UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Efficient Digital Health Solutions Using Wearable Devices

Permalink

https://escholarship.org/uc/item/1fp76657

Author

Rashid, Nafiul

Publication Date

2023

Copyright Information

This work is made available under the terms of a Creative Commons Attribution-NonCommercial-ShareAlike License, availalbe at https://creativecommons.org/licenses/by-nc-sa/4.0/

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

Efficient Digital Health Solutions Using Wearable Devices

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Nafiul Rashid

Dissertation Committee:
Professor Mohammad Abdullah Al Faruque, Chair
Associate Professor Marco Levorato
Assistant Professor Peter Tseng

2023

Portion of Chapter 3 © 2021 IEEE
Portion of Chapter 4 © 2020 IEEE
Portion of Chapter 5 © 2022 IEEE
Portion of Chapter A © 2020 IEEE
Portion of Chapter A © 2021 IEEE
All other materials © 2023 Nafiul Rashid

DEDICATION

To
My loving parents Lutfa Begum and Mohammad Bazlur Rashid
&
My wonderful wife Samia Shafique and dearest son Zayd Rashid
&
My amazing brother Mohammad Nizamul Rashid

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my gratitude to my academic advisor and committee chair, Professor Mohammad Abdullah Al Faruque, for all of his help and support throughout my research. Without his constant guidance, and valuable suggestions I would not have been able to complete this thesis today. He has been there for me through thick and thin.

I would also like to express my heartfelt appreciation to my committee members, Professor Marco Levorato and Professor Peter Tseng, for dedicating their valuable time to review my work and provide insightful comments.

I would like to extend my sincere thanks to colleagues in the Autonomous and Intelligent Cyber-Physical Systems (AICPS) laboratory, particularly Dr. Jiang Wan, Dr. Sujit Rokka Chhetri and Dr. Sina Faezi, Mohanad Odema, Berken Utku Demirel, Luke Chen, and Trier Mortlock, with whom I had the pleasure to collaborate with. I also thank my other collaborators Dr. Manik Dautta, and Abel Jimenez for their support in my research.

I would like to thank UCI's Department of Electrical Engineering and Computer Science for their support which allowed me to thrive on this exciting journey of research. I would also like to thank National Science Foundation (NSF) for partially funding my research under grants National Science Foundation (NSF) under awards CMMI-1739503 and CCF-2140154. My research work is also partially supported by the National Institutes of Health (NIH) grant R41DA049615 and the Graduate Assistance in Areas of National Need (GAANN) award from the United States Department of Education. Moreover, I would like to thank the Office of Naval Research to support my research partially under contract number N00014-16-C-2005. Any opinions, findings, conclusions, or recommendations expressed in this thesis are those of the author and do not necessarily reflect the funding agencies' views.

Part of this dissertation is a reprint of the materials as they appear in — Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT), IEEE Internet of Things Journal, Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Annual International Conference on Distributed Computing in Sensor Systems (DCOSS), Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC), and IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED). The content of Chapter 2 is a reprint of the materials as it appears in [40], and the authors hold the copyright through open access license. The content of Chapter 3 is a reprint of the materials as it appears in [88], and used with permission from IEEE. The content of Chapter 4 is a reprint of the material as it appears in [44], and used with permission from IEEE. The content of Chapter 5 is a reprint of the materials as it appears in [159, 160], and used with permission from IEEE. Finally, the portion of appendix Chapter A is a reprint of the materials as it appears in [52, 53, 110, 113], and used with permission from IEEE.

VITA

Nafiul Rashid

EDUCATION

Doctor of Philosophy in Electrical and Computer Engineering **2023**
University of California, Irvine *Irvine, California*

Master of Science in Computer Science and Engineering **2015**
Islamic University of Technology *Dhaka, Bangladesh*

Bachelor of Science in Computer Science and Engineering **2013**
Islamic University of Technology *Dhaka, Bangladesh*

RESEARCH EXPERIENCE

Graduate Research Assistant **2016–2021**
University of California, Irvine *Irvine, California*

PhD Research Intern **2022**
Samsung Research America *Mountain View, California*

PhD Research Intern **2017**
Siemens Corporate Research *Princeton, New Jersey*

TEACHING EXPERIENCE

Teaching Assistant **2021–2022**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

Stress Detection using Context-Aware Sensor Fusion from Wearable Devices **2023**
Under Review in IEEE Internet of Things Journal

AHAR: Adaptive CNN for Energy-efficient Human Activity Recognition in Low-power Edge Devices **2021**
IEEE Internet of Things Journal

HEAR: Fog-enabled Energy Aware Online Human Eating Activity Recognition **2020**
IEEE Internet of Things Journal

Wireless Qi-powered, multinodal and multisensory body area network for mobile health **2020**
IEEE Internet of Things Journal

Manufacturing supply chain and product lifecycle security in the era of industry 4.0 **2018**
Journal of Hardware and Systems Security

REFEREED CONFERENCE PUBLICATIONS

Template Matching Based Early Exit CNN for Energy-efficient Myocardial Infarction Detection on Low-power Wearable Devices **2022**
Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies

SELF-CARE: Selective Fusion with Context-Aware Low-Power Edge Computing for Stress Detection **2022**
18^{th} Annual International Conference on Distributed Computing in Sensor Systems (DCOSS 2022)

Feature Augmented Hybrid CNN for Stress Recognition Using Wrist-based Photoplethysmography Sensor **2021**
43^{rd} Annual International Conference of the IEEE Engineering in Medicine and Biology Society

EExNAS: Early-exit neural architecture search solutions for low-power wearable devices **2021**
2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)

LENS: Layer Distribution Enabled Neural Architecture Search in Edge-Cloud Hierarchies **2021**
2021 58^{th} ACM/IEEE Design Automation Conference (DAC)

Energy-Aware Design Methodology for Myocardial Infarction Detection on Low-Power Wearable Devices **2021**
Proceedings of the 26^{th} Asia and South Pacific Design Automation Conference, ASP-DAC '21

Energy-efficient Real-time Myocardial Infarction Detection on Wearable Devices **2020**
42^{nd} Annual International Conference of the IEEE Engineering in Medicine and Biology Society

A survivability-aware cyber-physical systems design methodology **2019**
2019 IEEE 15^{th} International Conference on Automation Science and Engineering (CASE)

Concept Design: Modeling and Synthesis from Requirements to Functional Models and Simulation **2018**
Design Automation of Cyber-Physical Systems, Springer International Publishing

Security trends and advances in manufacturing systems in the era of industry 4.0 **2017**
2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)

Modeling and simulation of cyberattacks for resilient cyber-physical systems **2017**
2017 13^{th} IEEE Conference on Automation Science and Engineering (CASE)

ABSTRACT OF THE DISSERTATION

Efficient Digital Health Solutions Using Wearable Devices

By

Nafiul Rashid

Doctor of Philosophy in Electrical and Computer Engineering

University of California, Irvine, 2023

Professor Mohammad Abdullah Al Faruque, Chair

With the advancement of technology and the prevalence of Internet of Things (IoT), wearable devices have been gaining huge momentum as consumer devices over the past few years. The wide adoption and multiple sensor modalities in those wearable devices have enabled them to be used for designing different digital health solutions for continuous and remote monitoring. The continuous use of wearable devices results in a myriad volume of physiological data being collected from multiple sensors. Transferring this collected data from wearable devices (edge) to server (cloud) introduces three major challenges - increased energy consumption, increased latency, and vulnerability to breach of privacy. To tackle these challenges, on-device computing solutions have been adopted for different wearable healthcare systems. However, the small form factor of wearable devices imposes three constraints on the on-device solutions. The solutions should be energy-efficient, memory-efficient, and provide maximum performance within the previous two constraints. Moreover, getting the ground truth label of the collected data from the user to update the classification models is a challenging task without user involvement. Furthermore, fusing data from multiple heterogeneous sensors may often introduce new challenges in terms of achieving maximum performance regardless of computing architecture.

To address these challenges, this thesis presents some efficient methodologies for different

wearable healthcare applications like - Myocardial Infarction Detection, Human Activity Recognition, Stress Detection, and Eating Activity Recognition. The first part of the thesis explores the Myocardial Infarction Detection application where we propose a Template Matching based Early Exit architecture that achieves energy and memory efficiency while outperforming state-of-the-art work. The second part proposes an adaptive convolutional neural network for Human Activity Recognition application. Our proposed solution outperforms state-of-the-art works while achieving energy and memory efficiency. In the third part of the thesis, we explore the Eating Activity Recognition application where we propose an online learning methodology to adapt to changing eating habits of the user. Our proposed online learned neural network outperforms other competitive offline methods while being energy-efficient. The fourth and final part of the thesis explores Stress Detection application using multiple heterogeneous sensor modalities. We propose a selective fusion approach using context-aware sensor selection from wearable devices which achieves better performance compared to state-of-the-art works.

Chapter 1

Introduction

1.1 Moore's Law and Internet of Things

"The number of transistors in a dense integrated circuit (IC) doubles about every two years."
– Gordon Moore (1965) [1]

The infamous Moore's Law states that since the number of transistors on a silicon chip roughly doubles every two years, the performance and capabilities of computers will continue to increase. Although Gordon Moore made this statement based on empirical observations in 1965, this has been considered as law over the years due to its correlation with the advancements in technology. With the increasing number of transistors, the devices become more powerful with smaller size at lower cost [2]. Additionally, the advancements of communication technology has led to the development of LTE, 4G, 5G networks [3, 4]. This enabled the devices to be connected leading to the revolution of **Internet of Things (IoT)** [5–7]. It is expected that by 2025 there will be approximately 27 billion connected IoT devices [8]. And the global IoT market is expected to be USD 650.5 billion by 2026 from USD 300.3 billion in 2021 at Compound Annual Growth Rate (CAGR) of 16.7% [9].

1.2 Wearable Devices in the Internet of Things Era

With the advancement of technology and prevalence of IoT technology, the wearable devices have been gaining huge momentum as consumer devices over the past few years [10, 11]. The small form factor of the devices coupled with their user friendly design have made the wide adoption of wearable devices possible among consumers [12, 13]. Starting from wrist watch, smart ring, earbud to wearable glass there has been significant effort going on to build and design wearable devices by the industry leaders like Apple, Google and Samsung. In general the wearable devices can be broadly classified into Wrist-wear, Eye-wear, Head-wear, Foot-wear, Neck-wear, and Body-wear [14]. According to a market research [15], the global wearable technology market had a valuation of USD 61.30 billion in 2022 which expected to grow at CAGR of 14.6% from 2023 to 2030.

1.3 Future of Digital Health using Wearable Devices

The wide adoption of smart wearable devices is driving the industry growth. This results in introducing wearable devices with more features and sensing capabilities [16]. Currently, the wearable devices are equipped with various sensors like – Accelerometer, Photoplethosmography, Electrocardiography, Electromyography, Electrodermal activity, Temperature sensors. These sensors aid in monitoring blood volume pulse, heart rate, movement, muscle contraction, oxygen saturation level, skin temperature and so forth [17]. These multiple sensor modalities of the wearable devices have enabled them to be used for designing digital health solutions for continuous and remote monitoring for different healthcare applications in real-time [18, 19]. The global digital health market size was valued at USD 211.0 billion in 2022 and is projected to grow at a CAGR of 18.6% from 2023 to 2030 [20].

1.4 Major Challenges and Thesis Contribution

The continuous use of these sensor rich wearable devices generate myriad volume of sensor data from multiple modalities. Usually, the collected data from wearable devices are sent over Bluetooth to a mobile phone (fog) or remote server (cloud) where all the processing takes. This form of computing is called fog [21, 22] or cloud computing [23] architecture. However, transferring this large amount of data results in increased energy consumption reducing the battery life of wearable devices [24]. Additionally, it also introduces latency, which is unsuitable for real-time monitoring [24, 25]. Moreover, passing the raw data to fog or cloud may make the users' data vulnerable to privacy breaches [26, 27]. To overcome those issues, researchers shifted to an alternative architecture to overcome these limitations, which is called 'edge computing' [28, 29], where it performs on device processing. Therefore, it reduces the energy consumption, latency, and vulnerability of privacy breaches [30–32]. However, the small form factor of wearable devices introduces three constraints for edge computing architecture for digital health solutions. The solutions should be energy-efficient, memory-efficient, and provide maximum performance within the previous two constraints [31, 32].

Moreover, to process and compute the collected data from users, different machine learning models are trained offline and used to make predictions on the run time. However, for many healthcare applications the user behavior may change overtime due to which may cause data drift or concept drift in the collected data [33, 34]. Due to this data drift the offline trained machine learning models may significantly suffer resulting in poor performance [35, 36]. Therefore, the models need to be updated online with new data from time to time to reflect the change in user behavior and become more personalized overtime. This requires to implement online learning for the machine learning models [37]. However, one of the key challenges of online learning is to generate the ground truth labels of the new data to update the machine learning model without much user involvement.

Furthermore, another notable challenge in using data from from multiple heterogeneous wearable sensors is that the data may be susceptible to substantial amounts of sensor noise due to various factors like motion or muscle contraction [38]. Fusing such noisy measurements can subsequently degrade the performance of the machine learning models [38, 39].

In summary, this thesis addresses the following challenges of designing digital health solutions using wearable devices:

- Designing an energy-efficient and memory-efficient solution for edge computing architecture while maintaining performance.
- Generating ground truth labels of collected data to update classification models online without much user involvement.
- Fusing noisy data from multiple heterogeneous sensors may degrade the machine learning models' performance.

To tackle the aforementioned challenges, this thesis makes the following contributions:

1. Propose various energy-efficient and memory-efficient solutions for edge computing architecture while maintaining performance.
2. Propose an online learning methodology to generate approximate ground truth labels without much user involvement.
3. Propose a selective fusion approach using context-aware sensor selection from wearable devices which achieves better performance compared to state-of-the-art works.

1.5 Thesis Structure

This thesis is structured as follows:

- *Chapter 2* explores the Myocardial Infarction Detection application where we propose a Template Matching based Early Exit architecture achieves energy and memory efficiency while outperforming state-of-the-art work.
- *Chapter 3* proposes an adaptive convolutional neural network for Human Activity Recognition application that outperforms state-of-the-art works while achieving energy and memory efficiency.
- *Chapter 4* explores the Eating Activity Recognition application where we propose an online learning methodology to adapt to changing eating habits of the user that generates approximate ground truth labels without much involvement from the user.
- *Chapter 5* explores Stress Detection application where we propose a selective fusion approach using context-aware sensor selection from wearable devices which achieves better performance compared to state-of-the-art works.
- *Chapter 6* concludes the dissertation with some remarks on the contributions and discussion on future directions.

Figure 1.1 maps the thesis contributions to the corresponding chapters.

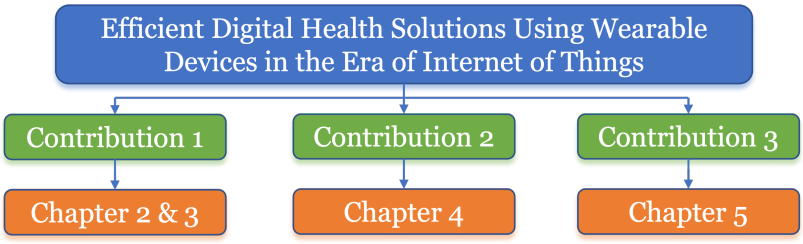


Figure 1.1: Mapping of thesis contributions to chapters

Chapter 2

Energy-efficient Myocardial Infarction Detection on Low-power Wearable Devices

2.1 Abstract

Myocardial Infarction (MI), also known as heart attack, is a life-threatening form of heart disease that is a leading cause of death worldwide. Its recurrent and silent nature emphasizes the need for continuous monitoring through wearable devices. The wearable device solutions should provide adequate performance while being resource-constrained in terms of power and memory. This chapter proposes an MI detection methodology using a Convolutional Neural Network (CNN) that outperforms the state-of-the-art works on wearable devices for two datasets – PTB and PTB-XL, while being energy and memory-efficient. Moreover, we also propose a novel Template Matching based Early Exit (TMEX) CNN architecture that further increases the energy efficiency compared to baseline architecture while maintaining

similar performance. Our baseline and TMEX architecture achieve 99.33% and 99.24% accuracy on PTB dataset, whereas on PTB-XL dataset they achieve 84.36% and 84.24% accuracy, respectively. Both architectures are suitable for wearable devices requiring only 20 KB of RAM. Evaluation of real hardware shows that our baseline architecture is 0.6× to 53× more energy-efficient than the state-of-the-art works on wearable devices. Moreover, our TMEX architecture further improves the energy efficiency by 8.12% (PTB) and 6.36% (PTB-XL) while maintaining similar performance as the baseline architecture. The findings in this chapter have been published in [40].

2.2 Introduction

Myocardial Infarction (MI) is one of the most fatal forms of heart disease being the leading cause of death worldwide. MI occurs when one of the coronary arteries responsible for supplying oxygenated blood to the heart muscle gets blocked. This blockage happens due to the deposition of plaques on the inner wall of the coronary arteries. Eventually, the heart muscle suffers from the shortage of oxygen and essential nutrients leading to a heart attack. About 805,000 people in the USA suffer from it every year. About 75% of them encounter a heart attack for the first time and the rest 25% experience recurrent heart attacks [41] which has a 6 times higher mortality rate than the first ones [42]. The higher risk associated with recurrent heart attacks requires continuous monitoring of those patients. Moreover, 1 out of 5 heart attacks is silent where victims are unaware of the damage [41]. And the mortality rate significantly increases by 41-62% if the treatment is delayed for more than 2 hours from MI initiation [43]. The aforementioned facts necessitate the real-time monitoring and detection of MI. Currently, most of the monitoring takes place in a clinical environment with bulky medical equipment which lacks portability. Therefore, wearable devices represent a more convenient solution for continuous monitoring on a daily basis.

Besides, a real-time monitoring solution using wearable devices enables physicians to keep track of their patients' health remotely. Currently, wearable device solutions for mobile health follow a cloud computing architecture where the raw data from wearable devices is offloaded to fog (mobile phones) or cloud (remote servers) where all the processing takes place [44]. This offloading consumes a huge amount of communication energy which reduces the battery life of wearable devices, as well as mobile phones [45]. In addition to that, it also introduces latency which hinders the real-time monitoring and detection in healthcare applications [46]. Moreover, offloading the raw data to a mobile phone or cloud makes the users' data vulnerable to privacy breaches. To overcome these aforementioned limitations, the 'Edge Computing' [47] paradigm has gained momentum in recent years where all the processing is done on the wearable devices and only the analyzed results are sent to the cloud for remote monitoring. Thus, it addresses the aforementioned issues related to energy consumption, latency, and vulnerability of privacy breaches.

The designed algorithms for wearable devices should be energy-efficient, memory-efficient, and provide acceptable performance within the previous two constraints. State-of-the-art works on MI detection are not wearable device compatible as they prioritize performance and do not consider the other two constraints. They use complex machine learning [48–50] and deep learning algorithms [51–55] to achieve high performance. Machine learning algorithms perform classification based on the extracted features from the data. However, the feature extraction processes are often time-consuming and require a huge amount of energy. Deep learning algorithms like Convolutional Neural Networks (CNN) do not require manual feature engineering and extraction as they automatically extract features through convolution. Moreover, the layered architecture of CNN provides flexibility to design a network by adding or removing layers as necessary in the training phase. Later, this architecture may be used to classify data during the inference phase. However, the full architecture from the training phase may not be needed at the inference phase as many of the data can be correctly classified using only the first few layers of the architecture. This early exit capability

of CNN should help to avoid redundant operations during the inference phase leading to energy efficiency for wearable device solutions while maintaining the performance.

2.2.1 Motivational Example

We conducted a small experiment to demonstrate the advantage of the early exit CNN architecture. We have created a multi-output CNN architecture with 2 convolution blocks and 2 output blocks. One output block is used after each of the convolution blocks to allow the early exit after any convolution block at the inference phase. Each convolution block consists of one convolution layer, one pooling layer, and one batch normalization layer. The details of the multi-output CNN architecture are provided in Section 3.4.2. Throughout the rest of the chapter, the first output block (FOB) is used to represent the CNN architecture that exits after the first convolution block. The second output block is considered as the baseline architecture that exits after the second convolution block. 5-fold cross-validation of the multi-output CNN architecture is performed with 62306 heartbeat segments extracted from 200 patients of the PTB diagnostic ECG database [56]. Figure 3.3 shows the blockwise statistics of the first and second output block. As demonstrated in Figure 3.3, the FLOP counts, execution time, and energy increases as performance increases from the first to second output block. We choose the second output block as the baseline architecture as it shows better performance of the two. Figure 3.3a shows that around 94% of the segments can be correctly classified by the FOB. Therefore, further convolution of those segments would be redundant. If we can avoid the redundant convolution operations, we can easily save some inference time and energy for the wearable devices. Table 3.1 shows the theoretical breakdown of the performance, FLOP counts, execution time, and energy of the early exit architecture to classify 62306 segments. Table 3.1 demonstrates that using early exit architecture, it is possible to save a total of $1143.41 \text{x} 10^6$ FLOPs, 2544.74 seconds of execution time, and 132.86 J of energy for 62306 segments. On average for each

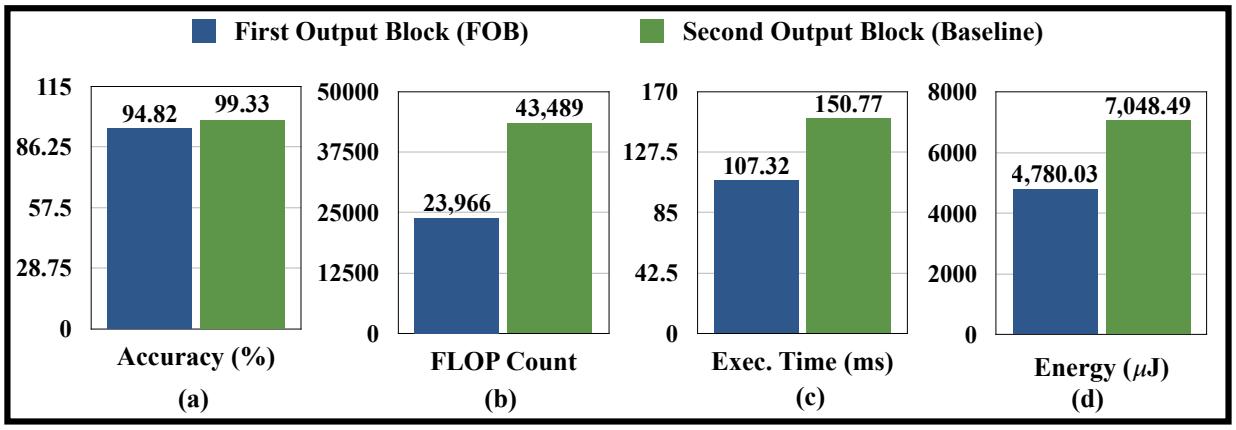


Figure 2.1: Blockwise Statistics of Multi-output CNN Architecture

Table 2.1: Difference between Baseline and Early Exit Architecture

Architecture Used	Output Block Used	% of Total Segments	# of Total Segments	Total FLOP Count	Total Exec. Time (S)	Total Energy (J)
Baseline	**Second**	**100**	**62306**	**2709.62x10^6**	**9393.87**	**439.16**
Early Exit	First	94	58567	1403.61x10^6	6285.41	279.95
	Second	6	3739	162.60x10^6	563.72	26.35
	Overall	**100**	**62306**	**1566.21x10^6**	**6849.13**	**306.30**
Theoretical Saving due to Early Exit				**1143.41x10^6**	**2544.74**	**132.86**
Average Saving per Segment due to Early Exit				**18351.52**	**40.84 (ms)**	**2132.37 (μJ)**

segment, we can save 18351.52 FLOPs, 40.84 ms of execution time, and 2132.37 μJ of energy using an early exit architecture compared to the baseline one. In summary, such early exit architecture would provide a much more energy-efficient solution than a baseline architecture while providing better or similar performance that is suitable for low-power wearable devices.

2.2.2 Problem Statement

Many state-of-the-art works [53, 57–60] have explored this early exit strategy for various applications. Usually, an extra output layer is added after each convolutional layer and a decision function is used to make the exit decisions for such early exit architectures. Some works [53, 57–59] use a simple decision function based on classification confidence. If the classification confidence at the output layer is above a certain threshold, the decision function makes the exit decision. Generally, this kind of decision function is extremely lightweight,

energy-efficient, and suitable for wearable devices. However, a substantial number of misclassified segments can cause a significant performance degradation compared to the baseline, given how the classification confidence at the earlier layers can be misleading, as will be demonstrated later in Section 2.6.2. Authors in [60] proposed to overcome this problem by using another machine learning model as the decision function. The use of the machine learning models as the decision function is suitable for deep architectures (with tens to hundreds of layers) for the projected computational gains. However, it is not suitable for wearable devices as the decision function's computational overhead may be comparable to the compact baseline classifier of the wearable devices. Moreover, state-of-the-art early exit strategies follow a brute force approach in which the decision function is invoked for every potential exit point that succeeds each convolution layer. Such needless invocations may lead to extra computational overheads if the majority of the input data segments eventually require to be processed by the baseline model. The aforementioned limitations prevent the state-of-the-art early exit strategies to be adopted for wearable device solutions for MI detection. Therefore, in this chapter, we introduce an output block selector (OBS) as our decision function to address these limitations, where it performs template matching using the simple Pearson correlation of a heartbeat segment against a template heartbeat. The template matching coefficient is used to select the output block, thus avoiding the brute force approach. Additionally, the same template matching coefficient is used for the exit decision-making criteria alongside the classification confidence to minimize the performance loss. What's more, the OBS is also lightweight enough to be adopted for wearable devices. To the best of our knowledge, we are the first to consider such a template matching based early exit (TMEX) architecture for MI detection that provides energy efficiency while ensuring similar performance as the baseline.

2.2.3 Novel Contributions

The novel contributions of this chapter are as follows:

- We propose an MI detection methodology to implement a baseline CNN architecture that outperforms the state-of-the-art works while being energy and memory-efficient.
- We introduce a novel Template Matching based Early Exit (TMEX) CNN architecture that further increases the energy efficiency of the baseline architecture while maintaining similar performance.
- Evaluation of our methodology on two well known datasets - PTB [56] and PTB-XL [61] from PhysioNet [62]. It shows that both the baseline and TMEX architectures outperform related works for both datasets.
- Evaluations on real hardware show that our baseline architecture achieves from 0.6× to 53× energy efficiency compared to the state-of-the-art work on wearable devices. Moreover, our TMEX architecture further improves the energy efficiency over the baseline by 8.12% (PTB) and 6.36% (PTB-XL) while maintaining similar performance.
- Hardware evaluations also demonstrate that our baseline and TMEX architectures are compatible with the low-memory requirement of wearable devices requiring only 20 KB of RAM.

2.3 Related Works

2.3.1 MI Detection Using Single Lead ECG

As our proposed solution is targeted for wearable devices using only one lead ECG data, we will discuss and compare against the works that used only one lead for MI detection. Authors in [48–50] used machine learning algorithms K-Nearest Neighbor (k-NN), Support Vector Machine (SVM), Random Forest (RF) for MI detection. All of these works applied 4-level Discrete Wavelet Transform (DWT) using Daubechies 6 (db6) basis function on the heartbeat segments which results in four detail and four approximation coefficients. Then for each of those 8 coefficients, they extracted normalized energy, Higuchi's fractal dimension, along with the following entropies: approximate, fuzzy, permutation, wavelet, Shannon, Renyi, and Tsalis which results in a total of 72 features. Then they applied an infinite latent feature selection algorithm [63] to sort those 72 features based on their relevance. Authors in [48] used first 47 of those features and achieved a very high performance (Accuracy = 98.80%, Sensitivity = 99.45%, Specificity = 96.27%) using k-NN. However, their work is intended for a clinical setup. Moreover, k-NN is not suitable for wearable device solutions as it requires all the training data to be stored locally. Authors in [49] used an event-driven 2-level SVM classifier to achieve energy efficiency on wearable devices. For the first level SVM, they only used the first 5 of those 72 features and for the second (full) level they used the first 47 features similar to [48]. Their event-driven 2-level SVM and full SVM achieved an accuracy of 90% and 95%, respectively. Authors in [50] also proposed an event-driven technique using a 5-level RF classifier where the first, second, third, and fourth levels use the first 5, 10, 15, and 20 features out of those 72 features, respectively. And the fifth/full level uses all of those 72 features. Their event-driven technique achieved relatively poor performance (accuracy = 80.32%, sensitivity = 81.02%, specificity = 79.63%) whereas the full RF achieved a slightly better performance (accuracy = 83.26%, sensitivity = 87.95%,

specificity = 78.82%).

The works in [49, 50] applied an event-driven technique to reduce the classifier complexity for energy efficiency. However, they along with [48], used extensive feature engineering to find optimal features for their classifiers which is a very difficult [64] task. Moreover, the feature extraction process is very expensive in terms of time and energy, making these methods unsuitable for real-time MI detection on wearable devices. In this scenario, deep learning algorithms like Convolutional Neural Networks (CNNs) [65] are a better alternative as they perform classification by automatically extracting features through convolution. Authors in [51] used 1-D deep CNN architecture with 4 convolutional layers, 4 max-pooling layers, and 3 fully connected layers. As their solution is intended for clinical setup, they only focused on performance without any resource constraints. Their method achieved an accuracy of 95.22%, a sensitivity of 95.49%, and specificity of 94.19%. Such a network is not suitable for wearable devices as it does not consider the energy and memory constraints. On the other hand, the authors in [52] developed a wearable device solution prioritizing energy and memory efficiency while sacrificing performance. Hence, they used a Binary Convolutional Neural Network (BCNN) with only 3 layers which achieved an accuracy of 90.29%, the sensitivity of 90.41%, and specificity of 90.16%. Another work in [53] used a CNN with an early exit strategy to develop a wearable device solution for MI detection. They applied neural architecture search to find an optimal network suitable for the wearable device while considering performance, energy efficiency, and memory efficiency as design objectives. Their baseline architecture achieved better performance (accuracy = 98.03%, sensitivity = 97.26%, specificity = 98.82%) than other wearable device solutions. Interestingly, their architecture with early exit strategy achieved a slightly better performance (accuracy = 98.54%, sensitivity = 97.66%, specificity = 99.44%) than the baseline, which may be attributed to the fact that the earlier exit's accuracy had a similar performance to the baseline one. Unfortunately, the details of their implemented architectures are not mentioned. However, their exit decision-making is based on the classification confidence only, potentially leading to a deterioration

in the overall performance when there is a considerable performance mismatch between the earlier layer's exit and the baseline's one, as will be detailed later in Section 2.6.1. In another work, the authors in [55] applied a deep LSTM architecture to detect MI using single lead (II) ECG data from the recently introduced PTB-XL dataset [61]. They achieved an accuracy, sensitivity and specificity of 84.17% ,78.37%, and 87.55%, respectively. However, their sizable architecture is not suitable for resource constrained wearable devices. In this chapter, we consider three metrics (performance, energy, memory) when designing a solution for wearable devices, and accordingly, we propose a methodology for implementing a CNN architecture that is energy- and memory-efficient while providing the maximum performance possible.

2.3.2 Energy-efficient CNN Design Approaches

Originally designed for computer vision applications, CNN has been widely adopted in other applications such as natural language processing, biomedical applications. Sometimes the deep learning nature of CNN allows the architecture size to grow extremely large having 100s of convolutional, pooling, and fully connected layers. However, such an architecture is very computationally expensive and not suitable for energy and memory constraint applications of mobile health such as wearable devices. Therefore, many researchers have been working on different approaches to reduce the architecture size to make it energy and memory-efficient while maintaining similar or competitive performance. Such approaches can be broadly classified into two categories - 1) Software-based approach, 2) Hardware-based approach.

Software-based approaches mainly focus on minimizing the network size or developing techniques to satisfy the energy or memory constraints while maintaining performance. The software-based approaches can be further classified into 2 phases - 1) Offline or training phase, 2) Online or inference phase. The software-based approaches in the training phase

can be broadly divided into 3 types - a) Neural Architecture Search (NAS), b) Network Pruning, c) Model Compression. NAS involves automatically finding the optimum network parameters from a search space using reinforcement learning [66, 67] or gradient-based methods [68] or multi-objective bayesian optimization [69, 70]. Network pruning involves random pruning of a portion of the big network, retraining/fine-tuning it, and repeat the process until it achieves the desired performance [71–73]. Finally, model compression involves binarization [74] or quantization [75, 76] of network weights to reduce the model size to make it memory-efficient. Whatever the methods are applied, the final model is considered as the baseline classifier to be used in the inference phase.

The software-based approach in the inference phase includes dynamic network pruning [77], slimmable neural network [78], dynamic quantization [79], and early exit strategies [57–60]. The dynamic network pruning [77] approach prunes the baseline network (weights/neurons) during the inference phase. Unlike pruning during the training phase, dynamic pruning does not perform retraining/fine-tuning after the pruning step and the network may suffer from performance loss. The slimmable neural network [78] on the other hand uses the same network but with a reduced number of filter kernels or active channels during the inference phase. This is similar to the dynamic pruning of neurons. In practice, Both dynamic pruning and slimmable neural network are suitable for large network architectures with redundant weights/neurons that can be pruned without a substantial performance degradation. However, due to energy and memory constraints, such large architectures (with redundant weights/neurons) are not suitable for wearable devices in the first place, that is, wearable devices require efficient and compact architectures with minimal/no redundant weights/neurons to whom the application of dynamic pruning or slimming may degrade performance considerably. This could have been tackled by fine-tuning/ retraining the network in the inference phase, however, this will incur significant computation overhead for the wearable device which may overshadow the benefits achieved from it. On the other hand, dynamic quantization [79] involves the quantization of the network (weights and activations)

in the inference phase. Similarly, the weights are not retrained after quantization, potentially exacerbating performance losses due to quantization errors. The early exit strategies [57–60] leverage the layered architecture of the neural network and introduce multiple exit (output) layers in the network. However, as mentioned in Section 2.2.2, the state-of-the-art early exit strategies have inherent limitations which prevent them from being adopted for wearable device solutions. Our work addresses those limitations by introducing template matching based early exit (TMEX) architecture. It implements an output block selector as the decision function that addresses the limitations of state-of-the-art early exit strategies by using simple and yet effective template matching by Pearson correlation coefficient.

It is to note that, the software-based approaches for the inference phase are independent of each other and may also be applied together. For example, a model can be dynamically quantized and pruned and then an early exit strategy can be adopted at the same time. As our proposed baseline architecture is designed for wearable devices and already compact, further pruning and quantization during inference phase leads to performance loss as will be shown later in 2.6.3. Therefore, in this chapter we implement our TMEX architecture directly on the baseline architecture.

On the other hand, hardware-based approaches focus on the design of custom hardware such as accelerators which are specifically designed for CNNs [80, 81]. These accelerators facilitate speeding up the inference process and thereby making it more energy-efficient. Both Software-based and Hardware-based approaches are independent of each other and can be either applied separately or combined in a HW/SW co-design-based approach. However, Software-based approaches are more commonly adopted as they can be applied to all commercially available computing platforms (CPUs, GPUs, MCUs) and do not require any customized hardware.

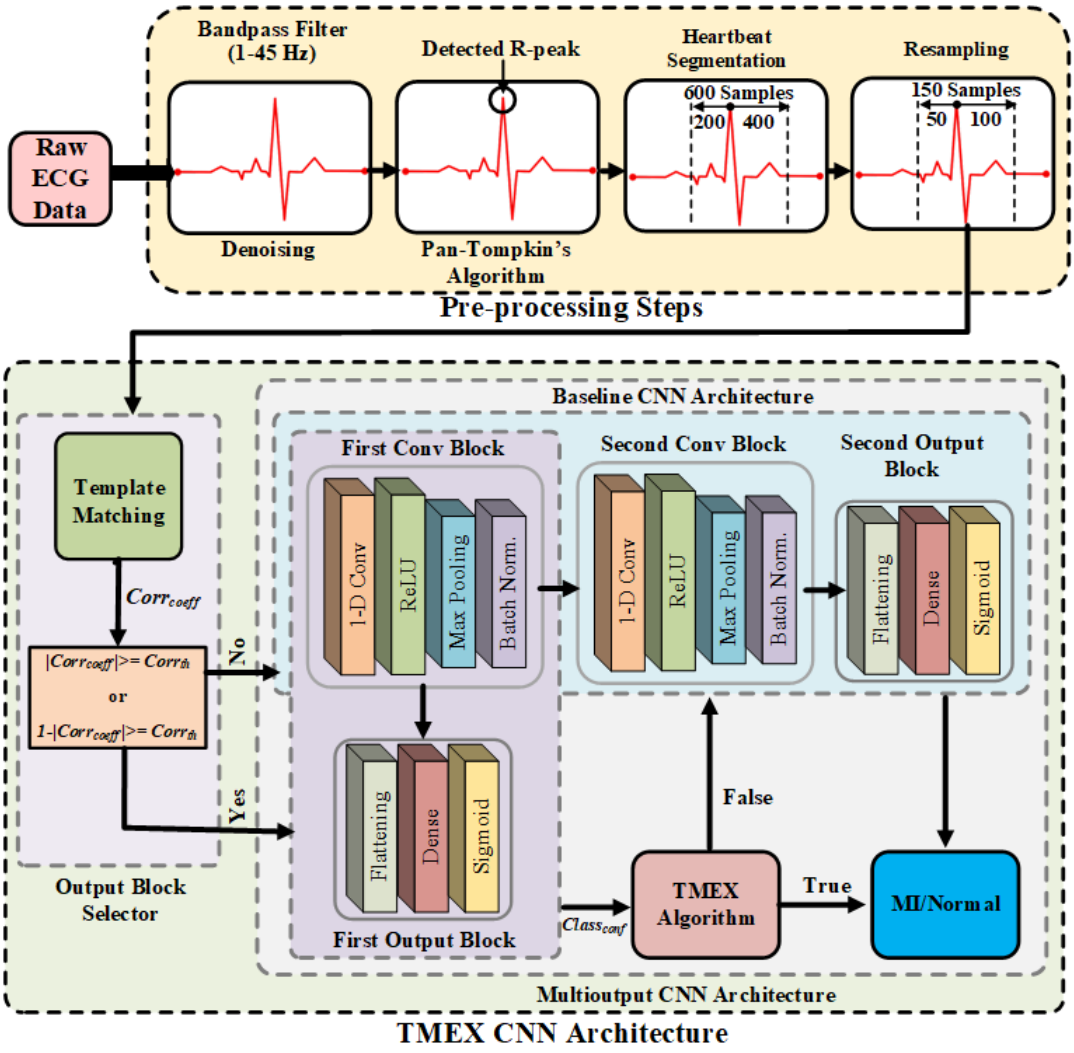


Figure 2.2: Overview of Our Proposed Methodology. Baseline CNN architecture refers to the full architecture with no dynamic model compression or early exit techniques applied to it.

