8)$$

To find the floating-point representation of any number, subsequent steps should be followed:

- Sign of the number should be determined
- The binary representation of the number should be calculated
- Normalize the number by putting the dot to the right of the most significant number and the right of the dot corresponds to the mantissa

- Add bias to the exponent and convert into binary (corresponds to the exponent)

For instance, the floating-point representation of 12.75 in single precision can be found by using the above steps.

1) The sign of the number is 0 since the number is positive. The rest would be the same if the number was negative.
2) Binary representation of $(12.75)_{10}$ is $(1100.11)_2$
3) When it is normalized, it becomes 1.10011×2^3 (the dot is moved by three points to the left). Removing the most significant bit produces the mantissa as 10011.
4) Adding 3 to 127 (the bias) yields 130 and conversion to the binary create the exponent which is 10000010.

Since the result should be in 32 bits, empty bit locations are padded with zero. The result is shown in Figure 21.

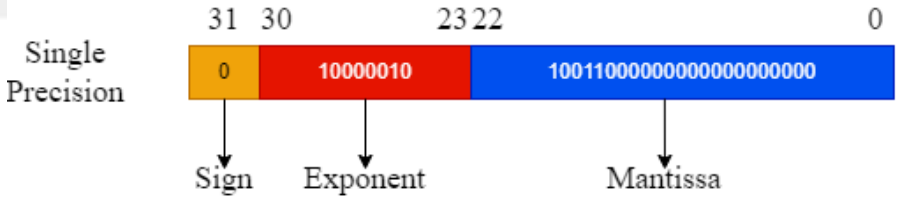


Figure 21. The floating-point representation of 12.75 in single precision

The floating-number conversion enables real numbers to be represented in very high precision in binary format, but it requires extensive operations to be used in hardware. If it is used in the FPGA, it would utilize lots of logic resources for floating-point arithmetic, and it would slow the operation that takes place on the FPGA. Therefore, instead of using the floating-point representation, the fixed-point conversion is employed in this thesis.

3.3.2. Fixed-point Representation

The fixed-point number means the number of bits used for representing the fractional part of the number. While the floating-number representation has higher precision, it has certain drawbacks such as speed and power consumption compared to the fixed-point representation. Another advantage of the fixed-point implementation over the floating-point representation is that it is simple to implement on the FPGA. Usually, Qm.n format notation is used to represent a fixed-point number, as illustrated in Figure 22, where m is the integer bit number including the sign bit and n is the bit number for representing the fractional part. For instance, 3.25 can be converted to a fixed-point number as (11.01).

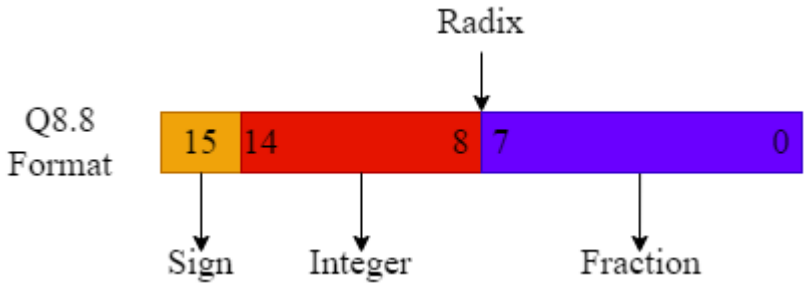


Figure 22. Fixed-point representation of Q8.8 format

The implementation of fixed-point arithmetic on the FPGA is not very complicated and does not use too much logic sources of the FPGA because standard integer arithmetic works for fixed-point numbers. The most critical part of fixed-point arithmetic is overflow. Suppose two numbers have m and n bits, respectively. The result of subtraction or addition of these two numbers requires (max (m, n) + 1) bits at maximum. On the other hand, the multiplication of these two numbers requires (m+n) bits in the worst-case scenario. If the required bit numbers are not arranged for the output, then most likely an overflow will occur. One of the drawbacks, in that case, each operation will increase the required bit numbers to avoid overflow. Since the resource is limited, the result may contain a fixed size by truncating the result. This

will affect the accuracy of the result as well as introduce the possibility of overflow. To overcome the overflow problem, two different approaches may be followed. The first approach would be detecting the overflow and when it occurs the result may be set to the maximum value. It is not a challenging task to detect overflow. For instance, if the mathematical operation of the same signed number produces the opposite signed result, then the overflow must have happened. The second approach would be testing the operations with the worst-case scenarios to figure out the required number of bits to be used in hardware design.

In this work, extracted parameters from the trained model are converted into fixed-point representations. Built-in MATLAB function called fi() is used for the conversion operation. The function takes 4 numbers which are the floating number to be converted into fixed-point representation, the sign of the output, word length, and the fraction length, respectively. The result of the fi() function is converted to the binary number with .bin() in MATLAB. For instance, fi(1.20, 1, 5, 3) produces 1.25, and the conversion to the binary yields a 5-bit binary representation of 1.20 as $(01.010)_2$.

Z. Li et al. (2018) suggested an 8-bit fixed-point CNN architecture which increases the speed and diminishes the consumption of logic sources, BRAM, and the power compared to a 32-bit implementation of the same design. In addition, it only loses 1% of accuracy. Hence, the fraction part is chosen to have 8 bits, so all the numbers (filter coefficients, weights, and biases) have an 8-bit representation. Using more bits increases the accuracy, but it utilizes more logic sources of the FPGA. Furthermore, to decide the required number of bits for the integer part, the model has been tested for various inputs and as a result, 7 bits for the representation of the integer part would guarantee that any mathematical operation would not be overflowed. Since the parameters of the model are not very large numbers, they do not require a 16-bit representation. Therefore, stored parameters in the FPGA have a different number of bits for the integer part, but all the parameters have an 8-bit fraction. For instance, Q2.8 notation is used for both the weights and the first convolution filter coefficients whereas Q1.8 notation is used for the second convolution filter coefficients. Even though parameters have different notations, they eventually converted into Q8.8

notation. In this notation, the fraction has 8 bits, the integer part has 7 bits, and 1 bit is used as the sign of the number. For example, the multiplication of two signed numbers having notations of Q5.8 and Q6.8 would produce a result with Q10.16 notation. Then, this result is truncated to Q8.8 notation which slightly reduces the accuracy.

