YAŞAR UNIVERSITY

GRADUATE SCHOOL

MASTER'S THESIS

MACHINE STATE DETECTION BY CNN ON A LOW POWER MICROCONTROLLER

DORUK ERDEMGİL

THESIS ADVISOR: PROF. DR. MUSTAFA GÜNDÜZALP

ELECTRICAL AND ELECTRONICS ENGINEERING

BORNOVA / İZMİR

AUGUST / 2024

JURY APPROVAL PAGE

We certify that, as the jury, we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Jury Members:	**Signature:**
Prof. (PhD) Mustafa GÜNDÜZALP Yaşar University	. . . . .
Assoc. Prof. (PhD) Burhan GÜLBAHAR Yaşar University	. . . . .
Assoc. Prof. (PhD) Yavuz ŞENOL Dokuz Eylül University	. . . . .

Prof. (PhD) Yücel Öztürkoğlu

Director of the Graduate School

ABSTRACT

MACHINE STATE DETECTION BY CNN
ON A LOW POWER MICROCONTROLLER

Erdemgil, Doruk
MSc, Electrical and Electronics Engineering
Advisor: Prof. (PhD) Mustafa GÜNDÜZALP
August 2024

Today, with the widespread adoption of the Industrial Internet of Things (IIoT), predictive maintenance (PdM) applications have become increasingly prevalent. The PdM plays a crucial role in assessing the conditions of industrial equipment and preventing breakdowns. In addition, with the introduction of Artificial Intelligence (AI) algorithms, AI has been implemented in PdM applications to increase the early detection capabilities. Traditionally, applications to identify potential errors in industrial peripherals have been implemented in high-power computer systems, which consume significant amounts of energy. Additionally, they often rely on server-side processing or data transfer, which require considerable energy. However, with the introduction of Tiny Machine Learning (TinyML) algorithms and microcontroller units (MCU) equipped with neural network accelerators, such as the MAX78000 MCU, these applications can now be implemented in low-power MCUs, resulting in substantial energy and cost savings. In this thesis, a Convolutional Neural Network (CNN) model was implemented that uses raw machine audio as input and utilizes the MAX78000 MCU to classify the audio of end mill cutter blades into four classes: fresh, moderate, broken, and base according to their wear rates. This thesis compares 1-dimensional (1D) and 2-dimensional (2D) CNN models with different parameters. The 1D CNN approach is shown to outperform the 2D CNN approach by 9% in terms of top-1 accuracy. It was found that the 1D approach has 93% accuracy for detecting classes, and the low-power MCU draws 12.88 mW power on average.

Keywords: Predictive Maintenance, IIoT, TinyML, Edge Computing

ÖZ

DÜŞÜK GÜÇLÜ MİKRO DENETLİYİCİDE EVRİŞİMSEL SİNİR AĞLARI KULLANARAK MAKİNE DURUM TESPİTİ

Erdemgil, Doruk
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği
Danışman: Prof. Dr. Mustafa GÜNDÜZALP
Ağustos 2024

Günümüzde, Endüstriyel Nesnelerin İnterneti'nin (IIoT) yaygın olarak benimsenmesiyle birlikte kestirimci bakım (PdM) uygulamaları giderek daha fazla önem kazanmaktadır. PdM, endüstriyel ekipmanların durumlarını değerlendirmede ve arızaları önlemede kritik bir rol oynamaktadır. Ayrıca, Yapay Zeka (AI) algoritmalarının tanıtılmasıyla, PdM uygulamalarında erken tespit yeteneklerini artırmak için AI uygulanmıştır. Geleneksel olarak, endüstriyel çevre birimlerinde potansiyel hataları belirlemek için kullanılan uygulamalar, yüksek enerji tüketen bilgisayar sistemlerinde gerçekleştirilmekteydi. Ayrıca, bu sistemler genellikle sunucu tarafı işleme veya veri aktarımına dayanmakta olup, bu da önemli miktarda enerji gerektirmektedir. Ancak, Tiny Machine Learning (TinyML) algoritmalarının ve MAX78000 MCU gibi nöral ağ hızlandırıcılarıyla donatılmış mikroişlemci ünitelerinin (MCU) tanıtılmasıyla, bu uygulamalar artık düşük enerji tüketen MCU'larda uygulanabilmekte ve bu da önemli enerji ve maliyet tasarrufları sağlamaktadır. Bu tezde, ham makine sesini girdi olarak kullanan ve parmak freze kesici bıçaklarının seslerini aşınma oranlarına göre dört sınıfa ayıran (yeni, orta, kırık ve temel) bir Evrişimsel Sinir Ağları (CNN) modeli uygulanmıştır. Bu çalışma, farklı parametrelerle 1-boyutlu (1D) ve 2-boyutlu (2D) CNN modellerini karşılaştırmaktadır. 1D CNN yaklaşımının, top-1 doğruluğu açısından 2D CNN yaklaşımından %9 daha başarılı olduğunu gösterilmektedir. 1D yaklaşımının sınıfları tespit etmede %93 doğruluğa sahip olduğu ve düşük güç tüketen MCU'nun ortalama 12.88 mW güç çektiği bulunmuştur.

Anahtar Kelimeler: Kestirimci Bakım, Endüstriyel Nesnelerin İnterneti (IIoT), TinyML, Kenar Bilişim

ACKNOWLEDGEMENTS

I would like to begin by expressing my sincere gratitude to everyone who has contributed to my academic and personal development throughout my Master's thesis journey.

First, I would like to sincerely thank my advisor, Prof. Dr. Mustafa Gündüzalp, for their guidance and support throughout this thesis. His insights and encouragement have been instrumental in my academic journey.

Second, I would like to extend my sincere thanks to all the faculty members of the Department of Electrical and Electronics Engineering at Yaşar University. Their guidance and support throughout both my undergraduate and graduate studies have been invaluable. I am deeply grateful for the knowledge and encouragement they provided, which have significantly contributed to my academic and personal growth.

Additionally, I am profoundly thankful to my colleagues at Yaşar University. Their unwavering encouragement and moral support during my Master's studies have been a constant source of motivation.

Finally, I would like to extend my heartfelt thanks to my family, especially my father, Hakan Erdemgil, and my mother, Gülsüm Elvan Erdemgil, for their continuous belief in me throughout my undergraduate and Master's studies. Their unwavering support has been an anchor for me.

I am also grateful to my friends for their emotional support and encouragement throughout this journey. Your presence and kindness have been deeply appreciated.

I would like to thank my girlfriend, Buse Pehlivan, for standing by my side and offering her endless support during my lowest and most anxious moments. Her motivation, love, and care have been a tremendous source of strength.

Doruk ERDEMGİL
İzmir, 2024

TEXT OF OATH

I declare and honestly confirm that my study, titled “MACHINE STATE DETECTION BY CNN ON A LOW POWER MICROCONTROLLER” and presented as a Master’s Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

DORUK ERDEMGİL

Signature:

_____

.....

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND SYMBOLS

ABBREVIATIONS:

PdM	Predictive Maintenance
IoT	Internet of Things
IIoT	Industrial Internet of Things
ML	Machine Learning
TinyML	Tiny Machine Learning
MCU	Microprocessor Unit
AI	Artificial Intelligence
PMON	Power Monitor
I^2S	Inter- Integrated Circuit Sound
SPI	Serial Peripheral Interface Bus
UART	Universal Asynchronous Receiver-Transmitter
GPIO	General-Purpose Input/Output
CNN	Convolutional Neural Network
DNN	Deep Neural Network
Conv1D	1-Dimensional Convolution
Conv2D	2-Dimensional Convolution
HL	Hidden Layer
ReLU	Rectified Linear Unit
QAT	Quantization Aware Training
MLP	Multi Layer Perceptron
LSTM	Long-Short Term Memory

SYMBOLS:

L_{train}	Train loss that is calculated over train dataset

Continued on the next page

SYMBOLS:

$y_{train,i}$	Binary indicator (0 or 1) if class label i is the correct classification for the current input in train set
$p_{train,i}$	Predicted probability of the input in train set being in class i
C	Number of classes in classification task
L_{val}	Validation loss that is calculated over validation dataset
$y_{val,i}$	Binary indicator (0 or 1) if class label i is the correct classification for the current input in validation set
$p_{val,i}$	Predicted probability of the input in validation set being in class i
L_{test}	Test loss that is calculated over test dataset
$y_{test,i}$	Binary indicator (0 or 1) if class label i is the correct classification for the current input in test set
$p_{test,i}$	Predicted probability of the input in test set being in class i

1. CHAPTER: INTRODUCTION

The aim of this thesis is to develop a real-time fault detection system for predictive maintenance and IIoT applications by using machine audio data to classify defects and faults of industrial peripheral and demonstrate the work on low-power microcontroller MAX78000. This chapter aims to provide information of the problem that will be addressed in this thesis alongside background information of the provided work. First, we review the structure of existing Predictive maintenance (PdM) applications. Second, we describe the problem that will be addressed in this thesis and present our approach alongside necessary background information. Third, we explain the relationship of this thesis with the existing literature. Fourth, we explain the contributions of this thesis to the state of the art. Fifth, we give the outline of this thesis.

1.1. Predictive Maintenance (PdM)

Predictive maintenance (PdM) is an innovative method for industrial equipment maintenance. PdM is a proactive maintenance technique that is based on Internet of Things (IoT) and Industrial Internet of Things (IIoT). There are two main PdM approaches: (1) to predict failures before they occur, (2) to identify any abnormal behaviours in machine data. The main goal of these approaches are: to reduce downtimes, lower maintenance costs and extend the lifespan of machinery, thereby improving overall operational efficiency. The key to predictive maintenance is the continuous monitoring of equipment using a network of IoT sensors that capture various operational parameters such as temperature, vibration, and pressure. In traditional methods, captured data is transmitted to a cloud or server based central system and analyzed by using manual methods or it is analyzed by using machine learning (ML) models. Machine learning models can handle vast amounts of data in real-time and identify the complex patterns in machine data. Hence, using machine learning models can improve the identification of patterns in machine behaviours. Thus, the system can detect faults or defects more accurate compared to manual methods (Jadhav et al., 2023). By leveraging IoT and machine learning, predictive maintenance not only boosts operational efficiency but also supports sustainable practices by reducing waste and conserving resources. As industries continue to adopt digital transformation, the role of predictive maintenance

will become increasingly important in maintaining the health and efficiency of industrial equipment.

1.2. Tiny Machine Learning (TinyML)

Traditional machine learning applications for industrial usage requires powerful computers or cloud-based servers to process the data and compute the algorithms. While these applications have a high computing power, they also have downsides such as high power consumption, high installation and operation costs and they often can not be used at the edge, because of their lack of mobility. To overcome these constraints, Tiny Machine Learning (TinyML) solutions have emerged.

TinyML is an approach that combines machine learning algorithms with microcontrollers. The main goal of this approach is to reduce power consumption, reduce operating costs and enable the use of machine learning algorithms in real-time at the edge. With TinyML applications, ML applications can be utilized on low power microcontrollers. These TinyML application consume less power compared to regular ML applications (Banbury et al., 2020).

To use TinyML applications, the deep learning or machine learning models must be quantized and optimized to minimize the amount of data to be computed by microprocessors and to utilize processors of microcontrollers most efficiently. Even though these applications can be run on general-purpose microcontrollers, there is an emerging technology for microprocessors integrated with neural network accelerators. These microprocessors are built specifically for these applications and they consume much less power, compared to general-purpose microprocessors, in addition to their significantly higher performance. In this thesis, MAX78000 Microcontroller is utilized for our TinyML application.

1.2.1 MAX78000 Microcontroller Unit (MCU)

The MAX78000 is a microcontroller unit (MCU) developed by Maxim Integrated, specifically designed for artificial intelligence (AI) and machine learning (ML) applications. It integrates a neural network accelerator, which enables efficient on-device inferencing for edge AI tasks. This MCU combines traditional microcontroller functions with advanced AI features, making it suitable for energy-efficient AI processing.

In Fig. 1.1, The MAX78000 Evaluation Kit is displayed. This is a development platform for AI application demonstrations. This device can be used for applications such as audio processing, object detection, object classification and time series data processing. Main components of this evaluation kit are; Arm Cortex M4 CPU, RISC-V Microcontroller

Core, Neural Network Accelerator, built-in microphone, camera module, TFT LCD Screen and Power Monitor (PMON) with a seperate LCD screen.

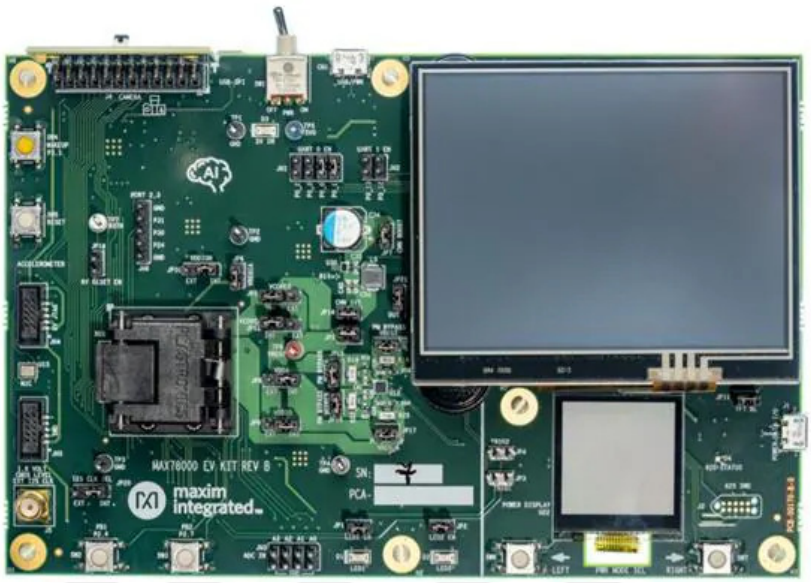


Figure 1.1. Image of MAX78000 Dev Kit

1.2.1.1 MCU Architecture

The architecture of the MAX78000 MCU is designed for optimal performance and low power consumption while enabling AI applications. The primary components of this architecture are given below.