2.4 Proposed Methodology

The following sections provide the details of our proposed methodology. The overview of our proposed methodology is demonstrated in Figure A.15.

2.4.1 Pre-processing Steps

Filtering

As shown in Figure A.15, the pre-processing step starts with the denoising of raw ECG data using a tenth-order Butterworth bandpass filter with cut-off frequencies f_1=1Hz and f_2=45Hz.

R-peak Detection

Once denoised, Pan-Tompkin's algorithm [82] is used to detect the R-peaks from the filtered ECG data.

Segmentation

Given a frequency value f, we take $f/5$ samples before and $2f/5$ samples after the R-peak. Thus, PTB dataset (f=1KHz) each segment consists of 600 samples and for PTB-XL dataset (f=500Hz) each segment consists of 300 samples representing a heartbeat.

Resampling

This step contributes a lot to the energy efficiency of our solution by reducing the number of samples to be processed by the CNN architecture for each segment. Therefore, it is important to carefully determine how much reduction is possible without compromising the performance. We determine the important frequencies from the power spectrum of the heartbeat segment. As shown in Figure 2.3, all the necessary frequency components are present within 125 Hz of the signal which means a sampling frequency of 250 Hz is good enough based on Nyquist theorem [83]. This is one-fourth of the sampling frequency used in

the PTB dataset (1 kHz) and half of the sampling frequency used in the PTB-XL dataset (500 Hz). Therefore, we can resample each PTB heartbeat segments by one-fourth and PTB-XL segments by half. Thus, the resampled segments contain 150 samples. The resampling is done by applying an anti-aliasing low pass filter to each segment using Kaiser window method. Then the segments are downsampled by 4 times. We use the resample function available in MATLAB to perform this operation.

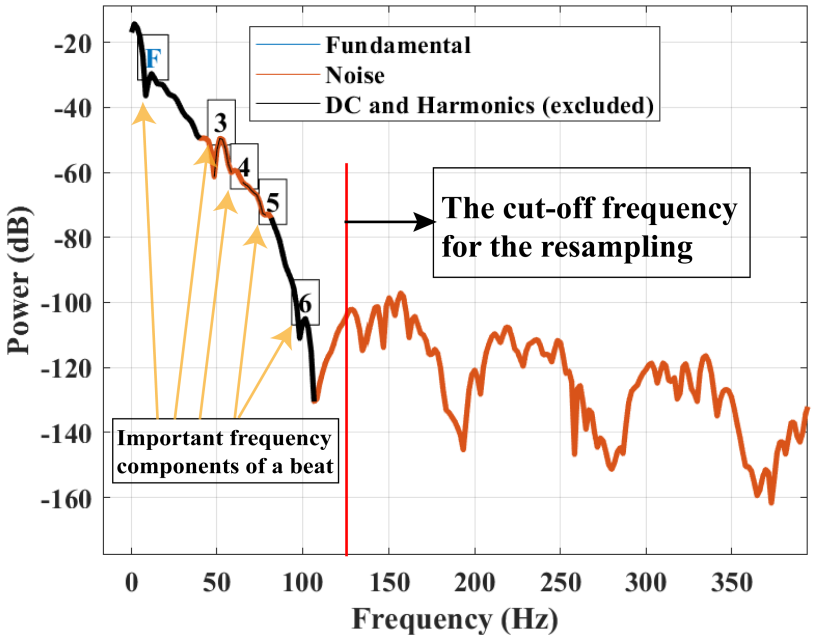


Figure 2.3: The Power Spectrum of Filtered ECG Signal

2.4.2 TMEX CNN Architecture

As shown in Figure A.15, our TMEX CNN architecture consists of three parts - 1) Multi-output CNN architecture that classifies the heartbeat segments, 2) Output block selector that decides which output block to start with based on the correlation of each heartbeat segment against the template beat, 3) TMEX algorithm that uses the correlation coefficient from the output block selector as an additional exit condition along with the classification confidence of the FOB.

Table 2.2: Multi-output CNN Architecture Details

Layer Name	**Kernel Size**	**Stride Size**	**Activation Function**	**Output Shape**	**# of Param.**
Input	-	-	-	150x1	0
Conv 1	31	1	ReLU	120x3	96
Pooling 1	2	2	-	60x3	0
Batch Norm.	-	-	-	60x3	12
Flatten 1	-	-	-	180	0
Dense 1	-	-	Sigmoid	1	181
Conv 2	7	1	ReLU	54x8	176
Pooling 2	2	2	-	27x8	0
Batch Norm	-	-	-	27x8	32
Flatten 2	-	-	-	216	0
Dense 2	-	-	Sigmoid	1	217
Total Number of Parameters after First Output Block					**289**
Total Number of Parameters after Second Output Block					**533**

Multi-output CNN Architecture

The multi-output CNN architecture is designed considering the resource constraints of the wearable devices. It consists of 2 convolution blocks and 2 output blocks where each convolution block is followed by one output block. Figure A.15 shows the architecture layout of our multi-output CNN architecture. Each convolution block consists of one convolution layer which is passed through ReLU activation, one max-pooling layer, and one batch normalization layer. Each output block consists of one flattening layer, and one dense layer which is passed through sigmoid activation. The details of the architecture parameters for each of the layers are given in Table 3.4. As shown in Table 3.4, the total number of parameters required to classify a heartbeat segment after the first, and second output block is 289, and 533, respectively.The two convolution blocks along with the second output block are considered as the baseline architecture in this chapter.

Output Block Selector

The Output Block Selector (OBS) is used to avoid the brute force method followed by traditional early exit CNN architecture. It implements the template matching mechanism using the Pearson correlation coefficient to determine whether we should try to exit FOB or not. We create a template beat from the average of 10 random MI heartbeat segments. It is to be noted that one can use a different number of heartbeat segments to create the template beat. However, using very few heartbeats (like 2 or 3) may not generalize the MI heartbeat template and may get biased to specific patients. Similarly, using too many heartbeat segments (100 or 200) might lose the MI pattern in the template beat. We also tried with 20 and 30 random beats to calculate the template beat. However, found no significant change in the performance of our TMEX architecture. Therefore, we decided to use the 10 random beats as it generalizes well and at the same time maintains the MI pattern on the template beat. If the absolute value of the Pearson correlation coefficient [84] between a segment and the template beat is greater than the correlation threshold, $Corr_{th}$, that means the segment is less complex and have a higher chance to be correctly classified by the FOB. Therefore, it selects the FOB to classify the segment. Otherwise, it directly uses the baseline architecture to classify a particular segment. It is to note that, even if the OBS decides to use the FOB, it does not guarantee early exit. The early exit decision is made by the TMEX algorithm as discussed in Section 2.4.2.

TMEX Algorithm

TMEX algorithm is used to make the final decision of early exit after FOB. Algorithm 1 shows the stepwise procedure followed to make the exit decision. As the template beat is created from MI heartbeats, $1 - |Corr_{coeff}|$ represents the correlation coefficient for the normal heartbeats. Similarly, as we are using the sigmoid function at the output block, $Class_{conf}$

Algorithm 1: TMEX Algorithm

Input: $Corr_{coeff}$: Pearson Correlation Coefficient from the OBS
Input: $Class_{conf}$: Classification confidence of the FOB
Output: $Exit_{flag}$: Early exit decision

Constant Variables:
$Corr_{th}$: Correlation Threshold used by OBS
$Conf_{th}$: Confidence Threshold for FOB
`// For MI segments`
if $|Corr_{coeff}| \geq Corr_{th}$ *and* $Class_{conf} \geq Conf_{th}$ **then**
    $Exit_{flag}$=True
end
`// For Normal segments`
else if $1 - |Corr_{coeff}| \geq Corr_{th}$ *and* $1 - Class_{conf} \geq Conf_{th}$ **then**
    $Exit_{flag}$=True
end
`// To move to next convolution block`
else
    $Exit_{flag}$=False
end
return $Exit_{flag}$

represents the classification confidence for MI segments. Therefore, $1 - Class_{conf}$ represents the classification confidence for normal segments. If both correlation co-efficient and classification confidence for either MI or normal segments are greater than the corresponding thresholds, the algorithm decides to exit early. Otherwise, it proceeds to the next convolution block. The use of both the correlation coefficient and the confidence thresholds allows us to overcome the limitations of traditional early exit architecture when the exit decision is solely based on classification confidence. For example, if FOB misclassifies a segment with high confidence it will exit after the FOB.

2.5 Experimental Setup

2.5.1 Database Used

We use two well known datasets – PTB diagnostic ECG database [56] and PTB-XL datastet [61] from PhysioNet [62]. The PTB database contains MI data from 148 subjects and normal

healthy control data from 52 subjects whereas the PTB-XL contains MI data from 5486 patients and healthy data from 9528 normal subjects. Each record includes 15 simultaneously measured signals: the conventional 12 leads (i, ii, iii, avr, avl, avf, v1, v2, v3, v4, v5, v6) together with the 3 Frank lead ECGs (vx, vy, vz). As our work is intended for the wearable device we use the single lead ECG data. We use the 11^{th} lead (v5) from PTB dataset and 2^{nd} lead (II) from PTB-XL dataset, to ensure fair comparison with the related works on the corresponding datasets. The signal in PTB and PTB-XL dataset is digitized at 1000 and 500 samples per second respectively. Table A.1 shows the summary of data distribution and the specific lead used in each of the related works.

2.5.2 Performance Metric

As the number of segments for different classes in both the dataset is highly imbalanced, only classification accuracy is not appropriate to measure performance. We use both the sensitivity and specificity metric to ensure a fair comparison with our related works as shown below:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.1}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{2.2}$$

$$Specificity = \frac{TN}{TN + FP} \tag{2.3}$$

Where TP, TN, FP, FN represents True Positives, True Negatives, False Positives, and False Negatives respectively. Sensitivity represents the true positive rate that measures the portion of the positive class (MI segments) that is correctly classified. Similarly, specificity represents the true negative rate that measures the portion of the negative class (Normal segments) that is classified correctly.

2.5.3 Training Multi-output CNN Classifier

To validate the performance of our Multi-output CNN classifier, we use data of 200 subjects (52 Normal, 148 MI) from PTB [56]. A total of 62306 (51880 MI, 10426 Normal) heartbeat segments are obtained after pre-processing steps from Lead 11 ECG data. The PTB dataset is highly imbalanced since the number of MI segments is approximately 5 times the normal segments. For the PTB-XL dataset data from 15014 subjects (9528 Normal, 5486 MI). Unlike PTB, PTB-XL provides a specific distribution of train, validation, test subjects to ensure a fair comparison of different related works. Thus, training and test data comes from completely different subjects. After pre-processing steps, we obtained 136136 (83765 Normal, 52371 MI) train segments, 17186 (10536 Normal, 6650 MI) validation segments, and 17319 (10575 Normal, 6744 MI) test segments. The PTB-XL dataset is also imbalanced but towards normal segments. To ensure proper training on the imbalance dataset, we assign class weights to each class during training using the following formula in Eq. A.9.

$$w_i = \frac{1}{N_i} * \frac{N}{n_c} \tag{2.4}$$

Here, w_i, and N_i represent the class weight and the number of segments belonging to class i, respectively. N is the total number of segments from all classes and n_c is the number of output classes which is 2 in our case.

We train the classifier with a batch size of 500. The models are trained for 150 epochs and select the model with minimum validation loss as the best one from those epochs. We use *Binary Crossentropy* as the loss function for each output block. *Adam* optimizer is used to train the models with a learning rate of .001. Similar to the related works in PTB, we also perform stratified 5-fold cross-validation on the total data where 80% data (4 folds) is used for training and 20% is used for testing. Moreover, 20% of the training data is used validation during training. Stratified 5-fold cross-validation ensures each fold contains

a similar distribution of data from each class. For the PTB-XL dataset, the model was trained, validated, and tested using the provided distribution of train, validation, and test subjects data.

2.5.4 Target Wearable Device

Our work is designed for low-power, low-memory wearable devices like SmartCardia [85]. For example, SmartCardia INYU [86] device is equipped with an ultra-low-power 32-bit microcontroller STM32L151 containing an ARM Cortex–M3 with a maximum clock rate of 32 MHz. It has 48 KB RAM, 384 KB Flash, and a standard 710 mAh battery. The device captures ECG signals using a single lead ECG sensor [86]. Our solution applies to any wearable device having the above or similar specifications.

2.6 Experimental Results and Analysis

2.6.1 Correlation and Confidence Threshold Analysis of TMEX Architecture

The correlation threshold, $Corr_{th}$ of the output block selector (OBS), and the confidence threshold, $Conf_{th}$ of the first output block (FOB) play the most important role in our TMEX architecture. Therefore, we conduct a detailed analysis of the different combinations of these two thresholds and their impact on the early exit decision, performance and energy efficiency. For the second output block, we always use the confidence threshold of 0.5 as it is the last output block in our architecture. Tables 2.3 and 2.4 show the performance and the different output blocks used to classify the segments of PTB and PTB-XL test dataset, respectively. We have also analysed the execution time, power, energy, and associated energy

Table 2.3: Correlation and Confidence Threshold Analysis of TMEX Architecture on PTB Dataset

Work	**OBS**	**FOB**	**Performance (%)**			**Output Block**		**Time**	**Pwr.**	**Energy**	**Saved**
	$Corr_{th}$	$Conf_{th}$	**Acc.**	**Sen.**	**Spec.**	**First**	**Second**	**(ms)**	**(mW)**	**(μJ)**	**(%)**
Baseline	–	–	99.33	99.25	99.74	0	62306	150.77	46.75	7048.49	0
Early Exit	–	0.5	94.82	94.22	97.83	62306	0	110.50	44.76	4945.98	29.83
		0.6	95.09	94.51	98.01	61860	446	110.81	44.77	4960.78	29.62
		0.7	95.37	94.80	98.23	61379	927	111.15	44.78	4976.74	29.39
		0.8	95.68	95.15	98.33	60691	1615	111.63	44.79	4999.59	29.07
		0.9	96.12	95.65	98.47	59434	2872	112.50	44.81	5041.36	28.48
Template Matching Based Early Exit (TMEX)	0.5	0.5	98.85	99.06	97.78	39740	22566	126.24	45.16	5700.88	19.12
		0.6	98.88	99.06	97.99	39494	22812	126.41	45.16	5709.18	19.00
		0.7	98.92	99.07	98.21	39238	23068	126.59	45.17	5717.81	18.88
		0.8	98.96	99.09	98.31	38855	23451	126.85	45.18	5730.74	18.70
		0.9	99.00	99.11	98.45	38139	24167	127.35	45.19	5754.91	18.35
	0.6	0.5	98.89	99.11	97.78	36097	26209	128.78	45.22	5823.93	17.37
		0.6	98.92	99.11	97.99	35866	26440	128.94	45.23	5831.74	17.26
		0.7	98.96	99.11	98.21	35618	26688	129.11	45.23	5840.13	17.14
		0.8	98.99	99.12	98.33	35264	27042	129.36	45.24	5852.11	16.97
		0.9	99.03	99.14	98.47	34598	27708	129.82	45.25	5874.65	16.65
	0.7	0.5	99.06	99.15	98.60	30006	32300	133.02	45.33	6030.40	14.44
		0.6	99.08	99.14	98.77	29814	32492	133.16	45.34	6036.92	14.35
		0.7	99.11	99.15	98.94	29607	32699	133.30	45.34	6043.95	14.25
		0.8	99.14	99.15	99.05	29304	33002	133.51	45.35	6054.25	14.11
		0.9	99.16	99.16	99.16	28726	33580	133.92	45.36	6073.90	13.83
	0.8	0.5	99.20	99.17	99.33	17723	44583	141.59	45.55	6449.55	8.50
		0.6	99.20	99.16	99.40	17621	44685	141.66	45.55	6453.05	8.45
		0.7	99.21	99.16	99.44	17509	44797	141.74	45.55	6456.88	8.39
		0.8	99.23	99.17	99.52	17326	44980	141.87	45.56	6463.16	8.30
		0.9	**99.24**	**99.18**	**99.55**	**16952**	**45354**	**142.13**	**45.56**	**6475.98**	**8.12**
	0.9	0.5	99.22	99.13	99.71	1260	61046	153.07	45.84	7017.20	0.44
		0.6	99.22	99.13	99.71	1256	61050	153.07	45.84	7017.33	0.44
		0.7	99.23	99.13	99.72	1251	61055	153.08	45.84	7017.51	0.44
		0.8	99.25	99.15	99.74	1241	61065	153.08	45.84	7017.85	0.43
		0.9	99.26	99.16	99.74	1229	61077	153.09	45.84	7018.27	0.43

saving achieved by the TMEX architecture for different $Corr_{th}$ and $Conf_{th}$. The details of the energy calculation is provided later in Section 2.6.6. As shown in Tables 2.3 and 2.4, the increasing value of $Corr_{th}$ increase the performance while increasing the energy consumption which in turn reduces the amount of energy saved by the TMEX architecture. This is because the OBS selects more segments to be classified by the second output block which has better performance but requires more energy. For each $Corr_{th}$ value, the same happens with the increase of the $Conf_{th}$ value. The performance increases at the cost of energy. This happens as the TMEX algorithm does not allow early exit after FOB as it requires higher classification confidence. Rather they are sent to the second output block which has better performance and higher energy requirement. This is why the number of segments classified by FOB decreases which increases the count for second output blocks. Tables 2.3 and 2.4 show that,

Table 2.4: Correlation and Confidence Threshold Analysis of TMEX Architecture on PTB-XL Test Data

Work	OBS	FOB	Performance (%)			Output Block		Time	Pwr.	Energy	Saved
	$Corr_{th}$	$Conf_{th}$	Acc.	Sen.	Spec.	First	Second	(ms)	(mW)	(μJ)	(%)
Baseline	–	–	84.36	78.60	88.03	0	17319	150.77	46.75	7048.49	0
Early Exit	–	0.5	84.12	74.02	90.55	17319	0	110.50	44.76	4945.98	29.83
		0.6	84.15	74.32	90.42	17108	211	111.03	44.77	4971.17	29.47
		0.7	84.18	74.57	90.32	16868	451	111.63	44.79	4999.84	29.07
		0.8	84.16	74.96	90.03	16557	762	112.41	44.81	5037.01	28.54
		0.9	84.05	75.37	89.59	16147	1172	113.44	44.83	5086.07	27.84
Template Matching Based Early Exit (TMEX)	0.5	0.5	84.08	74.87	89.95	12002	5317	123.84	45.10	5585.06	20.76
		0.6	84.05	74.97	89.84	11904	5415	124.09	45.11	5596.92	20.59
		0.7	84.09	75.22	89.75	11788	5531	124.38	45.11	5610.97	20.39
		0.8	84.10	75.61	89.52	11617	5702	124.81	45.12	5631.69	20.10
		0.9	84.05	76.11	89.11	11380	5939	125.40	45.14	5660.41	19.69
	0.6	0.5	84.16	75.43	89.72	9854	7465	129.23	45.24	5845.81	17.06
		0.6	84.11	75.53	89.58	9776	7543	129.42	45.24	5855.30	16.93
		0.7	84.11	75.77	89.43	9685	7634	129.65	45.25	5866.38	16.77
		0.8	84.06	76.11	89.13	9552	7767	129.99	45.26	5882.58	16.54
		0.9	84.02	76.62	88.75	9355	7964	130.48	45.27	5906.59	16.20
	0.7	0.5	84.21	75.98	89.47	6930	10389	136.56	45.42	6203.12	11.99
		0.6	84.17	76.08	89.33	6881	10438	136.69	45.43	6209.14	11.91
		0.7	84.16	76.29	89.18	6811	10508	136.86	45.43	6217.72	11.79
		0.8	84.07	76.53	88.88	6710	10609	137.12	45.44	6230.12	11.61
		0.9	84.05	77.00	88.55	6570	10749	137.47	45.45	6247.31	11.37
	0.8	0.5	84.50	76.81	89.41	3948	13371	144.05	45.61	6570.35	6.78
		0.6	84.47	76.93	89.28	3916	13403	144.13	45.62	6574.31	6.73
		0.7	84.42	77.05	89.13	3869	13450	144.24	45.62	6580.12	6.64
		0.8	84.31	77.19	88.84	3802	13517	144.41	45.62	6588.40	6.53
		0.9	**84.24**	**77.46**	**88.56**	**3707**	**13612**	**144.65**	**45.63**	**6600.15**	**6.36**
	0.9	0.5	84.55	77.65	88.95	1430	15889	150.36	45.77	6882.65	2.35
		0.6	84.54	77.73	88.88	1414	15905	150.40	45.77	6884.64	2.32
		0.7	84.51	77.77	88.80	1395	15924	150.45	45.78	6887.01	2.29
		0.8	84.40	77.85	88.58	1354	15965	150.55	45.78	6892.11	2.22
		0.9	84.30	77.91	88.38	1308	16011	150.67	45.78	6897.84	2.14

for $Corr_{th}$=0.5 and $Conf_{th}$=0.5, our TMEX archiecture achieves the lowest performance with the highest energy saving of 19.12% and 20.76% energy efficiency for the PTB and PTB-XL test dataset, respectively. Conversely, for $Corr_{th}$=0.9 and $Conf_{th}$=0.9, it achieves the highest performance with lowest energy efficiency of 0.43% and 2.14% on the respective datasets. This proves that there is a trade off between performance and energy efficiency for different values of correlation and confidence thresholds. Therefore, our goal is to find out the best combination of these two thresholds that maintains a similar performance of the TMEX architecture as the baseline one while providing descent energy efficiency. As shown in Tables 2.3 and 2.4, $Corr_{th} = 0.8$ and $Conf_{th} = 0.9$ achieves the best performance considering all three metrics while maintaining a decent energy efficiency of 8.12% and 6.36% for the respective datasets. Therefore, we consider these two values for the correlation threshold

of OBS and the confidence threshold of FOB in our TMEX architecture. However, TMEX architecture provides flexibility to achieve more energy efficiency at the cost of performance by choosing lower threshold values.

2.6.2 Performance Analysis of the Traditional Early Exit

To demonstrate the importance of the template matching based early exit architecture, we analyze the performance of the traditional early exit where exit decision is made solely based on the FOB confidence threshold. Tables 2.3 and 2.4 show the performance of traditional early exit architecture for different confidence thresholds ($Conf_{th} = 0.5$ to 0.9). As the tables show, the performance of traditional early exit increases with the increasing value confidence thresholds from 0.5 to 0.9. For the PTB dataset in Table 2.3, the best performance of traditional early exit architecture (for $Conf_{th} = 0.9$) is still lower than that of the lowest performance of our TMEX architecture (for $Corr_{th} = 0.5$, $Conf_{th} = 0.5$). And for the PTB-XL test dataset in Table 2.4, our TMEX architecture starting from $Corr_{th} = 0.5$ and $Conf_{th} = 0.8$ or any combination upto 0.9, outperforms traditional early exit architecture. However, traditional early exit architecture achieves higher energy efficiency while sacrificing the performance. They achieve upto 29.83% of energy saving compared to the 19.12% and 20.76% for our TMEX architecture. On the other hand, TMEX architecture ensures similar performance as the baseline while being as energy-efficient as possible.

2.6.3 Performance Analysis of the Other Dynamic Model Compression Techniques

This section evaluates the performance of other dynamic model compression techniques on our baseline architecture. As shown in Table 2.5, for all the dynamic compression techniques

the performance of the baseline architecture decreases to some extent. For slimmable neural network and dynamic pruning we have applied compression on the weights/neurons of the two convolution layers and the dense layer was kept intact. This is because, compressing dense layer causes even more performance degradation. For slimmable neural network, we have considered two configurations. In the first configuration we have slimmed the lowest magnitude filter kernel from the first convolution layer thus using only 2 of the 3 filter kernels. Similarly, in the second configuration we have slimmed the lowest magnitude filter kernel from the second layer thus using 7 of the 8 filter kernels. For dynamic weight pruning, we have pruned 20% of the weights with lowest magnitude. And for dynamic neuron pruning we have pruned 20% of the neurons with lowest magnitude. As shown in the Table 2.5, the baseline performance drops significantly for both the slimming and pruning options. This justifies the fact that our baseline architecture is already compact and does not have much redundant weights/neurons to prune or slim. On the other hand, dynamic quantization shows a better performance compared to pruning or slimming, as it does not change the baseline architecture during inference. Rather, it just quantizes the network weights while keeping the architecture intact. Similarly, traditional early exit architecture also suffers from performance loss compared to baseline one. And TMEX architecture outperforms other techniques while maintaining similar performance as the baseline one. It is to note that, the dynamic model compression techniques are independent of each other and can be used simultaneously. Our TMEX architecture can be implemented on a quantized and pruned version of the baseline architecture. However, we chose to implement the TMEX architecture on the baseline directly as other compression techniques reduce the baseline performance significantly.

Table 2.5: Performance Comparison with Dynamic Model Compression Techniques

Technique	Configuration	PTB			PTB-XL Test		
		Acc.	Sen.	Spec.	Acc.	Sen.	Spec.
Baseline [Ours]	–	**99.33**	**99.25**	**99.74**	**84.36**	**78.60**	**88.03**
TMEX [Ours]	$Corr_{th}$=0.8, $Conf_{th}$=0.9	**99.24**	**99.18**	**99.54**	**84.24**	**77.46**	**88.56**
Slimmable NN [78]	Filter: Conv1=2; Conv2=8	87.21	98.50	31.03	69.92	88.67	57.96
	Filter: Conv1=3; Conv2=7	95.47	99.25	76.61	66.44	93.29	49.31
Dynamic Pruning [77]	Weight Pruning (20%)	95.77	98.97	79.84	84.20	76.52	89.10
	Neuron Pruning (20%)	82.98	99.42	1.16	74.06	89.39	64.29
Dynamic Quantization[79]	int8	97.33	97.05	98.72	83.53	76.01	89.69
Trad. Early Exit [57, 59]	$Conf_{th}$=0.9	96.12	95.64	98.46	84.05	75.37	89.59

2.6.4 Performance Evaluation against Related Works on MI Detection

As shown in Tables A.1 and 2.7, our baseline architecture outperforms the related works on MI detection for both PTB and PTB-XL datasets. For PTB dataset (Table A.1), our baseline architecture achieves an accuracy of **99.33%**, sensitivity of **99.25%**, and specificity of **99.74%**. Our TMEX architecture shows a similar performance as the baseline one with an accuracy, sensitivity, and specificity of 99.24%, 94.18%, 99.54%, respectively. Both baseline and TMEX architectures significantly outperform the other state-of-the-art works [48–52] in almost all three metrics. Moreover, SVM and RF models, in general, are not suitable for wearable devices in terms of memory footprint which we will discuss in the next Section 2.6.5. The work [51] using deep CNN achieves comparatively better performance compared to other wearable device solutions [49, 50, 52]. However, our work still outperforms [51]

Table 2.6: Performance Comparison of Related Works on PTB Dataset

Work	PTB Patient			Classifier Type	Performance (%)		
	Normal	MI	Lead		Accuracy	Sensitivity	Specificity
[48]	52	148	11	k-NN	98.80	99.45	96.27
[49]	52	52	11	Full SVM	95	–	–
				2-level SVM	90	–	–
[50]	52	52	11	Full RF	83.26	87.95	78.82
				5-level RF	80.32	81.02	79.63
[51]	52	148	2	CNN	95.22	95.49	94.19
[52]	52	148	11	BCNN	90.29	90.41	90.16
[53]	52	148	11	Baseline CNN	98.03	97.26	98.82
				Early Exit ($Conf_{th}$=0.99)	98.54	97.66	99.44
Ours	52	148	11	FOB CNN	94.82	94.22	97.83
				Baseline CNN	**99.33**	**99.25**	**99.74**
				TMEX ($Corr_{th}$=0.8, $Conf_{th}$=0.9)	**99.24**	**99.18**	**99.54**

Table 2.7: Performance Comparison of Related Works on PTB-XL Dataset

Work	PTB-XL Patient			Classifier Type	Performance (%)		
	Normal	MI	Lead		Accuracy	Sensitivity	Specificity
[55]	9528	5486	2	Deep LSTM	84.17	78.37	87.55
Ours	9528	5486	2	FOB CNN	84.12	74.02	90.55
				Baseline CNN	**84.36**	**78.60**	**88.03**
				TMEX ($Corr_{th}$=0.8, $Conf_{th}$=0.9)	**84.24**	**77.46**	**88.56**

which is designed for clinical setups. The work [48] achieves the highest performance among the related works with accuracy, sensitivity, and specificity of 98.80% and 99.45%, and 96.27% respectively. However, the k-NN classifier is only suitable for clinical set up as all the training data should be loaded in the memory during the inference phase. The solution in [53] is implemented for wearable devices and achieves a close performance to ours. However, our architecture is much more energy-efficient than [53] which is detailed later in Section 2.6.6. As shown in Table 2.7, our baseline architecture also outperforms [55] on PTB-XL dataset achieving an accuracy, sensitivity, and specificity of 84.36%, 78.60.36%, and 88.03%, respectively. Our TMEX architecture shows a similar performance as the baseline one with an accuracy, sensitivity, and specificity of 84.24%, 77.46%, 88.56%, respectively. Moreover, the work [55] uses deep LSTM network which is not suitable for wearable devices.

2.6.5 Memory Footprint Evaluation on Real Hardware

We evaluate the memory footprint of all works mentioned in Table A.1 except for the work [48] that uses k-NN classifier. The k-NN classifier is not suitable for wearable devices as it requires all the training data to be loaded into the memory. For the machine learning approaches in [49, 50] the reported memory footprint is for the feature extraction and classification process. For the deep learning approaches using CNN [51, 52], we evaluate the memory footprint of the classification as it automatically extracts features during classification.

The work [49] and [50] uses 2-level SVM and a 5-level RF classifier respectively. Both

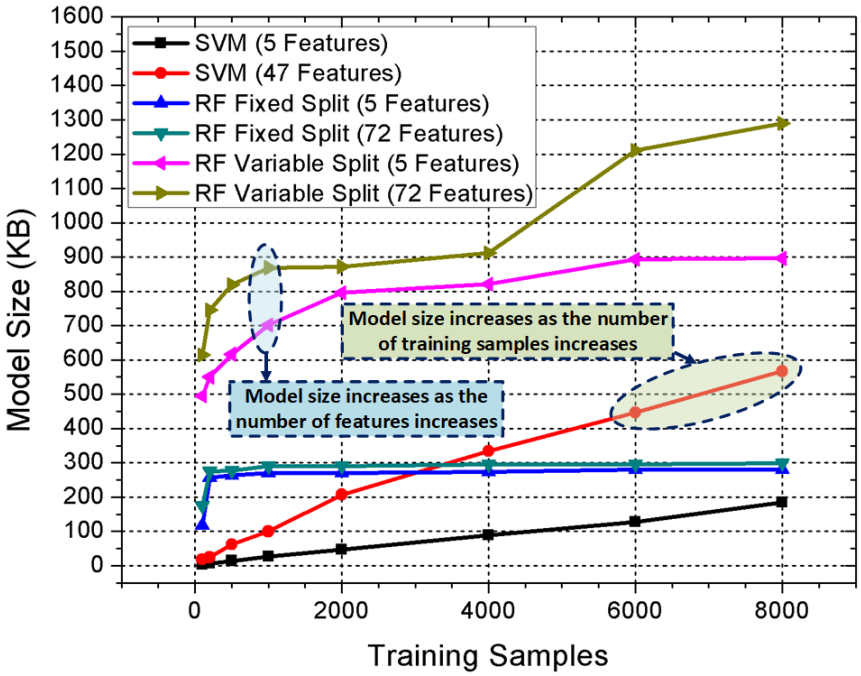


Figure 2.4: Variation of Model Size with Increasing Training Samples and Features

approaches have three major drawbacks in terms of memory. ***First***, the model size of the SVM and RF increases with the increasing amount of training data as well as the number of features. ***Second***, both of them need to have multiple models (2 for SVM and 5 for RF) loaded into memory. ***Third***, the feature extraction process for the full level SVM and RF classifier requires a huge amount of memory.

To demonstrate an example of the *first* drawback, we train the SVM and RF models with varying numbers of training samples. For SVM models, we train with both 5 (first level) and 47 (second/full) features. For RF models we train with 5 and 72 features corresponding to the first and fifth level classifiers in [50]. As shown in Figure 2.4, the size of the SVM model increases linearly with the increasing number of training samples. The model size also increases as the feature number increases from 5 to 47 for SVM. On the other hand, the RF model is trained with 100 weak learners as used in [50]. Moreover, they also used a variable-sized split for the RF where the tree grows until no further split is possible which also increases the model size exponentially. In Figure 2.4, we demonstrate the change of the

RF model size for both variable split and a fixed split of 20. As shown in Figure 2.4, the size of the RF model for both variable and fixed split increases with the increasing number of training samples as well as the increase of features from 5 to 72. Figure 2.4 shows that the SVM, fixed split RF, and variable split RF model trained with 8000 samples will have a model size of 185, 281, 896 KBs, respectively for only 5 features. For the same 8000 samples, the SVM model size increases from 185 KB to 567 KB when the number of features increases from 5 to 47. The same holds for the RF model with fixed and variable split as the number of features increases from 5 to 72, the model size changes from 281 KB and 896 KB to 299 KB and 1290 KB, respectively. Thus, both of these approaches are not suitable for low-memory wearable devices.

Moreover, the work [49] requires 2 of the SVM models and the work [50] requires 5 of the RF models to be loaded into the memory which gives us the perspective of the *second* drawback. Besides, both the approaches in [49, 50] require a huge amount of memory for the feature extraction process of their full level classifiers which brings us to the *third* drawback. We demonstrate the *third* drawback using the RAM footprint for each level of the SVM and RF classifier on the EFM32 Giant Gecko microcontroller which has 128 KB of RAM and 1 MB of flash memory. As our goal is to evaluate the memory requirement for the feature extraction of different levels of SVM and RF, we train the SVM with only 20 training samples and RF with only 100 training samples so that they can fit within the 128 KB of RAM. Also, for the RF model, we use only 10 weak learners with a fixed split of 10. As shown in Table 2.8, both the full level classifier in [49] and [50] requires almost 83 MB of RAM making them incompatible for wearable devices with lower memory.

One of the advantages of deep learning approaches in [51–53, 55] over the machine learning ones in [49, 50] is that the classifier model size does not change with the number of training samples. However, the memory requirement of the deep learning models still changes with architecture size, parameters, and input segment size. For example, the 11 layers deep CNN

Table 2.8: Memory Footprint and Energy Consumption Evaluation on EFM32 Giant Gecko Development Board

Works	**Classifier Level**	**RAM Footprint (B)**	**Exe. Time (ms)**	**Avg. Power (mW)**	**Energy (μJ)**
SVM[49]	First (5 Features)	78844	347.03	46.57	16161.18
	Full (47 Features)	Not Compatible: RAM Overflowed			
RF[50]	First (5 Features)	78844	345.35	46.57	16082.94
	Second (10 Features)	85948	3556.37	46.58	165655.71
	Third (15 Features)	87316	7669.7	46.81	359018.65
	Fourth (20 Features)	88132	8188.21	46.48	380588.00
	Full (72 Features)	Not Compatible: RAM Overflowed			
CNN[51]	-	114176	2036.82	46.97	95669.43
BCNN[52]	-	3568	253.73	44.45	11278.29
CNN[53]	Baseline	15972	–	–	28320
	Early Exit ($Conf_{th}$=0.99)	15972	–	–	28189
Deep LSTM[55]	–	Not Compatible: RAM Overflowed			
CNN[Ours]	OBS	2612	3.18	44.98	143.04
	FOB	11868	107.32	44.54	4780.03
	Baseline	**20160**	**150.77**	**46.75**	**7048.49**
	TMEX* (PTB)	**20160**	**142.13**	**45.56**	**6475.98**
	TMEX* (PTB-XL Test)	**20160**	**144.65**	**45.63**	**6600.15**

** TMEX energy measurements for $Corr_{th}$=0.8, $Conf_{th}$=0.9. Measurements for other thresholds are presented in Tables 2.3 and 2.4*

architecture used in [51] requires 114 KB of RAM. Whereas the work in [52] focuses on low memory wearable devices and requires only 3.5 KB of RAM. The approach in [52] focused on memory and energy efficiency while sacrificing performance. Authors in [53] developed wearable device solution with a RAM footprint of 15.97 KB. The deep LSTM architecture used in [55] has more than fourteen thousand parameters and encounters RAM overflow.

Table 2.8 shows the RAM footprint our baseline architecture is 20 KB. The OBS requires only 2.61 KB of RAM (which is less the 20 KB) without adding any extra memory overhead. The RAM footprint of our TMEX architecture is also 20 KB which is the maximum of RAM footprints of the output block selector and each of the output blocks. Thus, our proposed baseline and TMEX CNN architecture are compatible with any device with a minimum RAM of 32 KB.

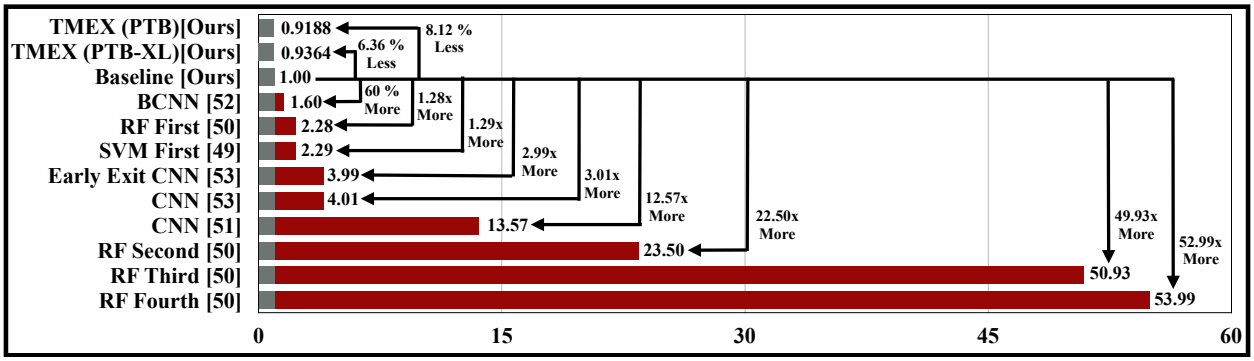


Figure 2.5: Comparison of Energy Consumption w.r.t Baseline Classifier

2.6.6 Energy Consumption Evaluation on Real Hardware

Evaluation of energy consumption is conducted on the same EFM32 Giant Gecko microcontroller as used for memory evaluation. The energy measurement is done using the integrated energy profiler [87] of the Simplicity Studio IDE for the EFM32 boards. While measuring the energy, the board is set to active/run mode (EM0-Energy Mode 0) with a clock speed of 48 MHz. To ensure a fair comparison, the same configuration is used for measuring the energy of all the state-of-the-art work implementations. For the machine learning approaches [49, 50], the energy for the feature extraction and classification are calculated. For deep learning ones using CNN, we evaluate the classification energy only as they automatically extract features during classification. The energy for the TMEX architecture is calculated through the summation of the products between the ratios segments classified by each output block and the corresponding energy consumption for that block. For the energy evaluation of work [49], we train the SVM classifier with only 20 training samples to just fit the trained model in the memory. Similarly, the RF classifier in [50] is trained with only 100 training samples, 10 weak learners with a fixed split of 10 just to fit the model into the RAM. The detailed analysis of execution time, power, and energy for one heartbeat segment are shown in Table 2.8.