3.4. 1D CNN Implementation on FPGA

The mathematical model of each layer in 1D CNN is established using Verilog on FPGA. The complete structure of the model is designed in a pipelined manner, so that high speed is achieved at the expense of utilizing more logic resources. Initially, the top module where all the operations are controlled is constructed. Since the model is developed through the bearing dataset, there is no real-time vibration data. Therefore, a sample from the training data converted into fixed-point representation is employed as the input for the model verification on the FPGA. It is written on a txt file consisting of 500 binary samples. The file is read by '$readmemb' and stored in the register called 'mem' which has the size of 500. In addition, the filter coefficients, weights, and biases are stored in registers.

1D CNN model begins with the first convolution layer. Since it has 8 neurons each having a kernel size of 3, the sub-module named 'single_conv' is instantiated 8 times in the top module. Each sub-module (neuron) consists of 3 'always' blocks which are employed for 1d convolution, ReLu, and max-pooing, respectively. It calculates the convolution output which passes through ReLu and max-pooing blocks and produces 8 samples of a 1-dimensional convolution array (8, 249). Although the input is predefined in the memory, it is not the case in a real-time application. To make sure that the design works in real-time, the input data will be sent one by one from the top module to the sub-module as if the input is read through the XADC port of the FPGA in real-time. The kernel has 3 coefficients which are multiplied with the corresponding input and summed to get a convolution output. The convolution is constructed in a single 'always' block consisting of 3 multiplication and 2 addition operations due to the size of the kernel being 3. When the positive edge of the clock occurs, the coming input data is stored in a memory, multiplication, and addition

operations are performed, and the result is sent to the next state (ReLu). Since the size of the input is 500, this means 500 samples will have been stored in the memory causes utilization of more logic resources. To overcome this problem, a circular ring buffer with 4 locations and a 2-bit pointer has been created. The pointer shows the location for the coming data. For each coming data, the offset pointer is increased by one. When it reaches the 3 which is the maximum number for a 2-bit number, it overflows making the pointer zero and the new coming data is located to the zeroth position of the circular buffer. Although an overflow is frustrating in most cases, here it is a useful paradigm that facilitates the design of a circular buffer. Instead of storing 500 samples, only 4 samples are enough for storage, and it utilizes fewer resources in this way.

Each convolution output is fed to the ReLu activation function which is constructed in the second ‘always’ block in the sub-module. In this module, input is checked whether it is a positive or a negative number. Simple if structure is enough to build a ReLu function. Since the max-pooling is defined in the model as having the kernel size and stride of 2, two consecutive ReLu outputs are provided to the max-pooling block which is the third ‘always’ block in the sub-module. The crucial part here is that max-pooling operation should be completed in pairs regarding the stride of 2 such as (1,2) or (3,4). To make sure that the pooling operation is done on correct pairs, a single if statement is employed to check the pair. Then, another if statement is used for finding the maximum of each pair which is the output of the first convolution layer. In this sub-module, all the blocks are parallelly executed. For instance, while the 6^{th} output of the convolution is being calculated in the first block at the 6^{th} positive edge of the clock, the 5^{th} convolution output determined in the previous cycle is fed into the ReLu function at the same time. In addition, the 3^{rd} and the 4^{th} convolution output are already passed through the ReLu, processes in the max-pooling block at the same time. In this way, the first convolution layer is just two cycles delayed from the upcoming input data, as illustrated in Figure 23.

Figure 23. Proposed FPGA architecture employing pipelined instructions

The second convolution layer has 4 neurons with 3 coefficients in the kernel followed by the ReLu and max-pooling. Since the output of the first convolution has 8 samples of 1d array, the second convolution has 96 (4 x 8 x 3) coefficients. To implement the second convolution in a pipelined technique, an inner module inside the sub-module is instantiated 4 times. Then, the convolution is calculated as explained in the previous section, but it is not the actual output of the second convolution. 32 convolution operations take place in the second layer, but it requires the addition of 8 convolution arrays to each other to produce a single output. The first inner modules (4) from each sub-module (8) should be summed to get the first output of the second convolution. Similarly, the rest of the second convolution can be calculated. Moreover, ReLu and max-pooling algorithms are the same as in the first convolution layer. The output of the second layer is 12 clock cycle delayed from the input and the result will be (4, 123) samples.

The next step is the fully-connected (dense) layer implementation. This is the most resource-consuming part of the FPGA due to storing all the weights and biases in the memory. It is composed of 10 neurons and each neuron is connected to the 492-

input coming from the second convolution layer. To implement this layer, a module called ‘neuron’ is instantiated 10 times from the top module. Since the whole procedure is fully pipelined, each clock cycle produces 4 single data from the second convolution layer. These 4 data are fed to all neurons in which they are multiplied with the corresponding weights and summed (4 multiplications and 3 additions). All the neuron outputs are initialized with their bias and each cycle the results are accumulated to get the actual output. Since each neuron has 492 inputs, all input arrives in 123 clock cycles. After the 123rd clock cycle, the ReLu is applied to all neuron outputs. Since all the operation until now is pipelined, only 14 cycle is delayed from the input. The output of the dense layer is achieved in 514 clock cycles.

The final step is the building of the output layer which has 4 neurons and the Softmax as the activation function. When the ReLu of the dense layer is completed, the output layer is ready to be calculated. Each neuron in the output is connected to the 10 neurons in the dense layer. Each clock cycle, the input to the output neuron is multiplied by the corresponding weight. 10 multiplications and the bias are summed for each neuron. The result of the neuron should go to the Softmax function which provides the probability of each class. Since the Softmax function is nonlinear, the implementation on the FPGA requires a significant number of sources. Instead of building the Softmax function, the so-called Hardmax function will be used because the Softmax function produces the maximum probability for the maximum output. Hence, it is basically finding the output neuron which possesses the maximum number. That specific neuron will be the class of the input. The total calculation takes place in 529 clock cycles.

CHAPTER 4: EXPERIMENTAL EVALUATIONS

This chapter demonstrates the simulation and implementation results of the methodology. It demonstrates the results of training the 1D CNN model using python and the simulations for different classes of the proposed FPGA architecture for the verification of the design using the Xilinx-Vivado development environment. All the training and the simulations are carried out on a personal computer having windows 10 as the operating system with Intel Core i7, 2.20 GHz, 8 GB of RAM, and GPU of NVIDIA GTX 1050 Ti. Next, two additional 1D CNN model is implemented on FPGA for the comparison of resource utilization on the Xilinx-Zybo FPGA. Later, the speed of the FPGA architecture is compared with its equivalent model on the CPU, GPU, and microcontroller. Finally, a real-time experimental setup is offered for the verification of the FPGA implementation.