Arm Cortex M4 CPU

MAX78000 uses Arm Cortex M4 CPU for its applications. Alongside this, RISC-V microcontroller core is used for handling general processing tasks and managing peripheral devices of the MAX78000 Evaluation Kit.

Neural Network Accelerator

A key component of the MAX78000 is the neural network accelerator, which is used for Convolutional Neural Network applications. This accelerator is used for real-time processing of complex neural networks.

Memory

MAX78000 has two different memory types, SRAM and flash memory. The neural network accelerator engine uses SRAM for temporary data storage during real-time operations, whereas the flash memory stores the program code and AI models.

Peripheral Interfaces

To enable the communication between microprocessors and external sensors and other devices, the MAX78000 is equipped with various peripheral interfaces, including I^2S (Inter-Integrated Circuit Sound), SPI (Serial Peripheral Interface Bus), UART (Universal Asynchronous Receiver-Transmitter), and GPIO (General-Purpose Input/Output).

1.3. Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a machine learning model type. Main applications for this model are; image processing and recognition, object detection and audio classification. A CNN model has different layer types, such as input layer, convolutional layer, dropout layer and flatten layer. In the input layer, data will be fed to the rest of the network. Rest of these layer types is outlined below.

1.3.1 Convolution Layer

The convolution layer is the core building block of a CNN. It is a set of learnable filters (or kernels) that is applied to the input data to create feature maps. Each filter activates certain patterns in the input, allowing the network to detect patterns, textures, and more complex features in deeper layers. There are 1D, 2D and 3D convolutional layers, in this thesis 1D and 2D convolutional layers are used. 1D convolutional layers are generally used for time-series or audio data applications while 2D convolutional layers are generally used for image processing tasks. Fig. 1.2 displays example 1D and 2D CNN models.

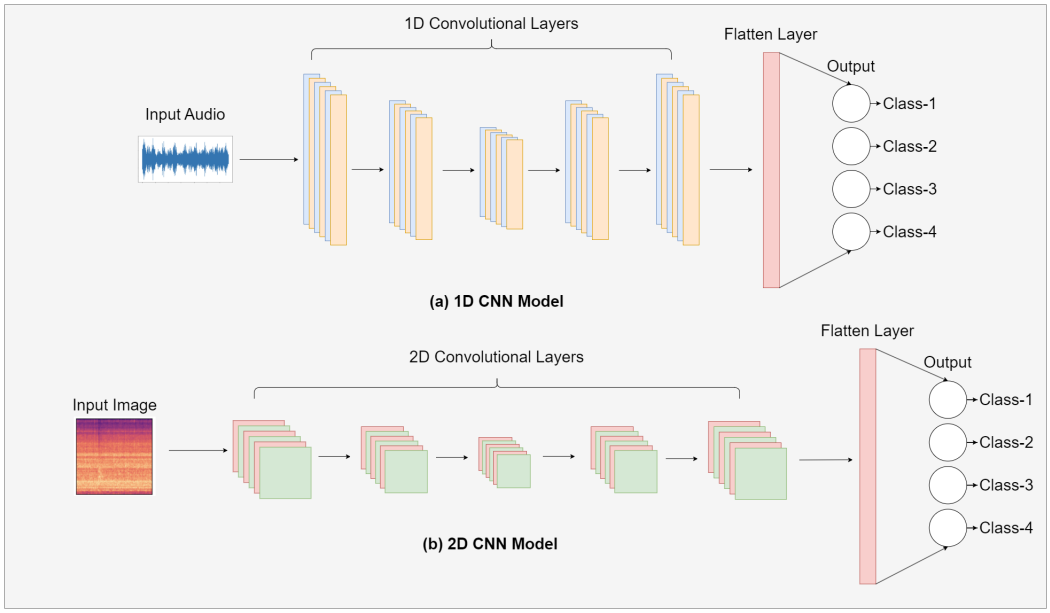


Figure 1.2. 1D and 2D Convolution Layers

In Fig. 1.2 (a), 1D CNN model is displayed. In Fig. 1.2 (b), 2D CNN model is displayed. In this figure, we see that in 1D model, convolutional layers are 1 dimentional while in 2D model, convolutional layers are 2 dimentional. They perform 1D and 2D convolutions to the input to extract features.

1.3.2 Dropout Layer

The dropout layer is a technique that is used to prevent overfitting in convolutional neural networks. During the training, according to the set dropout parameter, this layer drops the neurons randomly. This technique forces the network to learn redundant representations of the data, which improves its generalization capabilities and reduces the chance of overfitting to the training data. In order to show this technique, Fig. 1.3 is displayed. Fig. 1.3 (a) shows the neural network architecture without dropout layer applied and Fig. 1.3 (b) shows the neural network architecture with dropout layer applied with 0.5 probability.

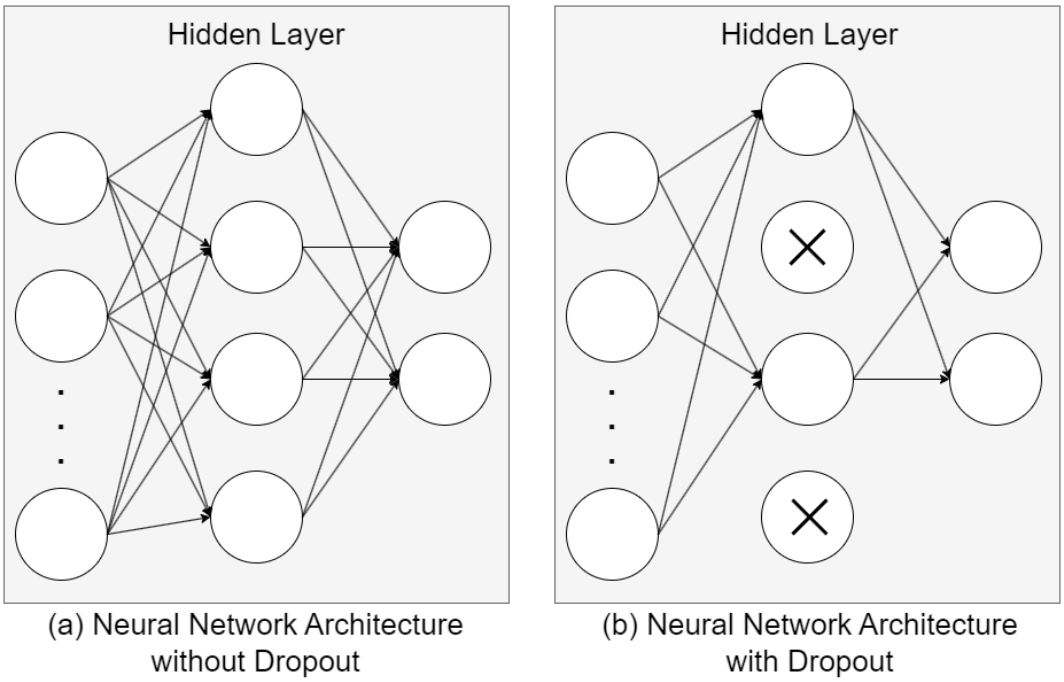


Figure 1.3. Depiction of Dropout Layer in Usage

1.3.3 Flatten Layer

The flatten layer is a layer that is used to convert multi-dimensional output of convolutional layers into a one-dimensional vector. This vector can then be fed into fully connected layers for classification tasks. Flattening is essential for transitioning from the convolutional layers to the dense layers, which require a fixed-size input.

1.3.4 Performance Metrics

Performance metrics are tools used to evaluate the accuracy and effectiveness of learning models. They offer a quantitative assessment of a machine learning model's performance. These metrics provide information about model's strength and weaknesses. In this thesis, Cross-Entropy Loss, Confusion Matrix and Top-1 Accuracy metrics are used to measure performances of provided CNN models.

1.3.4.1 Top-1 Accuracy

Evaluating the performance of a classification model involves several important metrics, each providing unique insights into different aspects of the model's effectiveness. One such crucial metric is Top-1 accuracy, which measures how often the model's top prediction is correct. To gain a comprehensive understanding of a model's performance, it is essential to assess Top-1 accuracy across different datasets, including the test, training, and validation datasets.

Top-1 Test Accuracy

Top-1 test accuracy is a performance metric commonly used in classification tasks to evaluate the effectiveness of a predictive model. It is defined as the proportion of instances where the model's highest-probability prediction (i.e., the class with the highest predicted probability) correctly matches the true class label. Mathematically, Top-1 test accuracy can be expressed as:

$$Top-1\ Test\ Accuracy = \frac{Number\ of\ Correct\ Top\ Predictions}{Total\ Number\ of\ Predictions} \times 100\% \quad (1.1)$$

where:

- **Number of Correct Top Predictions** is the count of instances where the model's top prediction matches the true class label.
- **Total Number of Predictions** is the total count of instances evaluated.

This metric provides a straightforward measure of how often the model's most confident prediction is accurate on a test dataset, making it a crucial indicator of the model's overall classification performance. A high Top-1 test accuracy indicates that the model frequently identifies the correct class as its top choice, reflecting strong predictive capabilities.

In addition to the Top-1 test accuracy, it is essential to evaluate performance of the model on the training dataset and the validation dataset. This distinction provides insights into

how well the model has learned from the training data and how well it generalizes to new, unseen data.

Top-1 Train Accuracy

Top-1 train accuracy is a performance metric that measures the proportion of instances in the training dataset where the model's highest-probability prediction matches the true class label. It is calculated for each training epoch as follows:

$$Top-1\ Training\ Accuracy = \frac{\begin{array}{c}Number\ of\ Correct\ Top\ Predictions\\ on\ Training\ Data\end{array}}{Total\ Number\ of\ Training\ Instances} \times 100\% \quad (1.2)$$

This metric provides insight into how well the model has learned from the training data, indicating the model's ability to accurately classify the instances it was trained on. A high Top-1 train accuracy suggests that the model has effectively captured the patterns and relationships within the training dataset.

Top-1 Validation Accuracy

Top-1 validation accuracy is a performance metric that measures the proportion of instances in the validation dataset where the model's highest-probability prediction matches the true class label. It is calculated for each training epoch as follows:

$$Top-1\ Validation\ Accuracy = \frac{\begin{array}{c}Number\ of\ Correct\ Top\ Predictions\\ on\ Validation\ Data\end{array}}{Total\ Number\ of\ Validation\ Instances} \times 100\% \quad (1.3)$$

This metric evaluates the model's generalization ability, reflecting how well the model performs on unseen data that was not used during training. A high Top-1 validation accuracy indicates that the model can generalize its learning to new, unseen instances, making it a crucial metric for assessing the model's real-world performance.

1.3.4.2 Cross-Entropy Loss

Cross-entropy loss, often referred to as log loss, is a crucial metric in the field of machine learning, particularly for classification tasks. It measures the performance of a classification model whose output is a probability value between 0 and 1. The goal is to minimize this loss, which in turn maximizes the likelihood of the model's predictions aligning with the true labels. Understanding cross-entropy loss is essential for evaluating

and improving models during training, validation, and testing phases. In the context of Convolutional Neural Networks (CNNs) used for image classification, cross-entropy loss plays a pivotal role in guiding the model to make accurate predictions.

Cross-Entropy Loss for Train Data (Train Loss)

Train loss refers to the cross-entropy loss calculated on the training dataset. It is an indicator of how well the model is learning from the training data. During training, the model adjusts its parameters to minimize this loss, thus improving its ability to classify the training samples correctly. Train loss is calculated at each epoch to monitor the model's learning progress.

For a binary classification problem, the cross-entropy loss for a single training example is given by:

$$L_{train} = -[y_{train}\, log(p_{train}) + (1 - y_{train})\, log(1 - p_{train})] \tag{1.4}$$

For multi-class classification, the loss is:

$$L_{train} = -\sum_{i=1}^{C} y_{train,i}\, log(p_{train,i}) \tag{1.5}$$

where C is the number of classes, $y_{train,i}$ is a binary indicator (0 or 1) if class label i is the correct classification for the current input, and $p_{train,i}$ is the predicted probability of the input being in class i.

Train loss is critical as it indicates the model's learning progress and helps in adjusting hyperparameters and model architecture. It is essential for diagnosing overfitting or underfitting by comparing it with validation loss.

Cross-Entropy Loss for Validation Data (Validation Loss)

Validation loss is the cross-entropy loss calculated on the validation dataset. This dataset is separate from the training data and is used to evaluate the model's performance during training without influencing the training process. Like train loss, validation loss is calculated at each epoch to monitor the model's performance on unseen data.

The formula for validation loss is the same as for train loss, but it is computed on the validation dataset:

$$L_{val} = -\sum_{i=1}^{C} y_{val,i}\, log(p_{val,i}) \tag{1.6}$$

Validation loss provides an unbiased evaluation of the model, helps in early stopping by monitoring when the validation loss stops decreasing, and assists in hyperparameter tuning and model selection.

Cross-Entropy Loss for Test Data (Test Loss)

Test loss is the cross-entropy loss calculated on the test dataset, which is used to evaluate the final model's performance. Unlike the training and validation phases, the test phase uses a separate dataset that the model has never seen before.

The formula for test loss is similar to the previous ones but computed on the test dataset:

$$L_{test} = -\sum_{i=1}^{C} y_{test,i}\, log(p_{test,i}) \tag{1.7}$$

Test loss provides a final measure of model performance, ensures the model's generalization capability to new, unseen data, and is important for reporting the model's accuracy and reliability.

In CNNs, which are widely used for image classification, cross-entropy loss is particularly important. It guides the convolutional layers to learn relevant features from images, ensures that the model's predictions are as close as possible to the true labels, and improves classification accuracy. Monitoring and minimizing cross-entropy loss during training, validation, and testing is crucial for developing CNN models that accurately classify images, leading to robust and reliable performance in real-world applications.