Figure 2.5 shows a summary of the energy consumption of various works reported in Table

2.8. The results are normalized with respect to the energy consumption of our baseline CNN classifier. As shown in Figure 2.5, RF first, second, third, and fourth level classifiers in [50] consumes 1.28×, 22.50×, 49.93×, and 52.99× more energy compared to our baseline classifier. All the 20 features calculated in RF fourth level are the same as the first 20 features out of the 47 features in full SVM classifier of [49]. This indicates that the full SVM and RF classifiers will consume much more energy compared to our baseline classifier. Even the first level SVM classifier in [49] consumes 1.29× more energy compared to our baseline classifier. It is to be noted that we used a small number of training samples to train the SVM and RF models to keep the model size small. The energy consumption of SVM and RF classifiers, like the model size, will increase with the increase of training samples. The BCNN [52] which has the lowest energy consumption among the related works also consumes 60% more energy than our baseline classifier. The work [51] using CNN classifier consumes 12.57× more energy than the baseline one. The work [53] is designed for wearable devices and still consumes 3.01× more energy than ours. The early exit version of [53] also consumes 2.99× more energy than ours. It is to note that the early exit version of [53] does not provide any significant energy saving from its baseline. This is because they used a confidence threshold of 0.99 which probably causes most of the segments to be classified by the baseline architecture. That is why the performance of the early exit is also similar to the baseline (as shown in Table A.1) as it hardly uses the early exit option. On the other hand, our TMEX architecture (for $Corr_{th}$=0.8, $Conf_{th}$=0.9) for PTB-XL and PTB dataset consumes 6.36% and 8.12% less energy compared to our baseline architecture as shown in Figure 2.5. Moreover, for lower correlation and confidence threshold values our TMEX architecture can achieve upto 19.12% (PTB) and 20.76% (PTB-XL) energy efficiency while sacrificing some performance (but still outperforming the traditional early exit strategy).

Table 2.9: Performance and Resource Consumption Analysis of the Baseline Architecture W/O Resampling

Dataset	Baseline	Performance (%)			Param Count	FLOP Count	Time (ms)	Energy (μJ)	RAM (Bytes)
		Acc.	Sen.	Spec.					
PTB	W/O Resampling	**99.35**	**99.43**	**98.92**	1429	212665	446.82	21188.2	101760
	With Resampling	99.33	99.25	99.74	**533**	**43489**	**150.77**	**7048.49**	**20160**
PTB-XL	W/O Resampling	83.42	77.12	87.44	829	99865	224.40	10598.41	44880
	With Resampling	**84.36**	**78.60**	**88.03**	**533**	**43489**	**150.77**	**7048.49**	**20160**

2.6.7 Ablation Study of the Segment Resampling

We perform an ablation study without resampling the segment for our baseline architecture to demonstrate the efficacy of the segment resampling in our proposed methodology. As mentioned in Section 2.4.1, before resampling there are 600 and 300 samples in a heartbeat segment for PTB and PTB-XL dataset, respectively. Therefore, we trained our baseline architecture as discussed in Section 3.4.2 with an input size of 600 and 300 samples instead of 150. As shown in Table 2.9 for PTB dataset, the accuracy, sensitivity, and specificity of our baseline architecture with resampling is very similar to that without resampling. This shows that resampling a segment from 600 samples to 150 does not cause any significant performance loss. Rather, resampling reduces the parameter and FLOP count of the baseline architecture by 2.68× and 4.89×, respectively as shown in Table 2.9. This in turn makes the inference time 2.96× faster requiring 3.01× less energy. Moreover, resampling reduces the memory footprint by 5.05× thus making the baseline architecture memory-efficient. For PTB-XL dataset, our baseline architecture with resampling outperforms the one without resampling as shown in Table 2.9. Also, it reduces the – parameter count, FLOP count, inference time, energy consumption, and memory footprint of the baseline architecture by 1.56×, 2.30×, 1.49×, 1.50×, and 2.23×, respectively. The ablation study shows that resampling in our methodology helps to achieve energy and memory efficiency without significant performance loss making it suitable for resource constraint wearable devices.

2.7 Summary

This chapter proposes an energy-efficient methodology for real-time MI detection on wearable devices using a Convolutional Neural Network (CNN). It involves novel pre-processing of the heartbeat segments to reduce the sample size that allows the baseline CNN to outperform the state-of-the-art works for two different datasets - PTB and PTB-XL, while being energy and memory-efficient. Moreover, we also propose a Template Matching based Early Exit (TMEX) CNN architecture that further increases the energy efficiency compared to baseline architecture while maintaining similar performance. On PTB dataset, our baseline and TMEX architecture achieve 99.33% and 99.24% accuracy, whereas on PTB-XL dataset they achieve 84.36% and 84.24% accuracy, respectively. Evaluation on real hardware shows that our baseline architecture achieves from **0.6x** to **53x** more energy efficiency while outperforming state-of-the-art works on wearable devices. Moreover, our TMEX architecture further achieves 8.12% (PTB) and 6.36% (PTB-XL) more energy efficiency compared to the baseline architecture while maintaining similar performance. To the best of our knowledge, the baseline and TMEX architecture of our methodology achieve the best performance on wearable devices while being energy-efficient with a RAM footprint of only 20 KB.

Chapter 3

Energy-efficient Human Activity Recognition in Low-power Wearable Devices

3.1 Abstract

Human Activity Recognition (HAR) is one of the key applications of digital health that requires continuous use of wearable devices to track daily activities. This chapter proposes an Adaptive CNN for energy-efficient HAR (AHAR) suitable for low-power edge devices. Unlike traditional adaptive (early-exit) architecture that makes the early-exit decision based on classification confidence, AHAR proposes a novel adaptive architecture that uses an output block predictor to select a portion of the baseline architecture to use during the inference phase. Experimental results show that traditional adaptive architectures suffer from performance loss whereas our adaptive architecture provides similar or better performance as the baseline one while being energy-efficient. We validate our methodology in classifying loco-

motion activities from two datasets- Opportunity and w-HAR. Compared to the fog/cloud computing approaches for the Opportunity dataset, our baseline and adaptive architecture shows a comparable weighted F1 score of 91.79%, and 91.57%, respectively. For the w-HAR dataset, our baseline and adaptive architecture outperforms the state-of-the-art works with a weighted F1 score of 97.55%, and 97.64%, respectively. Evaluation on real hardware shows that our baseline architecture is significantly energy-efficient (422.38x less) and memory-efficient (14.29x less) compared to the works on the Opportunity dataset. For the w-HAR dataset, our baseline architecture requires 2.04x less energy and 2.18x less memory compared to the state-of-the-art work. Moreover, experimental results show that our adaptive architecture is 12.32% (Opportunity) and 11.14% (w-HAR) energy-efficient than our baseline while providing similar (Opportunity) or better (w-HAR) performance with no significant memory overhead. The findings in this chapter have been published in [88].

3.2 Introduction

Human Activity Recognition (HAR) applications are useful tools for health monitoring, fitness tracking, and patient rehabilitation [89–91]. Since the HAR applications need continuous sensor data to infer user activity, advances in sensor technology [92] have enabled wide adoption of HAR applications in daily life. Smartphones have been significantly used for HAR in the past decade [93–95]. However, this kind of solution requires the user to continuously carry the phone which causes inconvenience. Moreover, the smartphone solutions consume higher energy in the range of watts [96] which may hinder the primary use of the phones reducing the battery life.

Therefore, wearable devices have gained much popularity for HAR applications [97]. Moreover, the use of wearable devices enable remote monitoring of patients suffering from critical diseases like movement disorders in Parkinson's disease [91]. However, most of the solutions

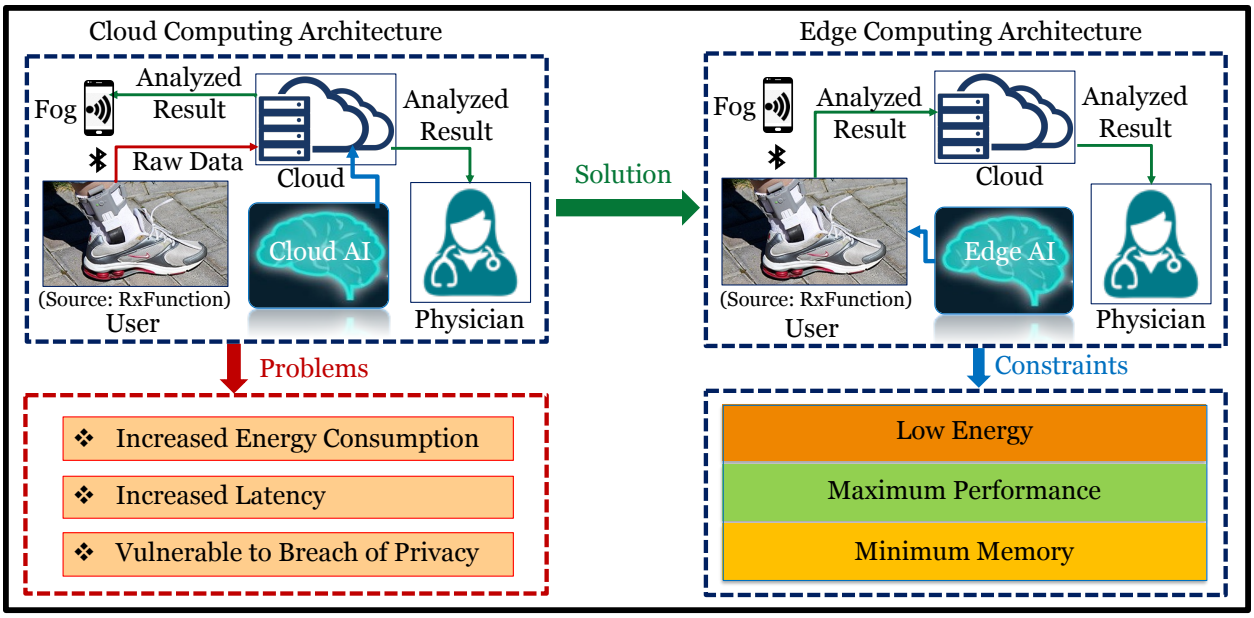


Figure 3.1: Shift from cloud computing to edge computing architecture

[98–106] using wearable devices follow a fog/cloud computing approach as shown in Figure 3.1. The collected data from wearable devices are sent over Bluetooth to a mobile phone (fog) [44] or remote server (cloud) where all the processing and classification takes place. The daily use of these devices generates vast amounts of raw data, and sending them over Bluetooth entails higher energy consumption [97]. Additionally, it also introduces latency, which is unsuitable for real-time monitoring. Moreover, passing the raw data to a mobile phone makes the users' data vulnerable to privacy breaches. Many researchers [107] followed a hierarchical approach where some simple activities are classified on the device where complex ones are transmitted over to the fog/cloud. Although this kind of solution saves computational energy to some extent, it still suffers from latency and privacy issues. Consequently, researchers shifted to an alternative architecture to overcome these limitations, which is called 'edge computing' [108], where all the processing is done on the device itself [52, 109]. Therefore, it reduces the energy consumption, latency, and vulnerability of privacy breaches. Figure 3.1 illustrates the shift from cloud to edge computing architecture.

The small form factor of wearable devices imposes three constraints on the processing algorithms as shown in Figure 3.1. The algorithms should consume low-energy, execute with

Table 3.1: Difference between Baseline and Adaptive Architecture for HAR Dataset

Architecture	Output block used	% of total segments	# of total segments	Correct classification (%)	Total FLOP count	Total exec. time (ms)	Total energy (μJ)
Baseline	**Second**	**100**	**4740**	**97.60**	**35,905,500**	**152,011.80**	**2,316,627.60**
Adaptive	First	97.13	4604	95.06	26,606,516	121,361.44	1,849,564.92
	Second	2.87	136	2.87	1,030,200	4,361.52	66,468.64
	Overall	**100**	**4740**	**97.93**	**27,636,716**	**125,722.96**	**1,916,033.56**
Total saving due to adaptive architecture					**8,268,784**	**26,288.84**	**400,594.04**
Average saving per segment due to adaptive architecture					**1,744.47**	**5.55**	**84.51**

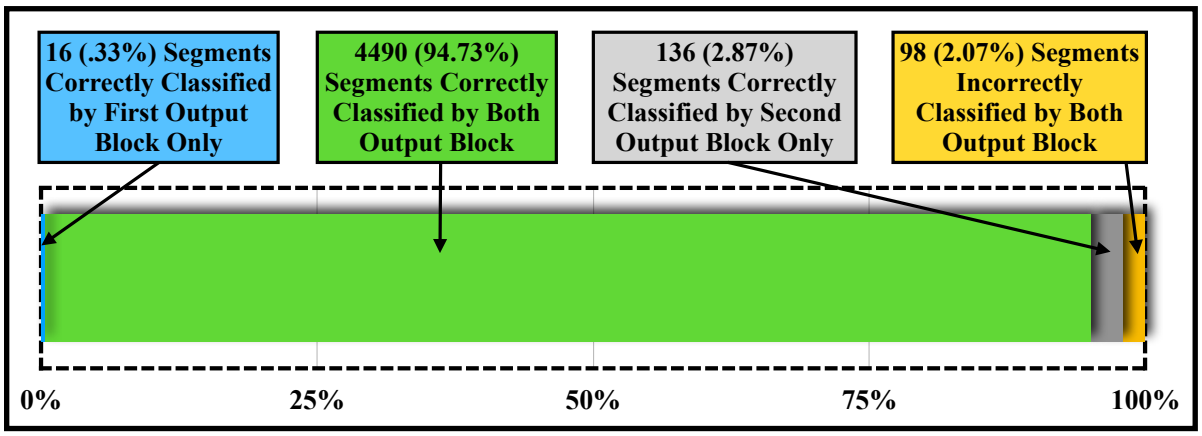


Figure 3.2: Blockwise multi-output CNN architecture performance breakdown

minimum-memory, and provide maximum performance within the previous two constraints. State-of-the-art works on HAR are mostly intended for fog/cloud platform where they use complex machine learning [98–100] and deep learning algorithms [101–106] to achieve high performance. They prioritize performance over the other two constraints, hence are not wearable device compatible. Machine learning algorithms perform classification based on the extracted features from the data which is often time and energy consuming, whereas, wearable device solutions should be fast and energy-efficient. Deep learning algorithms using Convolutional Neural Networks (CNN) [65, 110] have an advantage in this regard as they automatically extract features through convolution and do not require manual feature engineering or extraction. However, such deep networks require higher energy, memory, and execution time as they use a large number of layers. Therefore, for wearable device solutions CNN should be designed in such a way that satisfies the energy and memory constraints while maintaining reasonable performance. As CNN works in layers, it provides the flexi-

bility to design a network by adding or removing layers as necessary in the training phase which is used to classify data during the inference phase. However, the full architecture from the training phase may not be needed at the inference phase as many of the data may be correctly classified using only the first few layers of the architecture. Therefore, if we use a portion of the network as needed, it will help to avoid redundant operations of the CNN architecture leading to energy efficiency while maintaining the performance. This technique is called adaptive (early-exit) or Conditional Deep Learning Network (CDLN) architecture and was adopted by many researchers [57, 58] for image classification or computer vision applications. The traditional adaptive architectures or CDLN makes the early-exit decision based on the classification confidence at each output (exit) layer. If the classification confidence of an output layer for a particular class exceeds a threshold they exit the network. However, such architectures may suffer from performance loss than the baseline architecture when the earlier layer misclassifies a segment with higher confidence which is demonstrated later in Table 3.2. Therefore, implementing adaptive architecture based on classification confidence does not ensure similar performance as the baseline. This motivates us to propose an adaptive architecture that uses an output block predictor to make the early-exit decision which will ensure similar or better performance as the baseline while providing energy efficiency. Sections 3.2.1 and 4.4 provide a motivational example along with the observation to support our proposed adaptive architecture.

3.2.1 Motivational Example

To demonstrate the advantage of an adaptive architecture we have conducted a small experiment. We have created a multi-output CNN architecture with 2 convolution blocks and 2 output blocks. One output block is used after each of the convolution blocks so that we can exit the architecture after any convolution block at the inference phase. The first convolution block consists of one convolution layer, one pooling layer, and one batch normalization

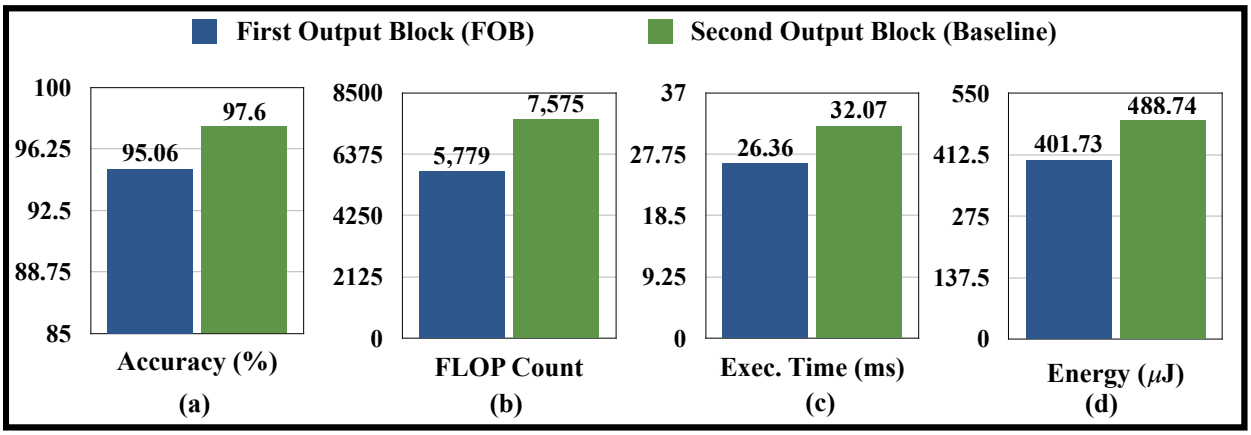


Figure 3.3: Blockwise statistics of multi-output CNN architecture

layer. The second convolution block consists of one convolution and one batch normalization layer only. The output blocks contain either one or two dense layers which represents the output layer. The details of the multi-output CNN architecture is provided in Section 3.4.2. Throughout the rest of this chapter, the first output block (FOB) is used as the portion of the CNN model that uses the first convolution block. The second output block is referred to as the CNN architecture that uses two blocks of convolution which is the baseline architecture. We performed a 5-fold cross-validation of the multi-output CNN architecture with 4740 activity segments from the w-HAR dataset [111]. Figure 3.2 shows the Venn diagram for the multi-output CNN architecture performance where 94.73% are correctly classified by both the FOB and baseline architecture. Only 0.33% and 2.87% of the segments are correctly classified by the FOB and baseline architecture respectively. Rest 2.07% segments are incorrectly classified by both of them. Figure 3.3a shows the blockwise performance breakdown. We find that the accuracy of the multi-output CNN architecture after the FOB, and baseline architecture are 95.06%, and 97.60%, respectively. Figure 3.3b shows the corresponding number of Floating Point Operations (FLOP) necessary to classify one activity segment after the FOB, and baseline architecture which are 5,799, and 7,575, respectively. Figure 3.3c shows the amount of execution time required to classify one activity segment on target wearable platform after FOB, and baseline architecture which are 26.36 μJ, and 32.07 μJ, respectively. Figure 3.3d demonstrates the amount of energy required to classify one

activity segment on target wearable platform after FOB, and baseline architecture which are 401.73 μJ, and 488.74 μJ, respectively.

3.2.2 Observation and Problem Statement

Figure 3.3 demonstrates that the FLOP counts, execution time, and energy increases as performance increases from the first to second output block. To get a better performance, one would choose the second output block as the baseline architecture (as in our case) at the cost of increased energy. However, Figure 3.2 shows that 94.73% (4490) segments that are correctly classified by the baseline architecture are also correctly classified by the FOB. Therefore, using the baseline architecture for those segments would be redundant. If we can avoid these redundant operations, we can easily save some inference time and energy of the wearable devices. Therefore, instead of using a fixed baseline architecture, it would be energy-efficient if we could adaptively decide at the inference phase up to which output block we should use. As shown in Figure 3.2, if we could adaptively use the FOB to classify the 95.06% (4490+16=4506) segments and use the baseline only for the 2.87% (136) segments, overall accuracy (97.93%) would be greater than that of the baseline architecture (97.60%) at a much lower energy consumption. Table 3.1 shows the theoretical breakdown of the performance, FLOP counts, execution time and energy of the adaptive architecture considering the FOB is also used for the 2.07% (98) segments those are misclassified by both output block. Table 3.1 demonstrates that using adaptive architecture, theoretically we can save a total of 8,268,784 FLOPs, 26,288.84 ms of execution time and 400,594.04 μJ of energy for 4740 segments. On average for each segment, we can save 1,744.47 FLOPs, 5.55 ms of execution time, and 84.51 μJ of energy using an adaptive architecture compared to the baseline architecture. In summary, an adaptive architecture would provide a much more energy-efficient solution than a baseline architecture while providing better or similar performance that is suitable for low-power wearable edge devices. On the other hand, traditional adaptive architectures

Table 3.2: Performance of CDLN for different FOB confidence threshold

Method	Weighted F1	Accuracy	Precision	Recall
CDLN (th = 0.5)	94.49	95.06	94.87	95.05
CDLN (th = 0.6)	94.71	95.25	95.09	95.24
CDLN (th = 0.7)	94.93	95.42	95.31	95.41
CDLN (th = 0.8)	95.01	95.48	95.37	95.47
CDLN (th = 0.9)	95.23	95.68	95.59	95.67
Baseline [Ours]	**97.55**	**97.60**	**97.57**	**97.60**
Adaptive [Ours]	**97.64**	**97.70**	**97.69**	**97.70**

or CDLN suffer from performance loss as the earlier layer misclassifies a segment with higher confidence and ends up exiting the network wrongly. As shown in Table 3.2, the performance of CDLN for various confidence thresholds at FOB. The maximum performance of CDLN is achieved for the confidence threshold of 0.9 which is still much less than our baseline architecture. Therefore, our adaptive architecture uses an output block predictor (instead of classification confidence) to make the early-exit decision. Table 3.2 shows that our adaptive architecture not only outperforms the traditional CDLN but also the baseline architecture for all performance metrics. It shows the efficacy of our adaptive architecture over traditional CDLN.

3.2.3 Novel Contributions

The novel contributions of this chapter are as follows:

- A novel Adaptive CNN architecture for HAR (AHAR) that uses an output block predictor to select a portion of the baseline architecture as needed during the inference phase. To the best of our knowledge, we are the first to investigate such an adaptive CNN architecture for HAR application.
- Evaluation of our methodology in classifying locomotion activities from Opportunity [112] and w-HAR [111] dataset. In comparison to the fog/cloud computing approaches

on the Opportunity dataset, both our baseline and adaptive architecture shows a comparable weighted F1 score of 91.79%, 91.57% respectively. For the w-HAR dataset, both our baseline and adaptive architecture outperforms the state-of-art-work with a weighted F1 score of 97.55% and 97.64%, respectively.

- Evaluation on real hardware shows that our baseline architecture is significantly energy-efficient (422.38x less) and memory-efficient (14.29x less) compared to the works on Opportunity dataset. For w-HAR dataset, our baseline architecture requires 2.04x less energy and 2.18x less memory compared to the state-of-the-art work on wearable devices.

- Experimental validation show that our adaptive architecture is 12.32% (Opportunity) and 11.14% (w-HAR) energy-efficient than our baseline while providing similar (Opportunity) or better (w-HAR) performance with no significant memory overhead.

3.3 Related Works

3.3.1 Works on Human Activity Recognition

The main goal of this chapter is to propose a wearable device solution for classifying locomotion activities. Therefore, to validate our proposed methodology, we have considered the Opportunity [112] and w-HAR [111] datasets that has labeled locomotion data from wearable devices. Accordingly, we will discuss and compare against the works mentioned in Table 3.3 that have used either of these two datasets for classifying the locomotion activities.

As shown in Table 3.3, works [99, 104–106] have used Opportunity dataset for classifying 4 locomotion activities - *Stand, Walk, Lie, Sit*. In [99] the authors proposed an activity-recognition algorithm based on the random forest classifier by extracting 4086 features which

Table 3.3: Summary of Related Works

Work	Data used	# of chan.	Classifier used	Adaptive	Computing platform
[99]	Opp.	117	RF (n=[40,95])	No	Fog/Cloud
[104]	Opp.	113	CNN, LSTM	No	Fog/Cloud
[105]	Opp.	6	2-D CNN	No	Fog/Cloud
[106]	Opp.	113	Deep CNN	No	Fog/Cloud
[109]	w-HAR	4	SVM, DT, NN	No	Edge
Ours	**Both**	**7**	**DT, 1-D CNN**	**Yes**	**Edge**

Opportunity (Opp.)

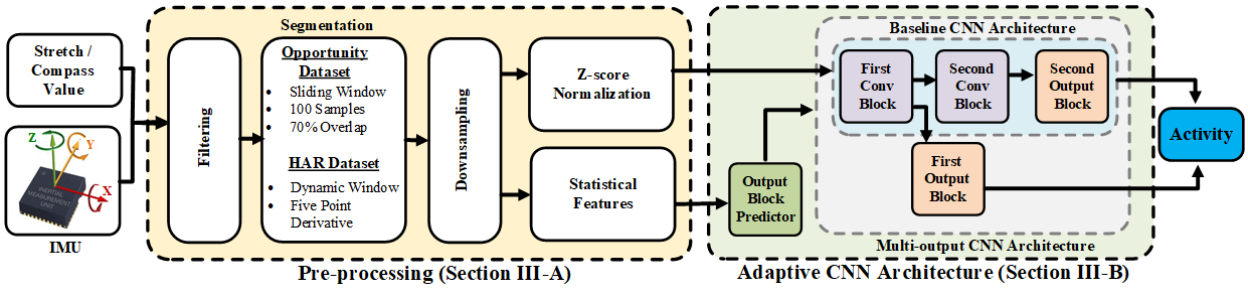


Figure 3.4: Overview of our proposed AHAR methodology

are from both time and frequency domain. They achieve a weighted F1 score of 90%. Authors in [104] use deep CNN architecture composed of 4 convolutional and 2 LSTM recurrent layers and achieves a weighted F1 score 93%. The work in [105] achieves an weighted F1 score of 92.57% using a two dimensional CNN architecture. Finally, the work in [106] used a CNN architecture that combines temporal and spatial convolutions to extract appropriate features to make it suitable for mobile devices. Their solution achieves a weighted F1 score of 92.5%.

On the other hand, the work in [109] used w-HAR dataset to propose a baseline and an activity-aware classifier for classifying 8 locomotion activities in wearable devices. The baseline and activity-aware classifier achieves an weighted F1 score of 94.96% and 97.37% respectively. The baseline architecture uses 120 statistical and frequency domain features whereas the activity-aware classifier works in hierarchical order. First, it classifies the activities as static (*Lie down, Sit, Stand*) or dynamic (*Jump, Walk, Stairs down, Stairs up, Transition*) by feeding 8 statistical features (mean, variance, minimum, maximum) to a support vector

machine (SVM) classifier. Then, if the activity is classified as static, a decision tree is used to classify it further with the same statistical features. Otherwise, the other 112 frequency domain features (FFT) are extracted and together 120 features are fed to a neural network (NN) to classify dynamic activities. Table 3.3 shows a summary of the related works.

3.3.2 Energy-efficient CNN Design Approaches

The deep architecture of CNN with hundreds of layers are very computationally expensive and not suitable for energy and memory constraint wearable devices. Therefore, different approaches have been introduced in the literature to make it energy and memory-efficient while maintaining similar or competitive performance. Such approaches can be broadly classified into two categories - 1) Software-based approach, 2) Hardware-based approach.

The software-based approaches can be further classified into 2 phases - 1) Offline or training phase, 2) Online or inference phase. The software-based approaches in the training phase can be broadly divided into 3 types - a) Neural Architecture Search (NAS), b) Network Pruning, c) Model Compression. NAS looks for optimum network parameters from a search space using reinforcement learning [67] or gradient-based methods [68] or multi-objective bayesian optimization [113–115]. Network pruning performs random pruning of a portion of the big network, retraining it, and repeating the process until it achieves the desired performance [73]. Finally, model compression involves binarization [74] or quantization [75] of network weights to reduce the model size to make it memory-efficient. Another model compression technique is knowledge distillation where a smaller network (student model) is taught, step by step, exactly what to do using a bigger already trained network (teacher model) [116]. Regardless of the methods used, the final model from the training phase is considered as the baseline classifier to be used at the inference phase.

Software-based approach designed for the inference phase is called adaptive (early-exit) or

Conditional Deep Learning Network (CDLN) architecture [57, 58]. If the input data is classified with enough confidence after a convolutional layer then it considers that as the final class without further proceeding to the next layers of convolution. However, they may suffer from performance loss if the earlier layer misclassifies a segment with higher confidence and exits the network wrongly.

It is to note that, the software-based approaches from the training and inference phase are independent of each other and they can be applied together as well. For example, during the inference phase, one can apply the early-exit mechanism to a baseline architecture that has been finalized at the training phase by using any of the NAS, network pruning, or model compression techniques.

Hardware-based approaches usually focus on the design of custom hardware such as accelerators which are specifically designed for CNN [80, 81]. The main goal is to make the inference phase faster thereby making it more energy-efficient.

In this chapter, we mainly focus on the inference phase of the software-based approach which allows early-exit. However, unlike the related works [57, 58], we propose a novel adaptive CNN architecture that uses an output block predictor (instead of classification confidence) to make the early-exit decision without any performance loss. To the best of our knowledge, we are the first to investigate such an adaptive CNN architecture for HAR application.

3.4 Proposed Methodology

3.4.1 Pre-processing Steps

Filtering

As shown in Figure A.15, the pre-processing starts with the denoising and smoothing. Raw data is filtered using a moving average filter with a window of 8 samples to smoothen it. Then the filtered data is segmented.

Segmentation

As the data from different datasets varies, we apply different segmentation technique for two datasets used in this chapter. For the Opportunity [112] dataset, the segmentation of filtered data is done using a sliding window of 100 samples with 70% overlap. As the data is collected at a sampling rate of 30 Hz, each segment of data captures 3.33 seconds of data. For the w-HAR [111] dataset, we follow the dynamic segmentation technique based on five-point derivative on the stretch sensor data as mentioned in [109]. The details of the datasets are given in Section 3.5.1.

Downsampling

Once segmented, we downsample each segment to 32 samples. Downsampling helps in two ways - 1) Lower number of samples in a segment requires less computation for the CNN architecture which makes the solution energy-efficient. 2) Downsampling to a fixed number of samples also helps when we perform dynamic segmentation as CNN requires a fixed size for the input segments.

Calculating Statistical Features

Next, we extract simple statistical features for each segment to be used by our output block predictor to implement our adaptive CNN architecture. The details of the output block predictor is given in Section 3.4.2. We have used a minimum number of features to ensure minimal overhead for our adaptive architecture. For the segments in the Opportunity dataset, we extract 4 features (mean acceleration along X and Z axis, minimum and maximum value of angular velocity along Z axis). For the segments in w-HAR dataset, we extract 6 features (mean acceleration along X and Z axis, minimum and maximum of gyroscope value along Z axis, minimum and maximum of Stretch sensor value). These extracted features will be used by our output block predictor to decide which output block to be used at the inference phase to classify a particular segment.

Z-score Normalization

Before passing the downsampled segments to our multi-output CNN architecture, we normalize each segment using Z-score normalization (Eq. 3.1) to reduce the effect of any outlier samples in the corresponding segments.

$$Z_i = \frac{X_i - \bar{X}}{S} \tag{3.1}$$

For a particular segment, Z_i is Z-score value of the i^{th} sample X_i whereas, $\bar{X}$ and S are the mean and standard deviation of the samples in that segment.

3.4.2 Adaptive CNN Architecture

Our designed adaptive CNN architecture consists of two parts - 1) Multi-output CNN architecture that classifies the segments of activity, 2) Output block predictor that decides which output block of the multi-output CNN architecture is to be used at inference phase based on some statistical features of each segment.

Multi-output CNN Architecture

As our target platform is the low-power edge devices, we design the multi-output CNN architecture considering the resource constraints of the wearable devices. Our multi-output CNN architecture consists of 2 convolution blocks and 2 output blocks. Each convolution block is followed by one output block. Figure 3.5 shows the architecture layout of our multi-output CNN architecture. The first convolution block consists of one convolution layer which is passed through *Leaky-ReLU* activation, one average-pooling layer, and one batch normalization layer whereas the second convolution block has one convolution layer which is passed through *Leaky-ReLU* activation, and one batch normalization layer. The first output block consists of one flattening layer, and one dense layer which is passed through *Softmax* activation. The second output block consists of one flattening layer, and 2 dense layers which are followed by the *Softmax* activation as well. The details of the architecture parameters for each of the layers are given in Table 3.4. As shown in Table 3.4, the total number of parameters required to classify a segment after first output block (FOB) and second output block (baseline architecture) is 240+(31*n_c), and 744+(17*n_c), respectively where n_c is the number of output classes. For the Opportunity dataset, we have 4 output classes and for the w-HAR dataset, we have 8 output classes.

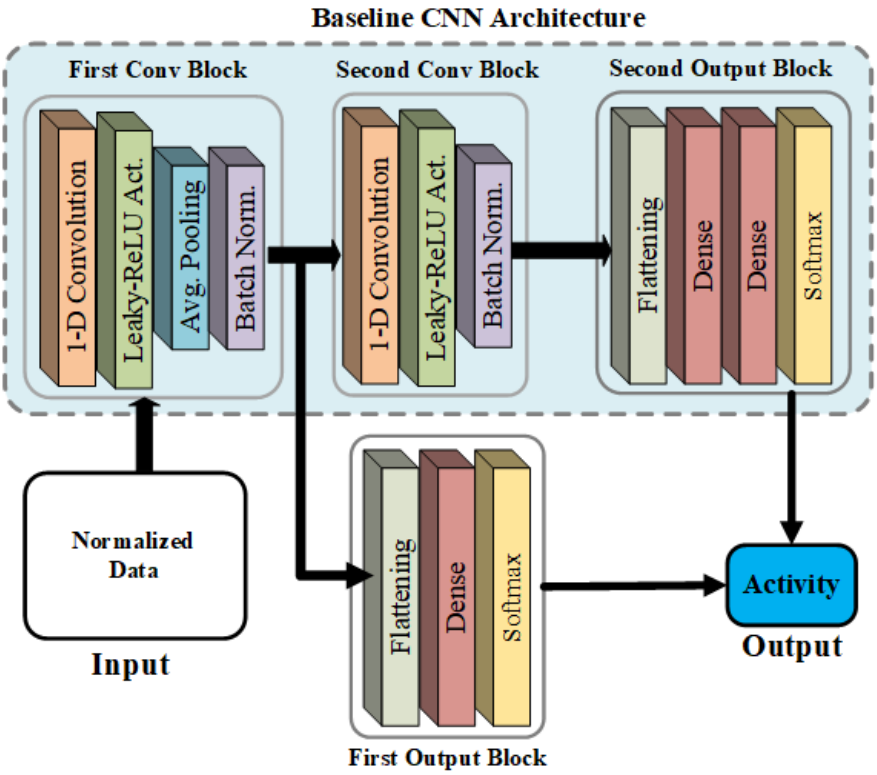


Figure 3.5: Multi-output CNN architecture layout

Output Block Predictor (OBP)

The output block predictor (OBP) is very crucial for our adaptive CNN architecture as the performance of the adaptive architecture greatly depends on the OBP. The better the performance of the OBP is, the better the performance of our adaptive architecture will be. We use a decision tree as our OBP to decide which output block to be used at the inference phase based on the statistical features for each segment. Therefore, instead of using the baseline architecture (second output block) to classify all the segments, we will adaptively use FOB or baseline based on the decision of output block predictor. This will help to avoid unnecessary computation up to the second output block of baseline architecture as some of the segments might be correctly classified just after FOB. As the goal of our adaptive architecture is to ensure energy efficiency compared to the baseline architecture, the OBP

Table 3.4: Multi-output CNN Architecture Details

Layer name	**Kernel size**	**Stride size**	**Act. func.**	**Output shape**	**# of param.**
Input	-	-	-	32x7	0
Conv 1	5	3	LR	10x6	216
Pool 1	2	2	-	5x6	0
BN 1	-	-	-	5x6	24
Flat 1	-	-	-	30	0
Dense 1	-	-	SM	n_c	31*n_c
Conv 2	4	1	LR	2x8	200
BN 2	-	-	-	2x8	32
Flat 2	-	-	-	16	0
Dense 2	-	-	LR	16	272
Dense 3	-	-	SM	n_c	17*n_c
Number of parameters after FOB					**240+(31*n_c)**
Number of parameters after baseline architecture					**744+(17*n_c)**

Batch Normalization (BN), Leaky-ReLU (LR), Softmax (SM)

should be designed in such a way that satisfies the following constraint:

$$[N \times E_{pred} + N_1 \times E_1 + N_2 \times E_2] < [N \times E_2] \tag{3.2}$$

Where E_{pred}, E_1, E_2 is the amount of energy - for the OBP and the FOB, and baseline architecture respectively. N_1, and N_2 are the number of segments that are classified by the FOB, and baseline architecture, respectively where, $N_1 + N_2 = N$. Equations 3.2 ensures that the total amount of energy needed to classify N segments using adaptive architecture should be less than that of the baseline one.