4.1. Training Results

The CWRU dataset is used for the training of the 1D CNN model to detect bearing faults in induction motors. The proposed model has 2 convolution and 2 MLP layers. The first convolution has 8 filters with a kernel size of 3 whereas the second one has 4 filters with 3 kernels. Both layers without having padding and bias are followed by the ReLu and the max-pooling layer. Moreover, while the first MLP layer has 10 neurons and the ReLu as the activation function, the second MLP has 4 neurons (4 classes) with the Softmax function.

The dataset is divided into segments with 500 samples. Then, the train set is composed of 80% of the dataset while the rest is used for the validation. The training process using the Adam optimizer is completed in 40 epochs to update the weights. When the training is completed, the model is validated. The training accuracy reaches around 98% while 94% is achieved in the validation as shown in Figure 24. In addition, the training loss decreases to around 6% whereas the validation loss is 18% in the 40^{th} epoch, as illustrated in Figure 25. Moreover, the training is completed in just 3 minutes.

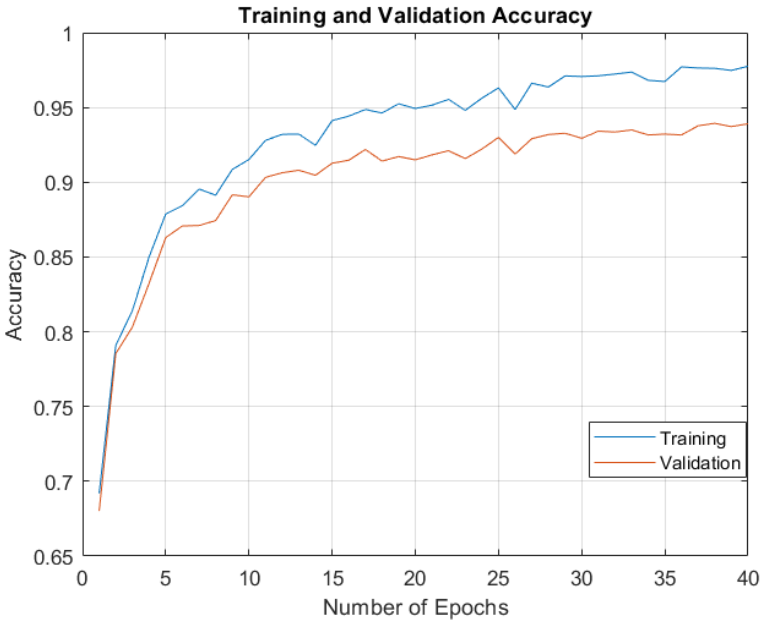


Figure 24. The training vs validation accuracy graph in the number of epochs

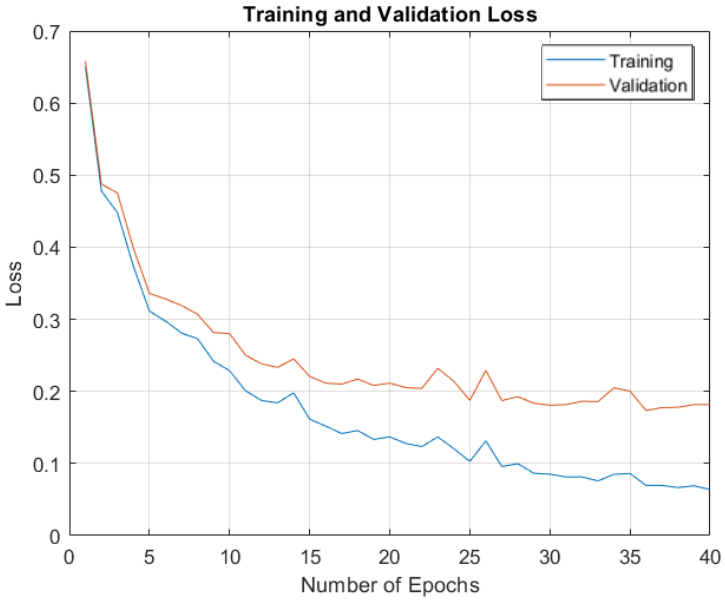


Figure 25. The training vs validation loss graph in the number of epochs

A confusion matrix, as given in Table 4, is created to show the number of true and the predicted values. By employing these numbers, the classification performance (given in Table 5) of the model can be evaluated with the utilization of performance metrics namely, Accuracy (Acc), Misclassification (Mis), Sensitivity (Sen), Specificity (Spe), Precision (Pre), and F1-score (F1). Accuracy and the misclassification are useful for evaluating the performance of the whole system whereas the other metrics are effective for an individual class. To give a better explanation for the calculation of these metrics, True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) terms should be elaborated. TP corresponds to both the true and the predicted values being positive, while TN indicates both being negative. On the other hand, FP means the model predicts a positive while it should be negative and conversely in FN. The terms are fruitful for elaborating the metrics.

Table 4. Confusion matrix of the proposed model for bearing fault detection

Confusion Matrix		PREDICTIONS			
		Healthy	Ball Bearing	Inner-race Fault	Outer-Race Fault
TRUE	Healthy	693	0	0	0
	Ball Bearing	0	679	58	32
	Inner-race Fault	1	23	692	63
	Outer-Race Fault	0	6	36	1315

Accuracy represents the ratio between accurate predictions and the total number of predictions, Acc = (TP + TN) / Total predictions; Mis-classification corresponds to (1 - Acc) which indicates the ratio between untrue prediction and the total number of predictions. Sensitivity, TP / (FN + TP), signifies how many of the true positives are classified as positive whereas Specificity, TN / (TN + FP), corresponds to negative predictions in the true negative. Precision indicates the ratio between true positive and the total positive prediction, Pre = TP / (TP + FP). Finally, the harmonic average of precision and the Sensitivity represents the F1-score, 2 x (Pre x Sen) / (Pre + Sen).

Table 5. Prediction performance of the proposed model for bearing faults

Classes	Classification Performance					
	Acc	Mis	Sen	Spe	Pre	F1
Healthy	93.91	6.08	100	100	99.85	99.92
Ball Bearing	93.91	6.08	88.29	96.88	95.90	91.93
Inner-race Fault	93.91	6.08	88.83	96.90	88.04	88.43
Outer-Race Fault	93.91	6.08	96.90	98.08	93.26	95.05

There are number of works in the literature that address the detection of bearing faults. The comparison of these studies is presented in Table 6. The proposed 1D CNN model does not accomplish the highest accuracy, but it produces competitive performance. Therefore, complex CNN architectures are not required to achieve high classification performance. Since the proposed method achieves fairly good accuracy without complex architectures, it is suitable for hardware implementation to monitor faults in real-time. Hence, the model is implemented on FPGA, the results will be given in the next part.