1.3.4.3 Confusion Matrix

A confusion matrix is a performance evaluation tool for detecting the performance of classification models generally used in multi-class detection applications. It provides a detailed breakdown of model predictions compared to actual class labels, allowing for a comprehensive analysis of classification accuracy and errors. The confusion matrix is structured as a square where the squares length is determined by the total class count. In a confusion matrix, rows represent actual classes and columns represent predicted classes. The diagonal line of confusion matrix displays the true predicted classes while off-diagonal boxes show misclassified predictions.

In order to explain this performance evaluation tool, Fig. 1.4 is displayed. In this figure, a 4x4 confusion matrix of a model with 4 classes for multi-class classification is shown. For classes, A, B, C and D, the diagonal line shows the true predictions while the off-diagonal elements show the misclassified predictions. For example, if actual class is A and it is misclassified as C, this indicates a misclassification. Confusion matrixes are often colored as heatmap to show the prediction counts of models.

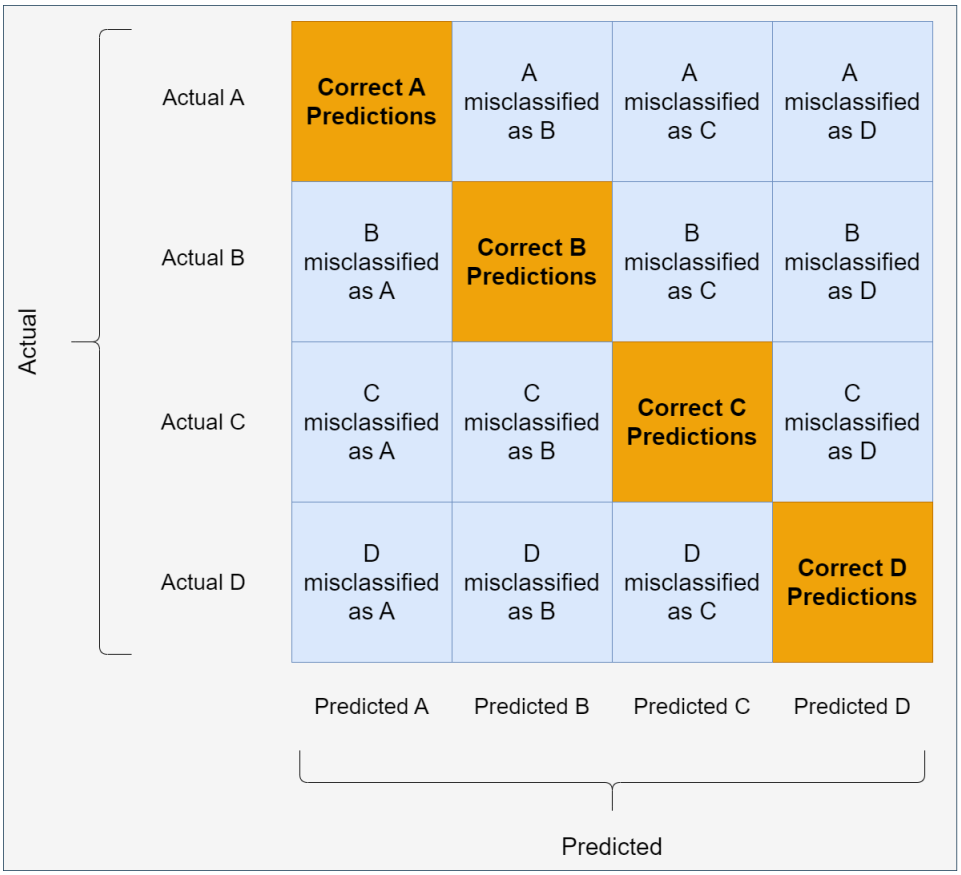


Figure 1.4. 4x4 Confusion Matrix Structure

Understanding the distribution of misclassifications is vital for improving the model's performance. By analyzing false positives and false negatives, we can identify patterns and areas where the model struggles, thereby guiding targeted improvements. For instance, a high number of 'A misclassified as B' cases might suggest the need for more training data for class A or a review of features that distinguish class A from class B.

1.4. Related Work

In this section, we categorize the existing literature into three groups: (1) studies that utilize the MAX78000 MCU for machine learning applications that include classification and detection purposes in IoT environments; (2) studies that focus on TinyML within

the scope of anomaly detection for PdM applications using general-purpose MCUs; and (3) studies on fault detection using industrial machine operating data, without involving edge computing and TinyML purposes.

First, we contrast our work with articles that utilize MAX78000 MCU for machine learning applications that include classification and detection purposes in IoT environments. Moss et al. (2022) benchmark the MCU's performance in running Deep Neural Network (DNN) models on tasks such as image recognition, classification, and segmentation. Cooper et al. (2022) describe an autonomous sorting system for the real-time image classification and sorting of recyclables. The system leverages the MCU's CNN accelerator to classify objects, such as plastic, metal cans, and paper, and uses mechanical arms to sort them on a conveyor belt.

In the study of Büyüksolak & Güneş (2023), they introduce a visual odometry system implemented on the MAX78002 MCU (follow-on product of MAX78000), utilizing a neural network named TinyVO. The system utilizes the CNN accelerator to perform real-time motion estimation from video sequences. Cooper et al. (2021) explore the implementation of edge machine learning for face detection. The system performs real-time face detection and classification. Lightbody et al. (2022) propose a lightweight CNN based Intrusion Detection System (IDS) for IoT devices. The system detects anomalies in power consumption data.

Rüegg et al. (2023) present KP2Dtiny, a quantized neural network for keypoint detection on the low-power MCU. Plozza et al. (2022) propose a low-power audio distortion circuit model using a quantized neural network deployed on the microcontroller. In the study of Moosmann et al. (2023) and Moosmann et al. (2024), they present TinyissimoYOLO, a CNN for object detection designed for deployment on low-power microcontrollers, including the MAX78000. In the study of Mnif et al. (2024), an edge computing approach is used for real-time hand gesture classification based on Electrical Impedance Tomography (EIT) measurements. Ingaleshwar et al. (2024) present an energy-efficient image classification system for real-time wildlife species recognition.

Ngo et al. (2023) propose a framework designed for network instrusion detection in IoT environments. The framework is designed for real-time anomaly detection on edge devices. In the study of Bakar et al. (2022), they present Protean, a computing platform that utilizes MAX78000 MCU as the primary processing unit. This work features a modular hardware design with an adaptive runtime system for plug-and-play features. Wang et al. (2022) introduce a real-time object detection system that uses CNN for image acquisition and object detection. In the study of Ulkar & Okman (2021), they design a keyword spotting (KWS) system by employing a CNN model trained on raw audio signals. Okman et al. (2022) introduce L3U-net, an image segmentation model. The study demonstrates handling high-resolution images by leveraging a data folding

technique for edge devices.

The references given in this category utilized the MAX78000 MCU to deploy energy-efficient, real-time machine learning applications, including image classification, object detection, intrusion detection, audio processing, and gesture recognition. In contrast, our work focuses specifically on utilizing the MAX78000 MCU for fault detection using machine sounds, highlighting its application in predictive maintenance for industrial equipment.

In the second category, we contrast our work against articles that use general-purpose MCUs and focuses on TinyML within the scope of anomaly detection for PdM applications. Anusha et al. (2024) present an edge machine learning-enabled predictive fault detection system for conveyor belt maintenance optimization. The system employs an ESP32 microcontroller and an accelerometer sensor to capture real-time vibration data, utilizing a four-layer Artificial Neural Network (ANN) model for predictive maintenance in industrial settings. Chen et al. (2023) introduce LOPdM, a predictive maintenance system that utilizes self-powered sensors. The system uses the ESP32 microcontroller and a piezoelectric sensor for real-time vibration monitoring, utilizing TinyML models for anomaly detection in industrial environments. In the study of Mihigo et al. (2022), they compare two predictive maintenance models, based on IoT, for real-time equipment health monitoring. The study uses the ESP32 microcontroller for on-device deployment to predict the remaining useful life (RUL) of industrial components. In the study of Athanasakis et al. (2022), they present a method based on TinyML for predicting the RUL of turbofan engines. This work benchmarks Long Short-Term Memory (LSTM) and CNN models on STM32F767ZI microcontroller. In the study of Antonini et al. (2022), an ESP32 microcontroller is utilized to detect anomalies such as unusual vibration patterns and temperature changes in industrial pumps in extreme industrial environments. Zhao & Wang (2024) introduce an edge computing technique for machine condition monitoring. The system uses an ESP32 microcontroller and a Microelectromechanical System Accelerometer (MEMS Accelerometer) to perform real-time vibration data processing and fault diagnosis for industrial motors. Njor et al. (2022) investigate techniques for optimizing TinyML-based predictive maintenance systems, focusing on both input and model aspects using the Arduino Nano 33 BLE Sense microcontroller.

References in this category perform TinyML-based predictive maintenance and fault detection by employing deep learning models such as LSTM and CNN using various microcontrollers, which have general-purpose memory designs and are not specifically designed for artificial intelligence (AI) applications. In contrast, in this work, we use MAX78000 MCU, which has a built-in neural network accelerator module designed for CNN applications. Our work benefits from the ultralow power consumption of MCUs during network inference and memory architecture, which are designed to handle large

models and datasets efficiently.

In the third category, we contrast our work against studies in which industrial machine sound classification is performed without involving the MAX78000 or TinyML purposes. Goundar et al. (2015) describe a real-time condition monitoring system that uses vibration and temperature data readings to predict motor bearing failures. The system employs piezoelectric sensors and a thermistor, interfacing with a Waspmote microcontroller for data acquisition, storage, and analysis via IoT. Talmoudi et al. (2019) utilize machine sound data to predict failures on an IoT system that employs vibration and sound sensors, storing and analyzing the data on a server to detect machine failures. The authors analyze the data by segmenting the raw sound data into time windows and applying Fast Fourier Transform (FFT). They use a correlation analysis to map the segments into a 2D space, clustering similar data points to detect anomalies and predict failures. Ganga & Ramachandran (2018) develop an IoT-based system for real-time monitoring of electrical machine conditions, using an IoT2040 Gateway to collect and transmit vibration data from DC motors, with analysis performed in the cloud using a statistical classification-based signal decomposition algorithm to determine vibration thresholds. In the study of Zhang et al. (2013), they develop a remote and online system for monitoring machine conditions and detecting faults using wireless sensor networks (WSNs) for motor systems. The system uses wireless vibration transmitters to continuously monitor motor status, with data being processed on a host computer using a system for fault diagnosis. In the study of Natesha & Guddeti (2021), they develop a machine fault monitoring system using fog computing with industrial controllers to process machine sound data. The system employs machine learning models, which are Support Vector Machine (SVM), Logistic Regression, and Random Forest to classify sounds as normal or abnormal based on Mel-Frequency Cepstral Coefficients (MFCC) features and Linear Predictive Coding (LPC). Yong & Nugroho (2022) develop an anomaly detection system for machine sounds using time-distributed deep learning models on Google Colab. The system combines CNN, Gated Recurrent Unit (GRU) and LSTM, to classify machine sounds based on features extracted using MFCC.

The references in this category focus on real-time condition monitoring and fault detection for industrial machines with statistical algorithms or machine learning models to predict and diagnose failures but rely on cloud or personal computer (PC) servers for data analysis and do not involve TinyML. In contrast, our work utilizes an ultra low-power MCU to analyze and compute data at the edge, using TinyML for low latency and improved energy efficiency. This approach eliminates the need for cloud or PC servers.

1.5. Contributions of this Thesis

The key technical challenges in designing state-of-the-art TinyML approaches are as follows. First, converting a convolutional Neural Network (CNN) model to work on TinyML platforms involves overcoming the limitations of processing capacity and memory constraints inherent in such devices. Second, machine wear classification difficulties arise owing to subtle differences in machine data, which are often very similar and require sophisticated techniques to distinguish effectively. Third, ensuring the robustness and accuracy of the model under varying operational conditions presents a challenge because industrial environments can introduce noise and other unpredictable factors that may affect the performance of the classification system. Addressing these challenges is crucial for successful implementation of TinyML solutions in real-world industrial applications.

The main contributions of this work to the state of the art approach are as follows:

- Our work successfully converts a CNN model to accelerate on the MAX78000 Microcontroller Unit (MCU) by processing audio data to fit within the input constraints of the device. The implementation utilizes optimized CNN functions to ensure efficient use of the microcontroller's limited processing capacity and memory, retaining all necessary data with minimum loss.
- We developed a system that effectively distinguishes between subtle differences in machine sounds by employing optimized CNN models, thereby enhancing the model's ability to accurately classify the wear rates of industrial peripherals on a MAX78000 MCU.
- Our implementation aims to ensure consistent performance in various industrial conditions by integrating the real-time processing capabilities of TinyML applications, thereby potentially improving the accuracy and dependability of the sound classification system in diverse operational environments.

1.6. Outline of this Thesis

The remainder of this thesis is organized as follows: In Chapter 2, we present our Real-Time Fault Detection by Industrial Machine Sounds Using a Low- Power Microcontroller work. First, we present our dataset and implementation process, which consists of our CNN models and training process. Second, we benchmark our training and validation results of different CNN models. Then, we evaluate these models, benchmark the evaluation results and present the real-life demonstration our work. In Chapter 3, we present our conclusions of this thesis, alongside our future work plan.