3.5 Experimental Setup

3.5.1 Datasets

Opportunity Dataset [112]

Opportunity dataset contains multimodal data from different wearable, object, and ambient sensors to benchmark the works on human activity recognition. The dataset contains a total of 6 hours of recording from 4 subjects. Each subject performs five sessions of Activities of Daily Living (ADL) and a drill session. The dataset is labeled for different gesture and locomotion activities. In our work, we use the locomotion activities (*Stand, Walk, Sit, Lie down*) as our goal is to propose a wearable device solution that can classify the locomotion activities on the device itself. Therefore, we use only 7 channels of data in total where 3 channels (*accX, accY, accZ*) are from accelerometer on the upper right knee and the other 4 channels (*AngVelBodyFrameX, AngVelBodyFrameY, AngVelBodyFrameZ, Compass*) are from the Inertial Measurement Unit (IMU) on the right shoe. The channels are selected as they are suitable for designing a wearable device where the sensors are in close proximity while collecting maximum information with minimum channels. All the data are collected at 30 Hz from all the sensors.

w-HAR Dataset [111]

w-HAR dataset contains wearable sensor data using IMU and stretch sensors from 22 subjects while performing 7 different locomotion activities (*Jump, Lie down, Sit, Stairs down, Stairs up, Stand, Walk*). Additionally, they also labeled the *Transition* between the activities. The dataset has 7 channels of data where 6 channels (*Ax, Ay, Az, Gx, Gy, Gz*) are from the IMU on the right ankle and 1 channel (*Stretch value*) is from the stretch sensor on the right knee.

Table 3.5: Data Labeling Mechanism for Output Block Predictor

Cases	FOB	Baseline	Assigned label
Both	✓	✓	1
FOB only	✓	×	1
Baseline only	×	✓	2
None	×	×	1

We use all 7 channels from this dataset as it is targeted towards wearable device design for locomotion activities. The IMU data is collected at 250 Hz and the stretch sensor data is collected at 25 Hz.

3.5.2 Training Multi-output CNN Classifier

As mentioned above, we train and test our multi-output CNN classifier on two different datasets. To ensure a fair comparison with the related works on locomotion activity recognition from the Opportunity dataset, we use similar distribution of training, testing and validation data as provided in the Opportunity challenge. Therefore, for training data we use - ADL1, ADL2, ADL3, ADL4, ADL5, DRILL data from subject 1; ADL1, ADL2, DRILL data from subject 2 and 3. The ADL3 data from subject 2 and 3 is used for validation. Finally, the classifier is tested on the ADL4 and ADL5 data from the subject 2 and 3. The classifier is trained for 100 epochs with *Sparse Categorical Cross Entropy* as the loss function. *Adam* optimizer is used to train the models with a learning rate of .007.

For the w-HAR dataset, we perform a stratified 5-fold cross-validation as there is no specific distribution of train test data. Therefore, 80% of the data is used for training, and the rest 20% is used for testing. Moreover, 20% of the training data is used for validation during training. For this dataset, the classifier is trained for 300 epochs with *Sparse Categorical Cross Entropy* as the loss function. *Adam* optimizer is used to train the models with a learning rate of 0.01.

Table 3.6: Confusion Matrix of Different Output Blocks on Opportunity Dataset

True label	FOB				Baseline				Adaptive			
	Stand	**Walk**	**Lie**	**Sit**	**Stand**	**Walk**	**Lie**	**Sit**	**Stand**	**Walk**	**Lie**	**Sit**
Stand	**1090**	108	1	13	**1103**	98	1	10	**1091**	107	1	13
Walk	103	**813**	0	7	107	**807**	1	8	103	**813**	0	7
Lie	0	117	**63**	5	0	0	**184**	1	0	1	**181**	3
Sit	18	6	2	**785**	21	1	9	**780**	18	3	8	**782**

3.5.3 Training Output Block Predictor

To train the output block predictor (OBP), we first generate a dataset based on the performance of the best multi-output CNN model for each of the Opportunity and w-HAR datasets. Then for each of the segments in the dataset, we determine which output block of the multi-output classifier can correctly classify them. For an activity segment, there are 4 different possible cases in our multi-output classifier as shown in Table 3.5. If the segment is correctly classified by both output blocks we would want to use the FOB to save energy hence it is labeled as 1. If it is correctly classified by either FOB or baseline architecture only, it will be labeled as either 1 or 2 respectively. Finally, if it is misclassified by both FOB and baseline architecture that should also be labeled as 1 to avoid unnecessary computation by second output block to classify that segment. Thus, the activity segments of each dataset are labeled which is used as the true label to train and test the OBP (decision tree).

And the input to the OBP is the statistical features for each activity segment as calculated in Section 3.4.1. For the Opportunity dataset, we use 4 features whereas for the w-HAR dataset we use 6 features. To train and test the OBP for Opportunity dataset, we use the same training and testing segments as used in training and testing the multi-output CNN architecture as mentioned in Section 3.5.2. For OBP of the w-HAR dataset, we use stratified 5-fold cross-validation where 80% data is used for training and 20% data is used for testing.

Table 3.7: Performance Comparison of Related Works on Opportunity Dataset for Locomotion (4 Activities)

Works	Weighted F1	Accuracy	Precision	Recall
RF[99]	90.00	-	-	-
CNN,RNN[104]	93.00	-	-	-
2-D CNN[105]	92.57	-	-	-
1-D CNN[106]	92.50	-	-	-
FOB [Ours]	87.24	87.86	88.54	87.86
Baseline [Ours]	91.79	91.79	91.80	91.79
Adaptive [Ours]	91.57	91.57	91.57	91.57

Table 3.8: Performance Comparison of Related Works on w-HAR Dataset for Locomotion (8 Activities)

Works	Weighted F1	Accuracy	Precision	Recall
Baseline[109]	94.96	94.87	95.14	94.87
Activity-aware[109]	97.37	97.34	97.45	97.34
FOB [Ours]	94.45	95.06	94.87	95.06
Baseline [Ours]	97.55	97.60	97.57	97.60
Adaptive [Ours]	97.64	97.70	97.69	97.70

3.5.4 Wearable Platform

Our proposed methodology is designed for low-power, low-memory wearable edge devices. Therefore, we evaluate our classifier on an ultra-low-power 32-bit microcontroller EFM32 Giant Gecko (EFM32GG-STK3700A) [117] which has an ARM Cortex–M3 processor with a maximum clock rate of 48 MHz. It has 128 KB of RAM, 1 MB of Flash.

3.6 Experimental Results and Analysis

As the number of segments for different activities in both the datasets are highly imbalanced, only classification accuracy is not appropriate to measure performance. Therefore, to ensure proper performance evaluation, we use precision, recall, and weighted F1 score in addition

Table 3.9: Confusion Matrix of First Output Block on w-HAR Dataset

True label	**First Output Block (FOB)**							
	J	**L**	**S**	**SD**	**SU**	**ST**	**W**	**T**
J	**445**	0	0	2	0	3	2	6
L	0	**474**	0	0	0	0	0	0
S	0	0	**687**	0	0	9	0	0
SD	0	0	0	**93**	0	0	6	0
SU	0	0	0	0	**106**	0	3	0
ST	1	1	5	0	0	**604**	6	3
W	3	2	0	3	2	6	**1983**	8
T	7	11	70	2	1	24	48	**114**

Table 3.10: Confusion Matrix of Baseline Architecture on w-HAR Dataset

True label	**Second Output Block (Baseline architecture)**							
	J	**L**	**S**	**SD**	**SU**	**ST**	**W**	**T**
J	**450**	0	0	1	0	1	2	4
L	0	**474**	0	0	0	0	0	0
S	0	0	**688**	0	0	8	0	0
SD	0	0	0	**94**	0	0	5	0
SU	0	0	0	1	**105**	0	3	0
ST	0	1	3	0	0	**607**	6	3
W	5	1	0	0	1	6	**1986**	8
T	4	3	14	0	0	16	18	**222**

to accuracy. The metrics used for evaluation are given below:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.3}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.4}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.5}$$

$$WF_1 = \sum_{i}^{n_c} 2 * w_i \frac{Precision_i.Recall_i}{Precision_i + Recall_i} \tag{3.6}$$

Where TP, TN, FP, FN represents True Positives, True Negatives, False Positives, and False Negatives respectively. The activity classes are indexed by i, and w_i=n_i/N. n_i is the number of activity segments in each class, and N is the total number of activity segments.

Table 3.11: Confusion Matrix of the Output Block Predictor (OBP)

True label	Opportunity		w-HAR	
	FOB	**Baseline**	**FOB**	**Baseline**
FOB	**2932**	44	**4582**	22
Baseline	36	**119**	11	**125**

Table 3.12: Confusion Matrix of Adaptive Architecture on w-HAR Dataset

True label	Adaptive architecture							
	J	**L**	**S**	**SD**	**SU**	**ST**	**W**	**T**
J	**455**	0	0	0	0	1	1	1
L	0	**474**	0	0	0	0	0	0
S	0	0	**688**	0	0	8	0	0
SD	0	0	0	**97**	0	0	2	0
SU	0	0	0	0	**106**	0	3	0
ST	1	1	4	0	0	**606**	6	2
W	3	2	0	1	2	6	**1986**	7
T	3	3	17	0	0	17	18	**219**

3.6.1 Performance Evaluation of Multi-output CNN Classifier

The performance for each output block of our multi-output CNN classifier is given in Tables 3.7 and 3.8. As shown in Table 3.7 for the Opportunity dataset, the FOB has overall accuracy, precision, recall, and weighted F1 score of 87.86%, 88.54%, 87.86% and 87.24% respectively, whereas; the baseline architecture shows higher overall accuracy, precision, recall, and weighted F1 score of 91.79%, 91.80%, 91.79%, and 91.79% respectively. The confusion matrices of the output blocks are presented in Table 3.6. It shows that the FOB performs poorly in classifying lying activity (63), whereas baseline architecture shows an improved performance (184). Similarly, for the w-HAR dataset, the baseline architecture achieves higher performance than the FOB. As shown in Table 3.8, the FOB achieves an overall accuracy, precision, recall, and weighted F1 score of 95.06%, 94.87%, 95.06% and 94.45% respectively, whereas; the baseline architecture achieves better accuracy, precision, recall, and weighted F1 score of 97.60%, 97.57%, 97.60%, and 97.55% respectively. Table 3.9 shows that the FOB can classify only 114 transition segments correctly, whereas the baseline

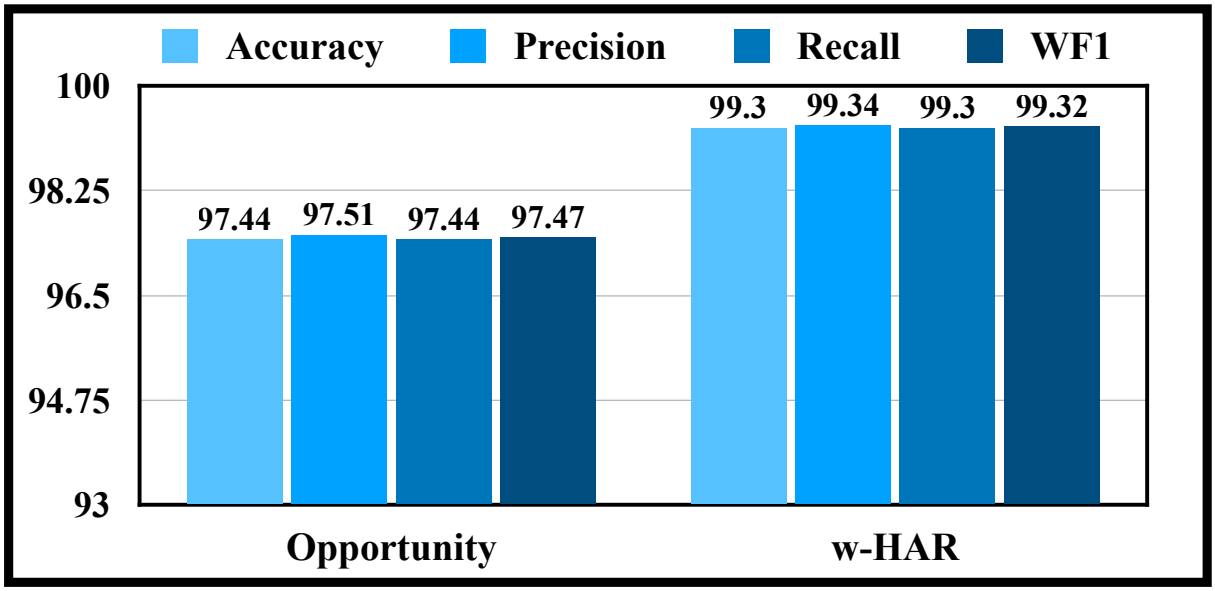


Figure 3.6: Performance of Output Block Predictor (OBP)

architecture shows a better performance while classifying 222 segments correctly as shown in Table 3.10.

3.6.2 Performance Evaluation of Output Block Predictor

To ensure a better performance of our adaptive architecture, our OBP has to perform better as well. As shown in Figure 3.6 for the Opportunity dataset, the OBP has an accuracy, precision, recall, weighted F1 score of 97.44%, 97.51%, 97.44%, 97.47% respectively. Table 3.11 shows the corresponding confusion matrix of the OBP in deciding which output block to use for the 3131 test segments of the Opportunity dataset. Similarly, for the w-HAR dataset, the OBP achieves an accuracy, precision, recall, weighted F1 score of 99.30%, 99.34%, 99.30%, and 99.32% respectively, as shown in Figure 3.6. The corresponding confusion matrix for the w-HAR dataset is shown in Table 3.11. Both confusion matrix shows that our OBP performs quite well in deciding the required output block to classify the activity segments.

3.6.3 Performance Evaluation of Adaptive Architecture

The performance of the adaptive architecture depends on the decision of OBP. We use the FOB to classify the segments that are predicted as 1 by the OBP. Similarly, the baseline architecture is used to classify the segments that are predicted as 2. The performance of adaptive architecture is determined by the combined performance of the FOB and baseline architecture in classifying the corresponding segments decided by the OBP. As shown in Table 3.7 for Opportunity dataset, our adaptive architecture achieves 91.57% performance for all four metrics- accuracy, precision, recall, and weighted F1 score. It shows that our adaptive architecture achieves very close performance as our baseline architecture while classifying most of the segments (2968) using the FOB. Table 3.6 shows how the adaptive architecture takes the advantage of both the output blocks. For example, 63 out of 185 lying activity segment is correctly classified by FOB whereas the baseline architecture can correctly classify 184 of them. And our adaptive architecture can classify 181 of them which is close to the baseline one. Moreover, both our baseline and adaptive architecture outperforms the work [99] and achieves a comparable performance with respect to [104–106] as shown in Table 3.8. It is to note that the works [99, 104–106] are designed for fog/cloud platform whereas our solution is designed for wearable platform.

For the w-HAR dataset, our adaptive architecture outperforms our baseline architecture with an accuracy, precision, recall, and weighted F1 score of 97.70%, 97.69%, 97.70%, 97.64% respectively as shown in Table 3.8. Moreover, both our baseline and adaptive architecture outperforms both the baseline and activity-aware classifier used in [109]. As shown in Table 3.12, the adaptive architecture can classify 455 out of 458 jump activity whereas the FOB and baseline architecture can classify 445 and 450 of them respectively. This is because there were jump activities that were being classified either by FOB or baseline architecture only. The adaptive architecture uses the best of the two which results in improved performance. Therefore, it proves that the adaptive architecture achieves similar (Opportunity) or better

Table 3.13: Energy and Memory Consumption Evaluation of the Works on Opportunity Dataset

Works	Classifier level	RAM (Bytes)	Exe. time (ms)	Avg. pwr. (mW)	Energy (μJ)
[**99**]	-	60932	11722.14	16.59	194470.31
[**104**]	-	Not compatible: RAM overflowed			
[**105**]	-	Not compatible: RAM overflowed			
[**106**]	-	Not compatible: RAM overflowed			
[**Ours**]	OBP	1120	1.61	15.26	24.56
	FOB	2688	24.57	15.25	374.69
	Baseline	4264	30.25	15.22	460.41
	Adaptive	4264	26.48	15.25	403.71

performance (w-HAR) with respect to our baseline architecture while using the FOB to classify most of the segments.

3.6.4 Energy and Memory Evaluation on Real Hardware

We evaluate the energy and memory consumption of our proposed architecture including the related works using the EFM32 Giant Gecko microcontroller as mentioned in Section 3.5.4. For the works [99, 109] that uses machine learning approaches, the reported execution time, power, energy, and RAM are for the feature extraction and classification together. For the works using CNN, we evaluate the classification as they automatically extract features during classification. The execution time, power, and energy values presented in the Table 3.13 and 3.14 are for one activity segment of data using the 14 MHz clock speed of the microcontroller.

As shown in Table 3.13, for Opportunity dataset the works [104–106] using deep CNN encountered RAM overflow and could not be executed on the target hardware. It shows that this kind of solution is only suitable for fog/cloud platforms with higher computational resources. Although the work [99] is designed for fog/cloud platform, it executes on the target hardware with around 60KB of RAM. It takes around 11.72 seconds to extract features and

Table 3.14: Energy and Memory Consumption Evaluation of the Works on w-HAR Dataset

Works	**Classifier level**	**RAM (Bytes)**	**Exe. time (ms)**	**Avg. pwr. (mW)**	**Energy (μJ)**
[109]	Baseline	9988	63.85	15.30	976.91
	Static	2164	31.93	15.31	488.84
	Dynamic	9988	85.55	15.30	1308.92
	A. aware	9988	65.07	15.3	995.77
[Ours]	OBP	1128	1.96	15.23	29.86
	FOB	3216	26.36	15.24	401.73
	Baseline	4568	32.07	15.24	488.74
	Adaptive	4568	28.50	15.24	434.29

classify an activity segment with 194.47 μJ of energy consumption. On the other hand, our FOB executes with only 2.62 KB of RAM. It takes only 24.57 ms with an energy consumption of 374.69 μJ to classify an activity segment. Our baseline architecture requires higher resources than the FOB as expected. As shown in Table 3.13, the OBP takes only 1.09 KB of RAM to execute. It takes only 1.61 ms to extract 4 statistical features from each segment and classify it with an energy consumption of 24.56 μJ. It shows that the OBP is very lightweight and does not add much overhead to implement our adaptive architecture. To evaluate the execution time and energy of our adaptive architecture, we calculate the average time and energy to classify 3131 test segments either by FOB or baseline architecture based on the decision of our OBP as presented in the confusion matrix of Table 3.11. Table 3.13 shows that the adaptive architecture takes only 26.48 ms with an energy consumption of 403.71 μJ which is less than our baseline architecture while providing similar performance.

For the w-HAR dataset, first, we evaluate the baseline classifier of the work in [109]. As shown in Table 3.14, the baseline classifier takes 63.85 ms to classify a segment with 976.91 μJ of energy. It takes 9.75 KB of RAM to execute. The baseline classifier in [109] involves extracting 120 statistical features from the activity segment and then classify it with a neural network. The activity-aware classifier uses different classifier for static - *Sit (S), Lie (L), Stand (ST)* and dynamic - *Stairs up (SU), Stairs down (SD), Jump (J), Walk (W),*

Transition (T) activities. For classifying a static activity, it takes 31.93 ms with 488.84 μJ energy. For the dynamic activities, it consumes higher energy of 1308.92 μJ with a longer execution time of 85.55 ms. The execution time and energy for the activity-aware classifier reported in Table 3.14 is the average time and energy to classify 4740 segments either by the static or dynamic classifier. Out of 4740 segments, the SVM classifier classifies 1810 segments as static and 2930 segments as dynamic as mentioned in [109]. Therefore, the total time and energy for classifying 1810 segments by the static classifier and 2930 segments by dynamic classifier is calculated and summed up. Next, the summation is averaged by 4740 which gives us the average time of 65.07 ms and energy of 995.77 μJ required by the activity-aware classifier. The activity-aware classifier also requires the 9.75 KB of RAM same as the baseline. This RAM is required for calculating the 120 features which is done in both baseline and activity-aware classifier.

On the other hand, our FOB executes with only 3.14 KB of RAM and takes only 26.36 ms with an energy consumption of 401.73 μJ to classify an activity segment of the w-HAR dataset. As expected, our baseline architecture requires higher resources - 4.46 KB of RAM and 32.07 ms to classify a segment with 488.74 μJ of energy. The OBP takes only 1.10 KB of RAM to execute which takes only 1.96 ms to extract 6 statistical features from each segment and classify it with an energy consumption of 29.86 μJ. Therefore, the OBP takes very minimum resources which ensures minimal overhead to implement our adaptive architecture. To evaluate the execution time and energy of our adaptive architecture, we follow the same procedure as the Opportunity dataset and do it for 4740 segments of the w-HAR dataset. As shown in Table 3.11, the OBP decides 4593 and 147 segments to be classified by the FOB and the baseline architecture respectively. Therefore, the summation of total time and energy taken by the OBP, FOB, and baseline architecture is averaged by 4740 which gives us the average time and energy to classify a particular segment by our adaptive architecture. Table 3.14 shows that the adaptive architecture takes only 28.50 ms with an energy consumption of 434.29 μJ which is less than our baseline architecture while

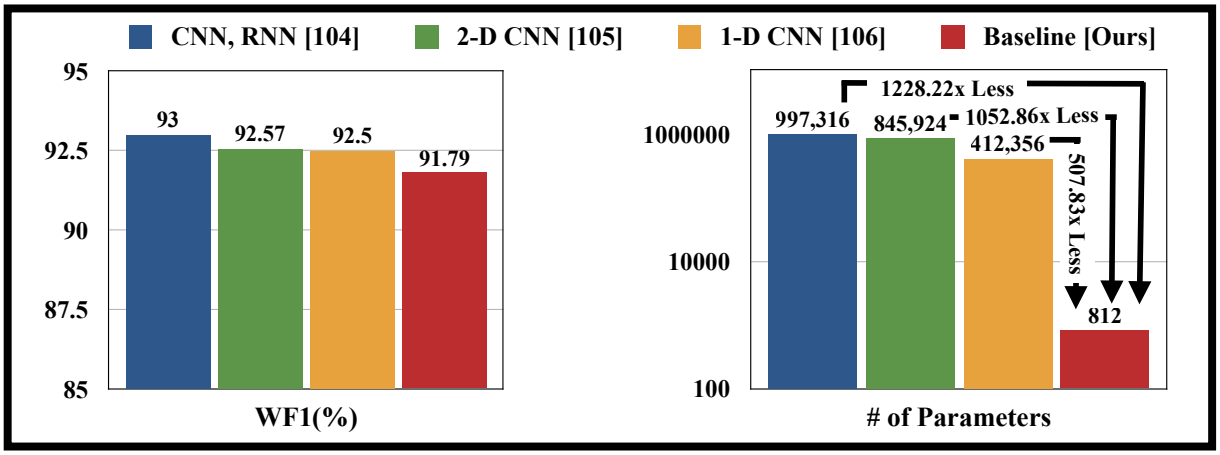


Figure 3.7: Benchmarking of the deep CNN works on Opportunity dataset

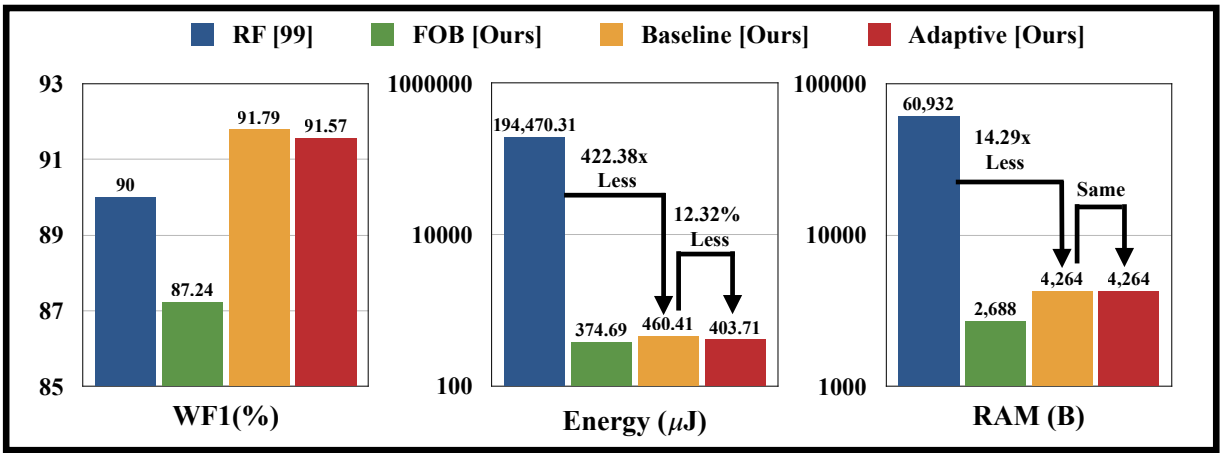


Figure 3.8: Benchmarking on Opportunity dataset

providing better performance.

3.6.5 Final Benchmarking

Finally, we make an overall comparison among the performance of different works along with the computational resources they require. As the deep CNN works [104–106] on Opportunity dataset are designed for the fog/cloud platform and do not fit into our target wearable platform, we compare their performance with the network parameter size to give a perspective. As shown in Figure 3.7, they achieve higher weighted F1 score of 93%, 92.57%,

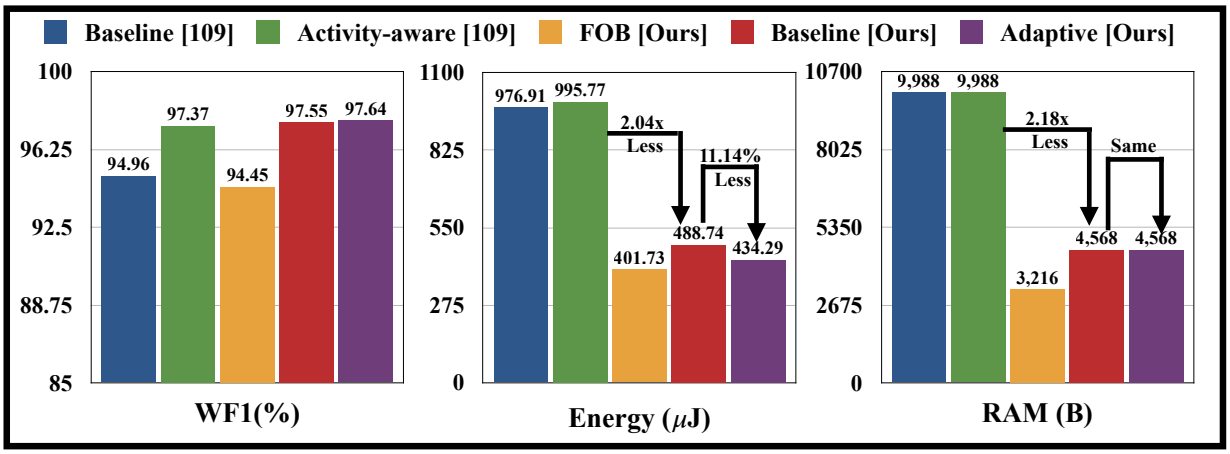


Figure 3.9: Benchmarking on w-HAR dataset

and 92.50%, whereas; our baseline architecture achieves a comparable weighted F1 score of 91.79% with 1228.22x, 1052.86x, and 507.83x less parameter size compared to [104–106] respectively. Besides, our baseline architecture outperforms the work [99] while consuming 422.38x less energy and 14.29x less RAM as shown in Figure 3.8. Moreover, our adaptive architecture achieves similar performance as the baseline while being 12.32% energy-efficient.

As shown in Figure 3.9 for the w-HAR dataset, our baseline architecture outperforms both the baseline (94.96%) and activity-aware (97.37%) classifier in [109] with a weighted F1 score of 97.55% while being 2.04x and 2.18x energy and memory-efficient compared to the activity-aware classifier. Moreover, our adaptive architecture outperforms our baseline while being 11.14% energy-efficient.

It is to note that, the 12.32% or 11.14% energy efficiency achieved by our adaptive architecture over the baseline one is only for 2 layers of convolution. The energy efficiency would be even more if we had deeper CNN architecture with multiple convolution layers. Therefore, our future plan is to investigate the potential of our adaptive CNN architecture for other applications that require multiple convolution layers.

3.7 Summary

This chapter proposes an Adaptive CNN for HAR (AHAR) to develop an energy-efficient solution for low-power edge devices. AHAR uses a novel adaptive architecture that decides which portion of the baseline architecture to be used during the inference phase based on the simple statistical features of the activity segments. Our proposed methodology is validated for classifying locomotion activities from Opportunity and w-HAR datasets. Compared to the fog/cloud computing approaches that use the Opportunity dataset, both our baseline and adaptive architecture shows a comparable weighted F1 score of 91.79%, 91.57% respectively. For the w-HAR dataset, both our baseline and adaptive architecture outperforms the state-of-art-work with a weighted F1 score of 97.55% and 97.64% respectively. Evaluation on real hardware shows that our baseline architecture is significantly energy-efficient (422.38x less) and memory-efficient (14.29x less) compared to the works on the Opportunity dataset. For the w-HAR dataset, our baseline architecture requires 2.04x less energy and 2.18x less memory compared to the state-of-the-art work. Moreover, experimental results show that our adaptive architecture is 12.32% (Opportunity) and 11.14% (w-HAR) energy-efficient than our baseline while providing similar (Opportunity) or better (w-HAR) performance with no significant memory overhead. To the best of our knowledge, we are the first to propose such adaptive CNN architecture for HAR in wearable devices that provides energy efficiency while maintaining performance.

Chapter 4

Fog-enabled Energy Aware Online Human Eating Activity Recognition

4.1 Abstract

Eating Activity Recognition (EAR) plays an important role in ensuring healthy eating habits. Recent advancements of the Internet of Things (IoT) have bolstered automated EAR through various wearable edge devices. State-of-the-art work uses some offline trained classifiers at the fog device to recognize eating activities. However, the eating habits of a person change quite frequently and vary from person to person. Therefore, the classifiers should be updated continuously with new data to adapt to these changes and be personalized over time through online learning. To the best of our knowledge, no state-of-the-art work has addressed this issue so far. In this chapter, we propose an online learning methodology called Human Eating Activity Recognition (HEAR) by introducing an online update phase. We also design an algorithm to be used in the online update phase that provides approximate true labels for the new data. Moreover, we also design a wearable neckband as the edge device to

capture eating activity data (*Chewing, Swallowing, Talking, and Idle*) in a lab environment. Through a detailed experimental evaluation on 12 users, we show that an Online Learned Neural Network (OLNN) classifier using our HEAR methodology performs better than any state-of-the-art offline trained classifier. We also demonstrate that our OLNN classifier is energy efficient compared to the competitive offline trained classifiers. The findings in this chapter have been published in [44].

4.2 Introduction and Related Work

Eating habits have a direct correlation with a healthy lifestyle and are the primary reason for many chronic diseases like obesity, diabetes, and hypercholesterolemia. Obesity itself cost more than 147 billion US dollars for medical treatment alone in the United States in 2008 [118]. According to the most recent National Health and Nutrition Examination Survey (NHANES), 18.5% of children and nearly 40% of adults in the United States had obesity in 2015-2016. These are the highest rates ever documented by NHANES [119]. A study in [120] estimated that overweight and obesity during childhood resulted in an excess lifetime cost per person of 4,209 euros (men) and 2,445 euros (women) in Germany and for the German population, the overall excess lifetime cost was 145 billion euros.

Food intake monitoring plays a primary role in establishing healthy eating habits because proper monitoring can ensure smooth functioning metabolism. Applications of food intake monitoring include i) Recognition of food eating activities (chewing, swallowing); ii) Classification of food type (solid, liquid); iii) Quantification of food (volume, weight, calorie) [121]. Among these applications, recognition of food eating activities is the most important as it lays the foundation for food classification and quantification.

The traditional way of food intake monitoring used to follow manual approaches of self-

reporting from the users. However, these methods often tend to be tedious and suffer from poor recall accuracy from the individuals [122]. The drawbacks of manual methods coupled with the recent advancements of the IoT and wearable devices have encouraged the researchers to dive into the automated approach of food intake monitoring.

Some researchers used acoustics as a sensing modality to capture eating activity [123–125]. Many researchers used other sensors like - strain gauge [126], piezoelectric [127–130], and proximity sensors [131] to pick up muscle movements around the larynx or outer ear for EAR to detect eating activity. Inertial sensors were also used to detect eating activity by capturing feeding gestures [132, 133]. Researchers also combined multiple modalities to better recognize eating activities. Authors in [134] used both strain gauge sensor and a microphone to detect eating activity. Researchers in [135] developed a smart necklace combining piezoelectric, inertial and microphone sensor. Another group in [136] developed a wearable system fusing inertial, proximity, and microphone sensor. Another group developed [137] a wearable necklace combining proximity, inertial, and an ambient light sensor. Among the aforementioned related works, the work in [135] is close to our work which uses a Random Forest classifier to detect eating or non-eating activities and achieves an F1-score of 80.8%.

Although, different researchers used different sensors or combination of sensors, they all follow a fog computing architecture [138–140] as shown in Figure 4.1. In EAR systems, the wearable device is the edge device primarily used for sensing and communication. Afterward,

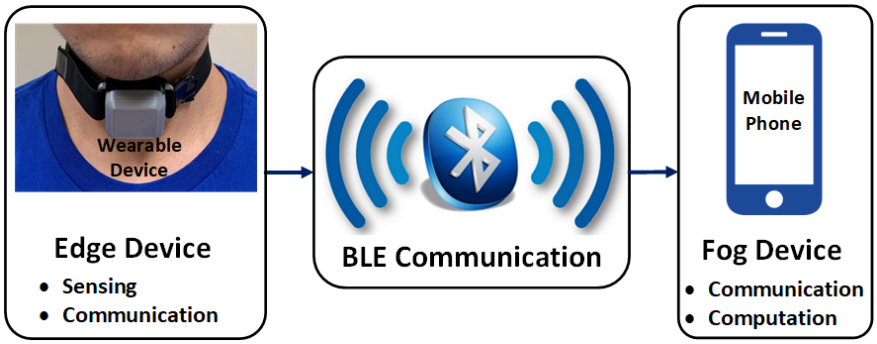
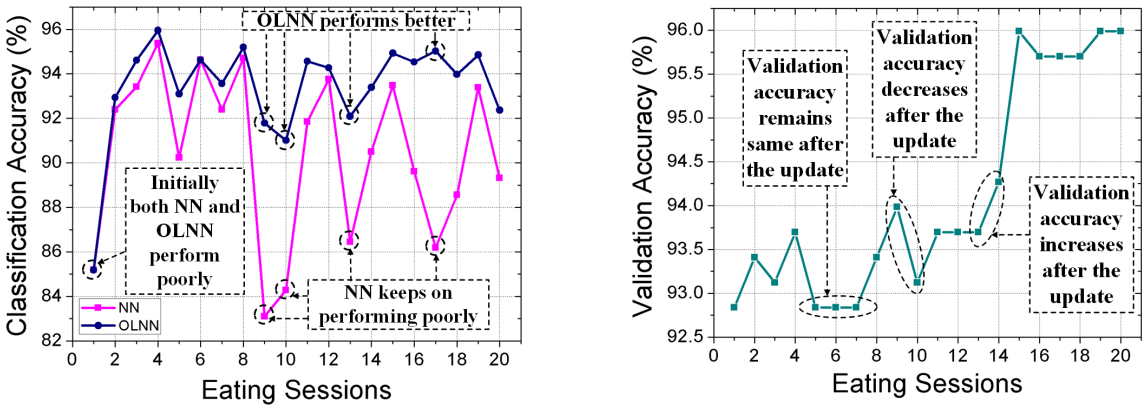


Figure 4.1: Fog Computing Architecture of EAR Systems

the data collected from the wearable device is sent over Bluetooth to a mobile phone used as a fog device. The fog device is mostly responsible for communication and computation. The computation in fog device mainly involves data processing, segmentation, feature extraction and finally classifier training. All these classifiers are trained offline then classify the online data coming from the edge device. However, the offline trained classifier is not suitable for EAR in the long run as the eating habits of a person change over time and vary from person to person. Therefore, the classifier should also be updated continuously to adapt to those changes and get personalized over time. To the best of our knowledge, no state-of-the-art work has addressed this issue so far. It is really important to keep the classifier updated because regardless of how accurately we design the wearable device to capture the data if the classifier is not good enough it will result in poor recognition accuracy.

4.2.1 Motivational Case Study

To demonstrate the limitation of using offline trained classifiers for EAR, we have conducted a small experiment. Data was collected from a male user using a wearable neckband (Section 4.4.1) in a lab environment as the user ate cookies and bread slices and drank water in between for around 8 minutes per session. We collected data for 4 sessions per day over 10 days (total of 40 sessions). The first half of the data was used as training and the second half as testing data. The training data was used to train a neural network (Section 4.3.1) with the training accuracy of 97.24%. We test the network both with and without updating it after each eating session. We call the first method offline trained Neural Network (NN), and the later method Online Learned Neural Network (OLNN). Figure 4.2a shows classification accuracy for both NN and OLNN. To check whether OLNN improves after every eating session, we also calculated its validation accuracy against a validation dataset of 4 eating sessions (each collected in 4 different days) from the same user as shown in Figure 4.2b.



(a) NN Vs. OLNN (b) OLNN

Figure 4.2: Motivational Case Study

4.2.2 Observation from Case Study

As depicted in Figure 4.2a, the performance of NN greatly varies on the test data collected in a different timeline. This is because the test data has a different pattern than the training data which implies that human eating patterns change from time to time. This gradual change in data properties, called *'Concept Drift'* [141, 142], cannot be handled by NN. It can, however, be handled quite well by OLNN as shown in Figure 4.2a. This is because OLNN learns from every new eating sessions. However, updating the classifier after every eating session might not be wise as the presence of unusual data may make the classifier worse. As shown in Figure 4.2b, the validation accuracy of the OLNN does not improve after every eating sessions. It may remain the same or even decrease sometimes. We need to make sure the validation accuracy does not decrease for the online classifiers.

To summarize, eating habits vary from time to time and from person to person. When it comes to personalized health care, we should develop a personalized classifier for each individual separately and update it continuously with new data to reflect daily changes in their eating habits. This method of updating the model with the new data is called *'Online Learning'* [143]. The need for online learning can be summarized with a simple analogy -

'*The way human body needs a continuous supply of food to be healthy and alive; similarly, a machine learning model needs to be fed continuously with new data to be healthy and alive*'.