Table 6. Comparison of different studies for bearing fault detection

	Classifier	Feature	Accuracy
Proposed work	1D CNN	-	93.91%
Yaqup et al., (2012)	K-nearest neighbor	Higher order cumulants and wavelet transform	91.23%
Konar et al., (2011)	ANN	Continuous wavelet transform	96.67%
Zhang et al., (2017)	DNN	-	94.4% - 100%
Eren et al., (2018)	Compact 1D CNN	-	93.2% (CWRU)

4.2. FPGA Simulations

Network parameters obtained from the trained model are converted into fixed-point representations to be stored in FPGA. Then, the mathematical representation of the proposed model is implemented on FPGA using Verilog on the Xilinx-Vivado program. To verify the FPGA implementation, a number of simulations are carried out with various inputs from each class (Figure 26 - 29). Since they are floating-point numbers, they are converted into fixed-point using MATLAB, and then fed to the FPGA. For each simulation, corresponding Python results are also compared. Table 7 gives the results obtained from both Python and FPGA for the output neurons and the predicted class. Even though, there is a small error in FPGA simulations compared to Python due to fixed-point implementation, classes are accurately predicted.

Table 7. Comparison of neurons at output layer and the prediction from both Python and FPGA

Class		Output Layer				
		Neuron 0	Neuron 1	Neuron 2	Neuron 3	Prediction
Class 0 Healthy	Python	**0.5128**	-18.5462	-8.3229	0.2937	0
	FPGA	**0.5371**	-18.4746	-8.4035	0.3033	0
Class 1 Ball	Python	-14.0254	**0.6969**	-4.3956	-1.6758	1
	FPGA	-14.0416	**0.7328**	-4.3599	-1.8373	1
Class 2 Inner	Python	-17.7814	2.8451	**7.5968**	-17.8198	2
	FPGA	-18.0036	2.7396	**7.6293**	-17.7436	2
Class 3 Outer	Python	-22.5142	0.4072	-13.6794	**5.4012**	3
	FPGA	-22.6191	0.3468	-13.4828	**5.5061**	3

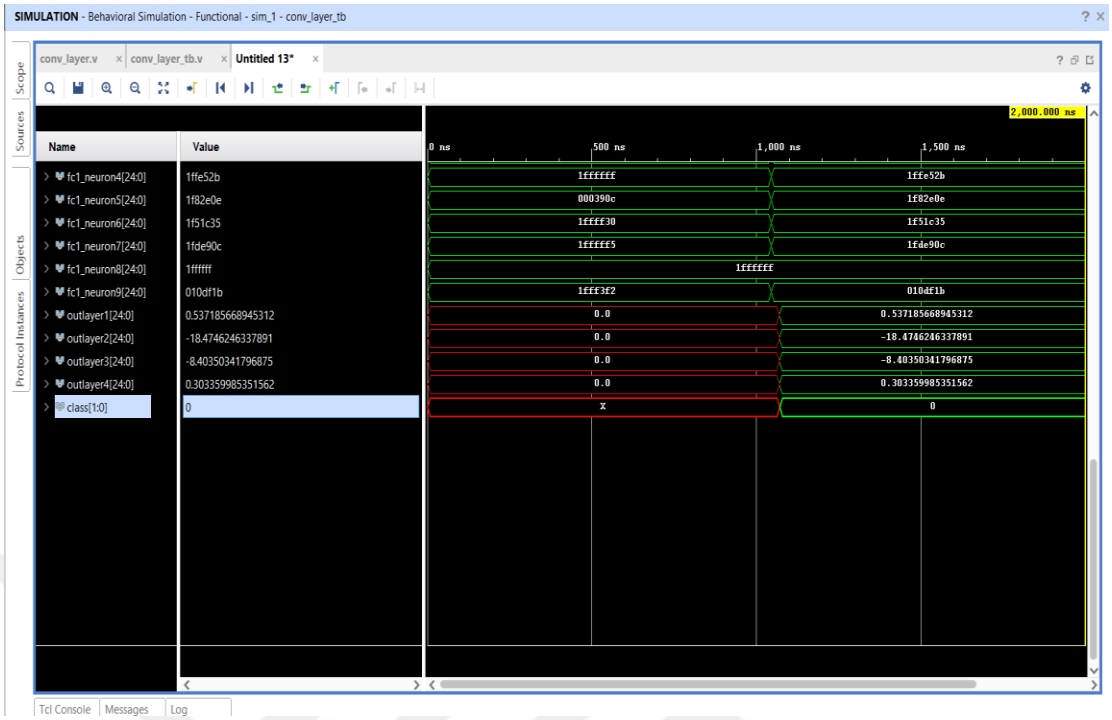


Figure 26. FPGA simulation for healthy input (class 0)

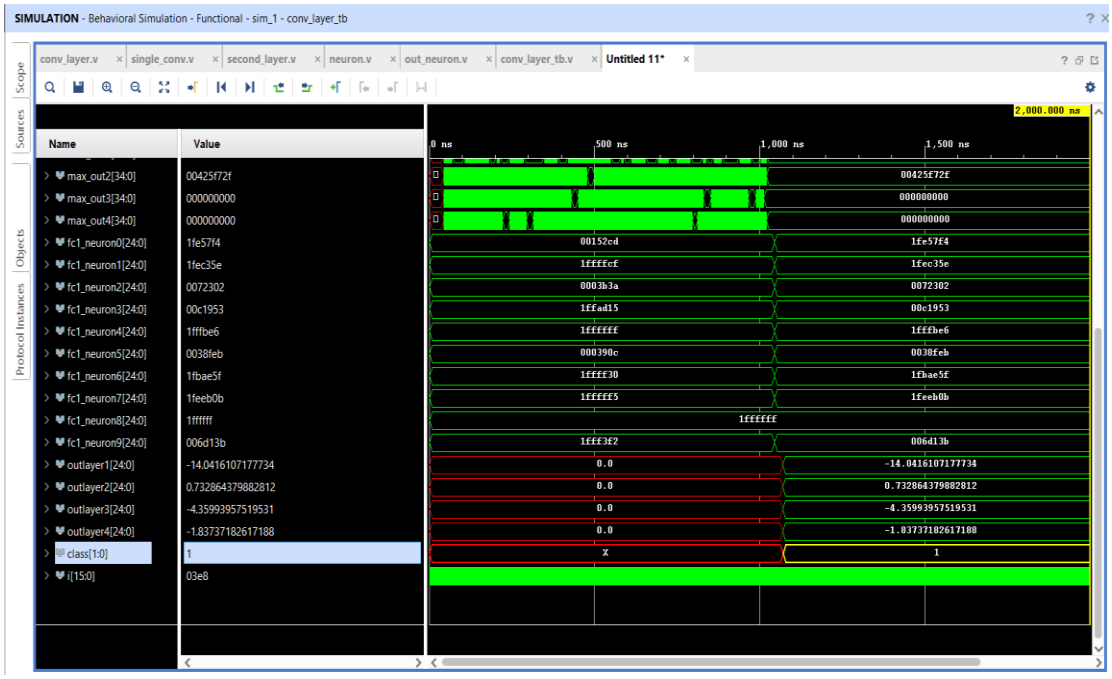


Figure 27. FPGA simulation for ball bearing (class 1)