2. CHAPTER: REAL-TIME FAULT DETECTION BY INDUSTRIAL MACHINE SOUNDS USING A LOW-POWER MICROCONTROLLER

2.1. Introduction

The demand for efficient and real-time monitoring solutions for industrial applications is increasing rapidly. One common problem is the difficulty in accurately diagnosing the conditions of the industrial peripherals. Traditional approaches often involve manual inspections or bulky monitoring systems that lack real-time analysis and early fault detection capabilities. These methods are not only time-consuming but also prone to human error. Predictive maintenance (PdM) is an advanced strategy that addresses these challenges by predicting equipment failures before they occur, thereby minimizing downtime and maintenance costs. This approach relies on continuous monitoring and data analysis to detect signs of wear and degradation early on. To detect these signs of wear, various data such as, audio recordings of machine sounds and images of machine degredations can be used.

Machine learning approaches are becoming widespread in modern PdM applications. These applications offer some improvements but typically require significant computational resources and are not always feasible for on-site, real-time applications. Therefore, a robust and low-power solution for these applications is required. Hence, Integrating predictive maintenance with state-of-the-art technologies such as Tiny Machine Learning (TinyML) offers a promising solution. TinyML integrates machine learning algorithms within resource-constrained environments, allowing real-time data processing and decision-making at the edge.

State-of-the-art approaches in TinyML offer several advantages over traditional and conventional machine learning methods. First, TinyML models are highly efficient (Abadade et al., 2023). Hence, it enables machine learning deployment on microcontrollers with low power usage. The MAX78000 microcontroller, with its efficient neural network capabilities and low power consumption, enhances this integration by providing a robust platform for implementing TinyML (Giordano et al., 2022). This combination allows for efficient and timely diagnosis, ensuring that maintenance can be performed proactively rather than reactively. The monitoring system can operate continuously and autonomously, providing accurate insights into the conditions of

industrial peripherals. Second, TinyML systems are cost-effective, reducing the need for expensive and bulky hardware traditionally required for machine-learning tasks. Third, TinyML enhances reliability by minimizing data transfer requirements, which not only speeds up processing, but also reduces potential points of failure associated with data transmission. For example, implementing a TinyML-based sound classification system for machine blade monitoring allows for continuous real-time diagnostics directly on the device, leading to timely maintenance and reduced downtime. These advantages underscore the transformative potential of TinyML for advancing industrial monitoring and diagnostic applications.

The developed algorithm processes audio data from trial peripherals in real time, allowing for the immediate classification of wear rates into four classes: Base, Broken, Fresh, and Moderate. These classes represent different conditions of the equipment, ranging from idle operation (base) to varying degrees of wear and damage (Broken, Fresh, Moderate). Using CNNs, the system captures essential features from audio signals, ensuring accurate classification. Integration with MAX78000 ensures that the model operates efficiently within the device's constraints, providing a practical solution for on-site monitoring and diagnostics.

The key novelty of this work lies in the effective real-time classification of audio from industrial peripherals using a power-efficient MAX78000 MCU. By operating on the edge, this approach significantly reduces time and power costs while addressing the challenging task of distinguishing subtle differences in machine sounds, thereby providing a practical solution for on-site monitoring and diagnostics.

The remainder of this chapter is organized as follows: in Section 2.2, we present our implementation and background steps, in Section 2.3, we present our results, in Section 2.4, we present our summary for this chapter.

2.2. BACKGROUND AND IMPLEMENTATION

In this section, we first describe the machine audio dataset used for the fault detection task, and we present our preprocessing steps for the dataset. Second, we provide the details of the training and network models. Third, we describe the implementation steps of MAX78000 MCU.

2.2.1 Dataset and Preprocessing

The "*Cutting Tool Wear Audio Dataset*" consists of 10-second wav files with a sample rate of 48,000 Hz (Soni et al., 2023). The dataset originally includes audio recordings of two different rpm values, 520 and 635. In this work, 520 rpm values are utilized. The

classes within this dataset are Base, Broken, Fresh, and Moderate. These audio files capture the sounds of an end mill cutter: "Base" represents the sound of the machine without cutting, "Fresh" indicates a sharp tool, "Moderate" denotes a slightly blunt tool, and "Broken" signifies a completely worn-out cutting tool. To accommodate the limitations of the MAX78000 MCU, the labeled files are divided into 1-second wav files, generating a total sample count of 7,065. In Fig. 2.1 one sample from each of the four labels is displayed as a) Base, b) Broken, c) Fresh and d) Moderate. In this figure, x axis represents time (in seconds) and y axis represents amplitude.

The original stereo audio files are converted to single-channel (mono) audio to fit the MAX78000's memory constraints. This conversion produces audio files shaped as 1x16,384. While this format is suitable for training on platforms like Google Colab, the MAX78000 microcontroller has an upper limit of 1,024 data points. To address this limitation, the audio files are reshaped to 128x128, ensuring minimal data loss.

After reshaping, the dataset was saved in a new format to ensure accurate data processing. The dataset was then split into training, validation, and testing subsets with an 80-10-10 distribution. Note that this split was randomly performed. These subsets are saved again to finalize the dataset preparation for the subsequent training and evaluation steps.

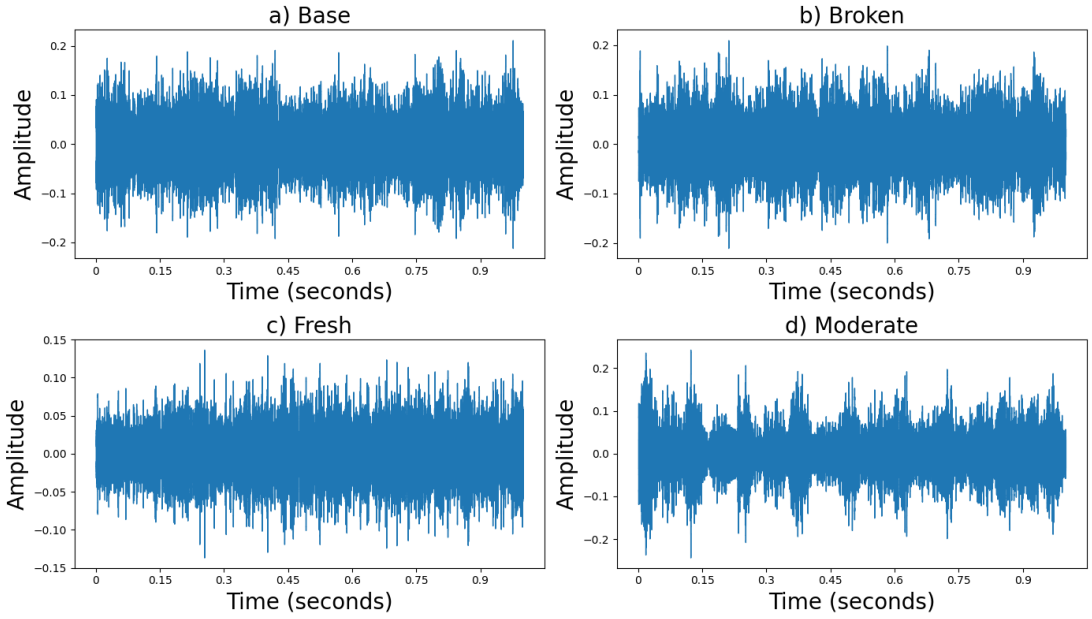


Figure 2.1. Audio Visualization of Classes (a) Base, (b) Broken, (c) Fresh and (d) Moderate

2.2.2 Training and Network

In this chapter, the CNN model and data loader were built using the ai8x library, which is a modified version of PyTorch to work on MAX78000 and MAX78002 MCUs. The

data loader was configured to feed the CNN with audio files prior to training. The created data loader includes built-in functions to apply data augmentation techniques to reduce overfitting by increasing data size. These techniques include time shifting and the addition of dynamic Gaussian noise. Thus, the size of the original dataset tripled. In addition, normalization and quantization were applied to the audio files before they were fed into the CNN. This normalization process, which is performed using the ai8x library, ensures that the data fit the requirements of the MCU. The data must be normalized to the expected range of the MCU, which is [-128, 127], for both the evaluation of simulated hardware and the actual deployment on the device.

2.2.2.1 Model Network

In this thesis, 1D convolution (Conv1D), 2D convolution (Conv2D) and dropout layers are used. An example architecture of 1D models[1] are illustrated in Fig. 2.2. In this figure, first, a 1D convolutional layer is applied to the input. Second, this layer is followed by a dropout layer. This process is repeated depending on the number of hidden layers (HL) until the final 1D convolutional layer. Third, a flatten layer is connected to the output of final 1D convolutional layer. In the output of the flatten layer, we see the audio classes for our model. Since our work involves audio data, the network model is kept relatively simple, without increasing complexity. Originally, this model was designed with separate layers and activation functions, but due to the memory limitations of the MCU and the ai8x training library, fused functions are used. Fused functions combine convolutional layers with their associated activation functions (Rectified Linear Unit (ReLU)) to address these limitations.

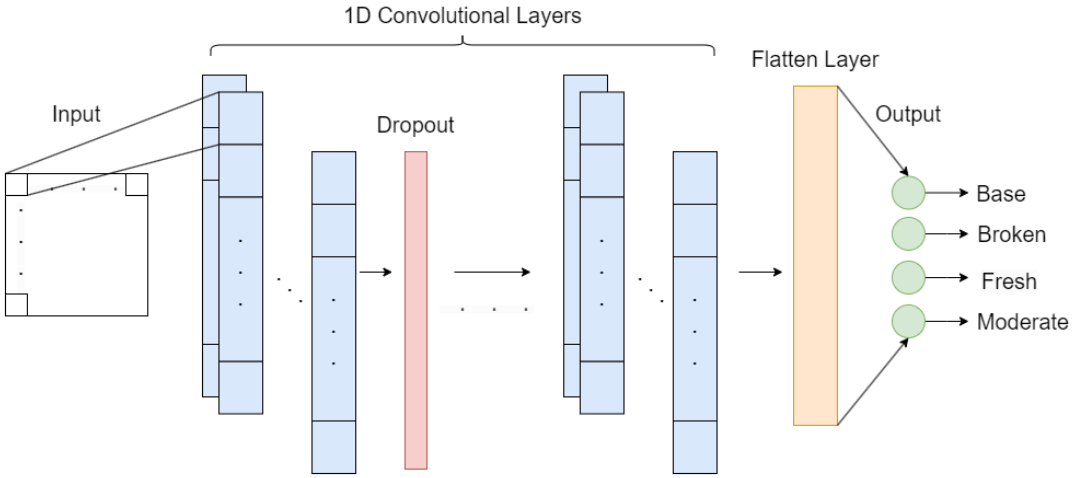


Figure 2.2. 1D CNN Model Architecture

[1]For 2D models, convolutional layer filters are two dimentional.

2.2.2.2 Training Process

In the training process, Quantization Aware Training (QAT) is utilized to reduce the size of the model. QAT is the process of simulating hardware behavior during the training and evaluation stages of ai8x training and synthesis implementation. In QAT, after a certain number of epochs, the weights are quantized to 8-bits. This step is crucial for hardware implementations because the weights must be quantized to 8-bits to be compatible with the hardware requirements.

2.2.3 Implementation on MAX78000 MCU

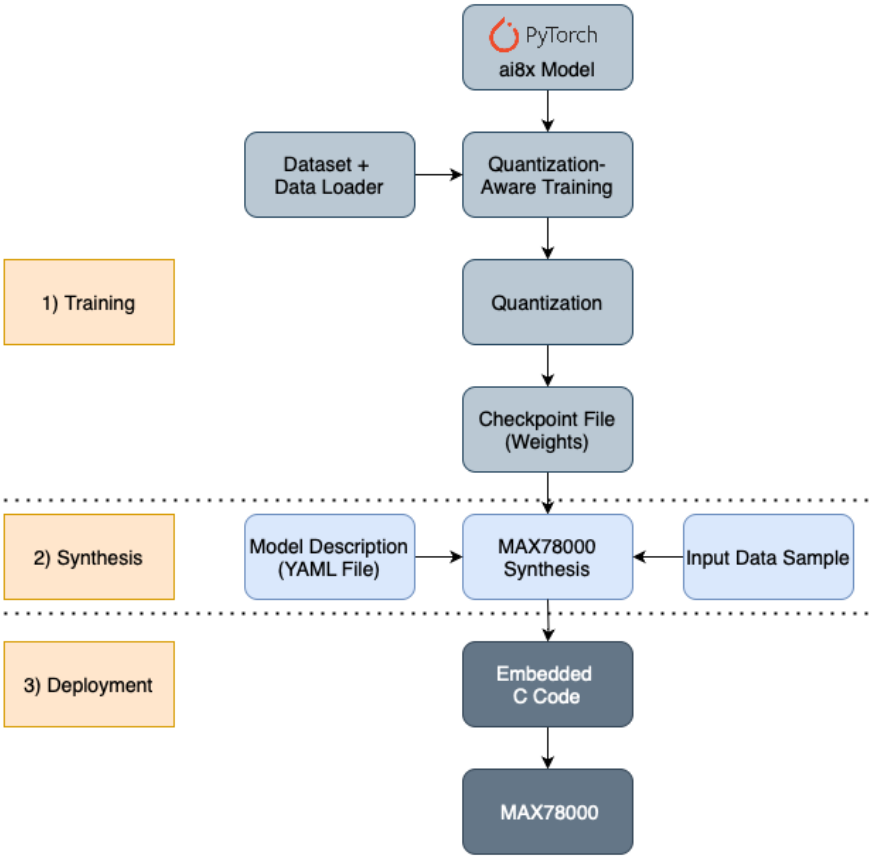


Figure 2.3. Implementation Flowchart of Classification Demo

In this section, we present the implementation steps of our study on the MAX78000 MCU. To this end, Fig. 2.3 is shown. In this figure, we first show the training blocks. In this block, the data loader feeds the dataset to the CNN model to begin the QAT. After training, the obtained weights were quantized to 8-bits. Second, we present the synthesis block, which is an intermediate step for running the CNN model on hardware. In this block, a model description file that configures the CNN model for C++ (YAML File) and the generated 8-bit weights are utilized by ai8x synthesis functions to generate a C++

file that contains the weights and model description. Finally, we present a deployment block. In this block, the generated C++ file and the main working algorithm of our application are configured and uploaded to MAX78000.