4.2.3 Research Challenges

1) One of the key challenges of online learning is the availability of true labels of the new data to update the model with. One way would be to ask users to continuously provide feedback [144] by labeling the data as they eat. This is not a feasible solution as it requires continuous human attention. Some researchers tackled this challenge by recording a video of users as they eat and processing it to get true labels [136]. However, this method has two major disadvantages. Firstly, it requires a lot of energy and memory and may even be more costly than EAR. Secondly, users may be uncomfortable being videoed every time they eat [145].

2) Another challenge of online learning is to protect the classifier against unusual data. Classifiers may worsen in classification accuracy by training on such data and suffer from an avalanche effect of poor classification accuracy.

3) Additionally while ensuring better performance compared to the offline trained classifiers, online learned classifiers should be efficient in terms of energy as well.

4.2.4 Goals and Novel Contributions

This chapter makes the following novel contributions to overcome the above mentioned challenges:

1. **Human Eating Activity Recognition (HEAR) Methodology (Section 4.3)**: In the fog side, we propose HEAR methodology that enables online learning by intro-

ducing an online update phase (Section 4.3.3). This phase overcomes the challenges of the online learning in two ways - 1) Implements a True Label Approximation (TLA) Algorithm (Section 15) to get approximate true labels for the new data; 2) Performs a validation check on the updated classifier to avoid the performance avalanche effect (Section 23).

2. **Wearable Neckband (Section 4.4.1)**: In the edge side, we develop an wearable neckband that captures the signals for four activities - *Chewing*, *Swallowing*, *Talking* and *Idle*.

3. **Experimental Evaluation (Section 4.5)**: We perform a detailed experimental analysis of our proposed methodology using an online learned neural network classifier (OLNN) which performs better than any offline trained state-of-the-art classifier (Section 4.5.2). We also demonstrate that our classifier is energy efficient compared to the competitive offline trained classifiers 4.5.3.

4.3 Proposed HEAR Methodology

As shown in Figure A.2, our HEAR methodology enables online learning by proposing an online update phase in addition to the offline training and online classification phase. Following sections discuss each phase in details.

4.3.1 Offline Training Phase

During the offline training phase, we design an initial classifier for the online classification phase to start with. The details of this phase are discussed as follows:

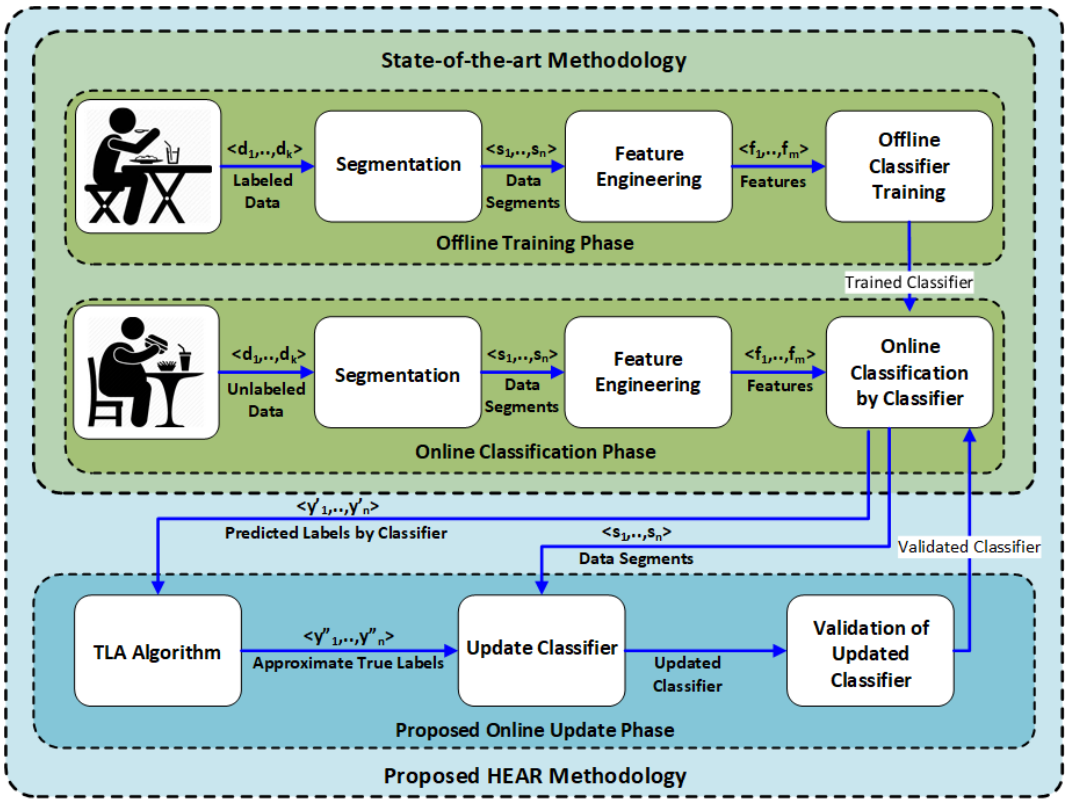


Figure 4.3: Overview of Proposed HEAR Methodology

Data Segmentation

To capture the correct data pattern for each activity we segment the sensor data coming from the wearable device. We use a sliding window of 20 samples with a 70% overlap in between the windows. In any window, if more than 60% of the samples belong to one activity, then the window is labeled as that activity. The window size of 20 is selected considering the sampling frequency of the wearable device which is 20 Hz.

Feature Engineering

The goal of feature engineering is to select the optimal features to be used for training the classifiers. In this step, we extract the features for each window generated during the segmentation step. From a total of 65 time-series features, we applied Correlation-based

Feature Subset (CFS) selection and selected 34 features[146]. From the selected features, we further applied FeatuRe Extraction based on Scalable Hypothesis tests (FRESH) [147] algorithm to select the optimal ones. Table 4.1 shows the list of 18 optimal features used for classifier training. The activities *(Chewing, Swallowing, Talking)* that we classify result in the spike of physiological signals from piezoelectric and acoustic sensors. Therefore, the optimal features are the ones that best characterize those spikes to better classify the corresponding activities. The total number of features used for classifier training is doubled as we use the same features for both the piezoelectric and the acoustic sensors.

Table 4.1: List of Optimal Features

Variance	Std. Deviation	Sum of Abs. Change
Skewness	1^{st} FFT Coeff.	Ratio of Unique Values
Minimum	Autocorrelation	Linear Trend for Std. Error
Maximum	Has Duplicate/Not	Percent of Re-occuring Val.
1^{st} Quartile	Mean Abs. Change	Complex-Invariant Distance
3^{rd} Quartile	Num. of CWT Peaks	Longest Strike Above Mean

Offline Classifier training

One of the goals of this chapter is to use a classifier that is energy efficient during online classification and online update phase. Therefore, we use a neural network (NN) [148, 149] that is small and simple enough to achieve the goal. NN can be easily updated with the new data without the need for previous training data. We use a 3 layered fully connected feed-forward architecture where the middle layer is the hidden layer with only 4 units plus the bias unit. The reason for choosing the fewer number of hidden units is to reduce the computational complexity of the NN while maintaining good accuracy. For the output layer, there are 4 units (one for each activity). We use *tanh* activation function in between the input layer and the hidden layer and for the output layer we use *Softmax* activation function. To optimize the NN weights we use a stochastic gradient-based optimizer called *'Adam'* [150].

Besides, state-of-the-art classifiers like - Decision Trees [151], Naive Bayes [152] and Ad-

aBoost [153] show poor performance on the new user data as shown in Figure 4.10. Other classifiers like Random Forest [154], Support Vector Machine [155], and k- Nearest Neighbors [156] show competitive accuracy, however; are not suitable for online learning as they are very computationally expensive to update. As shown in 4.5.3 the online classification phase for these classifiers are more computationally expensive than the online classification and online update phase of our neural network.

4.3.2 Online Classification Phase

In this phase, the sensor data from the wearable device is segmented in the same way mentioned in Section 4.3.1. Then the selected features from Section 4.3.1 are extracted for each segment. After that, the trained classifier from Section 4.3.1 classifies those segments to one of the four activities. We call this classification of segments as the predicted labels for the segments. All the segments generated during the online classification phase along with their predicted labels are used in the online update phase.

4.3.3 Online Update Phase

The online update phase uses the segments generated during the online classification phase and the corresponding predicted labels to update the classifier. Algorithm 2 shows the stepwise procedure followed during the online update phase. Following sections discuss the key steps during this phase.

True Label Approximation (TLA) Algorithm

One of the major goals of this chapter is to facilitate the availability of true labels of the online data without the involvement of the user. Also, we want to follow a simple and

Algorithm 2: Online Update Algorithm

Input: $Data_{new}$: Data from the last eating session
Input: $Labels_{pred}$: Predicted labels by the current NN
Constant Variables:
$Data_{valid}$: User specific validation data.
Static Variables:
NN_{cur}: Current NN (Initially it is the offline one)
$Accuracy_{cur}$: Accuracy of NN_{cur} on $Data_{valid}$
$Conf_{cur}$: Confusion matrix of NN_{cur} on $Data_{valid}$
Other Variables:
$Labels_{new}$: Approximate true labels by TLA Algorithm 3
NN_{new}: Updated NN_{cur} with $Data_{new}$ and $Labels_{new}$
$Accuracy_{new}$: Accuracy of NN_{new} on $Data_{valid}$
$Conf_{new}$: Confusion matrix of NN_{new} on $Data_{valid}$
if $Accuracy_{new} > Accuracy_{cur}$ **then**
    NN_{cur}=NN_{new}
    $Accuracy_{cur}$=$Accuracy_{new}$
    $Conf_{cur}$=$Conf_{new}$

lightweight method that does not add too much overhead during the online update phase. The TLA algorithm exploits the fact that an offline trained classifier has a certain amount of accuracy on the predicted labels. Therefore, it uses the classifier predicted labels of the online data and apply confusion matrix based heuristics on them to get approximate better labels than the predicted ones. It tries to find suspicious predictions in the predicted labels using the Lemma 1. Then it applies some heuristic on them with the help of a confusion matrix of the current neural network. The confusion matrix is calculated on the labeled validation dataset of the new user whose data is not used in the offline training phase. It holds information about how the classifier confuses between different activities of that user. The TLA algorithm looks into the label of the previous and next window of the suspicious prediction and tries to correct it using the confusion matrix. It also considers the fact that the previous and the next window might also be labeled incorrectly. That is why it uses the labels for the previous/next two consecutive windows to decide the labels of previous/next window for the current suspicious one.

Lemma 1. *If the sampling frequency is X Hz and the segmentation window is W (where $X, W \in \mathbb{R}$ and $W < X$) and also the minimum duration ($Duration_{min}$) of each activity is*

more than 1 second then there should not be any consecutive 2 transitions of activities in between the windows.

Proof: *When sampling frequency is X Hz that means there are X samples of data in 1 second. The segmentation window of W samples represents W/X seconds of data. If $Duration_{min} > n(W/X)$ where $n \geq 2$, then each activity should continue for at least n windows or more. Therefore, if there is a consecutive transition of activities in between windows that means one of the predicted activities was performed for W/X seconds whereas the $Duration_{min} > n(W/X)$ where $n \geq 2$. Therefore, there is an error in the prediction when there are 2 consecutive transitions in between the windows (Observation 3 in Figure 4.4).*

We design our novel TLA algorithm based on the following observations (See Figure 4.4) drawn from the classification errors made by the classifier -

1. Most of the classification errors happen during the transitions from one activity to another.
2. The classification errors that occur when the transition of activities happens every after multiple windows are hard to detect without the ground truth labels.
3. The classification errors that occur when the transition of activities happens every after one window can be approximately detected using Lemma 1.
4. Whenever one of the windows is classified wrongly, the true label of that window is one of the neighboring windows (either previous or the next).

Algorithm 3 shows the stepwise procedure followed by TLA. Among the four activities, *Chewing* and *Swallowing* are the eating-related activities and follows a physiological pattern in terms of duration. Table 4.2 shows their duration for the 12 users in our experiment. Their duration is also similar to the one reported through clinical study in [124, 157, 158].

Algorithm 3: True Label Approximation (TLA) Algorithm

Input: $Labels_{pred}$: Predicted labels by the NN_{cur}
Input: $Conf_{cur}$: Confusion matrix of NN_{cur} on $Data_{valid}$
Output: $Labels_{new}$: Approximate true labels using $Conf_{cur}$

```
n: Length of Labels_pred
Labels_new=Labels_pred
for i = 3 : n − 2 do
    if Labels_pred[i] ≠ Labels_pred[i − 1] and Labels_pred[i] ≠ Labels_pred[i + 1] and
     Labels_pred[i] == Chewing or Swallowing then
        if Labels_pred[i + 1] ≠ Labels_pred[i + 2] then
            if Conf_cur[Labels_pred[i + 1]][Labels_pred[i + 2]] ≥
              Conf_cur[Labels_pred[i + 2]][Labels_pred[i + 1]] then
                next=Labels_pred[i + 1]
            else
                next=Labels_pred[i + 2]
        else
            next=Labels_pred[i + 1]
        if Labels_pred[i − 1] ≠ Labels_pred[i − 2] then
            if Conf_cur[Labels_pred[i − 1]][Labels_pred[i − 2]] ≥
              Conf_cur[Labels_pred[i − 2]][Labels_pred[i − 1]] then
                prev=Labels_pred[i − 1]
            else
                prev=Labels_pred[i − 2]
        else
            prev=Labels_pred[i − 1]
        if Conf_cur[next][Labels_pred[i]] ≥ Conf_cur[prev][Labels_pred[i]] then
            Labels_new[i]=next
        else
            Labels_new[i]=prev
return Labels_new
```

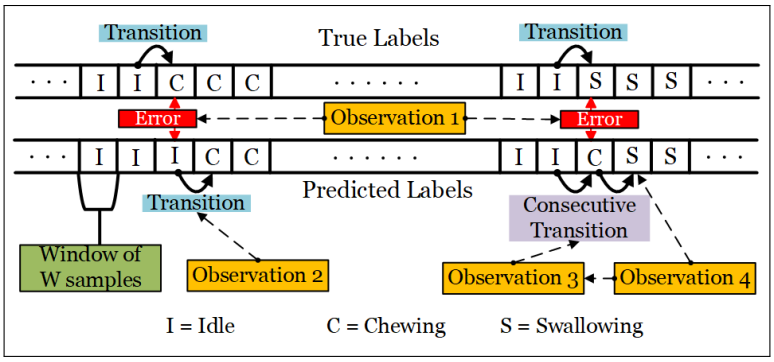


Figure 4.4: Observations Drawn from Classification Errors

Table 4.2: Duration of Chewing and Swallowing

Activity	**Average Duration ± Standard Deviation**
Chewing	4.368±2.802 (Seconds)
Swallowing	1.802±.686 (Seconds)

Besides, the *Talking* and *Idle* activities are non-eating activities which are used to distinguish between the eating and non-eating artifacts. Therefore, our TLA method tries to detect and correct the suspicious predictions for *Chewing* and *Swallowing* based on Lemma 1.

Operational Example of TLA Algorithm: We will demonstrate how TLA algorithm operates based on an example sequence as shown in Figure 4.5. The yellow marked windows - $(i-1)^{th}$, i^{th}, $(i+1)^{th}$ in predicted labels are the suspicious one based on Lemma 1. The red highlighted windows represent incorrect predictions. Here, we will show how the TLA algorithm works on the i^{th} window using the confusion matrix in Table 4.4. The predicted label for i^{th} window is *Chewing* which takes place for just one window in between $(i-1)^{th}$ and $(i+1)^{th}$ windows where both of them are *Swallowing*. Then to decide the label of $(i+1)^{th}$ window we look into the predicted labels of both $(i+1)^{th}$ and $(i+2)^{th}$ window which are *Swallowing* and *Chewing* respectively. As they are different, we look into the confusion matrix as <*True label, Predicted label*> tuple to decide the next label. Therefore, we look for the value of <*Swallowing, Chewing*> tuple and <*Chewing, Swallowing*> tuple from the confusion matrix in Table 4.4. As we can see, <*Swallowing, Chewing*> tuple has a higher value than the <*Chewing, Swallowing*> which means when the true label is *Swallowing* it

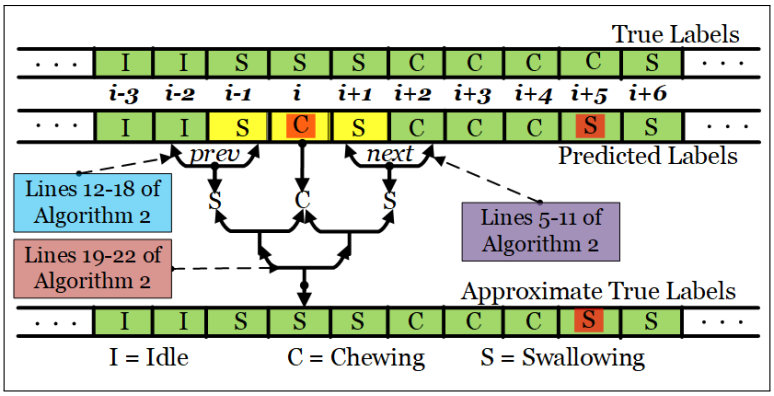


Figure 4.5: Operational Example of TLA Algorithm

is more often confused with *Chewing* than vice versa. So, the label for $(i+1)^{th}$ window is considered as *Swallowing*. Similarly, the label for $(i-1)^{th}$ window is also considered as *Swallowing*. As the label for both previous and the next window of i^{th} one is *Swallowing*, the i^{th} window is more likely to be *Swallowing* instead of *Chewing*. If they were not same, we would again take the help of confusion matrix to decide the label for i^{th} window. It is to note that, our TLA algorithm does not correct all the incorrect predictions. It only tries to correct the suspicious predictions based on Lemma 1. As shown in Figure 4.5, the incorrect prediction in $(i+5)^{th}$ window is not corrected. That is why the output of our TLA algorithm is called approximate true labels.

Updating the Classifier

Once the TLA method generates the approximate true labels, we use the data segments generated during the online classification phase and the corresponding approximate true labels to update the current NN. The weights of the NN are updated using the 'Adam' optimizer's update rule [150].

Validation of Updated Classifier

Once we update the current NN, we calculate the accuracy of this updated one for the validation dataset of the respective user. Then we use this validation accuracy as a measure to understand whether the updated network is better than the current one or not. If the validation accuracy of the updated one is better than the current one then we validate the update and replace the current NN by the updated one. Correspondingly, the confusion matrix of the updated one is used by the TLA algorithm in the next update phase. Otherwise, we discard the update and keep on using the current NN. Thus, we protect the classifier from being affected by unusual data and avoid the avalanche effect of poor classification. As the update decision is made based on how the updated classifier performs on the individual's data, having a better accuracy on the validation dataset means the classifier is getting personalized for that individual. And having a personalized model for each user is one of the key goals of personalized health care. It is to note that, the validation dataset for each individual should also be updated periodically to reflect the changing eating habits of the user.

4.4 Experimental Setup

4.4.1 Wearable Setup

We have developed a wearable neckband equipped with a piezoelectric strip sensor, and a microphone coupled with an amplifier integrated to a Bluetooth 4.0 LE enabled low powered RFDuino microcontroller. The embedded processor is an ARM Cortex M0 with 256kB of flash memory and 16 KB of RAM. The microcontroller is powered by a 250 mAh lithium coin cell battery placed in a battery holder. All the components (microcontroller, battery,

piezoelectric sensor, microphone) were integrated into the 3D printed box as shown in Figure 4.6. The box was mounted with an elastic rubber band around the neck using velcro patches. The neckband is designed in such a way that, the piezoelectric sensor is placed on the larynx and the microphone is placed in between the larynx and the laryngopharynx region of the throat. The microcontroller samples the data from the piezoelectric sensor and microphone at a sampling rate of 20 Hz and sends the data to a mobile phone over Bluetooth. The wearable neckband was tested for battery life and it lasted around 26 hours with a 250 mAh coin cell battery while continuously streaming the data.

4.4.2 User Information and Data Collection

A total of 12 users (6 female, 6 male) voluntarily participated in our experiment with the approval of the Institutional Review Board (IRB) of the university. The users were aged between 20 yrs to 33 yrs with an average body mass index (BMI) of 23.64±4.88 kg/m^2. The users were asked to eat different kinds of everyday foods like- a slice of bread, cookies, chips, almonds, and also drank water in between. Data for a total of 204 eating sessions were collected from 12 users over 1 month in a lab environment. Each eating sessions lasted for around 8 minutes. For experimental purpose, we have collected labeled data from the users. For that, we have developed a desktop application, where the users can see the readings from

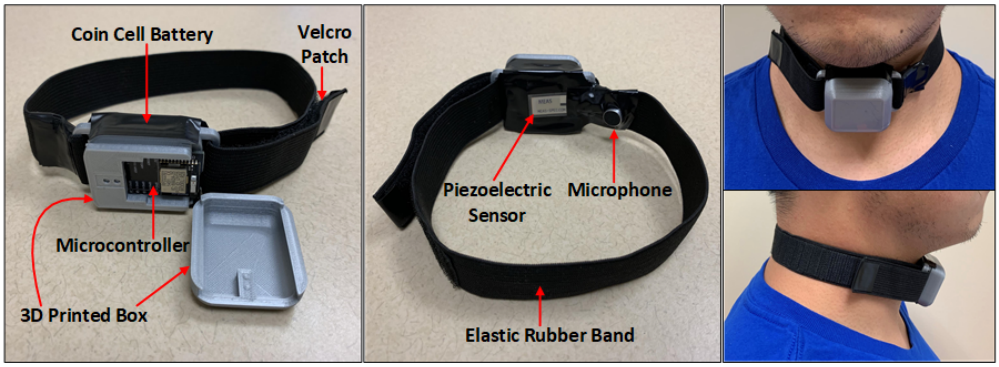


Figure 4.6: Wearable Neckband

different sensors of the neckband while eating. They also label activities while performing them by pressing the corresponding button for each activity. All the data were collected in the presence of 3 human experts who ensured the correct labeling of the data. Figure 4.7 shows a sample of the collected labelled data. As depicted in Figure 7, the spikes for *Talking* activity is more significant and regular in acoustic sensor data than the piezoelectric one which is expected. Similarly, the *Chewing* and *Swallowing* activities generate more significant spikes in piezoelectric sensor data than the acoustic one. Moreover, the spikes in piezoelectric sensor data for *Chewing* is sharper and more frequent than *Swallowing* which completely aligns with human eating behavior.

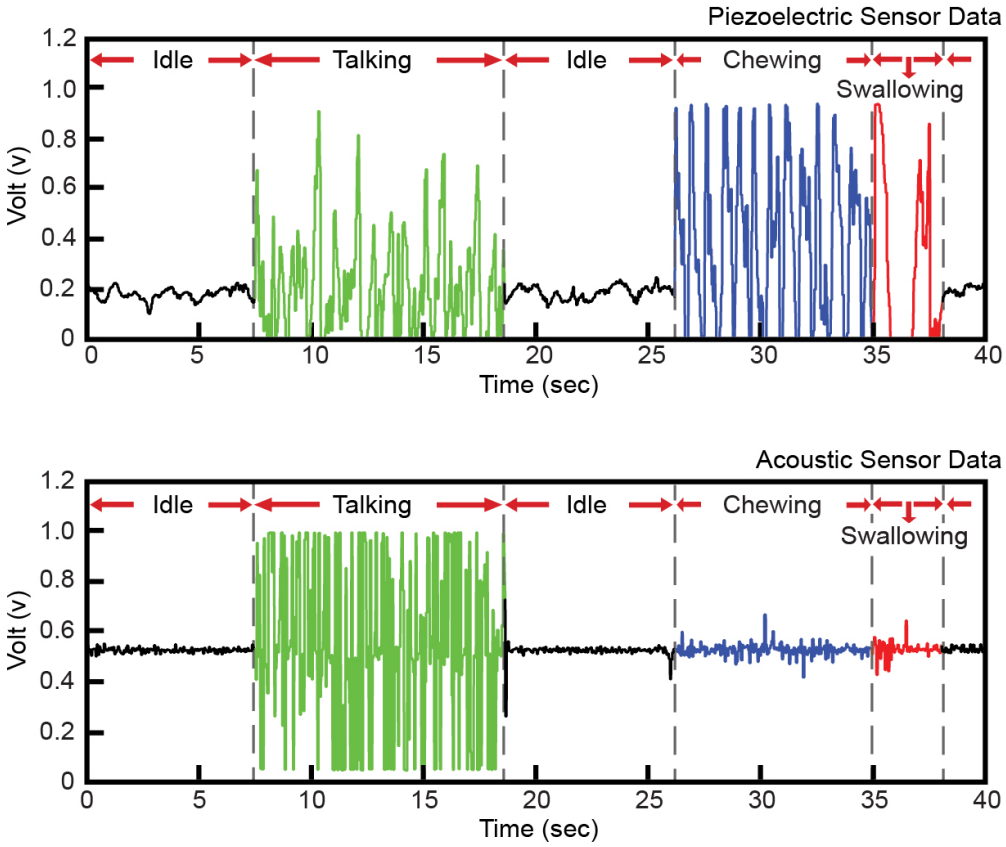


Figure 4.7: Sample of Collected Labeled Data

4.5 Experimental Evaluation

4.5.1 Offline Training Phase

For the offline training phase of the NN classifier, we used data from the 6 users (3 female, 3 male). Data for 10 eating sessions from each user resulting in a total of 60 eating sessions were used in this phase. Comparative analysis of our classifier against other related works is not possible as different works considered different sensors to recognize different activities. However, we do a comparative analysis of the NN with other state-of-the-art classifiers [151–156] during the offline training phase as shown in Table 4.3. As per the standard practice, 80% of the data were used for training and rest 20% were used for testing the classifiers.

Table 4.3: Accuracy Comparison of Offline Classifiers

Classifier	**Training Accuracy** (%)	**Testing Accuracy** (%)
k-NN (k=3)	96.43	91.15
Decision Trees	100	87.41
Random Forest (n=100)	99.99	91.42
Adaboost	76.85	74.33
Naive Bayes	91.36	87.34
SVM	92.78	91.21
Neural Network	94.06	91.78

4.5.2 Online Learning

The online learning involves the online classification phase and the online update phase of our proposed HEAR methodology. To evaluate our proposed methodology for online learning we have used data of 20 eating sessions from each of the 6 new users (3 female, 3 male). Users 1, 2, 3 represent male and users 4, 5, 6 are female. For each new user, we also maintain a separate validation dataset of 4 eating sessions of the respective user and the confusion matrix of the current NN on that validation dataset. The confusion matrix is used by the

TLA algorithm and the validation dataset is used to validate the update during online update phase.

Table 4.4: Confusion Matrix for Offline NN on Validation Dataset of User1

True Label	**Predicted Label**			
	Talking	Chewing	Swallowing	Idle
Talking	95.17%	0.48%	0.00%	4.35%
Chewing	0.00%	89.33%	10.67%	0.0%
Swallowing	0.00%	16.18%	78.11%	5.71%
Idle	0.60%	0.00%	2.41%	96.99%

Table 4.4 shows the confusion matrix of the offline trained NN (initial NN) on the validation dataset for User 1. For each user, the online classification phase initially starts with the offline trained NN and considered as the current one. For each eating session, the online classification phase is followed by the online update phase.

Figure 4.8, shows the percentage of the actual prediction errors detected by TLA algorithm using the Lemma 1. Accuracy of TLA algorithm in correcting the detected errors using the confusion matrix based heuristics is also shown in Figure 4.8. The detection and correction accuracy are calculated from the 20 eating sessions of each user and presented with 95% confidence interval. As different users have a different eating pattern, the detection and correction accuracy also varies greatly from user to user. As shown in Figure 4.8, the TLA algorithm has the highest detection and correction accuracy for user 5 which is 24.51% and 79.59% respectively. On the other hand user 4 has the lowest detection and correction accuracy which are 11.38% and 64.17% respectively. It means user 4 has a very different and unpredictable eating pattern compared to other users.

Figure 4.9 presents how the validation accuracy of the OLNN improves over 20 eating sessions for each new user. It indicates how the offline trained neural network gets better through online update phase (Section 4.3.3). It is to note that, the validation accuracy is calculated using the user specific validation dataset of each user. To reflect the changing eating habits

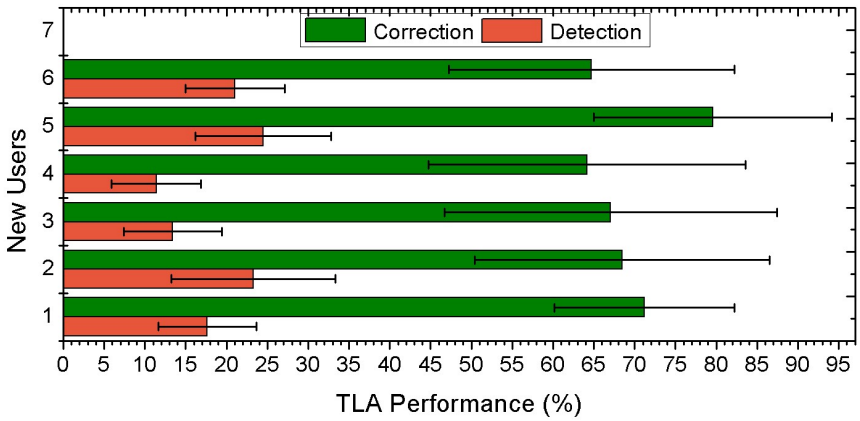


Figure 4.8: Performance of TLA Algorithm

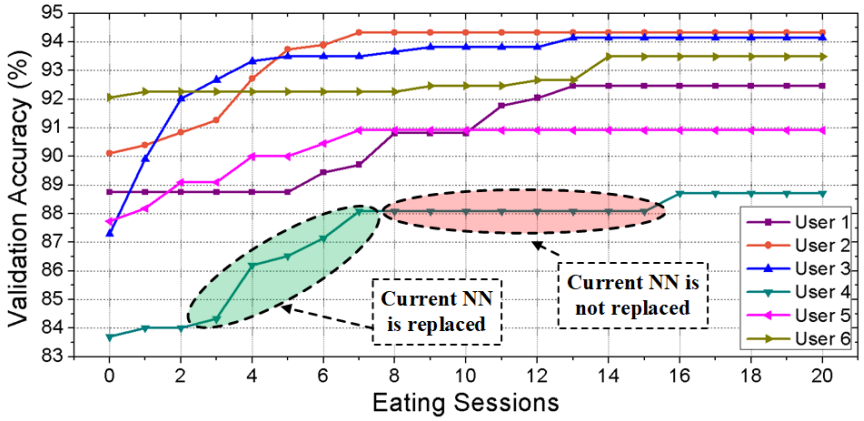


Figure 4.9: Change of Validation Accuracy for OLNN

of the user, the validation dataset is also updated. For our experiment, we updated the validation dataset in a first in first out (FIFO) manner after each day. Initially, we collect 4 eating sessions of data for validation dataset for each new user. After each day one new eating session of data is added to the validation dataset and the data of the earliest eating session from the validation dataset is removed. For 20 eating sessions of testing data collected over 5 days (4 eating sessions in each day) for each new user, the validation dataset is updated after each day. As shown in Figure 4.9, the validation accuracy for the 0^{th} eating sessions is for the offline trained NN. When the validation accuracy after an eating session is greater than the previous one, that means the current NN gets replaced by the updated one after that eating session. As depicted in Figure 4.9, the green marked region for user 4 indicates the eating

sessions after which the current NN is replaced by the updated one. On the other hand, the red marked region for user 4 represents the eating sessions where the validation accuracy does not change. That means the current NN does not get replaced by the updated one after those eating sessions. It applies for all the other users as well. It is to note that, OLNN has the over all lowest validation accuracy for user 4 compared to other users which means user 4 has a very different eating pattern. It also justifies the relatively poor performance of TLA algorithm in detecting and correcting the actual prediction errors for user 4 as mentioned in the previous paragraph. Another very important thing to notice from Figure 4.9 is, the validation accuracy of OLNN for each user either remains the same or increases but never decreases than before. Thus it proofs that, our HEAR methodology protects OLNN against unusual data.

Figure 4.10, shows the average classification accuracy of all the offline trained classifiers along with our OLNN with 95% confidence interval. For each of the 6 new users, the average classification accuracy is measured for 20 eating sessions. As depcited in Figure 4.10, our OLNN performs better than any other offline trained classifiers for each of the 6 new users. The overall classification accuracy of OLNN for 120 (6x20) eating sessions is 92.09% which is greater than offline classifiers - NN (89.39%), SVM (89.27%), k-NN (87.56%), Random Forest (86.15%), AdaBoost (77.72%), Naive Bayes (73.14%), Decision Tree (70.15%).

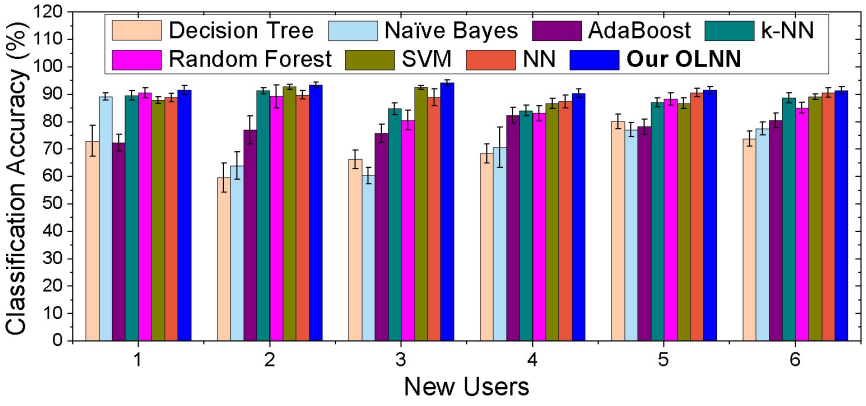


Figure 4.10: Various Classifiers' Accuracy on New Users

4.5.3 Power and Energy Evaluation

We evaluate the energy consumption of our OLNN with k-NN, SVM and Random Forest as they show competitive accuracy during the online classification phase as shown in Figure 4.10. The energy consumption was calculated in a mobile phone with an ARM Cortex-A53 (ARMv8) 64-bit processor with a clock speed of 1.2 GHz. It also has 1 GB of RAM with 64 GB of storage capacity. Table 4.5 shows the energy consumption of the competitive classifiers for each segment (1 second) of data. The energy consumption of OLNN combining the classification and update phase is 205.24 μJ which is 41.06% less than SVM, 59.38% less than Random forest and 76.63% less than k-NN classifier. The only overhead that our HEAR methodology incurs is the memory. We have to store the data of the last eating session, the validation dataset of the user and the corresponding confusion matrix. As the sampling frequency of wearable neckband is only 20 Hz, an eating session of 8 minutes generates around 150 KB of data. In that case, one eating session plus 4 eating sessions of validation dataset require 750 KB of storage in total. Given the higher memory and storage capacity of recent mobile devices, it should not be a problem as long as we follow fog computing architecture.

Table 4.5: Energy Consumption of the Competitive Classifiers

Classifiers		**Exec. Time** (ms)	**Avg. Power** (mW)	**Energy** (μJ)
k-NN Classification		1.956	449.08	878.40
Random Forest Classification		1.280	394.74	505.27
SVM Classification		1.012	344.08	348.21
OLNN	Classification	**.036**	**374**	**13.46**
	Update	**.448**	**428.08**	**191.78**
	Total	**.484**	**401.04**	**205.24**

4.6 Discussion & Future Work

The main goal of this chapter is to demonstrate the importance of online learning and its associated challenges in the food intake monitoring application. Although we investigated this problem concentrating on the food intake monitoring application, the challenges of online learning can be generalized for any healthcare application using wearable devices. We also propose the HEAR methodology that overcomes the challenges with no significant overhead. The advantage of our methodology is that it has the potential to be generalized for other healthcare applications like activity recognition where the average/minimum duration of tasks or activities are bound by the physiological properties of humans. Then with careful selection of the segmentation window, it is possible to formulate a lemma similar to lemma 1 in this chapter. We plan to explore this possibility in our future work. In that case, the TLA algorithm has to be modified accordingly based on the lemma specifically designed for the problem. To evaluate our proposed methodology, we have developed a wearable neckband as they are not commercially available. While many state-of-the-art works focused on the design, development, and usability of the neckband, this chapter mostly focuses on the methodology that is applicable regardless of the wearable devices. In the future, we plan to extend our methodology for the activity recognition problem.

4.7 Summary

Human eating habits change over time and vary from person to person. In this chapter, we propose a Human Eating Activity Recognition (HEAR) methodology which uses an online update phase to keep up with these changes. In this regard, we designed an algorithm that creates approximate true labels for the new eating data. We have also designed a wearable neckband to capture the eating activity related data in a lab environment. Through detailed

experimental evaluation of our methodology, we show that an online learned neural network classifier outperforms state-of-the-art offline trained classifiers while also being more energy-efficient.

Chapter 5

Stress Detection using Context-Aware Sensor Fusion from Wearable Devices

5.1 Abstract

Wearable medical technology has become increasingly popular in recent years. One function of wearable health devices is stress detection, which relies on sensor inputs to determine a patient's mental state. This continuous, real-time monitoring can provide healthcare professionals with vital physiological data and enhance the quality of patient care. Current methods of stress detection lack: (i) robustness—wearable health sensors contain high levels of measurement noise that degrades performance, and (ii) adaptation—static architectures fail to adapt to changing contexts in sensing conditions. We propose to address these deficiencies with SELF-CARE, a generalized selective sensor fusion method of stress detection that employs novel techniques of context identification and ensemble machine learning. SELF-CARE uses a learning-based classifier to process sensor features and model the environmental variations in sensing conditions known as the noise context. SELF-CARE uses noise context

to selectively fuse different sensor combinations across an ensemble of models to perform robust stress classification. Our findings suggest that for wrist-worn devices, sensors that measure motion are most suitable to understand noise context, while for chest-worn devices, the most suitable sensors are those that detect muscle contraction. We demonstrate SELF-CARE's state-of-the-art performance on the WESAD dataset. Using wrist-based sensors, SELF-CARE achieves 86.34% and 94.12% accuracy for the 3-class and 2-class stress classification problems, respectively. For chest-based wearable sensors, SELF-CARE achieves 86.19% (3-class) and 93.68% (2-class) classification accuracy. This work demonstrates the benefits of utilizing selective, context-aware sensor fusion in mobile health sensing that can be applied broadly to Internet of Things applications. The findings in this chapter have been published in [159, 160].