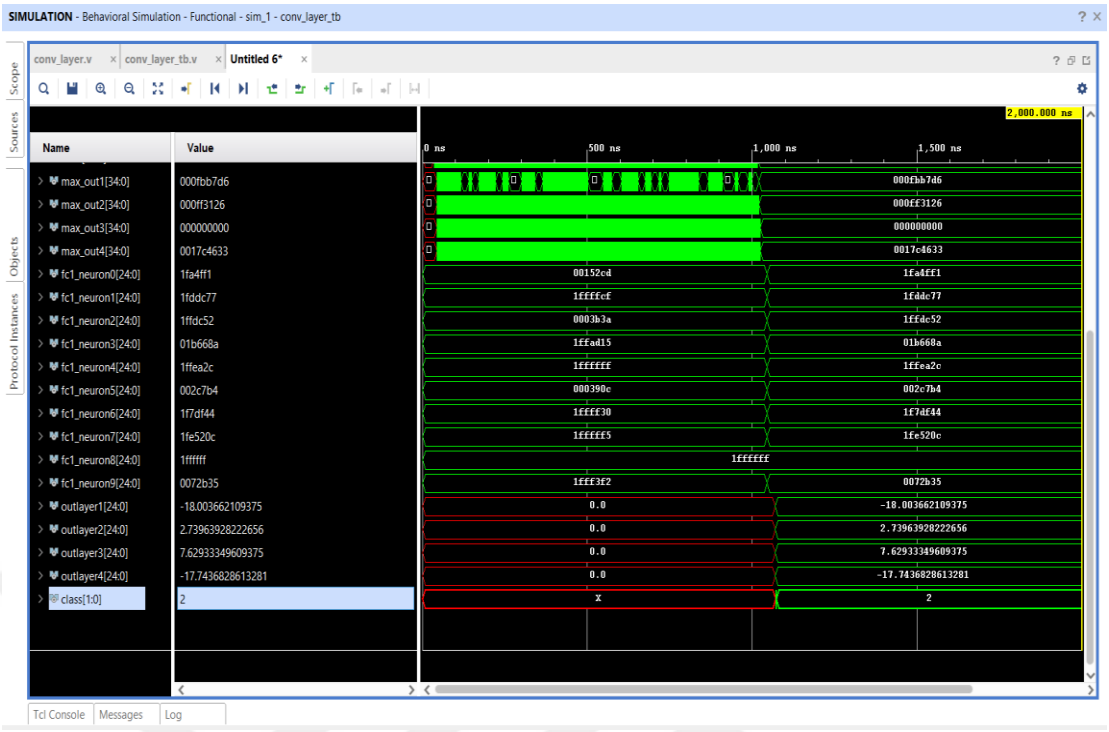


Figure 28. FPGA simulation for inner-raceway fault (class 2)

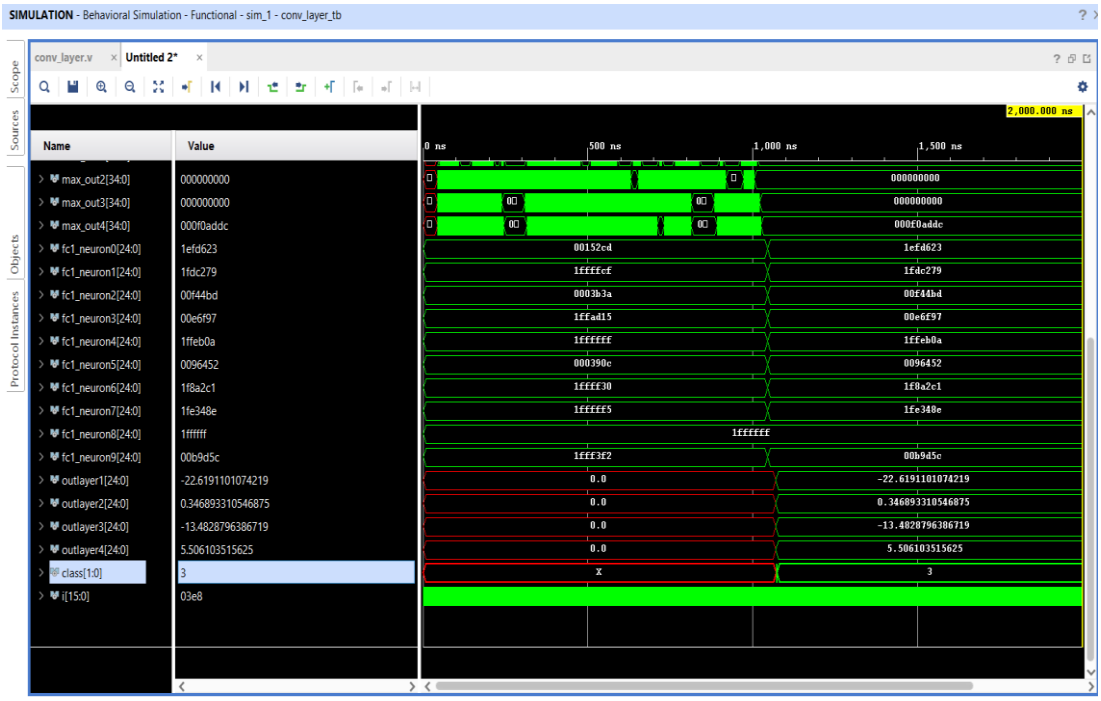


Figure 29. FPGA simulation for outer-raceway fault (class 3)

4.3. Resource Utilization of FPGA

When the verification of the design is completed, it is synthesized and implemented on the FPGA. Next, the resource utilization of the constructed model is acquired. To analyze resource utilization, another 1D CNN (model II) is established with a similar procedure. Model II lacks the second convolution operation compared to the proposed model. It has a single convolution layer which is consisted of 8 convolution filters with 3 kernels. Applying the ReLu and the max-pooling layer produces an array of (8x249). This is the input for the dense layer with 10 neurons followed by an output layer having 4 neurons. In this model, 19998 parameters should be stored in RAM whereas only 4998 parameters for the proposed model. The validation accuracy is obtained as 82.4%. Table 8 gives the resource utilization for both models. The highest difference is obtained in the utilization of the Lutram due to the required number of parameters to be stored in FPGA. Parameters are implemented on distributed ram (Lutram) in the proposed model whereas on BRAM in the second model. To compare the designs, the term FPGA-accuracy is also proposed. This corresponds to the comparison between python prediction and FPGA simulation. A total of 50 different inputs were applied to both models, and the result is represented in Table 8. Both models achieve an accuracy of 100% due to implementing an 8-bit fixed-point representation for the parameters. Reducing the number of bits for the fixed-point representation will cause a decrease in the so-called FPGA accuracy.