Fig. 2.4 provides an in depth explanation of the deployment block shown in Fig. 2.3. This figure illustrates a single loop of audio classification performed on the MAX78000 MCU. The process begins after the hardware detects audio input. In this figure, in Block A, first, audio is recorded using an Inter-IC Sound (I^2S) microphone. Second, a high-pass filter is applied to remove the DC offset of the microphone and audio is converted to single-channel. The processed audio is then fed into the Neural Network Accelerator (Block B), where the CNN weights are first loaded and then used to detect the class of audio input. The detected class and the input waveform of audio are shown on a separate display. The implementation of the CNN and the operation of the MAX78000 MCU were coded in C++ using the Eclipse IDE (Maxim SDK).

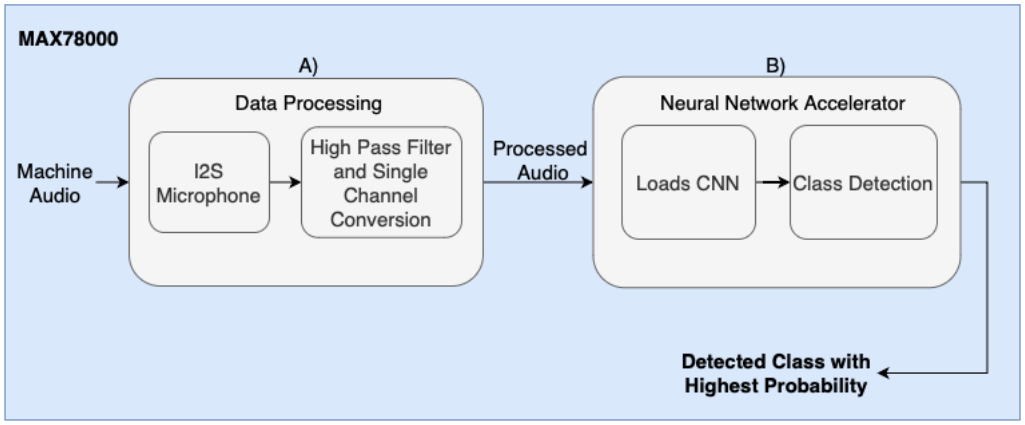


Figure 2.4. Working Principle of Demo on Hardware

Before deploying the generated weights and CNN model on the MAX78000 MCU, it is essential to optimize the CNN layers to enhance hardware efficiency and reduce evaluation time in practical applications. This optimization involves adjusting the processor count for each CNN layer. To implement the code in a real-world scenario, samples from the test dataset and generated weights must be used to initialize the hardware. This was achieved by configuring a YAML network file for hardware usage. The inputs of this YAML file are detailed definitions of CNN layers with related activation functions, processor counts for each layer, and generated 8-bit weights.

The detailed architecture of the YAML network is displayed in Fig. 2.5. Configuration of YAML network requires a series of steps. First, the CNN model architecture and the data loader must be configured to ensure that the structure of the network and the method for loading data are correctly defined. This involves specifying the layers, operations,

and parameters for each layer, such as the type of activation function, kernel size, and pooling. Second, for each layer in the network, the activation functions must be defined. Third, processor count must be configured to fit the according layer's output neuron count.

```yaml
---
arch: MachineAudioClass_v2
dataset: AudioClass

layers:
  # 1
  - pad: 0
    activate: ReLU
    out_offset: 0x2000
    processors: 0xffffffffffffffff
    data_format: HWC
    operation: Conv1d
    kernel_size: 1
  # 2
  - max_pool: 2
    pool_stride: 2
    pad: 1
    activate: ReLU
    out_offset: 0x0000
    processors: 0x000fffffffffffff
    operation: Conv1d
    kernel_size: 3
  # 3
  - max_pool: 2
    pool_stride: 2
    pad: 1
    activate: ReLU
    out_offset: 0x2000
    processors: 0x0000fffffffffffff
    operation: Conv1d
    kernel_size: 6
  # 4
  - avg_pool: 2
    pool_stride: 2
    pad: 1
    activate: ReLU
    out_offset: 0x0000
    processors: 0xffffffffffffffff
    operation: Conv1d
    kernel_size: 6
  # 5
  - flatten: true
    out_offset: 0x2000
    processors: 0xffffffffffffffff
    operation: MLP
    output_width: 32
    activate: None
```

Figure 2.5. YAML Network of 1D-CNN 5 HL Model

The network includes convolutional layers with specified kernel sizes, pooling layers with defined strides, and a flattening layer before the final multi-layer perceptron (MLP) layer. This comprehensive setup ensures that the network can process the input data effectively and perform the desired machine learning tasks accurately.

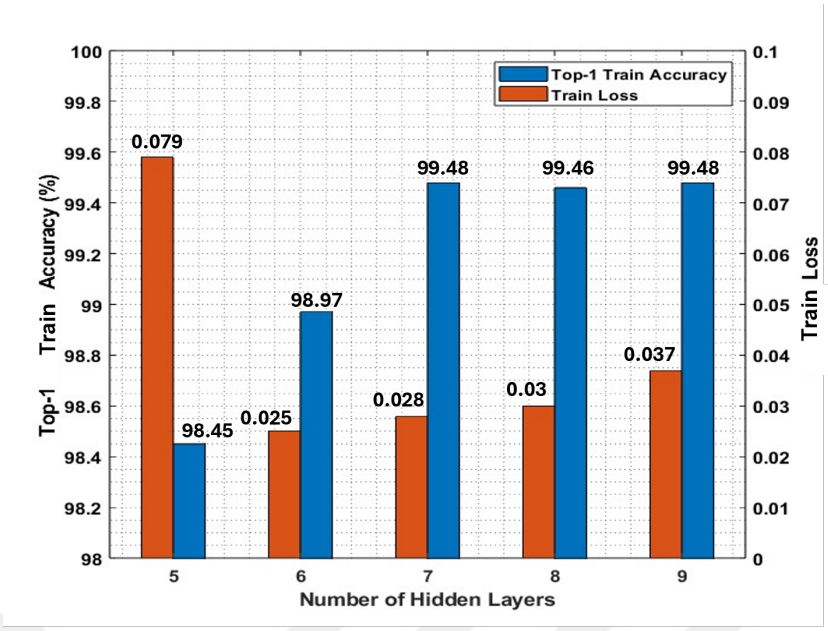
2.3. Results

In this section, we first present training results. Second, we evaluated the performance of the trained CNN model using hardware-based simulations. Third, we present the results of real-life application demonstration.

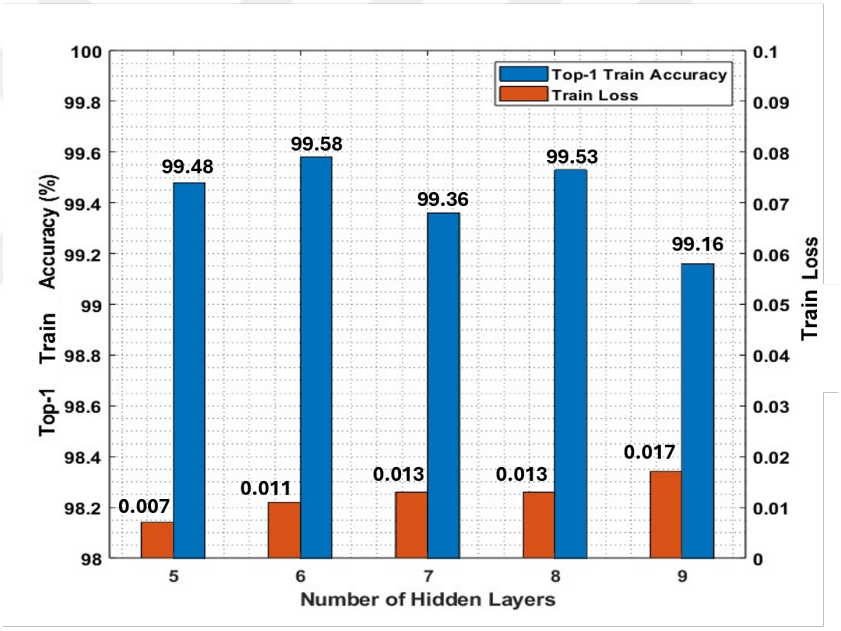
2.3.1 Training Results

In this section, we first determine the optimal CNN model for our study and then compare their performance metrics. An exhaustive search algorithm is used to determine the optimal HL count. For 1D CNN models, the HL parameter interval was set between 5 and 9. For 2D CNN models, the same approach was followed to determine the optimal HL counts. Exhaustive search algorithm was used to determine HL counts. The HL counts identified by this algorithm were then used to train the models. Training parameters are determined as follows: batch size set to 128, number of training epochs is set to 200, QAT start epoch is set to 35, Adam optimizer is utilized, the initial value of the adaptive learning rate is set to 0.001 to improve learning and to reduce training and validation losses. Epoch and batch size parameters rely on training machine's hardware availabilities. This training is done on AMD Ryzen 5600x CPU, but if any CUDA cores are available, training process will take a significantly lower time to finish. Epoch count was set to 200 because models models did not show significant improvement in terms of validation accuracy and validation loss after 200^{th} epoch. After determining the optimal hidden layer count, another exhaustive search algorithm was applied to find the optimal QAT start epoch. The interval for this search was set between 5 and 50, and the optimal QAT start epoch was found to be 35. In each training epoch, top-1 training accuracy, training loss, top-1 validation accuracy and validation losses are calculated.

In order to compare the top-1 train accuracy and train loss of each of 1D and 2D CNN models, Fig. 2.6 is displayed. In this figure, the x axis shows number of hidden layers, left y axis shows top-1 train accuracy percentages and right y axis shows train loss. Top-1 train accuracy is defined as the percentage of correct predictions made by the model on the training dataset. Train loss is calculated as Cross Entropy Loss of each model. These metrics are obtained by training these 5 different models for 200 epochs. These bars show the highest train accuracy and lowest train loss value of each model out of 200 epochs.



(a) 1D – CNN Top-1 Train Accuracy vs. Train Loss



(b) 2D – CNN Top-1 Train Accuracy vs. Train Loss

Figure 2.6. Top-1 Train Accuracy and Train Loss of 1D CNN Models and 2D CNN Models

In Fig. 2.6 (a), 1D-CNN Top-1 Train Accuracy vs. Train Loss as a function of number of hidden layers is displayed. In this figure, each blue bar represents the top-1 train accuracy of corresponding 1D CNN model and each orange bar represents train loss of corresponding 1D CNN model. In this figure, first, we see that, train loss increases as the hidden layer count increases, except for the 1D model with 5 HL. Second, we observe that, increasing the hidden layer count (between $[5, 7]$) results in an increase in

top-1 train accuracy. However, after the model with 7 HL, adding more HLs does not have an impact on top-1 train accuracy. This shows that, performance of the model can not be further improved as the complexity of the model increases over a saturation point.

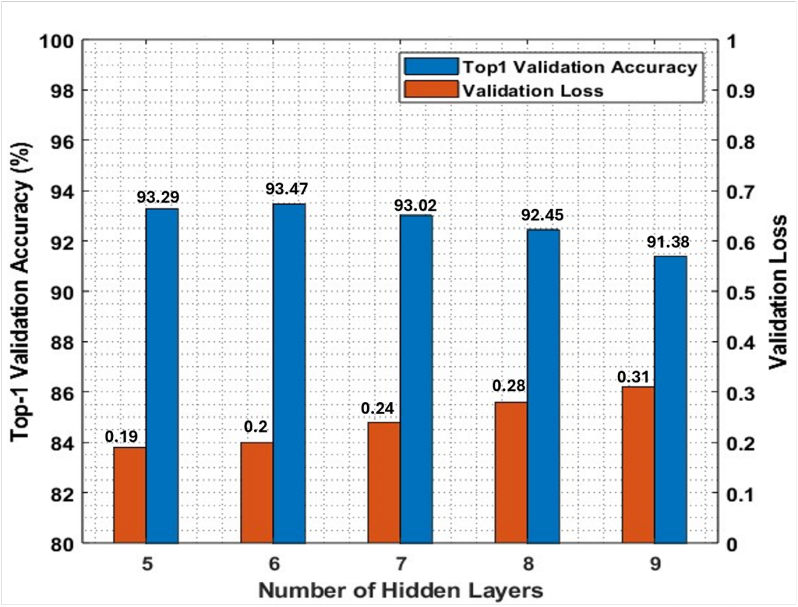
In 2.6 (b) 2D-CNN Top-1 Train Accuracy vs. Train Loss as a function of number of hidden layers is displayed. In this figure, each blue bar represents the top-1 train accuracy of corresponding 2D CNN model and each orange bar represents train loss of corresponding 2D CNN model. In this figure, we see that, as the hidden layer count increases, overall validation loss increases, whereas validation accuracy is not affected significantly. First, we observe that, different from Fig. 2.6 (a), top-1 train accuracy values does not follow a trend in increase or decrease as the HL count increases. Second, we see that the maximum top-1 value is achieved at model with 6 HL(99.58%) and minimum top-1 value is achieved at model with 9 HL(99.16%).

In order to compare the training times and number of epochs required for these 1D and 2D CNN models to reach their highest top-1 validation score, Table 2.1 is given. In this table, we see that training with 5 HLs takes approximately 1 hour while training with 9 HLs takes 1.5 hours for 1D CNN. Training for 2D CNN models takes 1.4 hours for 5HL and 2.3 for 9 HL. For 2D models, the top-1 validation accuracy typically stabilizes at an earlier epoch compared to 1D models, suggesting that 2D models may possess a comparatively lower learning capacity.