5.2 Introduction

Advancement in technology and the prevalence of Internet of Things (IoT) has led to the wide adoption of wearable medical devices in recent years. Wearable medical devices have shaped the study and practice of healthcare by allowing continuous, remote monitoring of vital physiological signs [40, 159, 161, 162]. Wearable health devices can also be used for stress detection, which uses inputs from body-worn sensors to analyze a patient's mental state [163–165].

Stress detection is of growing interest as recently the American Psychological Association issued a warning about long-term physical and mental health impacts due to stresses from the COVID-19 Pandemic, deeming it a *'a national mental health crisis'* [163].

Medically, stress is a physiological state that can be triggered by hormonal surges during moments of physical, cognitive, or emotional challenges [164]. Stress detection falls under the

umbrella of *affective computing*—the area of computing that allows machines to recognize and interpret human emotions [166]. Affective computing using wearable devices is a rapidly developing industry, the value of which is projected to expand from $29 billion to $140 billion—an increase of nearly five times—by 2025 [167].

5.2.1 Research Challenges

The increasing prevalence of wearable health technology—and the data that can be gleaned from this technology—has given rise to a body of academic literature focusing on stress detection [168–174]. The relationship between this sensor data and stress states is not governed by known physical equations. As a result, researchers have used classical machine learning models (e.g., random forests, decision trees) or deep learning models (e.g., convolutional neural networks, long short-term memory) to perform stress classification via supervised learning over labeled datasets with annotated stress states [110, 175–177]. Deep learning models have benefits in their ability to incorporate temporal modeling from the sensor data into the stress detection problem. Despite this, in stress detection, classical machine learning models have been more widely adopted compared to deep learning models due to the classical models' lower complexity levels, important for wearable on-device deployment [39]. However, both of these types of learning-based methods lack robustness when using single sensor modalities, since the coverage area of each sensing modality is limited by the domain in which the sensors operate [178].

Researchers commonly use sensor fusion across multi-modal physiological data to increase the performance of emotion recognition from wearable devices [179]. *Early* fusion (also known as feature-level fusion) focuses on combining data at the raw-data level. Alternatively, *late* fusion (also known as decision-level fusion) combines the final outputs of a system. Current methods of sensor fusion that employ combinations of early and late fusion still have limited

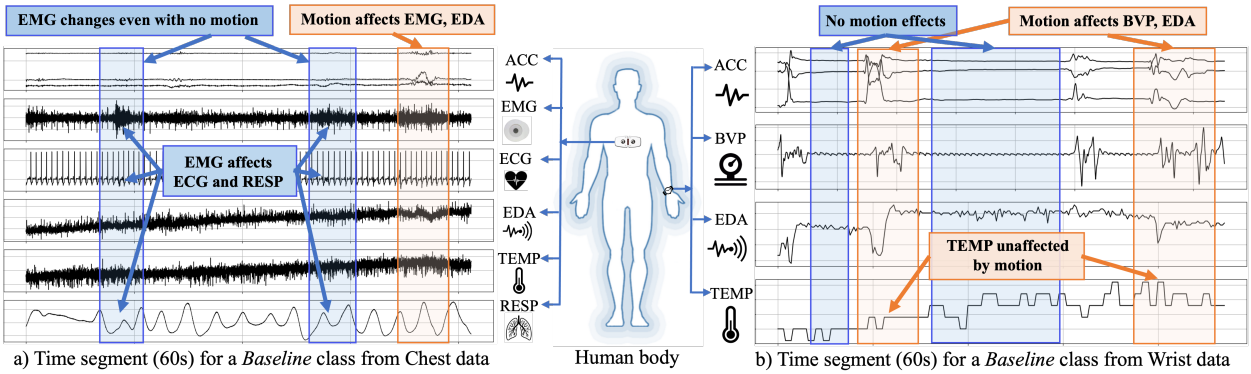


Figure 5.1: The context of noise on sensors depends on the respective sensor locations on the human body. a) Physiological signals from chest sensors. A baseline segment where EMG affects ECG and RESP even with no motion whereas, ECG remains unaffected even during motion. This shows that EMG is more suitable than ACC to understand the noise context from chest wearable devices. b) Physiological signals from wrist sensors. A baseline segment where BVP and EDA is affected due to motion. Hence, ACC is more suitable to understand the noise context in wrist wearable devices. Both sets of data in a) and b) are taken from wrist and chest sensors on one subject from the WESAD dataset.

efficacy due to the use of static architectures that cannot adapt to changing sensing conditions within the environment [180].

Another notable challenge in using data from these physiological signals for affective computing is that the data may be susceptible to substantial amounts of sensor noise due to physical motion or muscle contraction. Throughout the remainder of this chapter, we define the noise context of wearable health sensors as the group of external factors that can influence the variation in measurements and noise levels of the sensors. This context can be interpreted through intra-sensor relationships in the device as well as through sensing conditions surrounding the device (*e.g.*, the location of a wearable sensor on the body). And fusing data from multiple sensors without understanding the noise context may lead to performance degradation as found in [174].

The main research challenges we address in this work include: (i) how to effectively fuse multi-modal sensor data from wearable devices; (ii) how to develop an adaptive architecture to account for variations in sensing conditions; and (iii) how to model noise context in

wearable sensors to improve stress classification performance.

5.2.2 Motivation

In this subsection, we provide motivation and qualitative analysis regarding the challenges our approach addresses. Fig. 5.1 shows that the context of noise on sensors varies depending on the location of the wearable device. Fusing such noisy measurements can subsequently degrade the classification performance [39]. For example, Fig. 5.1 b) represents a baseline segment of data from four wrist sensing modalities: tri-axis accelerometer (ACC), blood volume pulse (BVP), electrodermal activity(EDA), skin temperature (TEMP). At several times during the segment, significant motion causes two of the sensors (BVP, EDA) to vary in their readings, which could cause a model to classify this segment *incorrectly* as stress. Therefore, it is important to understand the noise context when making sensor fusion decisions. Moreover, it also shows that motion sensors (ACC) have benefits for modeling the noise context in wrist-worn devices.

On the other hand, Fig. 5.1 a) shows data from six sensing modalities from the chest (ACC, electromyography: EMG, electrocardiogram: ECG, EDA, TEMP, respiration: RESP) for a baseline segment of the subject. While chest motion may affect EMG and EDA, it does not affect ECG. However, EMG may be affected even without any motion when the subject makes muscle contractions without moving. This may in turn affect ECG and RESP as shown in Fig. 5.1. Thus, for chest wearable devices, motion is not the best modality to understand the noise context for sensor fusion decisions. Rather, EMG is more suitable for chest-worn devices which is empirically validated later in Section 5.6.

The aforementioned examples motivate us to develop a context-aware sensor fusion technique that utilizes the noise context of wearable devices to make sensor fusion decisions, which will help us to maintain performance while avoiding misclassification. Moreover, the developed

method should be generalizable to both chest and wrist wearable devices as the noise context varies based on the location of wearable devices. Prior work has shown that stress detection using wrist-based wearable devices can be improved by modeling noise context [159], however, the differences in using chest-based wearable devices have yet to be examined.

5.2.3 Contributions

In this chapter, we propose SELF-CARE, a generalized stress detection method that utilizes the noise context of wearable devices to perform sensor fusion. We show that while motion-based noise context understanding works best for wrist-based wearable devices, muscle contraction works best for chest-based wearable devices. Through experimental evaluation, we demonstrate that EMG is better than ACC in understanding the noise context of chest-based wearable devices.

The key contributions of this chapter are as follows:

1. We introduce a generalized selective sensor fusion method, SELF-CARE, for stress detection from wearable health sensors. SELF-CARE implements a novel context identification method that models noise context based on the location of wearable devices (chest or wrist), and utilizes the noise context to dynamically adjust the sensor fusion performed across an ensemble of machine learning classifiers to improve classification performance.

2. We empirically demonstrate that noise context varies based on the location of wearable devices through experimentation across nine different wearable sensors. Our findings suggest that while motion (ACC) is most suitable to understand the noise context in wrist-worn devices, muscle contraction (EMG) is more suitable to determine noise context in chest-worn devices.

3. We propose a novel late fusion technique for classification over an ensemble of learners using a Kalman filter that incorporates temporal dynamics.

4. We perform an extensive performance evaluation of the different combinations of sensors from chest and wrist wearable devices for stress detection. This may serve as the benchmark for the research community to understand, evaluate, and compare the impact of sensor fusion in stress detection.

5. We validate our methodology on the WESAD dataset, showing that SELF-CARE is suitable for wrist-based and chest-based wearable devices and achieves state-of-the-art performance for the 3-class and 2-class stress detection problems.

5.2.4 Chapter Organization

The remainder of this chapter is structured as follows. In Section 5.3, we discuss related works in stress and emotion detection and sensor fusion. In Section 5.4, we describe the stress classification problem formulation. In Section 5.5, we introduce the methodology of our context-aware, selective sensor fusion approach. In Section 5.6 we show the results of our approach on a publicly available stress classification dataset. In Section 5.7, we highlight future directions and limitations, and in Section 5.8, we provide concluding remarks.

5.3 Related Works

As this chapter presents a context-aware sensor fusion technique for stress detection, we consider the related works from stress detection and sensor fusion. Therefore, we categorize the related works into two parts. In Section 5.3.1, we present some related works that consider stress and emotion detection using various sensor modalities. We also discuss the

availability of the dataset used in the corresponding works. In Section 5.3.2, we present and compare against the works that mainly focus on sensor fusion techniques for stress detection.

5.3.1 Stress and Emotion Detection

A number of studies [168–170] focus on detecting stress or emotion from physiological signals such as electrocardiograms (ECGs), electromyograms (EMGs), blood volume pulse (BVP), respiration (RESP), electrodermal activity (EDA), and skin temperature (TEMP). However, these datasets are not publicly available. Among works with publicly available datasets, authors in [171] detect stress while driving a vehicle, while [172] and [173] perform a more complex analysis on subjects' general emotional states. However, these datasets are limited in that they do not include data on both stress and additional emotions simultaneously.

Authors in [174] created the WESAD (Wearable Stress and Affect Detection) dataset, which includes data on both stress and amusement states from chest- and wrist-worn devices. Moreover, the authors compare the classification performance of multiple common machine learning methods using chest-worn sensors, wrist-worn sensors, and their combinations. They conclude that: (i) chest sensors perform better, and wrist sensors become redundant and sometimes even decrease performance, (ii) fusing multiple sensor modalities together can improve results, and (iii) the accelerometer can negatively impact classification performance. The third finding supports our claims that modeling the context as a learned abstraction of motion can be beneficial for wearable devices, and that sometimes fusing all available sensors together reduces performance. Authors in [175] use the WESAD dataset to present a translation method using a Generative Adversarial Network (GAN) to generate chest sensor features using the wrist sensors. However, the higher computational complexity of GANs, along with the requirement of chest data during training, limits the application for computing on a wrist-worn device. Authors in [110] propose a hybrid convolutional

neural network (CNN) architecture that uses both manually extracted and CNN features for classification, but only uses one sensing modality. The authors of [181, 182] and [176] explore the feasibility of deep learning models for stress and emotion detection using the WESAD dataset. However, traditional machine learning models are currently favored over deep learning approaches due to deep learning's increased computational complexity and lack of explainability [39, 179].

5.3.2 Sensor Fusion

Sensor fusion has many benefits when applied to both physiological signals and stress recognition [38, 179]. By combining raw-sensor data or features (early fusion), more information can be extracted from sensor measurements than would otherwise be available. Likewise, using an ensemble method of multiple learners (late fusion) can increase robustness to sensor/classifier errors. Performing late fusion on the outputs of multiple classifiers can improve performance, as each classifier can be specialized for its particular set of input data [183]. Traditional late fusion approaches typically use a voting method over the outputs of the classifiers to make a final decision. Other works have also proposed a learned late fusion method, such as the method discussed in [184]. The authors propose an adaptive fusion method, detailing the benefits of using event-related feature extraction techniques along with an adaptive framework. However, their approach does not consider the noise context of data for the sensor fusion decision as we do in our approach. Additionally, we also show that the noise context varies based on the location of wearable devices, which has not been addressed in their work. Furthermore, although their late fusion is adaptive, their method is static in that it requires a set number of classifiers. Our model, on the other hand, can dynamically adjust the number and type of classifiers used based on the performance-computation trade-off.

Lastly, sensor fusion presents additional benefits when fusing time-series data with temporal correlations, like the data present in physiological sensors. Kalman filters are tools for estimating unknown quantities by iteratively predicting and updating the estimated state of interest [185], which in our case is the predicted class. Some works propose using Kalman filters to solve classification problems, [186], while other works do not consider temporal aspects within their formulation. In this work, we present a novel late fusion method using a Kalman filter to take advantage of the temporal dynamics in the stress classification problem.

5.4 Problem Formulation

As discussed in Section 5.3, fusing multiple heterogeneous physiological signals has benefits for stress detection. The main sources of these physiological signals are generally either chest- or wrist-worn wearable devices. Between the two, wrist-worn wearable devices are more prone to noise induced by random movements of hands, and, as shown in Fig. 5.1, movements create varying impacts on different physiological signals. Fusing such noisy signals often deteriorates the classification performance [174]. On the other hand, chest-worn wearable devices are less prone to random movements due to their location, but signals may be affected or become noisy for other reasons, such as muscle contraction. Therefore, it is important to understand the context of the noise which varies based on the location of the wearable devices. Understanding the noise context can help to dynamically select the less impacted signals to be fused, which will eventually improve the classification performance. The problem formulation for stress detection in a selective approach is provided as follows.

For each input segment of sensor data, the goal of a classifier ϕ is to utilize the measurements from available sensors, $\mathbf{X}$, to classify the segment, $\mathbf{Y}$:

$$\mathbf{Y} = \phi(\mathbf{X}) = [p_1, p_2, \dots p_c] \tag{5.1}$$

$$\mathbf{X} = \{\mathbf{X}_i\}_{i=1...s} \tag{5.2}$$

where s is the number of available sensors; $\mathbf{X}_i$ represents the measurements from sensor i; and $\mathbf{Y}$ represents the classifier output which is comprised of the probabilities p of the c classes, (e.g., $c = 1$: baseline, $c = 2$: stress, $c = 3$: amusement). ϕ can be implemented via traditional sensor fusion techniques, a machine learning (ML) or deep learning (DL) model, or an ensemble of ML/DL models.

Since $\mathbf{X}$ represents data from multiple heterogeneous sensing modalities, sensor fusion can be used to fuse the data to provide a better estimate of $\mathbf{Y}$. In early fusion, the raw sensor inputs are fused before being passed through the classifier as follows:

$$\hat{\mathbf{Y}} = \phi(\psi(\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_s)) \tag{5.3}$$

where ψ represents the function for fusing the different inputs. In contrast, *late fusion*, involves fusing the outputs of an ensemble of sensor-specific classifiers as follows:

$$\hat{\mathbf{Y}}_1,\ \hat{\mathbf{Y}}_2,\ \ldots,\ \hat{\mathbf{Y}}_s = \phi_1(\mathbf{X}_1),\ \phi_2(\mathbf{X}_2),\ \ldots,\ \phi_s(\mathbf{X}_s) \tag{5.4}$$

$$\hat{\mathbf{Y}} = \phi(\hat{\mathbf{Y}}_1, \hat{\mathbf{Y}}_2, \ldots, \hat{\mathbf{Y}}_s) \tag{5.5}$$

The context of noise can vary dramatically based on the wearable device location and may have a range of impacts on different sensor modalities. This variance calls for the use of an adaptive ϕ that selects the sensor modalities to be fused based on the noise context—

for example, movements of hands in wrist-worn wearable devices or muscle contractions in chest-worn wearable devices. In this case, ϕ represents an ensemble of classification models, and ϕ^* represents the selected best subset of models in the ensemble for a given input $\mathbf{X}$. The context of the noise (either learned and modeled from the inputs or provided externally) is denoted as Ω. We introduce the context identification problem formulation as:

$$\Omega = \pi(\mathbf{X}), \tag{5.6}$$

$$\phi^* = \rho(\Omega), \tag{5.7}$$

where π represents a gating model that performs context identification, and ρ represents the mechanism for selecting ϕ^* given the identified context Ω. The goal of π and ρ is to select the optimal subset of branch models ϕ^* for the inferred context Ω to maximize stress classification performance for a given $\mathbf{X}$. In our specific case, context is defined as motion for wrist-worn wearable devices or as muscle contraction for chest-worn wearable devices. The inputs to π typically consist of measurements from the accelerometer (wrist-worn) or EMG (chest-worn) based on the wearable device location.

5.5 Methodology

In this Section we detail our method, SELF-CARE, depicted in Fig. 5.2. Our method performs stress classification given input sensor measurements from a specified time segment using four main blocks: (i) preprocessing, (ii) context identification, (iii) branch classifiers, and (iv) late fusion. SELF-CARE takes the form of a multi-branched architecture in which different "branches" represent stress detection classifiers using different combinations of sen-

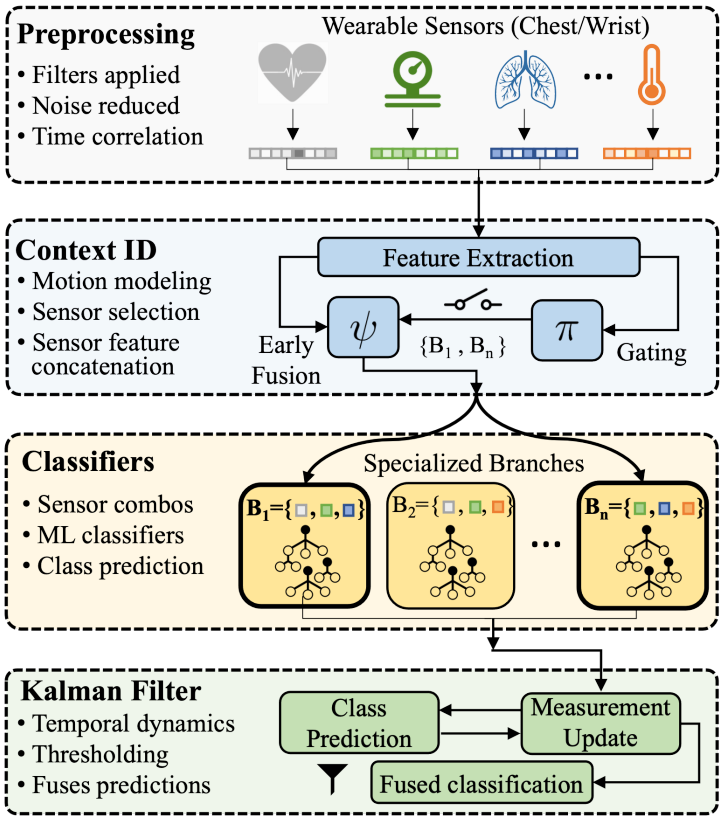


Figure 5.2: Proposed SELF-CARE Architecture. In this depiction different types of chest/wrist-worn sensors are used, the gating model selects the branches given the context, a Random Forest/AdaBoost classifier is used for the branch models, and a Kalman filter is used for the late fusion over the selected branches.

sors. Context identification selects which branches to execute, while late fusion is used to fuse the stress classification predictions if multiple branches are selected. The following subsections provide further details on the proposed method.

5.5.1 Preprocessing Step

SELF-CARE can take in data from varying numbers of heterogeneous or homogeneous physiological sensors as inputs. Preprocessing is a common step when dealing with raw, unfiltered sensor data. By applying various filters (e.g., band-pass filters or lowpass filters) to the input data, random noises are reduced, and important features are more easily extracted. The

preprocessing performed over each sensing modality is detailed in Section 5.6.

5.5.2 Context Identification

Feature Extraction

The purpose of the context identification block is to predict which branch classifier(s) will perform the best given an input set of sensor features that are used to model the context of the system. Contextual modeling can help illuminate the performance of various sensors in terms of their levels of noise under different situations and the locations of the wearable device on the human body. For wrist wearable devices, we use motion to model the context. Therefore, for wrist devices, we first extract only ACC features as they are directly related to the relative motion of the test subject. For chest wearable devices, on the other hand, the context is best modeled by muscle contraction, which is captured by EMG signal. We then extract EMG features for chest-worn devices for contextual modeling. Next, these features are processed by the gating model to select the best performing branch. The feature extraction of the other modalities takes place after the gating model has selected which branch(es) will be used for classification.

Gating Model (π)

The gating model trains a classifier that uses the ACC/EMG features as inputs to select one of the available branch classifiers according to wrist/chest-worn devices. For wrist-worn device, we shortlist these three branches:WB_1={BVP, EDA, TEMP}; WB_2={ACC, BVP, EDA}; WB_3={BVP, EDA} using Random Forest classifier for both 3-class and 2-class classification. Similarly, for chest-worn devices, we shortlist five branches for 3-class and 2-class classification using AdaBoost classifiers. For 3-class classification, the short-

listed branches are: CB_1={ECG, RESP, EMG, EDA, TEMP}; CB_{12}={ECG, EMG, EDA, TEMP}; CB_{14}={RESP, EMG, EDA, TEMP}; CB_{24}={ECG, EMG, EDA}; CB_{27}={EMG, EDA, TEMP}. For 2-class classification, the shortlisted branches are: CB_5={ACC, ECG, RESP, EDA}; CB_7={ACC, ECG, EMG, EDA}; CB_9={ACC, ECG, EDA, TEMP}; CB_{13}={ECG, RESP, EDA, TEMP}; CB_{20}={ACC, ECG, EDA}. The process for choosing these branches is discussed further in Section 5.6.2. We employ a Decision Tree (DT) classifier for our gating model because it is lightweight and adds minimum overhead to our architecture.

Performance-Computation Trade-off (δ)

An important feature of SELF-CARE is its ability to balance constraints between performance and computation. We introduce the term δ that aids the gating decision in considering this trade-off. The gating model outputs prediction probabilities for the available branches with $\bar{b}$ representing the maximum probability branch. δ has a range between 0 and 1, representing the range in which non-maximum branches are selected by allowing branches with probabilities greater than $\bar{b} - \delta$ to be also selected. Lower δ values indicate tighter computation constraints, with $\delta = 0$ indicating that only the highest probability branch from the gating classifier is selected, while higher δ values allow more branches to be selected, with $\delta = 1$ indicating that all possible branches are selected.

Early Fusion (ψ)

Once the branches are selected after applying δ on the gating model decision, the features for those branches will be extracted and concatenated together to be passed to the corresponding classifiers. For example, while using wrist modalities, if WB_1 and WB_3 are the selected branches by the gating model for either 3-class or 2-class classification, the features from BVP, EDA, and TEMP signals are concatenated together using early fusion for WB_1, while

features from BVP and EDA are fused for branch WB_3. Similarly, for 3-class classification using chest modalities, the features from ECG, RESP, EMG, EDA, and TEMP are fused together if the gating model selects the CB_1 branch.

5.5.3 Branch Classifiers

Next, the corresponding branch classifier(s) are used to classify the segment. For our approach, we use a Random Forest (RF) classifier for all three branches of wrist modalities for 3-class and 2-class classification. For chest modalities, we use the AdaBoost classifier for all five branches for 3-class and 2-class classification. The details of the classifier training and selection are provided below in Section 5.6.2. Currently, SELF-CARE operates using either only wrist sensors or only chest sensors, however, our method is capable of integrating both sets of branches with modifications to the context identification module. Each selected branch produces a classification prediction to serve as input for the late fusion method.

5.5.4 Late Fusion Method

The late fusion method is tasked with fusing the class predictions from the various selected branches, $\{\hat{\mathbf{Y}}_1, \hat{\mathbf{Y}}_2, \ldots, \hat{\mathbf{Y}}_s\}$, with the goal of producing higher accuracy predictions than any one individual branch by itself. Here we present our *Kalman filter-based* method for classification over an ensemble of classifiers.

Kalman filters are powerful and commonly used tools for sensor fusion and the broader field of estimation. They are designed to estimate the unknown state of a system along with the state's uncertainty by performing a series of recursive predictions and measurement updates. In the context of our problem, we consider a Kalman filter approach towards the multi-class classification problem like in [186], and we additionally model the temporal dynamics in the

stress classification problem for each sample at time k. The general form of the discretized linear dynamics of a system with state $\mathbf{x}$ and measurements $\mathbf{z}$ is given as:

$$\mathbf{x}(k) = \mathbf{F}x(k-1) + \mathbf{v}(k) \tag{5.8}$$

$$\mathbf{z}(k) = \mathbf{H}x(k) + \mathbf{w}(k) \tag{5.9}$$

where $\mathbf{F}$ is the state transition matrix; $\mathbf{v}$ is the process noise vector, which is modeled as zero-mean, normally distributed random variable with covariance, $\mathbf{Q}$; $\mathbf{H}$ is the measurement matrix relating the state to the measurements; and $\mathbf{w}$ is the measurement noise vector, which also is zero-mean with a normal distribution and covariance, $\mathbf{R}$.

During the prediction step of the Kalman filter, the state estimate and its estimation error covariance matrix, $\mathbf{P}(k)$, are propagated forward through the dynamics model with the added process noise. This step enforces the temporal dependency that the stress class probabilities at the current time step have on the future time step. The prediction equations are:

$$\mathbf{x}(k|k-1) = \mathbf{F}\,\mathbf{x}(k-1|k-1), \tag{5.10}$$

$$\mathbf{P}(k|k-1) = \mathbf{F}\,\mathbf{P}(k-1|k-1)\,\mathbf{F}^{\top} + \mathbf{Q}(k-1) \tag{5.11}$$

where the notation $(k+1|k)$ indicates the next time step given the current time step. Next, during the update step, measurements are processed and updated estimates of the states and their covariance are corrected according to the measurements. The measurement update

equations are as follows:

$$\mathbf{x}(k|k) = \mathbf{x}(k|k-1) - \mathbf{K}(k)[\mathbf{H}(\mathbf{x}(k|k-1) - \mathbf{z}(k)] \tag{5.12}$$

$$\mathbf{P}(k|k) = \mathbf{P}(k|k-1) - \mathbf{K}(k)\mathbf{H}\mathbf{P}(k|k-1) \tag{5.13}$$

$$\mathbf{K}(k) = \mathbf{P}(k|k-1)\mathbf{H}^{\top}[\mathbf{H}\mathbf{P}(k|k-1)\mathbf{H}^{\top} + \mathbf{R}(k)]^{-1} \tag{5.14}$$

with $\mathbf{K}$ representing the Kalman gain. The prediction and update step are iterated to produce an estimate of the state, $\mathbf{x}$, and its associated estimation error covariance, $\mathbf{P}$, representing the uncertainty involved with the state estimate.

For our case, we abstract the multi-class classification problem as follows. The unknown state our filter is attempting to estimate is the probability of each class during each segment. Thus, $\mathbf{x}$ is a c dimensional vector of estimated class probabilities. Additionally, the predictions from each separate classifier are the measurements $\mathbf{z}$, which are processed sequentially per time step. This allows for s^* measurement updates per iteration where s^* is adaptively selected per sample by the gating model. We additionally provide some measurement thresholding during the filter updates that are detailed in Section 5.6.2. Finally, we arrive at our late fusion output using the Kalman filter-based method:

$$\hat{\mathbf{Y}}_{kf} = \arg\max_{c} \mathbf{x} \tag{5.15}$$

where $\mathbf{x}$ is the state vector from the Kalman filter. To validate our Kalman-filter based method, we benchmark its performance against commonly used voting mechanisms for late

fusion: *hard-voting* and *soft-voting* [183]. The method of hard-voting assigns the final class based on the class most commonly voted by each classifier, whereas soft-voting selects the class with the highest average value across all the classifiers. Our results comparing different late fusion approaches are presented in Figures 5.4, 5.5, 5.6, and 5.7 of Section 5.6.

5.6 Experimental Analysis

This section presents the experimental findings of SELF-CARE on a wearable health device stress detection dataset. First, we describe the dataset used for evaluation. Second, we explain the training and implementation of our models. Third, we describe our evaluation metrics and analyze experimental results.

5.6.1 Dataset

SELF-CARE is validated on the publicly available WESAD dataset [174]. This dataset was selected because it contains data from both wrist- and chest-worn wearable devices, which makes it an ideal dataset for understanding the noise context devices worn on different parts of the body. The dataset contains data for a total of 15 subjects, from both chest- (RespiBAN) and wrist- (Empatica E4) worn sensors. The chest sensors used in RespiBAN are ACC, ECG, RESP, EMG, EDA, TEMP. The wrist sensors from the Empatica E4 are ACC BVP, EDA, TEMP. The dataset has three types of classes related to emotional states: (i) baseline (neutral), (ii) amusement, and (iii) stress. For the 2-class problem, baseline and amusement are grouped together in the non-stress class.

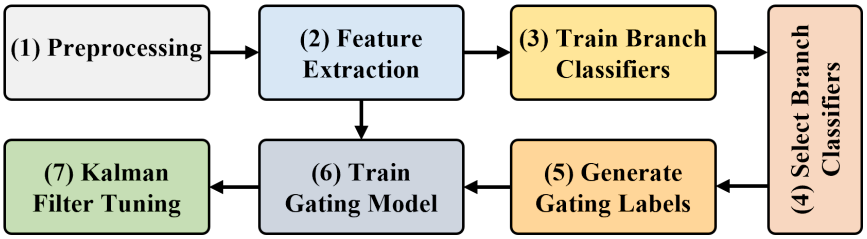


Figure 5.3: SELF-CARE training and implementation procedure.

5.6.2 SELF-CARE Training and Implementation

This section describes the training and implementation details for the SELF-CARE architecture, shown in Fig. 5.3.

Preprocessing Step

The preprocessing step involves raw data processing to filter out typical noises.

Wrist Modalities: The ACC data is passed through a Finite Impulse Response (FIR) filter with a length of 64 with a cut-off frequency of 0.4 Hz. Following the work in [110], the raw BVP signal is filtered by a Butterworth band-pass filter of order 3 with cutoff frequencies (f_1=0.7 Hz and f_2=3.7 Hz), which takes into account the heart rate at rest (≈40 BPM) or high heart rate due to exercise scenarios or tachycardia (≈220 BPM) [187]. The raw EDA signals are filtered using a Butterworth lowpass filter of order 6 with cut-off frequency of 1 Hz. Finally, we use a Savitzky-Golay filter (window size=11, order=3) to smooth the raw TEMP signals.

Chest Modalities: Because the chest data is collected at a very high sampling rate (700 Hz), the signals are first smoothed using a Savitzky-Golay filter. The ACC data is smoothed using a window size of 31 with an order of 5. The other signals (ECG, EMG, EDA, RESP, and TEMP) are smoothed using a window size of 11 and an order of 3. Similar to wrist BVP,

the ECG signal is further filtered by a Butterworth band-pass filter of order 3 with cutoff frequencies (f_1=0.7 Hz and f_2=3.7 Hz) that takes into account the heart rate at rest (≈40 BPM) or high heart rate due to exercise scenarios or tachycardia (≈220 BPM) [187]. The EDA signals are filtered using a Butterworth lowpass filter of order 2 with a cutoff frequency of 5 Hz. To extract some of the peak features (number of peaks, peak amplitude), the EMG signal is passed through a Butterworth lowpass filter of order 3 and a cutoff frequency of 0.5 Hz. We extract other EMG features from the smoothed EMG signal. The RESP signal is filtered by a Butterworth bandpass filter of order 3 with cutoff frequencies f_1=0.1 Hz and f_2=0.35 Hz.

The filtered signals from both the wrist and chest are segmented by a window of 60 seconds of data with a sliding length of 5 seconds following [175]. This process produces a total of 6458 segments for each signal across all subjects of the WESAD dataset.

Feature Extraction

We extract the same wrist and chest sensor features as used in [174], some of which include mean/standard deviations, correlations, slope, and dynamic ranges, peak and power frequencies, and absolute integrals. We note that this feature extraction is only performed across the sensors that are selected to run by the gate for a given input sample. Table 5.1 contains the list of extracted features. We refer readers to [174] for further details of extracted features per sensor.

Train Branch Classifiers

To train the individual branch classifiers within SELF-CARE, we train using different combinations of input sensor data.

Table 5.1: List of Extracted Features

Feature Symbol	Feature Names
ACC Features	
$\mu_{ACC,i}$, $\sigma_{ACC,i}$ $i \in \{x, y, z, 3D\}$	Mean and STD of each axis and summed over all axes
$\lvert \int_{ACC,i} \rvert$, $i \in \{x, y, z, 3D\}$	Absolute integral for each/all axes
$f^{peak}_{ACC,j}$, $j \in \{x, y, z\}$	Peak frequency of each axis
ECG/BVP Features	
μ_{HR}, σ_{HR}	Mean and STD of HR
μ_{HRV}, σ_{HRV}	Mean and STD of HRV
$NN50$, $pNN50$	Number and percentage of HRV intervals differing more than 50 ms
rms_{HRV}	Root mean square of the HRV
f^{x}_{HRV} $x \in ULF, LF, HF, UHF$	Energy in ultra-low, low, high, ultra-high frequency band of the HRV
$f^{LF/HF}_{HRV}$	Ratio of LF and HF component
$\sum^{f}_{x}$ $x \in ULF, LF, HF, UHF$	$\sum$ of the frequency components in ULF-HF
rel^{f}_{x}	Relative power of freq. components
LF_{norm}, HF_{norm}	Normalised LF and HF component
EMG Features	
μ_{EDA}, σ_{EDA}	Mean and STD of EMG
$range_{EDA}$	Dynamic range of EMG
$\lvert \int_{EMG} \rvert$	Absolute integral
$\tilde{\pi}_{EMG}$	Median of EMG
P^{10}_{EMG}, P^{90}_{EMG}	10^{th} and 90^{th} percentile
μ^{f}_{EMG}, $\tilde{f}_{EMG}$, f^{peak}_{EMG}	Mean, median, and peak frequency
PSD(f_{EMG})	Energy in seven bands
$\#^{peaks}_{EMG}$	Number of peaks
μ^{amp}_{EMG}, σ^{amp}_{EMG}	Mean and STD of peak amplitude
$\sum^{amp}_{EMG}$, $\sum^{amp}_{EMG}$	$\sum$ and norm. $\sum$ of peak amplitude
EDA Features	
μ_{EDA}, σ_{EDA}	Mean and STD of EDA
min_{EDA}, max_{EDA}	Min and max value of EDA
δ_{EDA}, $range_{EDA}$	Slope and dynamic range of EDA
μ_{SCL}, σ_{SCL}, σ_{SCR}	Mean and STD of SCL/SCR
$Corr_{SCL,t}$	Correlation between SCL and time
$\#\ SCR$	Number of SCR segments
$\sum^{amp}_{SCR}$, $\sum^{t}_{SCR}$	$\sum$ of SCR magnitudes and duration
$\int_{SCR}$	Area under SCR segments
RESP Features	
μ_x, σ_x $x \in I, E$	Mean and STD of inhalation (I) exhalation (E) duration
I/E	Inhalation/exhalation ratio
vol_{insp}, $range_{RESP}$	Volume and range of RESP
$rate_{RESP}$, $\sum_{RESP}$	Respiration rate and duration
TEMP Features	
μ_{TEMP}, σ_{TEMP}	Mean and STD of TEMP
min_{TEMP}, max_{TEMP}	Min and max of TEMP
δ_{TEMP}, $range_{TEMP}$	Slope and dynamic range of TEMP

Standard Deviation (STD), Skin Conductance Response (SCR), Skin Conductance Level (SCL), Heart Rate (HR), Heart Rate Variability (HRV)

For Wrist Modalities, we use five different early fusion combinations of wrist sensors as input branches: WB_1={BVP, EDA, TEMP}; WB_2={ACC, BVP, EDA}; WB_3={BVP, EDA}; WB_4={ACC, BVP}; WB_5={ACC, EDA} as shown in Tables 5.2 and 5.3. For chest

modalities, we tried forty-two different combinations of chest sensors as input branches as shown in Tables 5.4 and 5.5.

We evaluate each branch on five different machine learning classifiers—Decision Tree (DT), Random Forest (RF), AdaBoost (AB), Linear Discriminant Analysis (LDA), K-Nearest Neighbor (KNN). We selected these classifiers to ensure a fair comparison with the original WESAD work [174]. Following the work in [174], we use the same configurations for the classifiers. We use DT as the base estimator for the RF and AB ensemble classifiers, and use 100 base estimators for both RF and AB. In order to measure the splitting quality of the decision nodes, we used information gain and set the minimum number of samples to split a node to 20. For KNN, the K value is set to 9. All classifiers are trained using leave-one-subject-out (LOSO) validation.

Select Branch Classifiers

We select the branches with the least amount of training loss to be used. The training loss is calculated from the classification confidence of the trained classifiers on the training samples using the categorical cross-entropy, $CE = -\sum_{i}^{n_c} y_i \log \hat{y}_i$, where y is the one hot encoded true label of a sample, $\hat{y}$ is the corresponding classification output for that sample, and n_c is the number of classes. CE is then calculated for all the training samples across all rounds of LOSO validation.

Next, out of the 25 (5 branches x 5 classifiers per branch) possible branch classifiers for wrist modalities, RF classifiers for input branches WB_1, WB_2, and WB_3 are selected as the branch classifiers for both 3-class and 2-class classification. Similarly, for chest modalities, out of 210 (42 branches x 5 classifiers per branch) possible branches, AB classifiers for input branches CB_1, CB_{12}, CB_{14}, CB_{24}, and CB_{27} are selected for 3-class classification. And for 2-class classification, we select AB classifiers for input branches CB_5, CB_7, CB_9, CB_{13}, and CB_{20}

Table 5.2: Early Fusion Performance of Wrist Modalities in WESAD Dataset for 3-Class (Baseline vs. Stress vs. Amusement)

Modality Used	DT		RF		AB		LDA		KNN	
	MF1	Acc.	MF1	Acc.	MF1	Acc.	MF1	Acc.	MF1	Acc.
WB_1={BVP, EDA, TEMP}	56.23	62.32	**62.73**	**76.62**	63.78	75.78	52.62	61.79	58.3	69.04
WB_2={ACC, BVP, EDA}	58.46	48.27	**62.88**	**77.71**	62.39	76.63	60.23	69.63	58.9	68.55
WB_3={BVP, EDA}	55.14	59.02	**61.02**	**73.96**	60.67	72.54	56.55	69.8	65.73	53.44
WB_4={ACC, BVP}	51.54	60.66	56.86	71.38	57.83	71.96	58.67	68.36	55.51	67.05
WB_5={ACC, EDA}	47.98	54.5	52.97	70.15	56.47	71.31	57.71	68.6	58.75	64.87

Table 5.3: Early Fusion Performance of Wrist Modalities in WESAD Dataset for 2-Class (Stress vs. Non-stress)

Modality Used	DT		RF		AB		LDA		KNN	
	MF1	Acc.	MF1	Acc.	MF1	Acc.	MF1	Acc.	MF1	Acc.
WB_1={BVP, EDA, TEMP}	74.1	84.27	**84.66**	**89.01**	85.29	88.96	71.46	77.32	83.74	86.56
WB_2={ACC, BVP, EDA}	69.44	77.06	**85.08**	**88.76**	85.44	88.45	85.66	87.92	80.25	83.62
WB_3={BVP, EDA}	80.8	84.48	**86.37**	**89.33**	86.13	89.26	83.77	86.55	79.7	83.66
WB_4={ACC, BVP}	74.97	79.94	76.43	82.45	79.77	84.21	82.37	85.07	76.49	80.13
WB_5={ACC, EDA}	65.65	76.1	72.77	82.42	75.39	83.52	78.66	84.19	73.72	77.55

for use within our SELF-CARE methodology. These classifier selections are informed by the extensive experiments we performed across the classifiers variations, which we benchmark in Tables 5.2, 5.3, 5.4, and 5.5.