Table 8. Resource utilizations and the FPGA-accuracy for 2 different models

	LUT	LUTRAM	Flip-Flops	DSP	I/O	FPGA-Acc
Proposed Model	13750 78.13%	1728 28.80%	7914 22.48%	74 92.50%	3 3%	100%
Model II	13245 75.26%	192 3.20%	4600 13.07%	80 100%	3 3%	100%

4.4. Speed Comparison

The execution speed of the proposed model is compared with its equivalent representation on the CPU, GPU, and Microcontroller to show that it can be employed in real-time for monitoring or diagnosis of induction motors. Table 9 gives the execution speeds for four implementations. The execution speed of FPGA is acquired from the simulation which takes 529 clock cycles. Since the clock frequency of FPGA is 125 MHz (8 ns), the speed is (529 x 8 ns = 4.232 μ seconds). Speed of the CPU and GPU are obtained through the validation time using Python. It has been validated 20 times and the average is calculated. Lastly, the speed of the equivalent model on STM32 microcontroller is obtained (Kilickaya, 2022). Since the FPGA implementation is the fastest among them, it can be used in real-time not just for the bearing fault detection but for other applications utilizing 1D CNN in the literature.

Table 9. Speed comparison of four different design

	Speed
FPGA	4.232 μsec
CPU	73.87 μsec
GPU	64.76 μsec
Microcontroller	20.327 msec

4.5. Real-time Data Acquisition Through UART Communication

Since the proposed model is developed through the CWRU dataset, there is no real-time vibration signal. It is possible to store test samples in RAM to test the FPGA in real-time, but it is not an efficient way to implement it. If the vibration signal was available, it could be read through the ADC of the FPGA to test the model. To overcome this problem, the UART communication protocol could be employed for

sending data to the FPGA in real-time. Xilinx SoC supports the UART communication, but it takes the data through the ARM processor in the SoC. Acquiring data from ARM does not make the system a generic design for other FPGAs. Hence, a USB to TTL converter is employed for the data transmission. The setup is given in Figure 30. The transmitter of the converter is connected to one of the digital inputs of the FPGA. Realterm application is used for the data transfer to the FPGA, but it only supports 8 bits at a time. Therefore, two consecutive data is used for a single input. The receiver module is implemented using Verilog with 9600 baud rates. The Verilog implementation can have two states such as idle and receive. In an idle state, the input is one (logic High) and no action is required. In receive state, the transition of 1 to 0 occurs, so the data can be read. Since the baud rate of 9600 is used, each data takes 104 μ seconds. Since the clock frequency is 125 MHz, a counter is created to count 104 μ seconds each time. When 8 consecutive data bits are acquired, the counter is set to zero. The receiver module is integrated into the 1D CNN model to store 500 samples in the RAM of the FPGA along with the constraint file for the inputs and output. Since the model can produce 4 different outputs, each led on the FPGA corresponds to the state of the motor faults. Then, the 1D CNN model process the input and produce the corresponding output by turning on the led.

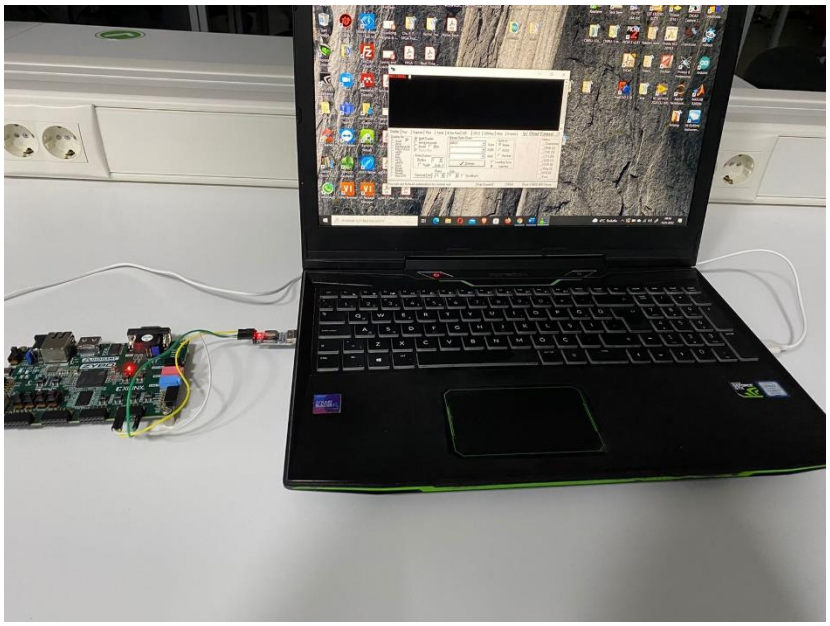


Figure 30. Real-time data acquisition through UART communication

CHAPTER 5: CONCLUSION

Electrical motors play a vital role in numerous applications in the industry because of their cost and ease of repairing. Since it has been extensively used, failures of these motors are a serious problem. It may cost a significant amount of money because of the motor failure causing a delay in the manufacturing process or even worse, it may be dangerous to human life.

Induction motors contain roller bearings that increase the number of revolutions and efficiency by lowering the friction between rotating parts, and it is the most statistically confronted failure. Despite the fact that detecting those failures is a complex task, it prevents severe expenses if immediately diagnosed. There are many studies available in the literature for the early detection of bearing faults, but none of them develops special hardware (ASIC/FPGA) that solves the problem. The main motivation of this thesis is to build a model on FPGA for the early diagnosis of bearing faults.