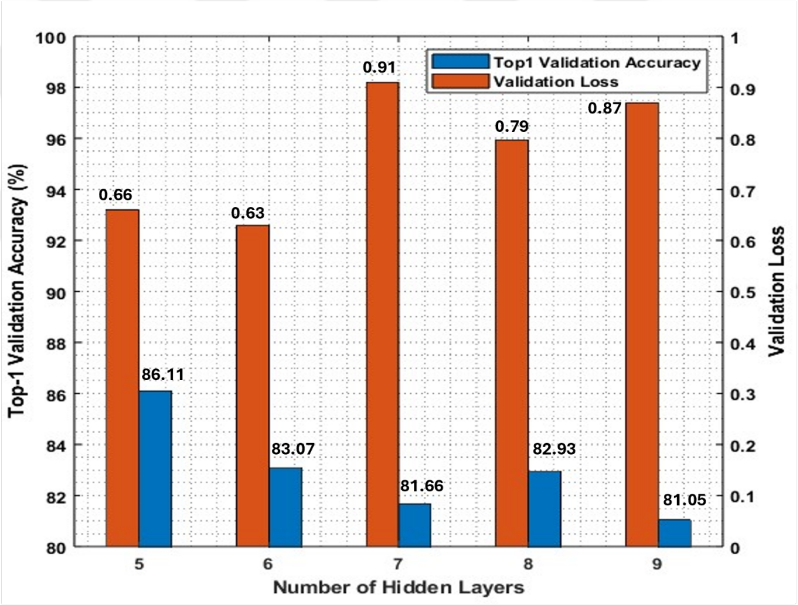
Table 2.1. Number of Epochs Required for Different Models to Reach Their Highest Top-1 Score

	Model	Training Time	Top1 Accuracy Achieved @Epoch
1D	5 HL	1.06 hr	184
	6 HL	1.25 hr	180
	7 HL	1.34 hr	192
	8 HL	1.39 hr	199
	9 HL	1.52 hr	155
2D	5 HL	1.41 hr	176
	6 HL	1.99 hr	37
	7 HL	2.09	192
	8 HL	3.26 hr	145
	9 HL	2.33 hr	127

In order to compare the top-1 validation accuracy and validation loss of each of 1D and 2D CNN models to examine the generalization ability of these models, Fig. 2.7 is displayed. In this figure, the x axis shows number of hidden layers, left y axis shows top-1 validation accuracy percentages and right y axis shows validation loss. Top-1 validation accuracy is defined as the percentage of correct predictions made by the model on the validation dataset. Validation loss is calculated as Cross Entropy Loss of each model.



(a) 1D – CNN Top-1 Validation Accuracy vs. Validation Loss



(b) 2D – CNN Top-1 Validation Accuracy vs. Validation Loss

Figure 2.7. Top-1 Validation Accuracy and Validation Loss of 1D and 2D CNN Models

In Fig. 2.7 (a), 1D-CNN Top-1 Validation Accuracy vs. Validation Loss as a function of number of hidden layers is displayed. In this figure, each blue bar represents the top-1 validation accuracy of corresponding 1D CNN model and each orange bar represents validation loss of corresponding 1D CNN model. In Fig. 2.7 (a), we see that, as the hidden layer count increases, overall validation loss increases, whereas validation accuracy decreases. The reason is that adding more hidden layers increases models complexity, hence it results in decrease of top-1 validation accuracy and increase of

validation loss.

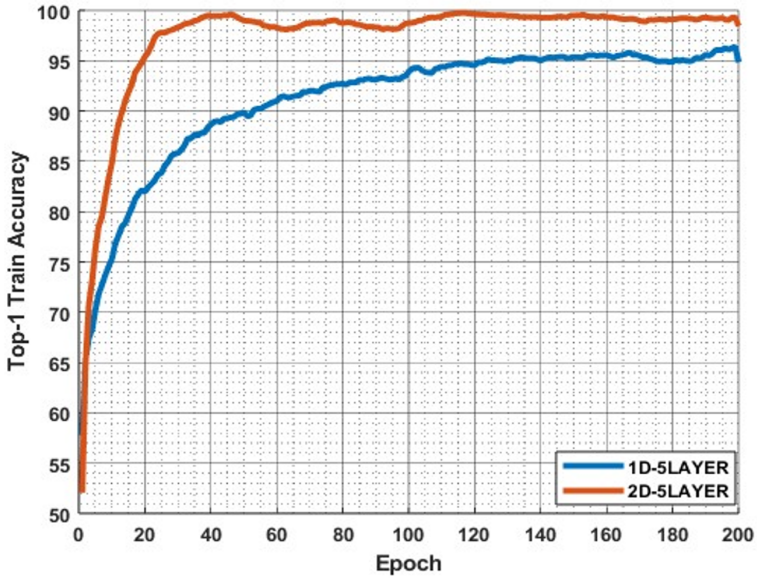
In Fig. 2.7 (b) 2D-CNN Top-1 Validation Accuracy vs. Validation Loss as a function of number of hidden layers is displayed. In this figure, each blue bar represents the top-1 validation accuracy of corresponding 2D CNN model and each orange bar represents validation loss of corresponding 2D CNN model. First, we see that, compared to Fig. 2.7 (a), validation losses of all models have increased by 0.53 on average. Second, we observe that top-1 validation accuracies of all models have decreased by 9.75% on average. These results show that, for this task, 1D CNN models outperforms 2D CNN models in terms of top-1 validation accuracy and validation loss.

In order to compare the learning performance of 1D and 2D convolution approaches, Fig. 2.8 is displayed. In this figure, we show top-1 train accuracy and train loss for models with 5 HLs as a function of number of epochs. Models with 5 HLs for both 1D and 2D convolution approaches are chosen, because they have relatively higher top-1 validation accuracy and lower validation loss among the models with higher hidden layer numbers. In Fig. 2.8 (a), the blue line represents the top-1 train accuracy of 1D 5HL model and the orange line represents the top-1 train accuracy of the 2D 5HL model. We can see that 2D model outperforms the 1D model in terms of top-1 train accuracy by approximately 1.5%. In Fig. 2.8 (b), blue line represents top-1 train loss of 1D 5HL model and orange line represents top-1 train loss of 2D 5HL model. In this graph, 2D model results in lower train loss compared to 1D model.

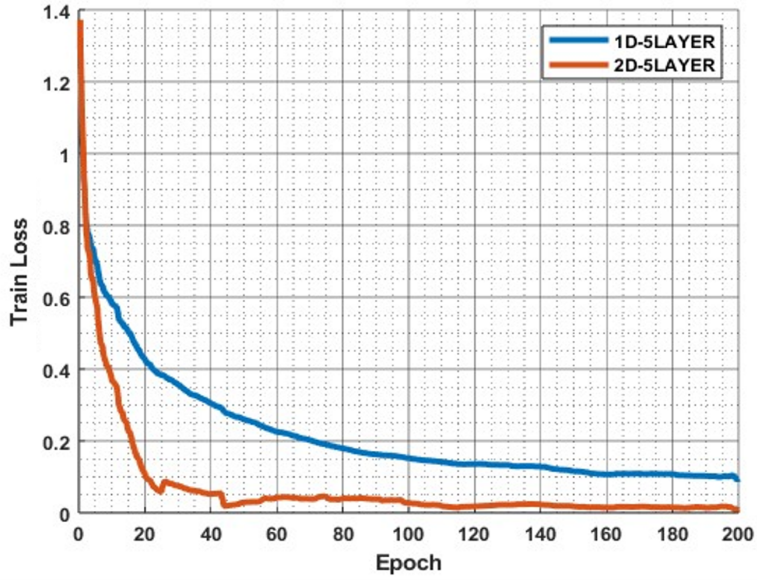
In order to compare the generalization performance 1D and 2D convolution approaches, Fig. 2.9 is displayed. To this end, in this figure, we evaluate top-1 validation accuracy and validation loss for models with 5 HLs. The reason is that, models with 5 HLs for both 1D and 2D convolution approaches have relatively higher accuracy and lower loss among the models with higher hidden layer numbers.

In Fig. 2.9 (a), the blue line represents the top-1 validation accuracy of 1D 5HL model and the orange line represents the top-1 validation accuracy of the 2D 5HL model. We can see that 1D model outperforms the 2D model in terms of top-1 validation accuracy by 7.2%. At epoch 35, the QAT starts. After the start of QAT, the validation accuracy of the 2D model suddenly stopped increasing, started decreasing around 40^{th} epoch and then converged to 85% at the end of 200^{th} epoch whereas the 1D model continued to improve in terms of top-1 validation accuracy and reached 93% at the end of 200^{th} epoch. This shows that 1D model was not affected from QAT while 2D model was negatively affected.

In Fig. 2.9 (b), blue line represents top-1 validation loss of 1D 5HL model and orange line represents top-1 validation loss of 2D 5HL model. In this graph, 1D model outperforms 2D model in terms of validation loss. After the start of QAT at 35^{th} epoch, a sharp



(a) 1D-CNN Top-1 Train Accuracy vs. 2D-CNN Top-1 Train Accuracy

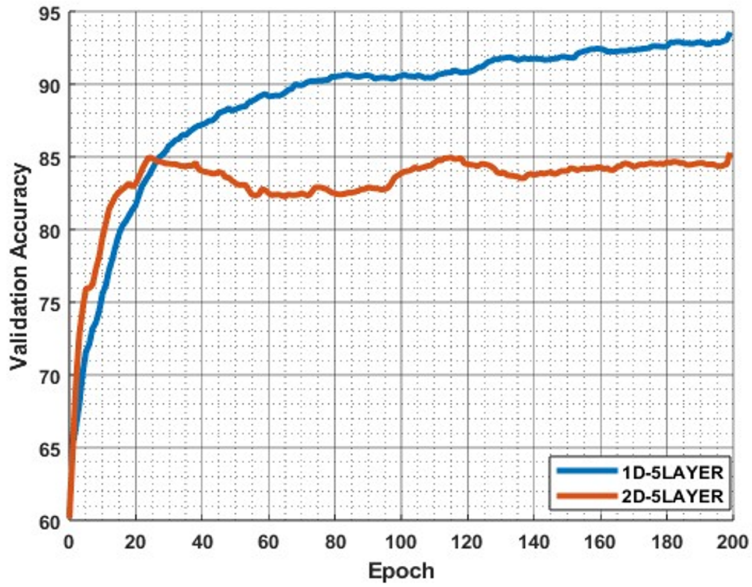


(b) 1D-CNN Train Loss vs. 2D-CNN Train Loss

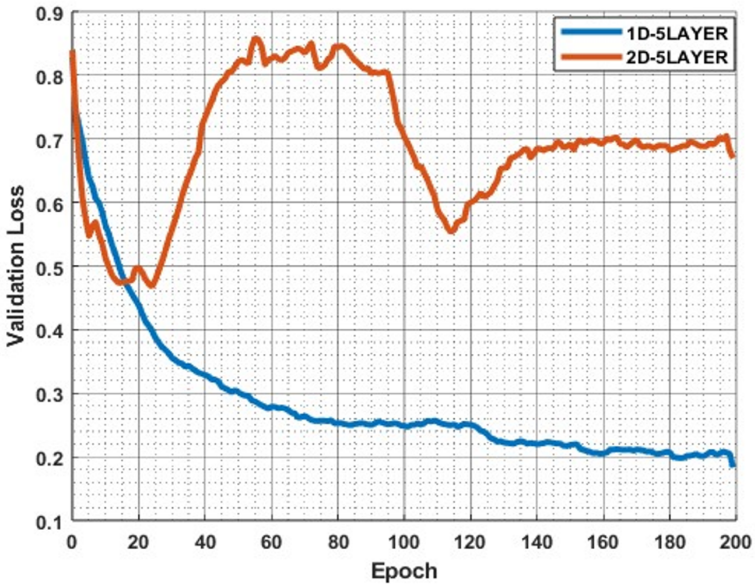
Figure 2.8. Top-1 Train Accuracy and Train Loss Comparison of 1D and 2D CNN Models with 5 HLs

increase is observed in validation loss of 2D model while 1D model is not affected. The reason behind this is that 2D model has a more complex structure compared to 1D model, thus it affects learning of 2D model more than 1D model.

In this section, we observe that, 2D models outperform 1D models in terms of top-1 train accuracy by a small margin. In addition, they have lower train loss compared to 1D models. However, 1D models outperforms 2D models in terms of top-1 validation



(a) 1D-CNN Top-1 Validation Accuracy vs. 2D-CNN Top-1 Validation Accuracy



(b) 1D-CNN Validation Loss vs. 2D-CNN Validation Loss

Figure 2.9. Top-1 Validation Accuracy and Validation Loss Comparison of 1D and 2D CNN Models with 5 HLs

accuracy and validation loss. The reason is that, for training, models utilize a similar data package in each epoch, whereas for validation, models utilize a data package which they have not used in training. This process shows if the model has learned to accurately classify a data sample outside of the training set. Observed results show that, 1D models have a better generalization performance than 2D models.

2.3.2 Evaluation Results

In this section, we present the results from hardware-based simulations conducted in the ai8x environment. These simulations use 8-bit weights that are generated with Quantization Aware Training (QAT). The model's class prediction performance was evaluated using test samples.

Test dataset is used to simulate hardware-based behaviour. This dataset completely differs from training and validation sets, thus it gives an overview of models perceptions. In order to illustrate the evaluation results of this dataset for 1D models, Fig. 2.10 is displayed. Fig. 2.10 shows the top-1 test accuracies (purple bars) and test losses (yellow bars) of 1D models as a function of number of HLs. These evaluation results are obtained by testing the trained models with the test set. In this figure, we can see that top-1 accuracies of models with HL counts 8 and 9 have significantly dropped, and the accuracy of 5 HL model is improved slightly compared to Fig. 2.7 (a) in which top-1 validation accuracy and validation loss is measured. A similar pattern can be seen on test loss values of models with 6 HLs and 9 HLs, as their loss rates have increased.