Generate Gating Labels

The objective of the gating model is to predict one or a subset of branch classifiers from the classifiers listed in Section 5.6.2 to be used in our SELF-CARE methodology. For each of the training samples, we generate gating labels representing the branch that has the least amount of training loss. These gating labels will be used to train the gating model. For each round of LOSO validation, gating labels are generated based only on the training data, and no test data is used to ensure the validity of our approach.

Train Gating Model

The gating model interprets the context of a sample by modeling the movement (for wrist-worn devices) or muscle contraction (for chest-worn devices) that occurred during that seg-

Table 5.4: Early Fusion Performance of Chest Modalities in WESAD Dataset for 3-Class (Baseline vs. Stress vs. Amusement)

Modality Used	DT		RF		AB		LDA		KNN	
	MF1	Acc.	MF1	Acc.	MF1	Acc.	MF1	Acc.	MF1	Acc.
CB_1={EC,RE,EM,ED,TE}	54.26	62.07	58.68	71.39	**65.63**	**76.53**	30.14	38.2	53.87	65.1
CB_2={AC,EC,RE,EM,TE}	49.41	57.01	53.55	69.1	60.18	71.88	55.71	72.0	43.15	51.5
CB_3={AC,RE,EM,TE}	45.8	55.14	53.46	68.58	57.11	69.14	54.93	66.07	44.79	55.49
CB_4={AC,EC,RE,EM}	48.29	55.55	50.37	63.36	54.52	65.55	51.8	62.8	43.04	51.56
CB_5={AC,EC,RE,ED}	44.33	53.75	52.32	69.15	55.02	73.09	44.05	54.93	43.73	52.43
CB_6={AC,EC,EM,TE}	48.96	56.55	53.98	69.58	58.98	70.88	55.31	71.41	42.33	49.93
CB_7={AC,EC,EM,ED}	51.22	61.09	57.0	72.64	59.54	73.35	45.79	56.21	46.82	56.17
CB_8={AC,RE,ED,TE}	42.79	51.74	54.6	71.06	52.34	67.32	22.66	24.62	40.91	48.78
CB_9={AC,EC,ED,TE}	41.94	51.1	54.75	72.57	57.36	75.99	38.59	48.37	45.53	54.2
CB_{10}={EC,RE,EM,TE}	47.17	52.8	58.11	70.71	61.8	71.92	54.51	68.6	51.17	59.94
CB_{11}={EC,RE,EM,ED}	51.14	59.24	57.7	68.9	60.47	70.68	50.34	60.93	51.54	62.99
CB_{12}={EC,EM,ED,TE}	53.95	61.41	57.95	71.12	**62.09**	**74.51**	31.52	39.52	53.31	63.83
CB_{13}={EC,RE,ED,TE}	51.75	60.09	55.02	72.85	57.73	73.45	31.98	38.97	57.14	70.26
CB_{14}={RE,EM,ED,TE}	48.93	54.44	60.87	71.41	**63.68**	**74.16**	31.59	37.16	51.69	64.38
CB_{15}={AC,ED,TE}	41.23	49.34	53.69	69.18	52.91	68.95	24.18	25.84	42.29	50.57
CB_{16}={AC,EM,ED}	48.81	58.07	54.59	69.32	54.7	69.25	35.84	45.83	44.99	54.87
CB_{17}={AC,RE,ED}	43.23	51.69	51.5	67.36	49.91	66.85	35.15	45.74	39.96	49.74
CB_{18}={AC,EC,RE}	40.4	50.19	48.55	61.61	50.11	65.04	51.39	61.91	39.85	48.3
CB_{19}={AC,RE,EM}	45.19	52.93	48.03	61.65	50.84	62.98	43.41	54.94	42.54	54.79
CB_{20}={AC,EC,ED}	44.66	53.48	53.36	69.41	53.78	72.28	43.15	53.74	42.98	50.43
CB_{21}={EC,RE,EM}	43.25	49.07	51.13	58.51	52.17	59.63	54.68	65.37	47.82	57.02
CB_{22}={EC,ED,TE}	52.6	62.66	55.06	72.59	57.63	72.9	33.23	39.41	56.58	68.1
CB_{23}={EC,RE,ED}	49.02	55.73	51.74	65.97	50.34	65.1	46.24	59.24	52.38	64.95
CB_{24}={EC,EM,ED}	52.58	60.19	57.89	68.58	**61.69**	**71.25**	49.41	59.77	50.23	60.75
CB_{25}={RE,EM,ED}	42.09	49.83	56.13	64.04	61.46	69.09	39.07	49.63	48.25	61.33
CB_{26}={RE,ED,TE}	45.95	54.85	56.76	74.1	54.56	71.05	22.48	23.51	50.27	64.98
CB_{27}={EM,ED,TE}	49.94	55.45	61.32	71.51	**64.72**	**74.36**	32.73	40.29	51.23	62.91
CB_{28}={AC,RE}	42.27	51.12	48.39	60.41	45.06	58.18	43.3	56.92	40.93	52.46
CB_{29}={AC,EM}	44.36	52.12	47.96	61.39	50.41	63.09	42.04	53.55	42.18	53.4
CB_{30}={AC,EC}	39.41	48.76	49.48	63.37	48.79	62.9	51.14	61.68	37.45	45.01
CB_{31}={AC,ED}	42.26	49.04	51.56	67.42	47.84	65.89	32.8	42.47	40.46	50.72
CB_{32}={AC,TE}	39.86	48.43	49.58	62.29	46.81	59.66	52.05	63.16	42.78	52.02
CB_{33}={EC,RE}	42.24	47.3	44.13	54.5	45.72	55.92	52.31	66.59	45.34	56.68
CB_{34}={EC,EM}	40.99	46.91	51.12	57.73	50.89	58.93	54.25	64.96	48.46	55.93
CB_{35}={EC,ED}	51.3	57.86	50.4	64.85	50.02	64.46	44.76	57.57	50.21	61.31
CB_{36}={EC,TE}	41.34	48.0	48.82	63.43	51.94	63.68	53.47	69.58	50.07	59.97
CB_{37}={ED,TE}	47.52	56.72	55.39	72.88	53.17	69.89	23.22	24.12	47.97	59.91
CB_{38}={RE,ED}	39.88	47.63	50.19	60.34	45.98	54.46	30.8	42.41	48.9	64.12
CB_{39}={RE,EM}	42.73	50.93	47.94	57.68	47.65	57.34	44.9	57.34	45.69	56.73
CB_{40}={RE,TE}	41.78	53.36	51.85	69.44	49.91	59.96	56.65	71.94	48.43	62.27
CB_{41}={EM,ED}	42.9	51.26	55.5	63.64	60.54	68.68	36.4	45.9	49.92	60.11
CB_{42}={EM,TE}	40.63	45.37	62.12	71.95	63.16	70.91	58.63	67.99	50.41	59.67

AC = ACC, ED = EDA, EC = ECG, EM = EMG, RE = RESP, TE = TEMP

ment. Therefore, we use the ACC (wrist) or EMG (chest) features as input data to train the gating model with the labels generated from the previous Section 5.6.2. We use a DT classifier as the gating model where the minimum number of samples to split a node is set to 20. The DT classifier is very lightweight and helps to minimize the overhead of SELF-CARE. Once the gating model is trained, the test subject data is used to test our architecture as shown in Fig. 5.2. For wrist-worn devices, the gating model outputs the probability of using

Table 5.5: Early Fusion Performance of Chest Modalities in WESAD Dataset for 2-Class (Stress vs. Non-stress)

Modality Used	DT		RF		AB		LDA		KNN	
	MF1	Acc.	MF1	Acc.	MF1	Acc.	MF1	Acc.	MF1	Acc.
CB_1={EC,RE,EM,ED,TE}	73.17	75.85	82.02	83.24	81.45	84.14	46.32	48.77	74.68	78.81
CB_2={AC,EC,RE,EM,TE}	67.25	73.11	80.12	83.86	77.07	82.75	77.53	79.98	63.81	69.48
CB_3={AC,RE,EM,TE}	68.85	76.61	78.15	83.44	73.16	81.56	74.03	77.17	61.97	70.88
CB_4={AC,EC,RE,EM}	66.93	72.16	70.57	78.06	72.98	80.57	75.66	79.83	64.12	69.86
CB_5={AC,EC,RE,ED}	70.39	75.81	82.41	84.21	**83.21**	**85.64**	68.46	75.02	74.6	77.75
CB_6={AC,EC,EM,TE}	66.06	72.22	80.13	83.88	75.77	82.36	77.13	79.64	62.97	68.0
CB_7={AC,EC,EM,ED}	72.45	77.72	83.49	85.64	**82.29**	**85.72**	69.6	75.2	72.05	75.86
CB_8={AC,RE,ED,TE}	68.71	73.26	81.03	84.16	72.19	79.34	29.71	32.23	66.42	74.28
CB_9={AC,EC,ED,TE}	65.13	70.79	84.47	86.12	**82.15**	**85.2**	59.5	61.72	75.02	78.4
CB_{10}={EC,RE,EM,TE}	52.55	54.77	76.87	80.08	76.02	80.39	76.89	79.17	68.32	73.94
CB_{11}={EC,RE,EM,ED}	74.89	77.51	81.23	82.93	82.25	84.79	70.15	75.69	73.76	77.89
CB_{12}={EC,EM,ED,TE}	72.35	75.25	82.05	83.39	79.64	82.7	47.71	50.14	73.03	77.14
CB_{13}={EC,RE,ED,TE}	73.62	75.58	79.2	80.24	**83.31**	**84.78**	49.66	51.9	79.22	82.04
CB_{14}={RE,EM,ED,TE}	70.82	73.11	81.8	84.15	77.41	81.44	47.13	50.71	67.67	76.19
CB_{15}={AC,ED,TE}	66.82	72.0	80.51	83.08	71.5	79.53	32.61	35.96	66.95	73.5
CB_{16}={AC,EM,ED}	69.7	76.11	78.93	83.67	74.92	82.07	54.66	61.85	64.89	71.65
CB_{17}={AC,RE,ED}	66.46	73.42	77.17	81.81	70.91	79.16	53.28	62.59	63.82	71.44
CB_{18}={AC,EC,RE}	62.03	69.17	72.22	78.51	75.78	81.16	74.68	79.53	65.87	69.49
CB_{19}={AC,RE,EM}	63.76	71.48	66.44	77.19	64.21	75.5	63.29	71.03	60.84	70.04
CB_{20}={AC,EC,ED}	69.46	74.81	84.11	85.62	**84.0**	**86.37**	66.96	73.53	73.19	76.18
CB_{21}={EC,RE,EM}	61.32	65.25	68.82	73.78	67.5	74.26	75.91	80.2	65.96	72.18
CB_{22}={EC,ED,TE}	73.09	75.06	78.33	79.4	81.01	82.4	52.84	55.01	77.92	80.7
CB_{23}={EC,RE,ED}	70.15	72.45	78.83	80.05	79.93	81.62	67.9	74.41	76.45	79.31
CB_{24}={EC,EM,ED}	74.75	77.26	80.69	82.39	82.03	84.58	69.03	74.81	71.75	75.88
CB_{25}={RE,EM,ED}	60.6	65.44	74.27	78.99	73.3	79.62	54.6	62.95	67.22	76.13
CB_{26}={RE,ED,TE}	69.55	71.23	77.57	78.93	77.44	79.83	35.86	39.07	70.25	75.78
CB_{27}={EM,ED,TE}	70.92	73.2	80.65	83.39	77.88	82.14	50.88	54.17	66.11	73.9
CB_{28}={AC,RE}	64.32	70.8	68.92	77.16	66.29	75.53	62.96	75.13	65.24	72.01
CB_{29}={AC,EM}	61.76	69.37	66.36	76.85	64.19	75.86	60.91	69.19	60.51	68.4
CB_{30}={AC,EC}	63.41	70.31	71.71	78.03	75.3	80.99	74.32	79.37	62.87	66.02
CB_{31}={AC,ED}	65.63	73.35	77.51	81.63	69.97	78.57	50.03	59.14	64.28	70.83
CB_{32}={AC,TE}	65.69	72.52	76.55	82.0	68.5	78.26	73.3	76.77	67.49	74.41
CB_{33}={EC,RE}	65.21	69.46	72.23	77.74	73.98	78.63	77.57	82.62	68.9	72.35
CB_{34}={EC,EM}	59.41	64.16	68.24	73.16	66.69	74.49	75.07	79.67	65.89	71.05
CB_{35}={EC,ED}	73.98	75.83	79.27	80.54	79.02	80.46	66.44	73.23	74.62	77.1
CB_{36}={EC,TE}	60.01	62.86	74.62	77.53	76.0	79.02	77.6	79.43	69.8	72.64
CB_{37}={ED,TE}	67.64	69.64	76.97	78.79	73.82	76.4	42.8	45.87	68.43	72.55
CB_{38}={RE,ED}	56.46	59.02	74.86	77.79	66.92	71.09	48.47	59.08	71.71	78.38
CB_{39}={RE,EM}	57.13	66.44	53.69	69.12	54.17	69.16	61.04	70.26	57.0	69.98
CB_{40}={RE,TE}	54.42	56.14	73.9	77.39	73.96	77.51	73.24	76.48	70.44	74.84
CB_{41}={EM,ED}	61.9	67.19	76.18	80.4	72.54	79.26	51.34	59.27	66.06	74.2
CB_{42}={EM,TE}	56.15	60.49	77.06	81.6	72.14	78.9	73.38	76.76	62.5	71.6

AC = ACC, ED = EDA, EC = ECG, EM = EMG, RE = RESP, TE = TEMP

the three final branch classifiers based on the test subject's ACC features. Similarly, for chest-worn devices, EMG features are used by the gating model to determine the probability of using the five final branch classifiers as mentioned in Section 5.6.2. One, two, or all of the final classifiers may be selected for final classification depending on the value of δ, as discussed earlier in Section 5.5.2. For our 3-class (2-class) classification using wrist-worn devices, we set $\delta = 0.40$ ($\delta = 0.10$). And for the chest-worn devices, we set $\delta = 0.20$ for

3-class and $\delta = 0.15$ for 2-class classification. The model extracts additional features based on the required input of the selected branch classifiers, and applies a late fusion method to the classification output of the selected branches to generate the final result.

Kalman Filter Tuning

The Kalman filter-based method is the only late fusion method in our implementation that requires tuning. As described in Section 5.5.4, Kalman filters require an initial state ($\mathbf{x}_0$), state covariance ($\mathbf{P}_0$) and process noise and measurement noise vectors, $\mathbf{v}$ and $\mathbf{w}$, respectively. For the 3-class (2-class) classification using wrist-worn devices, we initialize $x_0 = [0.8, 0.1, 0.1]^\top$ ($x_0 = [0.8, 0.2]^\top$). Similarly, for the 3-class and 2-class classification using chest-worn devices, x_0 is initialized to $[0.93, 0.21, 0.01]^\top$ $[1.0, 0.55]^\top$. For 3-class (2-class) classification, we initialize $P_0 = 0.01 \cdot \mathbf{I}_{3x3}$ ($P_0 = 0.01 \cdot \mathbf{I}_{2x2}$) for both wrist-worn and chest-worn devices. The state transition matrix $\mathbf{F}$ and measurement matrix $\mathbf{H}$ are identity matrices for the respective problems. The $\mathbf{Q}$ for both problems is modeled as a discrete time white process noise with variance set at 5e-4. The measurement noise is modeled as a function of each measurement to allow the filter to adjust the confidence of the measurements according to each reported class probability: $\mathbf{R} = ((\mathbf{1} - \mathbf{z}) \cdot 2 \cdot \mathbf{I}_{3x3})^2$ ($\mathbf{R} = ((\mathbf{1} - \mathbf{z})/2 \cdot \mathbf{I}_{2x2})^2$). Lastly, a tunable threshold technique was used to process the measurements which involved (i) an ϵ parameter to select measurements which had a maximum predicted probability above the threshold and (ii) a γ factor to scale the measurements to account for the imbalanced class distribution in the dataset. This thresholding process allows the filter to weigh each measurement it receives with a different degree of noise while also attempting to resolve issues that arise from imbalanced datasets. For the 3-class (2-class) classification using wrist-worn devices, we set $\epsilon = 0.4$ ($\epsilon = 0.7$) and $\gamma = [0.278, 1, 1]^\top$ ($\gamma = [0.667, 1.1]^\top$). For the 3-class (2-class) classification using chest-worn devices, we set $\epsilon = 0.5$ ($\epsilon = 0.5$) and $\gamma = [1.35, 1.5, 1.6]^\top$ ($\gamma = [0.915, 0.995]^\top$).

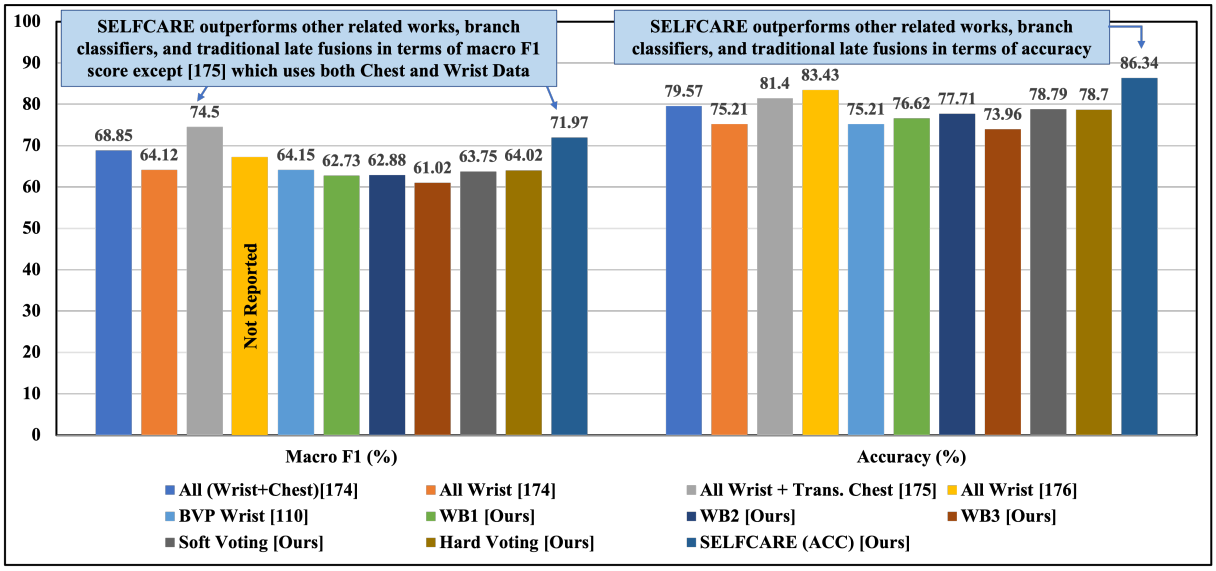


Figure 5.4: Overall Performance Comparison of Related Works using LOSO Validation on Wrist Data 3-Class

5.6.3 Evaluation Metrics

As stated previously, the WESAD dataset is highly imbalanced in terms of the number of segments per class. For this reason, F1 score is also used along with accuracy to measure the classification performance. To ensure a fair comparison with other works, we use the macro F1 score. The metrics used for evaluation are given below:

$$Accuracy = (TP + TN)/(TP + FP + TN + FN) \tag{5.16}$$

$$P = TP/(TP + FP),\ R = TP/(TP + FN) \tag{5.17}$$

$$Macro\ F_1 = \frac{1}{n_c}\sum_{i}^{n_c} 2 * \frac{P_i.R_i}{P_i + R_i} \tag{5.18}$$

where TP, TN, FP, FN represents True Positives, True Negatives, False Positives, and False Negatives, respectively; and P and R represent Precision and Recall, respectively. The classes are indexed by i, and n_c is the number of output classes.

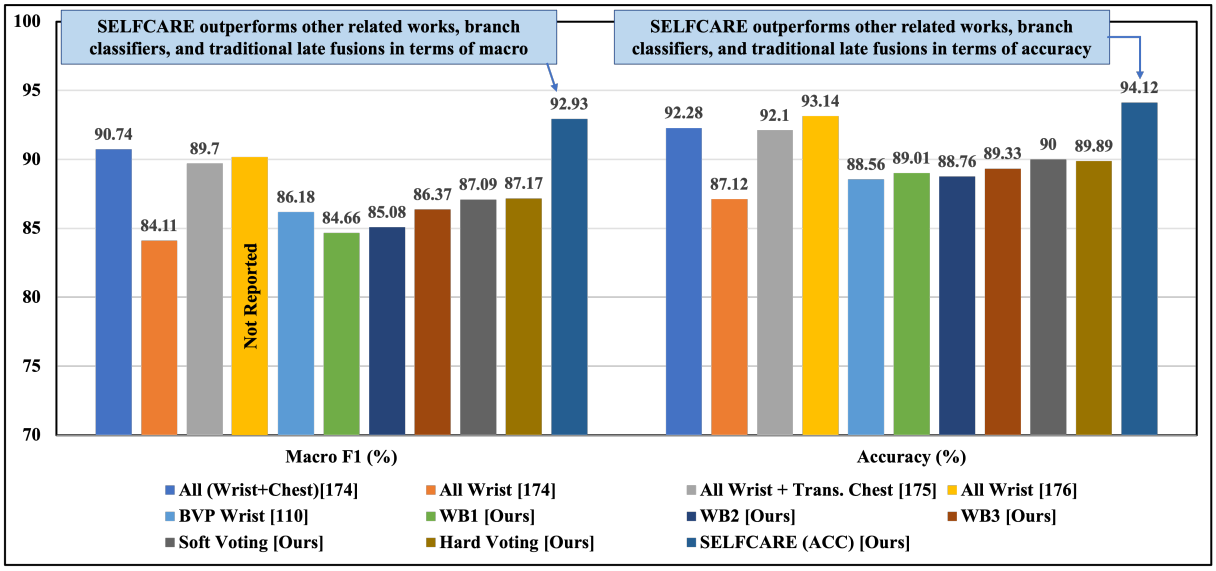


Figure 5.5: Overall Performance Comparison of Related Works using LOSO Validation on Wrist Data 2-Class

5.6.4 Experimental Results

This section presents the performance of SELF-CARE for stress detection in 3-class and 2-class classification using wrist and chest modalities.

Performance Evaluation of Wrist Modalities

Tables 5.2 and 5.3 show the performance analysis of different classifiers for various input branches for the 3-class and 2-class problems, with each branch representing different combinations of input sensors. The RF classifier for branches WB_1, WB_2, and WB_3 shows better or competitive performance compared to the other classifiers for both 3-class and 2-class. The RF classifiers also achieved minimum training loss for these input branches during training, leading to our selection of these three branches with the RF classifier for our approach.

As shown in Fig. 5.4, for 3-class classification, the SELF-CARE method outperforms other related works [110, 174, 176], the branch classifiers, and the traditional late fusion methods

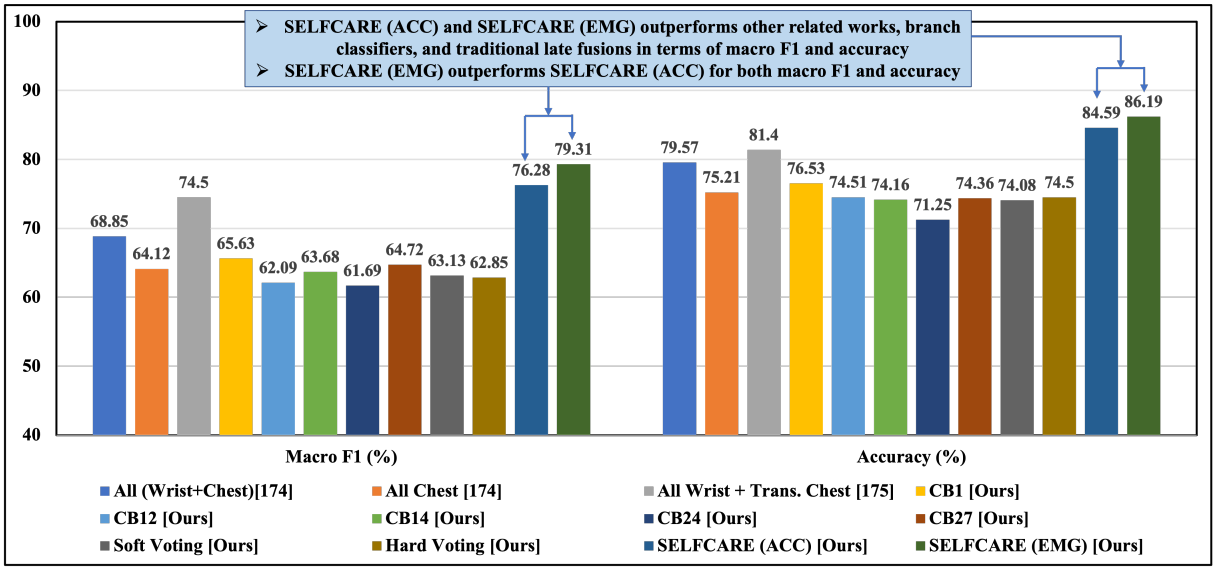


Figure 5.6: Overall Performance Comparison of Related Works using LOSO Validation on Chest Data 3-Class

in terms of both accuracy and macro F1 score achieving a performance of 86.34% and 71.97%, respectively. Compared to [175], SELF-CARE achieves better accuracy—though [175] achieves a better macro F1 score, as this work uses both wrist and chest sensors for stress classification. For 2-class classification, the SELF-CARE method achieves an accuracy of 94.12% and macro F1 score of 92.93%, outperforming the related works [110, 174–176], the branch classifiers, and the traditional late fusion methods in terms of both accuracy and macro F1 score (as shown in Fig. 5.5). For the three selected branch classifiers, we apply soft- and hard-voting methods, showing performance improvements compared to the individual branch classifiers for both 3-class and 2-class classifications. SELF-CARE also uses Kalman filter-based late fusion to further improve the performance for 3-class and 2-class classification compared to these traditional late fusion methods.

Performance Evaluation of Chest Modalities

Tables 5.4 and 5.5 show the performance analysis of different classifiers for various input branches for the 3-class and 2-class problems, with each branch representing different combi-

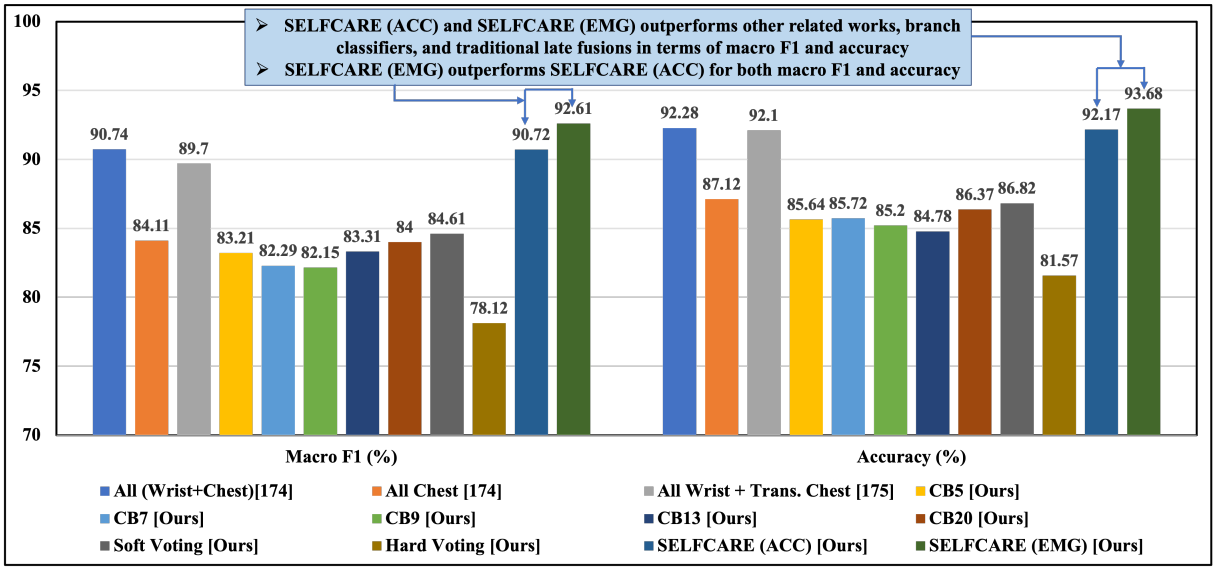


Figure 5.7: Overall Performance Comparison of Related Works using LOSO Validation on Chest Data 2-Class

nations of input sensors. The AB classifier for branches CB_1, CB_{12}, CB_{14}, CB_{24}, and CB_{27} shows better or competitive performance compared to the other classifiers 3-class classification. Similarly, for 2-class classification, the branches CB_5, CB_7, CB_9, CB_{13}, and CB_{20} showed better performance that other classifiers. The AB classifiers also achieved minimum training loss for these input branches during training, which led to the selection of five branches for the SELF-CARE framework. The soft- and hard-voting methods applied to the five selected branch classifiers do not show performance improvements compared to the individual branch classifiers for both 3-class and 2-class classifications. However, incorporating Kalman filter-based late fusion significantly improves the performance for 3-class and 2-class classification compared to these traditional late fusion methods.

As shown in Fig. 5.6 and 5.7, for both 3-class and 2-class classification, the SELF-CARE method, using either muscle contraction (EMG) or motion (ACC) for context understanding, outperforms other related works [110, 174–176], the branch classifiers, and the traditional late fusion methods in terms of both accuracy and macro F1 score. This study also demonstrates that even with motion-based context understanding, SELF-CARE outperforms other works.

However, the model's performance improves by 2-3% while using muscle contraction for context understanding compared to motion. This illustrates that the impact of movement on other sensors depends on the location of wearable devices. Therefore, movement is not always the best choice for contextual understanding as we observe the results while using chest modalities for stress detection.

5.7 Limitations and Future Directions

One of the main goals of this chapter is to explore how the context of noise varies depending on the location of wearable devices. For this reason, we modeled the noise context of sensor modalities from stand-alone devices, choosing not to combine the wrist and chest sensor modalities. However, future research could explore this issue further. Understanding the relation between the noise context of multiple wearable devices from physically different locations and fusing cross modal sensors based on that relation may produce interesting scientific findings that can be leveraged for methods of affective computing. Modeling the noise context of wearable health sensors can lead to further levels of human emotion understanding as information from the health sensors becomes increasingly useful when interpreted on a contextual basis.

Further, SELF-CARE is limited by the manual design of sensor fusion branch configurations. Though domain knowledge is required to determine which sensor data to fuse together, future works could explore using machine learning to make these determinations instead. Additionally, the energy efficiency of wearable health devices is an important constraint that could be examined in future works. SELF-CARE implements a configurable parameter for balancing computation and performance, but future works could examine the efficiency trade-offs between chest and wrist sensing modalities. This chapter did not focus on deep learning models, but SELF-CARE's modular design allows for the implementation of any

learning-based classifier, including deep learning branches. Further, SELF-CARE could be applied more broadly in the domain of affective computing to include additional tasks beyond stress detection and emotion recognition. Moreover, it can also be applied to wearable healthcare applications like human activity recognition [44, 53, 88], myocardial infarction detection [40, 52, 113] and various others [188–192] etc., that involves data from multiple wearable sensors. Lastly, SELF-CARE's use of a specialized set of ensemble classifiers has broad applicability to IoT sensing, including the domains of sensor networks [193], and transportation [194–197] and others [198–201].

5.8 Summary

In this chapter, we propose SELF-CARE, a generalized selective sensor fusion method for stress detection that utilizes the noise context in the chest- and wrist-worn devices to dynamically adjust the sensor fusion performed to maximize classification performance. SELF-CARE determines the noise context using muscle contractions (EMG) or motion (ACC) of a subject, and performs an intelligent gating mechanism to select which sensor fusion schema to use depending on the location of the sensor.We also show that, while determining the noise context based on motion works best for wrist-based wearable devices, it is not the best for chest-based wearable devices. Through experimental evaluation, we conclude that EMG is better than ACC in understanding the noise context of chest-based wearable devices. To the best of our knowledge, SELF-CARE achieves state-of-the-art performance on the WESAD dataset for both chest and wrist-based sensors among methods that use LOSO validation. Using wrist-based sensors our methodology achieves 86.34% (3-class) and 94.12% (2-class) classification accuracy while outperforming current state-of-the-art works. Similarly, for chest-based wearable sensors, our methodology outperforms existing models with 86.19% (3-class) and 93.68% (2-class) classification accuracy.

Chapter 6

Conclusion

This thesis presents some efficient methodologies for designing digital health solutions using wearable devices for various healthcare applications like – Myocardial Infarction Detection, Human Activity Recognition, Human Eating Activity Recognition, and Stress Detection. Particularly, this thesis contributes by addressing the following research challenges - 1) Designing energy and memory-efficient edge computing solutions while maintaining performance; 2) Generating ground truth labels of collected data for online learning without much user involvement; and 3) Degradation of model's performance due to the fusion of noisy data from multiple heterogeneous sensors.

Chapter 2 addresses the first challenge by proposing an energy-efficient methodology for real-time MI detection on wearable devices using a Convolutional Neural Network (CNN). It proposes a Template Matching based Early Exit (TMEX) CNN architecture that further increases the energy efficiency compared to baseline architecture while maintaining similar performance. On PTB dataset, our baseline and TMEX architecture achieve 99.33% and 99.24% accuracy, whereas on PTB-XL dataset they achieve 84.36% and 84.24% accuracy, respectively. Evaluation on real hardware shows that our baseline architecture achieves from

0.6x to **53x** more energy efficiency while outperforming state-of-the-art works on wearable devices. Moreover, our TMEX architecture further achieves 8.12% (PTB) and 6.36% (PTB-XL) more energy efficiency compared to the baseline architecture while maintaining similar performance. To the best of our knowledge, the baseline and TMEX architecture of our methodology achieve the best performance on wearable devices while being energy-efficient with a RAM footprint of only 20 KB.

Chapter 3 also addresses the first challenge by proposing an Adaptive CNN for HAR (AHAR) to develop an energy-efficient solution for low-power edge devices. AHAR uses a novel adaptive architecture that decides which portion of the baseline architecture to be used during the inference phase based on the simple statistical features of the activity segments. Our proposed methodology is validated for classifying locomotion activities from Opportunity and w-HAR datasets. Compared to the fog/cloud computing approaches that use the Opportunity dataset, both our baseline and adaptive architecture shows a comparable weighted F1 score of 91.79%, 91.57% respectively. For the w-HAR dataset, both our baseline and adaptive architecture outperforms the state-of-art-work with a weighted F1 score of 97.55% and 97.64% respectively. Evaluation on real hardware shows that our baseline architecture is significantly energy-efficient (422.38x less) and memory-efficient (14.29x less) compared to the works on the Opportunity dataset. For the w-HAR dataset, our baseline architecture requires 2.04x less energy and 2.18x less memory compared to the state-of-the-art work. Moreover, experimental results show that our adaptive architecture is 12.32% (Opportunity) and 11.14% (w-HAR) energy-efficient than our baseline while providing similar (Opportunity) or better (w-HAR) performance with no significant memory overhead. To the best of our knowledge, we are the first to propose such adaptive CNN architecture for HAR in wearable devices that provides energy efficiency while maintaining performance.

Chapter 4 explores the second challenge addressed in the thesis. This chapter proposes a Human Eating Activity Recognition (HEAR) methodology which uses an online update

phase to keep up with the changes of eating habits through online learning. In this regard, we designed an algorithm that creates approximate true labels for the new eating data. We have also designed a wearable neckband to capture eating activity data in a lab environment. Through detailed experimental evaluation of our methodology, we show that an online learned neural network classifier outperforms state-of-the-art offline trained classifiers while also being more energy-efficient.

Finally, chapter 5 addresses the third challenge of maintaining model's performance while fusing noisy data from multiple heterogeneous sensors. In this chapter, we propose SELF-CARE, a generalized selective sensor fusion method for stress detection that utilizes the noise context in the chest- and wrist-worn devices to dynamically adjust the sensor fusion performed to maximize classification performance. SELF-CARE determines the noise context using muscle contractions (EMG) or motion (ACC) of a subject, and performs an intelligent gating mechanism to select which sensor fusion schema to use depending on the location of the sensor.We also show that, while determining the noise context based on motion works best for wrist-based wearable devices, it is not the best for chest-based wearable devices. Through experimental evaluation, we conclude that EMG is better than ACC in understanding the noise context of chest-based wearable devices. To the best of our knowledge, SELF-CARE achieves state-of-the-art performance on the WESAD dataset for both chest and wrist-based sensors among methods that use LOSO validation. Using wrist-based sensors our methodology achieves 86.34% (3-class) and 94.12% (2-class) classification accuracy while outperforming current state-of-the-art works. Similarly, for chest-based wearable sensors, our methodology outperforms existing models with 86.19% (3-class) and 93.68% (2-class) classification accuracy.

Overall, this thesis uses one of the aforementioned applications as use cases to address one of the challenges mentioned above. However, novel methods and algorithms introduced in this thesis may also be applicable to other various wearable healthcare applications that are not

used in this thesis. Other researchers or even engineers interested in designing digital health solutions may find the content of this thesis useful for their novel applications. Moreover, the findings of this thesis may also attract researchers from other domains such as transportation systems, environment monitoring, smart city, and agriculture to name a few that involves the integration of different sensors through the Internet of Things.

Appendix A

Secondary Thesis Contributions

Apart from the key thesis contributions, the findings in this thesis have also contributed to several other research works which are summarized in the following sections. Besides, while working on this thesis, the author also contributed to several other research works [202–207].