In this thesis, the FPGA implementation of the 1D CNN model for detecting bearing faults in induction motors is presented. The proposed architecture is constructed using the benchmark CWRU dataset. It is the collection of vibration signals acquired for 4 different classes which are healthy, ball bearing, inner-raceway, and outer-raceway faults. 1D CNN which does not require hand-crafted features is chosen as the classifier of the model because it can understand special features directly from the raw data. Then, the 1D CNN model is developed using Python in the Jupyter notebook. The model is established with 2 Convolution and 2 MLP layers. The Softmax activation function is employed at the output layer whereas the rest of the model utilizes the ReLu as the activation function. Furthermore, 80% of the dataset is randomly taken for the test set, and the rest is employed for the validation of the model. The model is trained with the test set for 40 epochs and the Adam optimizer is utilized for updating the weights. The training and validation accuracies are attained as 98% and 94%, respectively. The validation accuracy produces a fair result compared to the studies published in the literature. Since the model does not involve manual feature

extraction and is not a deep and complex architecture, it is suitable for real-time implementation.

When the training is completed, the parameters are extracted from the trained model as a 32-bit floating-point. The next step is to deploy parameters into the FPGA, but the parameters should be described in binary form. There are two options for number representations namely: the floating-point and fixed-point. The floating-point representation achieves higher accuracy and provides more precision than the fixed-point representation, but it consumes more logic resources of the FPGA and takes a longer time compared to the fixed-point representation. Hence, the parameters are converted into 8-bit (for the fractional part) fixed-point representations using MATLAB.

The last part is to develop the mathematical representation of the 1D CNN model (feed-forward) on FPGA. Initially, the parameters are stored in FPGA. Then, each layer is implemented in a pipelined manner using Verilog on the Xilinx-Vivado tool. The design is verified through simulations while comparing the results from the Python prediction. Then, the design is synthesized, implemented, and sent to the FPGA through a bit-stream file for programming. Finally, the speed of the FPGA model is achieved as 4.232 μ seconds which makes it a faster design than its equivalent model on the CPU, GPU, and the Microcontroller. The FPGA model can be applied to the input batch of 500 samples or for each sampled data up to 236 kHz (sampling frequency) which corresponds to the bandwidth of 118 kHz. Therefore, the model is also appropriate for real-time audio signal processing.

As for future works, the experimental setup for the bearing fault detection will be created for real-time vibration signal acquisition. The data will be gathered for the model generation. When the model is trained, it will be tested in real-time by measuring vibration signal through the XADC port of the FPGA. Then, the IP (intellectual property) core containing the parametric 1D CNN model will be established, so that this IP can be employed for other applications in the literature.

REFERENCES

Case Western Reserve University. (2004) Bearing Data Center, seeded fault test data. Available at: https://engineering.case.edu/bearingdatacenter/download-data-file (Accessed: 25 August 2021)

Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., and Temam, O. (2014) *DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning*, SIGARCH Comput. Archit. News, Vol. 42(1), pp. 269–284.

Chow, M.Y., Yee, S.O., and Mangum, P.M. (1991) *A Neural Network Approach to Real-Time Condition Monitoring of Induction Motors*, IEEE Transactions on Industrial Electronics, Vol. 38, pp. 448–453.

Contreras-Hernandez, J.L., Almanza-Ojeda, D.L., Ledesma, S., and Ibarra-Manzano, M.A. (2019) *Motor fault detection using Quaternion Signal Analysis on FPGA*, Measurement: Journal of the International Measurement Confederation, Vol. 138, pp. 416–424.

Dai, X., and Gao, Z. (2013) *From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis*, IEEE Transactions on Industrial Informatics, Vol. 9, pp. 2226–2238.

[Digilent]. (2021) Documentation for ZYBO FPGA Board [Web-based visual]. Available at: https://digilent.com/reference/programmable-logic/zybo/start (Accessed: 10 December 2021)

Eren, L., and Devaney, M.J. (2004) *Bearing Damage Detection via Wavelet Packet Decomposition of the Stator Current*, IEEE Transactions on Instrumentation and Measurement, Vol. *53*, pp. 431–436.

Eren, L., Ince, T., and Kiranyaz, S. (2019) *A Generic Intelligent Bearing Fault Diagnosis System Using Compact Adaptive 1D CNN Classifier*, Journal of Signal Processing Systems, Vol. 91, pp. 179–189.

Farabet, C., Poulet, C., Han, J.Y., and LeCun, Y. (2009) *CNP: An FPGA-based processor for Convolutional Networks*, In FPL 09: 19th International Conference on Field Programmable Logic and Applications, pp. 32–37.

Filippetti, F., Bellini, A., and Capolino, G.A. (2013) *Condition monitoring and diagnosis of rotor faults in induction machines: State of art and future perspectives*, In Proceedings - 2013 IEEE Workshop on Electrical Machines Design, Control and Diagnosis, WEMDCD 2013, pp. 196–209.

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M.A., and Dally, W.J. (2016) *EIE: Efficient Inference Engine on Compressed Deep Neural Network*, In Proceedings - 2016 43rd International Symposium on Computer Architecture, ISCA 2016, (Institute of Electrical and Electronics Engineers Inc.), pp. 243–254.

Ince, T., Kiranyaz, S., Eren, L., Askar, M., and Gabbouj, M. (2016) *Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks*, IEEE Transactions on Industrial Electronics, Vol. 63, pp. 7067–7075.

Janssens, O., Slavkovikj, V., Vervisch, B., Stockman, K., Loccufier, M., Verstockt, S., Van de Walle, R., and Van Hoecke, S. (2016) *Convolutional Neural Network Based Fault Detection for Rotating Machinery*, Journal of Sound and Vibration, Vol. 377, pp. 331–345.

Karim, E., Memon, T.D., and Hussain, I. (2019) *FPGA based on-line fault diagnostic of induction motors using electrical signature analysis*, International Journal of Information Technology (Singapore), Vol. 11, pp. 65–169.

Kilickaya, S. (2022). "*Microcontroller-based real-time motor bearing fault detection and diagnosis using 1D convolutional neural networks*." Master Thesis. Izmir University of Economics.

Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., and Inman, D.J. (2021) *1D convolutional neural networks and applications: A survey*, Mechanical Systems and Signal Processing, Vol. 151, pp. 107398.

Kiranyaz, S., Ince, T., and Gabbouj, M. (2016) *Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks*, IEEE Transactions on Biomedical Engineering, Vol. 63, pp. 664–675.

Konar, P., and Chattopadhyay, P. (2011) *Bearing fault detection of induction motor using wavelet and Support Vector Machines (SVMs)*, Applied Soft Computing Journal, Vol. 11, pp. 4203–4211.

Kuon, I., Tessier, R., and Rose, J. (2007) *FPGA architecture: Survey and challenges*, Foundations and Trends in Electronic Design Automation, Vol. 2, pp. 135–253.