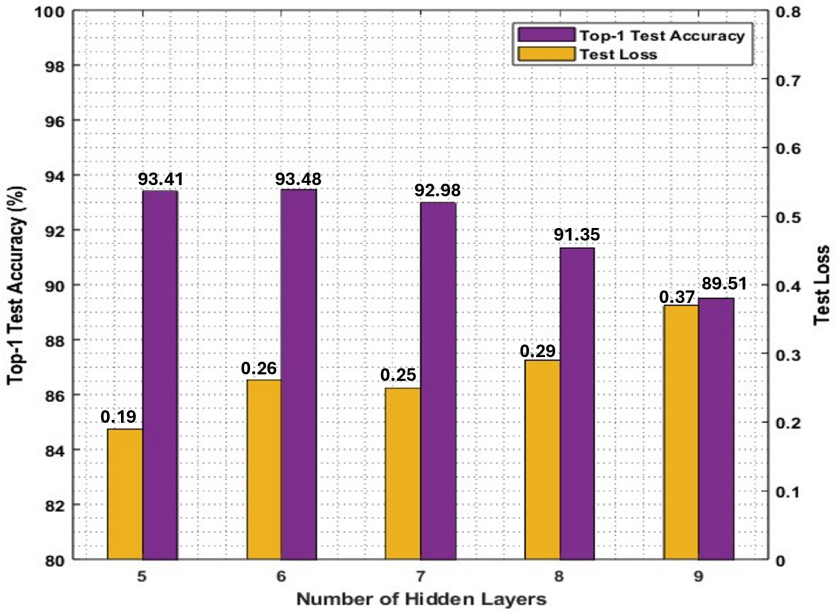


Figure 2.10. Top-1 Test Accuracy and Test Loss of 1D Models

In order to measure the evaluation performance of 2D models, Fig. 2.11 is displayed. This figure shows the top-1 test accuracies (purple bars) and test losses (yellow bars) of each model. In this figure, we observe a fluctuation in top-1 test accuracy of 2D models, as the number of hidden layers increase. Still, the general trend for top-1 test accuracy is downward. Besides, there is a sharp increase in test loss when the number of hidden layers increase from 5 to 6. After this sharp increase (from 0.71 to 1.21), test

loss decreases to a lower test loss value (0.93) and stabilizes around that point. Top-1 accuracy values of 2D models show a similar pattern compared to that of Fig. 2.7 (a), hence, we can say that validation and test result are consistent with each other.

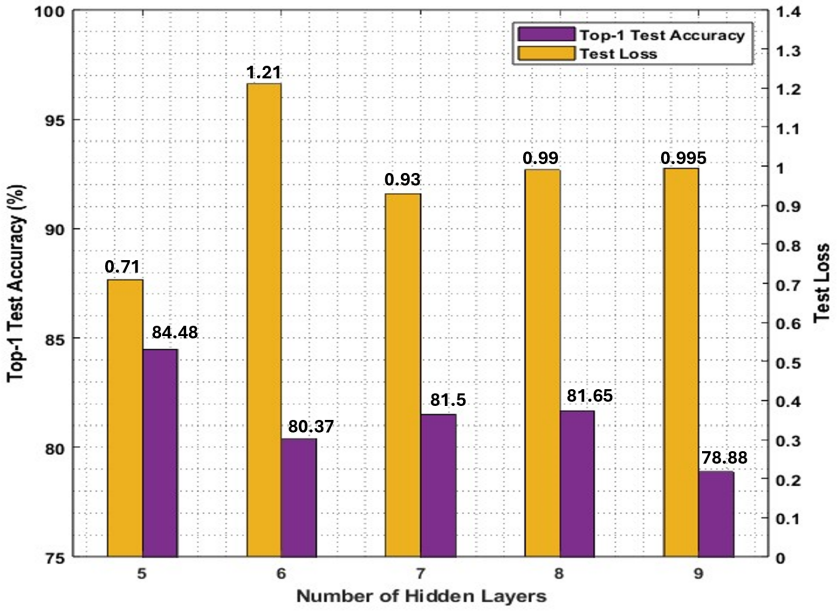


Figure 2.11. Top-1 Test Accuracy and Test Loss of 2D Models

In order to provide detailed breakdown of 1D and 2D model's performance, Fig. 2.12 and Fig. 2.13 are displayed. Fig. 2.12 illustrates the confusion matrix of the 1D model with 5 hidden layers. The diagonal elements represent the true positive predictions, while the off-diagonal elements indicate misclassified predictions. The figure shows that the true positive values are notably higher than the other values, suggesting that the model is reasonably effective at correctly identifying true positive cases.

Fig. 2.13 illustrates the confusion matrix of the 1D model with 5 hidden layers. When we compare this figure against Fig. 2.12, we see that the true positive values of confusion matrix of 1D model are greater for each of the four classes. While both models show similar results for true positives of base and broken classes, there is a significant difference between 1D and 2D model's true positive predictions for fresh and moderate classes. The results suggest that, base and broken classes are more predictable compared to fresh and moderate classes.

In this section, we observe that, first, top-1 test accuracies for 1D CNN models are approximately 9% higher compared to 2D CNN models. Second, 1D CNN model with 5 HL show better prediction results compared to 2D CNN model with 5 HL for each of the classes. Considering these results, we choose 1D CNN model with 5 HL for our demonstrations on hardware.

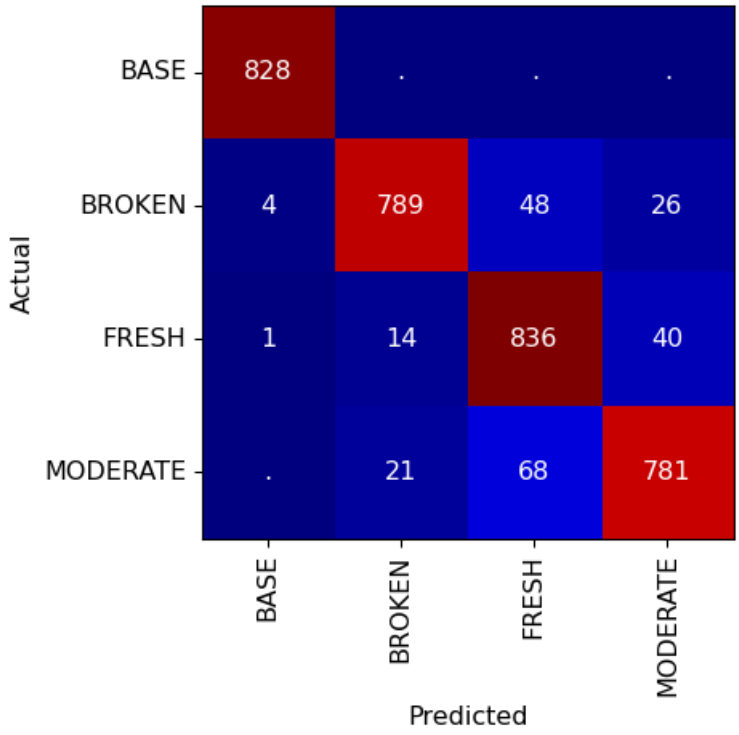


Figure 2.12. Confusion Matrix of 1D CNN Model with 5 Hidden Layers

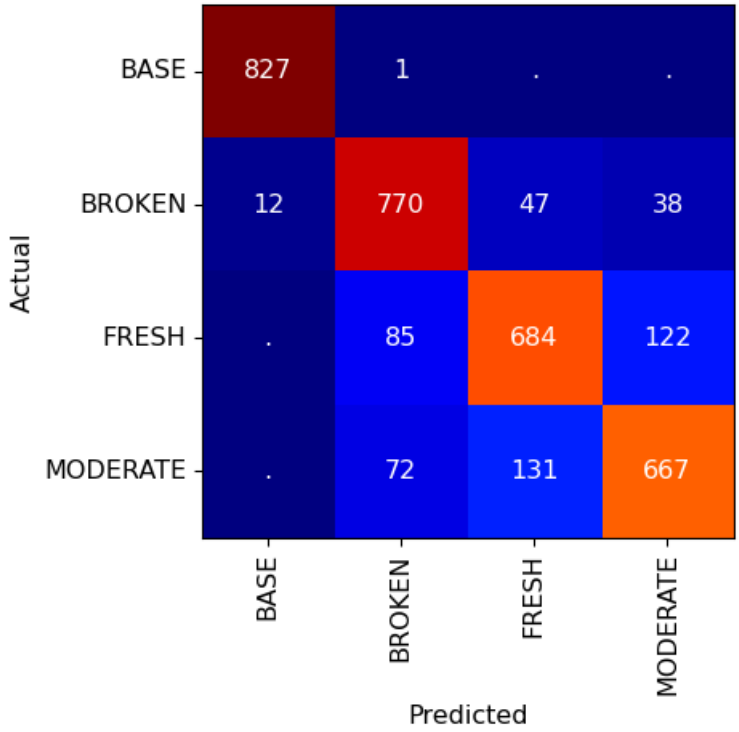


Figure 2.13. Confusion Matrix of 2D CNN Model with 5 Hidden Layers

2.3.3 Experimental Results

In this section, first, we present the experimental setup and demonstrations of the provided CNN models on MAX78000 MCU. Second, we compare their CNN inference times and power usage metrics.

2.3.3.1 Demonstrations

The MAX78000 Evaluation Kit is equipped with a TFT LCD screen, built-in microphone and Power Monitor (PMON). The workflow of our demonstration is illustrated in Fig. 2.14. In this figure, first, the MAX78000 device is started and the built-in I2S microphone and TFT LCD screen are initialized. The device requires user input to start recording the audio. After the user input, the built-in microphone starts recording the required audio samples. In order to prevent any audio detection errors, audio threshold for detection and I^2S buffer limit values are adjusted. After acquiring samples, hardware loads the CNN and waits until the processing is complete. After CNN processing is complete, CNN loads are unloaded to save energy and the classification results are displayed on both the TFT LCD screen and serial communication monitor. On this TFT LCD screen, our input audio data is shown as waveform to inform the user. CNN inference times are written to the serial communication port and displayed.

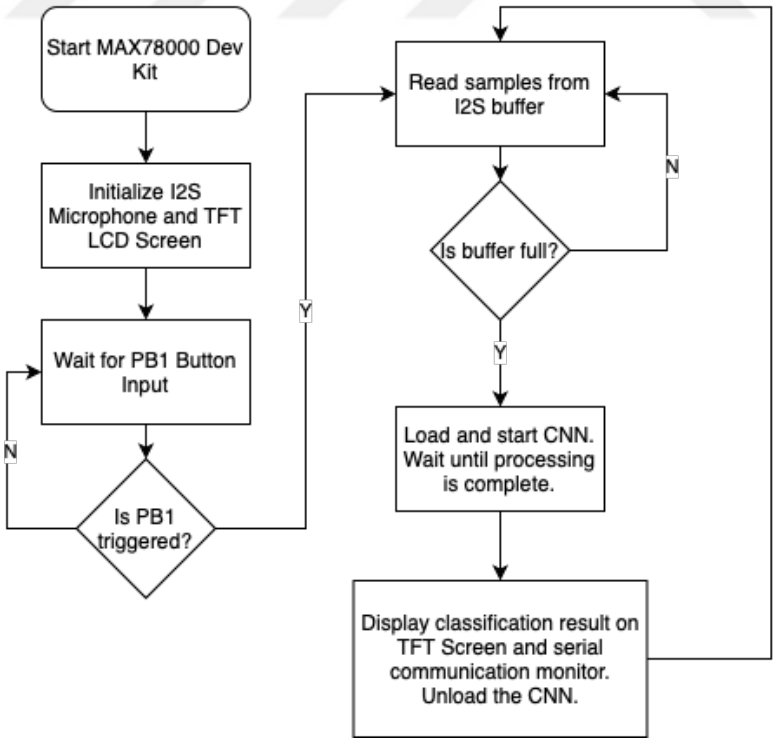


Figure 2.14. Demonstration Work Flow

A real-time demonstration on MAX78000 Dev Kit is shown in Fig. 2.15. In this figure,

the TFT LCD screen of the MAX78000 Dev Kit displays the input audio in waveform format with prediction probability and decibel (dB) of the audio input. The input audio was selected from the test dataset and fed into the built-in microphone of MAX78000 Dev Kit.

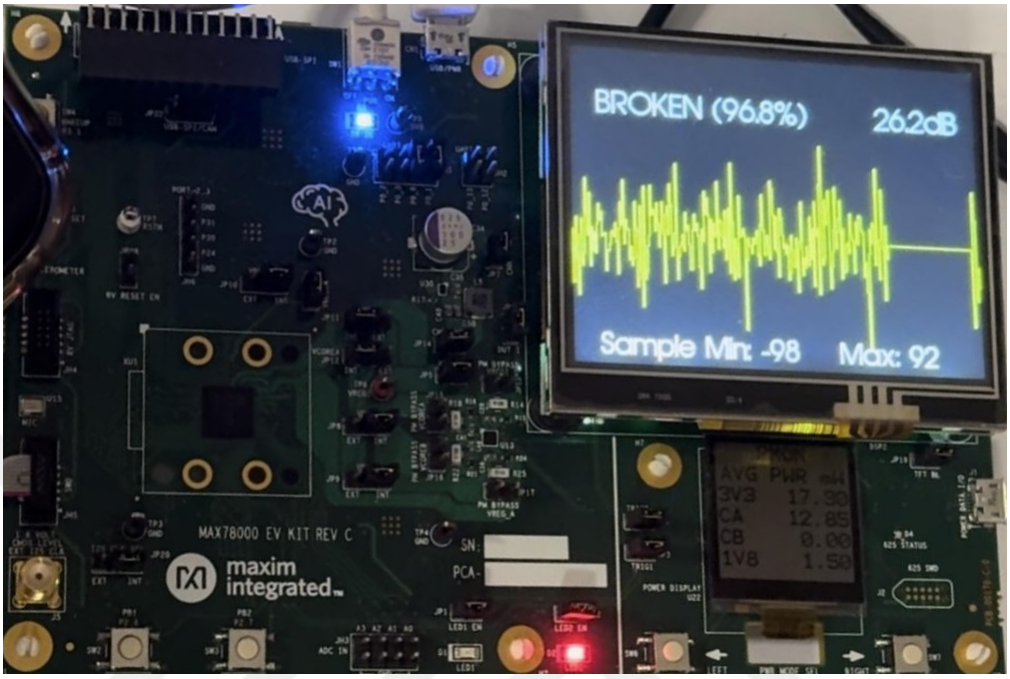


Figure 2.15. Demonstration on MAX78000 Dev Kit

2.3.3.2 Model Comparison

In this section, we present the CNN inference times, energy and average power rates of different models that were tested on the MAX78000 Dev Kit, in order to compare the models with different number of hidden layers. The PMON on MAX78000 Dev Kit was used to measure the energy (μJ) and average power (mW) draws of the different CNN models. The CNN inference time was measured using the timer on the MCU in the main C++ loop.