Lee, J., Qiu, H., Yu, G., and Lin, J., (2007) Rexnord Technical Services IMS, University of Cincinnati. Bearing Data Set, NASA Ames Prognostics Data Repository. Nasa Ames Research Center: Moffet Field, CA, USA, Available at: http://ti.arc.nasa.gov/project/prognostic-data-repository (Accessed: 10 January 2022)

Lessmeier, C., Kimotho, J.K., Zimmer, D., and Sextro, W. (2016) *Condition monitoring of bearing damage in electromechanical drive systems by using motor current signals of electric motors: a benchmark data set for data-driven classification*, Third European Conference of the Prognostics and Health Management Society, pp. 152-156.

Li, Y., Wang, X., Si, S., and Huang, S. (2020) *Entropy Based Fault Classification Using the Case Western Reserve University Data: A Benchmark Study*, IEEE Transactions on Reliability, Vol. 69, pp. 754–767.

Li, Z., Wang, L., Guo, S., Deng, Y., Dou, Q., Zhou, H., and Lu, W. (2018) *Laius: An 8-bit fixed-point CNN hardware inference engine*, In Proceedings - 15th IEEE International Symposium on Parallel and Distributed Processing with Applications and 16th IEEE International Conference on Ubiquitous Computing and Communications, ISPA/IUCC 2017, (Institute of Electrical and Electronics Engineers Inc.), pp. 143–150.

Lizarraga-Morales, R.A., Rodriguez-Donate, C., Cabal-Yepez, E., Lopez-Ramirez, M., Ledesma-Carrillo, L.M., and Ferrucho-Alvarez, E.R. (2017) *Novel FPGA-based methodology for early broken rotor bar detection and classification through homogeneity estimation*, IEEE Transactions on Instrumentation and Measurement, Vol. 66, pp. 1760–1769.

Malhi, A., and Gao, R.X. (2004) *PCA-based feature selection scheme for machine defect classification*, IEEE Transactions on Instrumentation and Measurement, Vol. 53, pp. 1517–1525.

MPFT. (2013) Condition based maintenance fault database for testing of diagnostic and prognostics. Available at: https://www.mfpt.org/fault-data-sets/ (Accessed: 10 January 2021)

Nectoux, P., Gouriveau, R., Medjaher, K., Ramasso, E., Morello B., Zerhouni, N., Varnier, C. (2012) *PRONOSTIA: An Experimental Platform for Bearings Accelerated Life Test*, IEEE International Conference on Prognostics and Health Management, PHM'12, pp. 1-8.

Neupane, D., and Seok, J. (2020) *Bearing fault detection and diagnosis using case western reserve university dataset with deep learning approaches: A review*, IEEE Access, Vol. 8, pp. 93155–93178.

[Nvidia]. (2022). Nvidia DGX-1. Efficient instrument for AI research. Available at: https://www.nvidia.com/en-us/data-center/dgx-1/ (Accessed: 20 January 2022)

Pekel, E., and Kara, S.S. (2017) *A Comprehensive Review for Artificial Neural Network Application to Public Transportation*, Sigma Journal of Engineering and Natural Sciences, Vol. 35, pp. 157–179.

Rawat, W., and Wang, Z. (2017) *Deep convolutional neural networks for image classification: A comprehensive review*, Neural Computation, Vol. 29, pp. 2352–2449.

Soualhi, A., Clerc, G., and Razik, H. (2013) *Detection and diagnosis of faults in induction motor using an improved artificial ant clustering technique*, IEEE Transactions on Industrial Electronics, Vol. 60, pp. 4053–4062.

Tao, J.H., Du, Z.D., Guo, Q., Lan, H.Y., Zhang, L., Zhou, S.Y., Xu, L.J., Liu, C., Liu, H.F., Tang, S., et al. (2018) *BenchIP: Benchmarking Intelligence Processors*, Journal of Computer Science and Technology, Vol. 33, pp. 1–23.

Tu, J.V. (1996) *Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes* Journal of Clinical Epidemiology, Vol. 49, pp. 1225–1231.

Ulloa, M.A.J., (2020) Forward and Back Propagation over a CNN [Web-based visual]. Available at: https://www.linkedin.com/pulse/forward-back-propagation-over-cnn-code-from-scratch-coy-ulloa (Accessed: 20 December 2021)

Wang, Y.S., Ma, Q.H., Zhu, Q., Liu, X.T., and Zhao, L.H. (2014) *An intelligent approach for engine fault diagnosis based on Hilbert-Huang transform and support vector machine*, Applied Acoustics, Vol. 75, pp. 1–9.

Wang, B., Lei, Y., Li, N., and Li, N. (2020) *A Hybrid Prognostics Approach for Estimating Remaining Useful Life of Rolling Element Bearings*, IEEE Transactions on Reliability, Vol. 69(1), pp. 401-412.

Waziralilah, N. F., Abu, A., Lim, M. H., Quen, L. K., & Elfakharany, A. (2019*) A Review on Convolutional Neural Network in Bearing Fault Diagnosis*, MATEC Web of Conferences, Vol. 255, pp. 6002.

Wen, L., Li, X., Gao, L., and Zhang, Y. (2018) *A New Convolutional Neural Network-Based Data-Driven Fault Diagnosis Method*, IEEE Transactions on Industrial Electronics, Vol. 65, pp. 5990–5998.

Wovk, V. (1991) *Machinery Vibration, Measurement and Analysis*, New York: McGraw-Hill.

Yaqub, M.F., Gondal, I., and Kamruzzaman, J. (2012) *Inchoate fault detection framework: Adaptive selection of wavelet nodes and cumulant orders*, IEEE Transactions on Instrumentation and Measurement, Vol. 61, pp. 685–695.

Yeh, C. C., Sizov, G. Y., Sayed-Ahmed, A., Demerdash, N. A. O., Povinelli, R. J., Yaz, E. E., & Ionel, D. M. (2008) *A reconfigurable motor for experimental emulation of stator winding interturn and broken bar faults in polyphase induction machines*, IEEE Transactions on Energy Conversion, Vol. 23(4), pp. 1005–1014.

Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., and Cong, J. (2015) *Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks*, In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15). Association for Computing Machinery, New York, pp. 161–170.

Zhang, R., Peng, Z., Wu, L., Yao, B., and Guan, Y. (2017) *Fault diagnosis from raw sensor data using deep neural networks considering temporal coherence*, Sensors (Switzerland), Vol. 17(3), pp. 549-566.

Zhang, W., Zhang, F., Chen, W., Jiang, Y., and Song, D. (2019) *Fault State Recognition of Rolling Bearing Based Fully Convolutional Network*, Computing in Science and Engineering, Vol. 21, pp. 55–63.