Table 2.2. Time and Energy Consumption of Each Model

Model	Inference Time (μs)	Energy (μJ)	Average Power (mW)
5 HL	939	246	12.88
6 HL	1127	248	12.95
7 HL	1271	249	13.04
8 HL	1438	252	13.12
9 HL	1853	256	13.20

Table 2.2 shows the Inference Times (μs), energy usage (μJ) and average power (mW) draws of the 1D models with 5, 6, 7, 8 and 9 HL. The energy usage rates of the models

show slight differences. However, the 9 HL model stands out by displaying significantly higher latency compared to the other models. Additionally, it draws slightly more power than other models. The increase in power consumption and latency are attributed to the complexity of the model; as the number of hidden layers increases, both the average power draw and inference time tend to increase.

2.4. Summary

In this chapter, we have developed a CNN model on ultra low-power MAX78000 MCU that is spesifically designed for ANN applications with its CNN accelerator. First, we have explained the cutting tool wear audio dataset which has four distinct classes: "base", "fresh", "moderate", and "broken". We also give the preprocessing details of this dataset. Second, we have presented the CNN model structure that we have implemented on the MCU. Then, we have explained the training process that includes QAT to fit the memory limitations of the MCU. Third, we have presented our results. We have compared the performance of both 1D and 2D CNN models with different number of HLs. We have found that 1D CNN models outperform 2D CNN models by a significant margin in terms of the top-1 validation and test accuracy while 2D CNN moldes outperform 1D CNN models in terms of the top-1 train accuracy. In addition, we have observed similar trend in train, validation and test losses. These results suggest that even if the 2D CNN models have high performance on the train dataset, 1D CNN models have better generalization ability on validation and test sets, which are unseen data by the model. Fourth, we have presented our experimental results and demonstration. We have compared the inference time, energy and average power drawn by 1D CNN models with 5, 6, 7, 8, and 9 HLs. We have found that 1D-5 HL model consumes the lowest power and runs faster in real-time among other models with higher hidden layer numbers.

3. CHAPTER: CONCLUSIONS

In this thesis, we present a novel approach for predictive maintenance in industrial facilities by utilizing an optimized 1D CNN model implemented on the MAX78000 MCU. This model predicts the wear rates of end mill cutters by processing raw audio data and classifying them based on their wear rates. Our 1D model outperforms the 2D models, achieving a 93% accuracy. This work serves as an example of TinyML and edge computing, demonstrating the effective application of machine learning on low-power, resource-constrained devices for real-time analysis directly at the data source.

In Chapter 2, first, we explained the background information of the implementation steps and we present these steps in detail. Our dataset consists of 4 different classes; base, broken, fresh and moderate, these classes represent the blade conditions of the end mill cutter machine. Second, we present our CNN models and training process. In this thesis, we implemented 1D and 2D CNN models to classify the machine sounds on MAX78000 MCU. In the training process, we used QAT approach to reduce the size of the model to fit the memory contraints of the MCU. Third, we explained the hardware implementation steps. In this step, we configured a YAML network to optimize the MCU's processor utilization. Fourth, we presented our training, validation and evaluation results. In this part, we compared 1D and 2D CNN models in terms of Top-1 Accuracy and Cross-Entropy Loss. We observed that, for this task, 2D CNN models have outperformed 1D CNN models in terms of Top-1 training accuracy and train loss. However, 1D CNN models have outperformed 2D CNN models in terms of Top-1 validation by 10% and in terms of Top-1 test accuracy by 9%. These results show that, although 2D CNN models have better learning performance on the train dataset, 1D CNN models have better generalization performance for this task. In addition, we compared the 1D and 2D CNN models in terms of number of hidden layers. This comparison shows that, as the number of hidden layer increase, Top-1 validation and Top-1 test accuracies decrease. The reason is that, adding more hidden layers increase the overall complexity of the model, hence, results in decrease in accuracy. Fifth, we demonstrated the 1D CNN models that have 5, 6, 7, 8 and 9 HLs on MAX78000 MCU Dev Kit. We found that the energy consumption and average power draw rates are similar, however, adding more hidden layers increase the energy consumption and average power draw slightly. In addition, inference times increase as the hidden layer count increase.

These results show that, using a CNN model with lower hidden layer count results in a relatively less complex model with better results compared to higher hidden layer count CNN models. Hence, this offers an advantage for TinyML applications.

In our future work, we plan to develop a predictive maintenance system using TinyML MCUs equipped with neural network accelerators capable of implementing various machine learning models, such as MLP and LSTM. The system will be equipped with multiple sensors related to industrial equipment to enhance the accuracy of these machine learning models.

REFERENCES

Abadade, Y., Temouden, A., Bamoumen, H., Benamar, N., Chtouki, Y., & Hafid, A. S. (2023). A comprehensive survey on tinyml. *IEEE Access*.

Antonini, M., Pincheira, M., Vecchio, M., & Antonelli, F. (2022). A tinyml approach to non-repudiable anomaly detection in extreme industrial environments. In *2022 ieee international workshop on metrology for industry 4.0 & iot (metroind4. 0&iot)* (pp. 397–402).

Anusha, P., Swapna, P., & Reddy, D. R. K. (2024). Edge machine learning-enabled predictive fault detection system for conveyor belt maintenance optimization in industrial settings. *Journal of Electrical Systems*, *20*(3), 2031–2041.

Athanasakis, G., Filios, G., Katsidimas, I., Nikoletseas, S., & Panagiotou, S. H. (2022). Tinyml-based approach for remaining useful life prediction of turbofan engines. In *2022 ieee 27th international conference on emerging technologies and factory automation (etfa)* (pp. 1–8).

Bakar, A., Goel, R., De Winkel, J., Huang, J., Ahmed, S., Islam, B., ... Hester, J. (2022). Protean: An energy-efficient and heterogeneous platform for adaptive and hardware-accelerated battery-free computing. In *Proceedings of the 20th acm conference on embedded networked sensor systems* (pp. 207–221).

Banbury, C. R., Reddi, V. J., Lam, M., Fu, W., Fazel, A., Holleman, J., ... others (2020). Benchmarking tinyml systems: Challenges and direction. *arXiv preprint arXiv:2003.04821*.

Büyüksolak, O., & Güneş, E. O. (2023). Ai-based visual odometry implementation on an embedded system. In *2023 10th international conference on electrical and electronics engineering (iceee)* (pp. 47–51).

Chen, Z., Gao, Y., & Liang, J. (2023). Lopdm: A low-power on-device predictive maintenance system based on self-powered sensing and tinyml. *IEEE Transactions on Instrumentation and Measurement*.

Cooper, G., Benenati, V., Long, B., Copeland, K., Ekaireb, T., Kumar, S., ... Isukapalli, Y. (2022). Autonomous system for sorting objects at the edge..

Cooper, G., Manjunath, B., & Isukapalli, Y. (2021). Edge machine learning for face detection..

Ganga, D., & Ramachandran, V. (2018). Iot-based vibration analytics of electrical machines. *IEEE Internet of Things Journal*, *5*(6), 4538–4549.

Giordano, M., Piccinelli, L., & Magno, M. (2022). Survey and comparison of milliwatts micro controllers for tiny machine learning at the edge. In *2022 ieee 4th international conference on artificial intelligence circuits and systems (aicas)* (pp. 94–97).

Goundar, S. S., Pillai, M. R., Mamun, K. A., Islam, F. R., & Deo, R. (2015). Real time condition monitoring system for industrial motors. In *2015 2nd asia-pacific world congress on computer science and engineering (apwc on cse)* (p. 1-9). doi: 10.1109/APWCCSE.2015.7476232

Ingaleshwar, S., Thasharofi, F., Pava, M. A., Vaishya, H., Tabak, Y., Ernst, J., . . . others (2024). Wildlife species classification on the edge: A deep learning perspective. In *Icaart (3)* (pp. 600–608).

Jadhav, A., Gaikwad, R., Patekar, T., Dighe, S., Shaikh, B., & Patankar, N. S. (2023). Predictive maintenance of industrial equipment using iot and machine learning. In *2023 4th international conference on computation, automation and knowledge management (iccakm)* (p. 1-5). doi: 10.1109/ICCAKM58659.2023.10449546

Lightbody, D., Ngo, D.-M., Temko, A., Murphy, C., & Popovici, E. (2022). Host-based intrusion detection system for iot using convolutional neural networks. In *2022 33rd irish signals and systems conference (issc)* (pp. 1–7).

Mihigo, I. N., Zennaro, M., Uwitonze, A., Rwigema, J., & Rovai, M. (2022). On-device iot-based predictive maintenance analytics model: Comparing tinylstm and tinymodel from edge impulse. *Sensors*, *22*(14), 5174.

Mnif, M., Sahnoun, S., Kaaniche, M., Atitallah, B. B., Fakhfakh, A., & Kanoun, O. (2024). Ultra-fast edge computing approach for hand gesture classification based on eit measurements. In *2024 ieee international symposium on robotic and sensors environments (rose)* (pp. 1–7).

Moosmann, J., Giordano, M., Vogt, C., & Magno, M. (2023). Tinyissimoyolo: A quantized, low-memory footprint, tinyml object detection network for low power microcontrollers. In *2023 ieee 5th international conference on artificial intelligence circuits and systems (aicas)* (pp. 1–5).

Moosmann, J., Müller, H., Zimmerman, N., Rutishauser, G., Benini, L., & Magno, M. (2024). Flexible and fully quantized lightweight tinyissimoyolo for ultra-low-power edge systems. *IEEE Access*.

Moss, A., Lee, H., Xun, L., Min, C., Kawsar, F., & Montanari, A. (2022). Ultra-low power dnn accelerators for iot: Resource characterization of the max78000. In *Proceedings of the 20th acm conference on embedded networked sensor systems* (pp. 934–940).

Natesha, B., & Guddeti, R. M. R. (2021). Fog-based intelligent machine malfunction monitoring system for industry 4.0. *IEEE Transactions on Industrial Informatics*, *17*(12), 7923–7932.

Ngo, D.-M., Lightbody, D., Temko, A., Pham-Quoc, C., Tran, N.-T., Murphy, C. C., & Popovici, E. (2023). Hh-nids: Heterogeneous hardware-based network intrusion detection framework for iot security. *Future Internet*, *15*(1). doi: 10.3390/fi15010009

Njor, E., Madsen, J., & Fafoutis, X. (2022). A primer for tinyml predictive maintenance: Input and model optimisation. In *Ifip international conference on artificial intelligence applications and innovations* (pp. 67–78).

Okman, O. E., Ulkar, M. G., & Uyanik, G. S. (2022). Lˆ 3u-net: Low-latency lightweight u-net based image segmentation model for parallel cnn processors. *arXiv preprint arXiv:2203.16528*.

Plozza, D., Giordano, M., & Magno, M. (2022). Real-time low power audio distortion circuit modeling: a tinyml deep learning approach. In *2022 ieee 4th international conference on artificial intelligence circuits and systems (aicas)* (pp. 415–418).

Rüegg, T., Giordano, M., & Magno, M. (2023). Kp2dtiny: Quantized neural keypoint detection and description on the edge. In *2023 ieee 5th international conference on artificial intelligence circuits and systems (aicas)* (pp. 1–5).

Soni, N., Kumar, A., & Patel, H. (2023). Acoustic analysis of cutting tool vibrations of machines for anomaly detection and predictive maintenance. In *2023 ieee 11th region 10 humanitarian technology conference (r10-htc)* (p. 43-46). doi: 10.1109/R10-HTC57504.2023.10461855

Talmoudi, S., Kanada, T., & Hirata, Y. (2019). An iot-based failure prediction solution using machine sound data. In *2019 ieee/sice international symposium on system integration (sii)* (pp. 227–232).

Ulkar, M. G., & Okman, O. E. (2021). Ultra-low power keyword spotting at the edge. *arXiv preprint arXiv:2111.04988*.

Wang, G., Bhat, Z. P., Jiang, Z., Chen, Y.-W., Zha, D., Reyes, A. C., ... others (2022). Bed: A real-time object detection system for edge devices. In *Proceedings of the 31st acm international conference on information & knowledge management* (pp. 4994–4998).

Yong, L. Z., & Nugroho, H. (2022). Acoustic anomaly detection of mechanical failure: Time-distributed cnn-rnn deep learning models. In *Control, instrumentation and mechatronics: Theory and practice* (pp. 662–672). Springer.

Zhang, Y., Zeng, P., Yang, G., & Li, J. (2013). Online and remote machine condition monitoring and fault diagnosis system using wireless sensor networks. In *2013 15th ieee international conference on communication technology* (pp. 259–265).

Zhao, X., & Wang, P. (2024). A deployable edge computing solution for machine condition monitoring. In *2024 ieee international instrumentation and measurement technology conference (i2mtc)* (pp. 1–6).