A.1 Real-time Myocardial Infarction Detection on Wearable Devices

In this work [52], we propose an MI detection methodology using Binary Convolutional Neural Network (BCNN) that is fast, energy-efficient, and outperforms some of the state-of-the-art works on wearable devices. We validate the performance of our methodology on the well known PTB diagnostic ECG database from PhysioNet. Evaluation on real hardware shows that our BCNN is faster and achieves up to 12x energy efficiency compared to the state-of-the-art work. The main contributions of this work are as follows:

1. Real-time MI methodology, using a Binary Convolutional Neural Network (BCNN),

that is faster, achieves up to **12x** energy-efficiency and provides better performance compared to state-of-the-art work on wearable devices.

2. Validating the performance of our approach on well known PTB diagnostic ECG database (PTBDB) [56] from PhysioNet [62].

3. Validating the energy-efficiency of our BCNN on real hardware.

A.1.1 Methodology

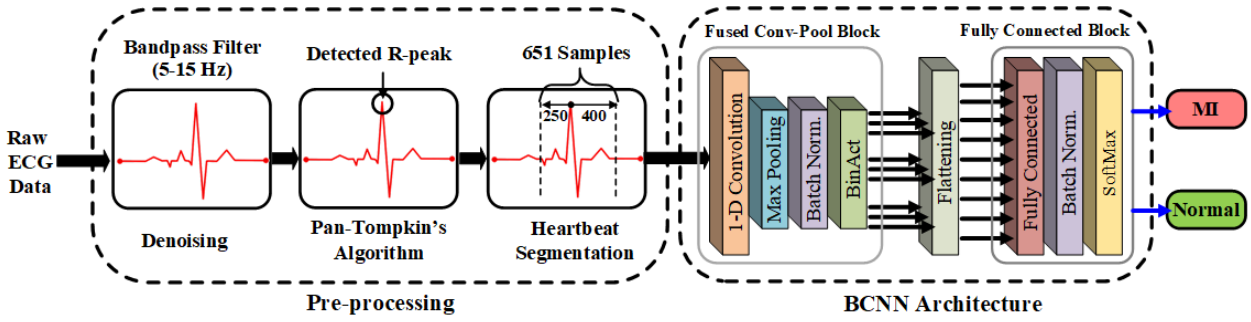


Figure A.1: Our MI Detection Methodology

Preprocessing

As shown in Figure A.15, the preprocessing step starts with the denoising of raw ECG data using a bandpass filter with cut-off frequencies f_1=5 and f_2=15 Hz. Then the Pan-Tompkin's algorithm [82] is used to detect the R-peaks from the filtered ECG data. Similar to other works [48–51], we take 250 samples before and 400 samples after the R-peak. Thus, each segment consists of 651 samples representing a heartbeat.

1-D BCNN

We design our BCNN with the minimum number of layers possible to maintain the low-power and low-memory constraint while maintaining acceptable performance for wearable devices.

Rationale of memory efficiency: Despite the enormous benefits of CNNs, their higher memory requirement makes them unsuitable for embedded/wearable device applications. To reduce the memory requirements of CNNs, we use the weight binarization technique from the Binarized Neural Network (BNN)[208] where all the filter weights in a layer are represented as -1 or 1 instead of 32-bit floating-point values. Thus the binary representation of weights enables 32 times memory efficiency compared to floating-point CNNs. Besides, we also use the binary activation function that clamps all negatives inputs to -1 and all positive to 1. Although the weights in CNNs are represented using binary values, the temporaries generated in between the CNN layers are still represented in floating-point values which requires a lot of memory.

To reduce the memory overhead of the temporaries, we reorganize the computation order of CNN layers following the technique of embedded Binarized Neural Networks (eBNNs) [209]. eBNN reorganizes the computation order of standard BNNs. In standard BNNs, the results of the convolution layer are stored (in floating-point values) before passing through batch normalization, binary activation, and pooling as shown in Figure 2(a). Whereas, in eBNN the result of the convolution is not stored in memory but is directly sent to pooling followed by batch normalization and binary activation as shown in Figure 2(b). Further details can be found in [209]. This binary representation of weights and temporaries not only makes our BCNN memory efficient but also energy-efficient. Because binary representation allows for faster execution and reduces computational complexity, it uses less energy.

Architecture: As shown in Figure A.15, we use one fused convolution-pool block and one fully connected block. The fused convolution-pool block consists of one 1-D convolution layer

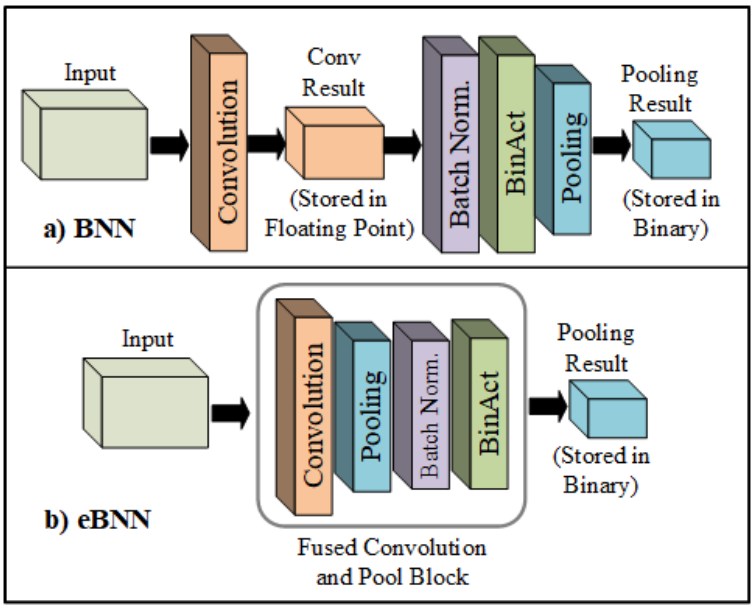


Figure A.2: Memory Efficiency due to Fused Convolution and Pool Block

(kernel=100, stride=2), one max-pooling layer (kernel=2, stride=2), one batch normalization layer, one binary activation layer. We use 3 feature maps for the convolution operation. The fully connected block consists of one fully connected layer with 2 neurons (for 2 output class), one batch normalization layer and finally the softmax activation layer. To train the BCNN we use *Adam* optimizer with a learning rate of .007. We use softmax cross-entropy as the loss function for the optimizer.

A.1.2 Experimental Setup and Evaluation

Wearable Platform

Our work is designed for low-power, low-memory wearable devices like SmartCardia INYU [85]. The device is equipped with an ultra-low-power 32-bit microcontroller STM32L151 containing an ARM Cortex–M3 with a maximum clock rate of 32 MHz. It has a 48 KB RAM, 384 KB Flash, and a standard 710 mAh battery. The device captures ECG signals using a single lead ECG sensor [210]. Our solution applies to any wearable device having

the above or similar specifications.

Performance Evaluation

To validate the performance of our BCNN, we use data from 200 subjects (52 Normal, 148 MI) from PTDDB [56]. We obtained a total of 50371 (44214 MI, 6157 Normal) heartbeat segments after preprocessing. Since the number of MI segments is approximately 7 times more than the number of normal segments, the model achieves an accuracy of 88% even when it predicts all input data to be from MI patients. Thus sensitivity and specificity are important metrics to judge model performance. To ensure proper training, we split the MI segments into 7 parts, each consisting of around 6316 MI segments (close to the total number of normal segments). One part of these MI segments is combined with all the normal segments and a 10-fold cross-validation is performed on the combined data. In each fold, 90% of the combined data is used for training and validation for 100 epochs. The remaining 10% is used for testing. The performance of the model is averaged across the 10 folds. This entire process is repeated for all the MI segment parts. The overall performance of the model is the average of all the 10-fold cross-validations. Our methodology achieves an accuracy of **90.29%**, the sensitivity of **90.41%**, and specificity of **90.16%** as shown in Table A.1. Our methodology outperforms the other state-of-the-art works [49, 50] which are intended for wearable devices in all three metrics. We significantly improve on the sensitivity and specificity compared to [50]. Although [49] is a close competitor for accuracy, it does not report sensitivity and specificity. As noted above, it is easy for models to achieve high accuracy on this data but it is harder to achieve high performance on all the metrics. As expected, the work [48, 51] achieves a very high performance as they are designed for clinical set up not for wearable devices.

Table A.1: Performance comparison of related Works

Related Works	PTBDB Data			Performance (%)		
	Normal	MI	Lead	Acc.	Sen.	Spe.
[48]	52	148	11	98.80	99.45	96.27
[51]	52	148	2	95.22	95.49	94.19
[49]	52	52	11	90	–	–
[50]	52	52	11	–	81.02	79.63
Our Work	**52**	**148**	**11**	**90.29**	**90.41**	**90.16**

Energy Consumption Evaluation

We evaluate the energy consumption of our BCNN using an EFM32 Leopard Gecko (EFM32LG-STK3600) microcontroller which has similar specifications as the SmartCardia device. Energy profiling is done using the simplicity studio software that comes with the EFM32 microcontrollers. As BCNN automatically extracts features and classifies them, we compare the energy consumption of our BCNN against the feature extraction and classification steps of other wearable device solutions [49, 50]. [49] uses a 2-level SVM classifier where the first level uses 10 features and the second level uses 47 features. [50] uses a 5-level Random Forest classifier where the first level uses only 5 features and the final level uses 72 features. We compare using the first level classifier of both the works as their first level classifiers are the most energy-efficient ones. Table A.2 shows the energy consumption of all the wearable device solutions to classify one heartbeat segment. Our BCNN consumes **536.80 μJ** whereas [49] consumes **4641.62 μJ** and [50] consumes **6478.64 μJ**. Thus, our BCNN achieves **8x** and **12x** energy-efficiency compared to [49] and [50] respectively. Moreover, our BCNN execute with only 3.3 KB of RAM and takes only 7.05 KB of flash thus leaving enough space for other application to run parallelly on wearable devices.

Table A.2: Energy Consumption Analysis on Real Hardware

Related Works	**Exe. Time (ms)**	**Avg. Curr. (mA)**	**Avg. Power (mW)**	**Energy (μJ)**
[49]	333.93	4.24	13.90	4641.62
[50]	465.42	4.25	13.92	6478.64
Our Work	**37.83**	**4.33**	**14.19**	**536.80**

A.1.3 Summary

In this work, we propose a methodology for real-time MI detection on wearable devices using Binary Convolutional Neural Network (BCNN). Evaluation on real hardware shows that our BCNN is faster and achieves up to **12x** energy efficiency while providing better performance compared to the state-of-the-art work.

A.2 Energy-Aware Design Methodology for Myocardial Infarction Detection on Wearable Devices

In this work [113], we propose a methodology to incorporate Neural Architecture Search (NAS) [211] to co-optimize the design of BCNNs for MI detection with regard to accuracy and energy using Multi-Objective Bayesian Optimization (MOBO). Figure A.3 shows the generalized design flow using our methodology where MOBO performs a systematic design space exploration of a BCNN inspired search space to sample the most efficient models satisfying both objectives. Each sampled model is trained to estimate its accuracy before being deployed on the target device to retrieve the related energy measurements. The Bayesian models are updated each iteration with new data in order to improve the search strategy. Finally, our methodology renders a set of Pareto optimal neural architectures that represent the most suitable models for deployment on the target wearable devices. The main contributions of this work are summarized as follows:

- A methodology is proposed to automate the design of BCNNs for MI detection on wearable devices through co-optimizing both accuracy and energy using real-hardware target device measurements.
- To the best of our knowledge, we are the first to propose a NAS based design methodology working with time-series ECG signals.
- The performance of our methodology is validated using PTB diagnostic ECG database [56] from PhysioNet [62].
- In comparison with the state-of-the-art works for wearable devices, one of our explored models achieves the highest accuracy of **91.22%** while others achieve up to **8.26**× more energy efficiency.

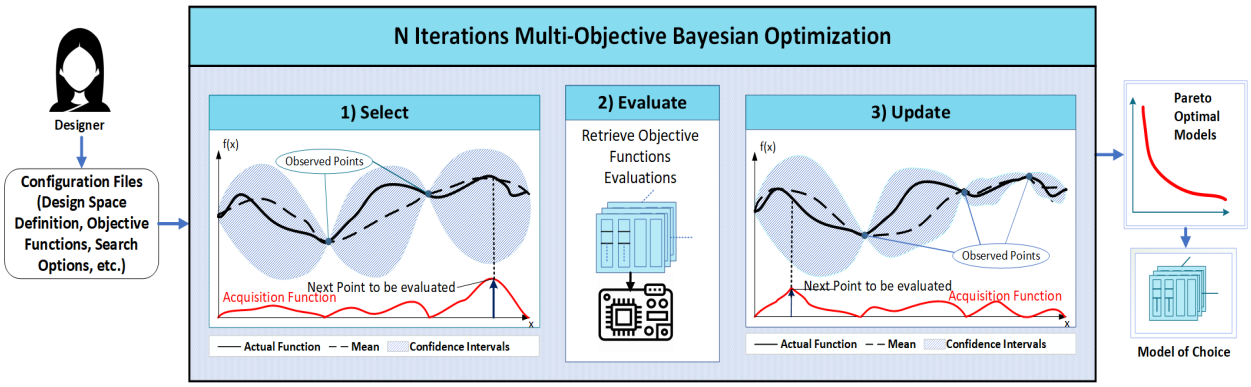


Figure A.3: The design flow process using MOBO. The actual function is unknown in reality. Instead, a Gaussian Process (GP) model is constructed for each objective function and updated each iteration based on the information collected so far.

A.2.1 Methodology

Overview

The multi-objective optimization problem can be formulated as -

$$min_{x \in X}(error(x), energy(x)) \tag{A.1}$$

Where the goal is to find a network architecture parameterized by $\boldsymbol{x}$ from the search space $\boldsymbol{X}$ that minimizes two objective functions: MI detection error and energy consumption on the target device. As shown in Figure A.11, our methodology does not assume direct closed-form models for both functions with respect to the neural architectural parameters. Instead, the problem is treated as a black box optimization one. This requires that for each sampled architecture, the accuracy loss and energy consumption should be evaluated to understand better how they relate to the architectural search parameters $\boldsymbol{x}$. While the classification loss is estimated computationally, energy is obtained through measuring power and execution time directly from the target device. However, as the two objectives are conflicting in nature, improving on one objective will negatively impact the other. Therefore, the outcome of this problem would have to be a set of Pareto optimal architectures $\boldsymbol{X}^{\star}$ which dominate

all other explored architectures but not each other. Formally in a minimization context, a point $\boldsymbol{x}$ is said to dominate $\boldsymbol{x'}$ if for every objective function $\boldsymbol{f_k}$, $\boldsymbol{f_k(x) \leq f_k(x')\forall k}$ and at least one inequality is strict.

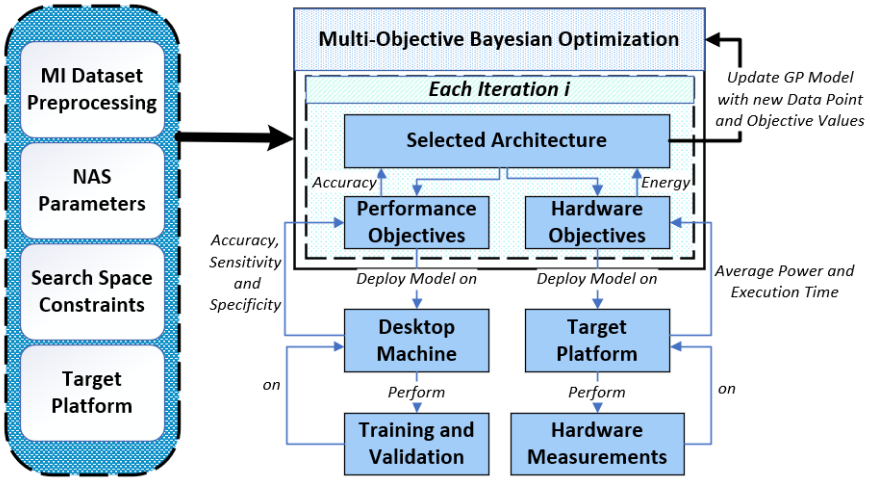


Figure A.4: Our proposed energy-aware design methodology of neural architectures for MI detection on wearable devices.

To solve this problem, we exploit MOBO [212] for our black-box optimization problem. Bayesian optimization methods provide efficient design space exploration in order to sample the most promising candidates that meet the minimization requirements of the objective functions. This is extremely useful when the search space is large and when evaluating an objective function is costly (like computing the loss function in our case). In this problem, once an architecture is sampled from the search space, the objective functions are evaluated to determine whether this candidate architecture should belong to the optimal set or not. Also, with each evaluation, the search strategy is updated to find better architectures in the following iterations.

Binary Convolutional Neural Network

Our search space is inspired by the BCNN architecture proposed in [52]. Their aim was to design an efficient CNN that can fit into wearable devices with limited memory while

conserving energy resources. To achieve this, the model weights are limited only to +1 or -1. Moreover, only a binary activation function is used to clamp the inputs to either +1 or -1 as introduced in the binarized neural networks [208]. This binary representation of weights achieves 32× memory efficiency compared to the standard floating-point representation. Although the weights are in binary, temporaries generated between convolutional layers are still represented in floating-point. They require a lot of working memory resources which can still present an issue for wearable devices. To handle this, the computation order of inference in a binarized neural network has been modified following the work in [209]. Unlike in the traditional order, the resulting temporaries after the convolution layer are not stored in memory. Alternatively, they are directly passed to the pooling layer followed by batch normalization and binary activation layers. This makes the models not only memory efficient but also energy efficient because of the faster and less complex binary operations. Figure A.5 shows the modified order of computation in one BCNN block for processing heartbeat segments.

Multi-Objective Bayesian Optimization

Given previous evaluations for each of the $\boldsymbol{k}$ objective functions $\boldsymbol{f_k(x)}$, the goal is to find samples that provide more information about the Pareto optimal set $\boldsymbol{X^\star}$. MOBO serves this purpose by performing a sequential design space exploration where each objective function is replaced by a surrogate, cheaper to evaluate, probabilistic Gaussian Process (GP) model. Let $\boldsymbol{D_n = \{(x_i, Y_i)\}_{i=1}^n}$ represent the set of all the queried points up to iteration $\boldsymbol{n}$; where for each step $\boldsymbol{i}$, $\boldsymbol{x_i}$ represents the sampled architecture at iteration $\boldsymbol{i}$ and $\boldsymbol{Y_i}$ represents its corresponding vector of $\boldsymbol{k}$ real evaluated values from the $\boldsymbol{k}$ objective functions. It is assumed that for each function $\boldsymbol{f_k(x)}$, evaluations $\mathbf{f_k} := \boldsymbol{Y_{1:n}[k]}$ are jointly Gaussian with mean $\boldsymbol{m}$ and co-variance $\boldsymbol{K}$, *i.e.*, $\mathbf{f_k}|\boldsymbol{x_{1:n}} \sim \boldsymbol{N(m, K)}$. This means each GP model at iteration $\boldsymbol{n}$ represents a distribution over all the possible functions of $\boldsymbol{f_k(x)}$ based on the data collected

so far. This distribution is known as the posterior, and it represents the current belief about the shape of functions that most likely fit $\boldsymbol{D_n}$.

The next sample from the search space is selected using an acquisition function $\boldsymbol{\vartheta(x)}$. The merit in using $\boldsymbol{\vartheta(x)}$ is that, unlike the $\boldsymbol{k}$ objective functions, it is analytically available, making it much cheaper to evaluate than any $\boldsymbol{f_k(x)}$. Hence for each iteration $\boldsymbol{n}$, $\boldsymbol{\vartheta_n(x)}$ is constructed using one of the $\boldsymbol{k}$ GP models to identify which point should be queried next. The GP model selected to construct $\boldsymbol{\vartheta_n(x)}$ is chosen based on the improvement potential with regard to that specific objective function. Once the GP model is chosen, $\boldsymbol{\vartheta_n(x)}$ is formulated to yield high values where the uncertainty of the probabilistic model is high (exploration), and around where the GP has had the best evaluations (exploitation). Then, the sample that maximizes $\boldsymbol{\vartheta_n(x)}$ is selected to be the next query point $\boldsymbol{x_{n+1}}$. In the following iteration $\boldsymbol{n+1}$, the objective functions are evaluated yielding $\boldsymbol{Y_{n+1}}$. Given this new data pair and the previous ones, the GP models are updated using the new dataset $\boldsymbol{D_{n+1} = D_n \cup (x_{n+1}, Y_{n+1})}$. MOBO proceeds with this select-evaluate-update loop until the final iteration $\boldsymbol{N}$ is reached, and the Pareto set at that iteration is rendered as the final solution.

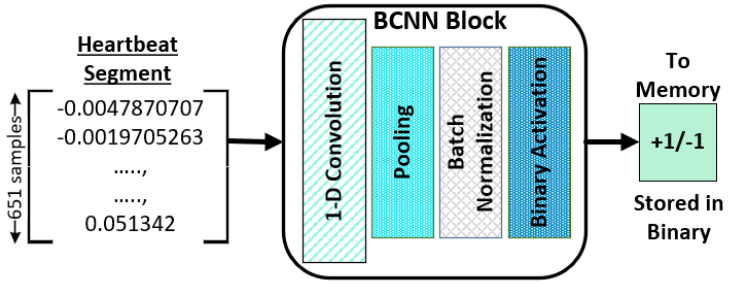
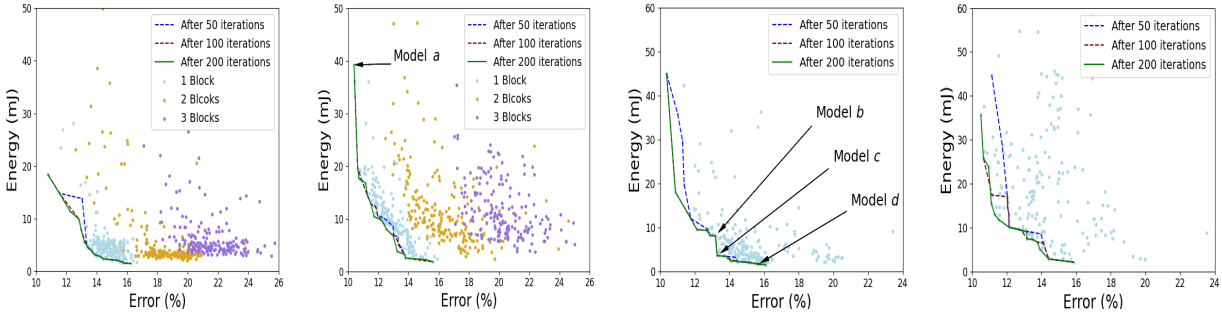


Figure A.5: Processing heartbeat segments through the layers of the BCNN block and the final result is stored in binary.

Table A.3: Ranges of the Architectural Search Parameters.

Parameter	Search Parameters Ranges
# BCNN blocks	*[1-3]*
# filters	*[2-5], [2-5], [2-5]*
Conv. layer kernel length	*[10-120], [10-70], [5-20]*
Conv. layer kernel stride	*[1-2], [1-2], [1-2]*
Pool. layer kernel length	*[2-3], [2-3], [2-3]*



(a) (b) (c) (d)

Figure A.6: Results from our experiments. Sub-figures (a) and (b) show MOBO and normalized-MOBO over 3 reference architectures, respectively. While (c) and (d) compare MOBO and random sampling, respectively, over one block reference architecture.

A.2.2 Experimental Setup

The multi-objective Bayesian optimization is built on top of Dragonfly [213]. We use Thompson sampling [214] for our acquisition function. Wrapper scripts are implemented around the objective functions to automate the selected models' generation, training, and deployment onto the target board. The details are provided below:

Training Process

Lead 11 ECG dataset from PTB diagnostic ECG database [56] is used for training and testing the models during and after the search process. It contains data for 200 subjects where 148 subjects suffer from MI, and the remaining 52 are normal. Out of the obtained heartbeat segments, 44214 segments are classified as MI while 6157 are normal. Since the

number of MI segments are 7× the number of normal ones, we ensure proper training by dividing the segments into 7 groups. Each group will always contain all the normal segments combined with around 6316 MI segments. Then for each group, a 10 fold cross-validation is performed. In this scheme, each group is divided into 10 folds where for each fold, a unique 10% of that group's segments are used for testing while the remaining 90% are for training and validation. Each model selected during the search process is trained for 20 epochs with Adam optimizer, a learning rate of 0.007, and softmax cross-entropy as the loss function. For each group, the model's performance is averaged across all folds. Finally, the model's overall performance is estimated as the average across the entire 10 fold cross-validations in all groups.

Target Device

Our proposed design methodology targets low-power medical wearable devices like Smart-Cardia INYU [85]. This device was used by the related works in [49, 50]. It is equipped with an ultra-low-power Microcontroller STM32L151 running on an ARM Cortex-M3 with a maximum clock frequency of 32 MHz. The device also has a 48 KB RAM, 384 KB of Flash memory, and 710 mAh battery. The device also possess an ECG sensor to retrieve ECG signals through a single lead. For our experiments, we utilize both a desktop machine with a GeForce RTX 2070 SUPER and an EFM32 Leopard Gecko [215] as the low-power target device. The Bayesian search and the accuracy estimation procedures are performed on the desktop machine. Then to retrieve the relative hardware measurements, the model is converted into its corresponding C code implementation and automatically flashed onto the EFM32 board.

The EFM32 Leopard Gecko development board has been chosen for our experiments as it runs on the same ARM Cortex-M3 as SmartCardia INYU and has similar specifications. Estimating the energy consumption from hardware measurements can be detailed as follows:

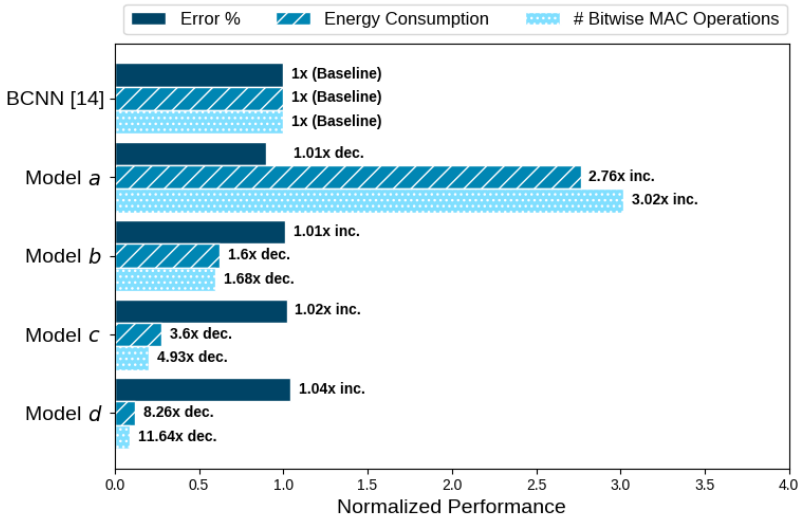


Figure A.7: Analysis of Bitwise MAC Operations Count, Energy Consumption and Error for the Binary Based Models.

First, the execution time for a single inference of an ECG segment is calculated once the cycle count per inference is retrieved. After that, the target device is reset. Then, the average power over the calculated execution time for a single inference is measured. Finally, the energy consumption per inference can be computed directly by multiplying both the average power and the execution time.

A.2.3 Results and Discussion

Experiments

We have performed multiple experiments to assess the effectiveness of MOBO within this problem context, as shown in Figure A.6. The first experiment incorporated conducting MOBO over a BCNN-inspired search space. The chosen architectural search parameters are defined in Table A.6, and their ranges are shown. Multiple ranges indicate the respective range of values for each consecutive BCNN block in an architecture. For convenience, this experiment was divided into 3 child experiments where for each one, the number of blocks

was fixed as either one, two, or three to manage the dependency of the search parameters on the number of blocks. The search spaces for each child experiment accounted for $\mathbf{1.78 \times 10^3, 1.73 \times 10^6,}$ and $\mathbf{4.44 \times 10^8}$ possible architectures, respectively. Each child experiment was run for 200 iterations, and their combined results are shown in Figure A.6a. The evolution of the ***combined*** Pareto frontier over 50, 100 and 200 iterations from each child experiment is shown. Two observations can be made here. The first is that MOBO tends to explore more around models that minimize energy consumption because the potential for improvement with respect to energy is greater than that with respect to error. The second observation is about how the single block architecture models dominate those from the other two architectures with respect to both objective functions.

Based on the first observation, the second experiment is designed to allow biasing the search in favor of one objective function over the other. Hence, rather than just directly using the real function evaluations, we add the option to normalize those evaluations in the Bayesian search process. This required modifying each function evaluation at every iteration $\boldsymbol{n}$ from $\boldsymbol{f_{kn}(x) := Y_n[k]}$ to $\boldsymbol{f_{kn}(x) := \alpha_k \times \frac{Y_n[k]-min_k}{max_k-min_k}}$, where $\boldsymbol{\alpha_k}$, $\boldsymbol{min_k}$, and $\boldsymbol{max_k}$ are the bias constant, minimum, and maximum values of the $\boldsymbol{k^{th}}$ objective function, respectively. Since the first experiment was more biased towards energy, we set $\boldsymbol{\alpha_k}$ for all objectives to 1 and use the *min* and *max* values from the previous experiment and re-run it. Figure A.6b shows that the sampled architectures are more spread out than those in Figure A.6a, indicating that MOBO has become more neutral in its search with respect to both objective functions.

The final experiment was to assess the effectiveness of the Bayesian search in terms of design space exploration. Based on the second observation from the first experiment, we re-run that non-normalized experiment twice but only for the one-block architecture. Bayesian search is used for the first run while the second employs random sampling. Figures A.6c and A.6d show their respective results. It can be observed that the Bayesian approach is much more systematic in its search to minimize the objective functions. This is evident through the

Table A.4: Models' Architectural Parameters

Model	**# filters**	**Conv. len.**	**Conv. str.**	**Pool. len.**
BCNN [52]	3	100	2	3
a	4	117	1	2
b	3	55	2	2
c	4	13	2	2
d	2	11	2	2

Table A.5: Comparison between Our Models and Previous Works with regard to Performance and Energy Metrics

	Performance			**Avg. Power (mW)**		**Exec. Time (ms)**		**Energy (mJ)**	
Model	Acc.	Sen.	Spec.	*14 MHz*	*48 MHz*	*14 MHz*	*48 MHz*	*14 MHz*	*48 MHz*
SVM [155]	90	-	-	14.24	46.92	13049.14	4303.28	185.82	201.91
RF [50]	83.26	87.95	78.82	14.34	46.98	13278.69	4378.69	190.42	205.71
BCNN [52]	90.29	90.41	90.16	14.52	46.71	893.14	279.86	12.97	13.07
Model *a*	**91.22**	**91.57**	**90.86**	**14.47**	**47.07**	**2477.77**	**846.39**	**35.85**	**39.84**
Model *b*	**89.63**	**90.01**	**89.24**	**14.63**	**46.97**	**553.68**	**176.74**	**8.10**	**8.30**
Model *c*	**88.26**	**87.27**	**89.27**	**15.30**	**47.08**	**235.2**	**80.54**	**3.60**	**3.79**
Model *d*	**86.92**	**85.91**	**87.96**	**15.31**	**47.14**	**104.30**	**36.23**	**1.57**	**1.71**

rapid convergence of the Pareto frontier in the Bayesian experiment, as it is almost the same after 100 and 200 iterations.

Final Benchmarking

The four models pointed out in Figures A.6b and A.6c are the ones we use for our final benchmarking. Their architectural search parameters values are presented in Table A.7. Regarding their memory footprint, our **models *a***, ***b***, ***c***, and ***d*** use up around **19.33**, **19.12**, **19.17**, and **19.05** KB of flash and **3.69**, **3.54**, **3.63**, and **3.52** KB of RAM, respectively. This indicates that models from our design space comply with the low memory requirements of medical wearable devices like SmartCardia INYU. Next, we re-train those models to the full 100 epochs and compare them against the BCNN implementation in [52]. As shown in Figure A.7, as the complexity of the model grows, so does the number of bitwise Multiply and Accumulate (MAC) operations. This, in turn, leads to increased energy consumption. However, as complexity is reduced, the energy savings are significant in comparison to the

loss in accuracy. For instance, our model ***d*** incurs **1.04**× more detection error than the BCNN, yet it is **8.26**× more energy efficient.

Finally, we benchmark our retrained models against the SVM [49], RF [50], and BCNN [52] works. We compare their performance in terms of accuracy, sensitivity, and specificity metrics. Additionally, we also re-implement these works on the EFM32 board to ensure consistency of the energy consumption estimation across them all. However, it should be noted that although we report the best performance values for the SVM and RF, we only implement their first level classifiers for the energy-related evaluations. This is justifiable since the first level classifiers are the most efficient in terms of the execution time and energy consumption. The energy-related readings are measured at 14 MHz (default) and 48 MHz (maximum) operating frequencies of the EFM32 board for validation. Table A.8 shows all measurements across all performance and energy metrics. Our **Model *a*** achieves the highest scores across the 3 performance metrics, whereas our remaining models are the most energy-efficient at the cost of some performance drop.

A.2.4 Summary

Adding intelligence to low-power wearable devices presents a design conundrum regarding the trade-off between high performance and energy efficiency. To address this, our proposed methodology provides a systematic automated design space exploration of efficient neural networks for MI detection on wearable devices. Our MOBO-based methodology allows for co-optimizing both detection error and energy consumption on the target device to render a Pareto optimal set of binarized models, allowing designers to choose their most suitable architectural design. Also, designers would be able to bias the search in the design process towards one objective or the other based on their preferences. To adhere to the memory limitations, our methodology explores the design space of variants of the BCNN architecture

suitable for deployment on wearable devices. Experimental evaluation shows that one of our explored models achieves an accuracy of **91.22%**, outperforming the MI detection state-of-the-art performance on wearable devices. Other explored models trade off some accuracy to conserve more energy (as high as **8.26×**).

A.3 Early Exit Neural Architecture Search for Wearable Devices

Equipping wearable devices with intelligence is essential for promoting mobile healthcare applications. However, challenges remain due to the resource limitations of these devices. In this work [53], we introduce EExNAS, a methodology for designing high-performance and resource-efficient dynamic Neural Architecture solutions for wearable devices. The methodology incorporates a platform-aware Neural Architecture Search (NAS) that accounts for energy efficiency at runtime through an Early-Exit (EEx) option. We showcase our methodology's merit across 2 wearable applications, Myocardial Infarction (MI) detection and Human Activity Recognition (HAR). Solutions from EExNAS are compared against those from related works in terms of accuracy and performance. For MI detection, our final solutions with EEx capability could reach **98.54%** accuracy on the PTB ECG dataset.

To show the merits of a 1D CNN solution with EEx option, we provide a case study using a *model a* whose 1D CNN-based architecture follows the template in Figure A.8. We provide an analysis in Figure A.9 on how data segments from the PTB ECG database [56] for Myocardial Infarction (MI) detection problem are processed using *model a*. The classification probability estimates at the EEx block indicate the confidence of the classification decision and all evaluations are normalized with respect to the relative baselines with no

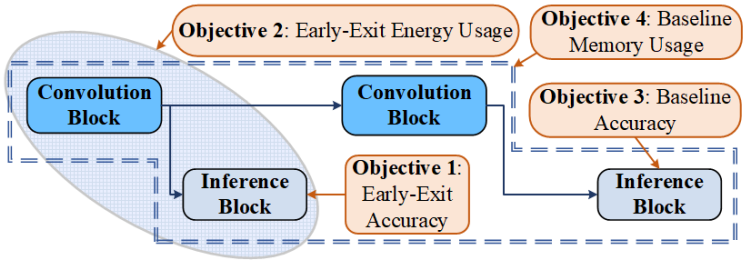


Figure A.8: The template baseline architecture from EExNAS with potential objective functions associations.

EEx option. In the top bar-chart, note how most segments in both conditional models with decision thresholds (***th***) of 0.7 and 0.99 are directed to take the EEx, indicating that the probability estimates are high enough to exceed the ***th*** values. This led to significant energy gains compared to the baseline reaching **7.34**× and **9.13**×, respectively. However, these performance improvements came at the expense of a slight degradation of accuracy.

Addressing the accuracy drop, rather than taking the probability estimates as they are, we apply temperature scaling to address possible mismatch between the probability estimates and true confidence values [216]. In the lower bar-chart of Figure A.9, note how the number of data segments not invoking the EEx option is clearly larger than that in the uncalibrated models, indicating how the original probability values can overestimate the true confidence values. Another interesting observation is that the accuracy of the conditional models surpassed that of the baseline. Mainly because more complex interpretations of relatively simple segments can occur at deeper parts of the architecture, leading to occasional misclassifications. And since the models are calibrated, these segments can be classified at the EEx block with high confidence using simpler representations, improving the overall accuracy. Note how calibration alters the accuracy and energy responses over confidence thresholds in Figure A.10. Although the energy savings are lower compared to the uncalibrated cases (**2.23**× at th=**0.7**), calibration offers a more generic sustainable approach applicable to multiple wearable applications in which decision making is critical, and high accuracy is needed. Therefore, the final takeaway here is that a calibrated wearable solution with an EEx capability can offer an overall more accurate and energy-efficient solution compared to a fixed solution.

Based on the previous arguments, the research challenges we aim to address in this work include:

- What techniques should be utilized to provide a generic design methodology for wear-

able applications that can render accurate and resource-efficient solutions?

- How to include potential EEx benefits at design time within the global design optimization problem?

Addressing the above-mentioned challenges, we propose a platform-aware Multi-Objective Neural Architecture Search (NAS) approach, namely EExNAS, that explores a pre-defined search space of architectural parameters to provide optimal model implementations with EEx capability. The most promising architectures are identified through their estimated evaluations over a designated set of objective functions. These objective functions can be accuracy- or performance-related (e.g., energy consumption and memory utilization). Also, they can be defined at different parts of the backbone architecture to promote the EEx capability, as shown in Figure A.8. Our research contributions can be summarized as follows:

- We propose EExNAS, a Multi-Objective NAS-based design methodology to develop resource-efficient solutions for wearable applications employing time-series data.
- We separately associate objective functions at the EEx block to optimize its implementation.
- We demonstrate the effectiveness of EExNAS across two wearable applications, Myocardial Infarction (MI) detection and Human Activity Recognition (HAR).
- On the PTB ECG dataset [56], EExNAS final solutions achieve state-of-the-art accuracy for MI detection on wearable devices, reaching **96.5%** and **98.54%**.
- On the w-HAR dataset [111], EExNAS final solution incurs a **0.584%** accuracy drop from the state-of-the-art but is **47.076%** more energy-efficient.

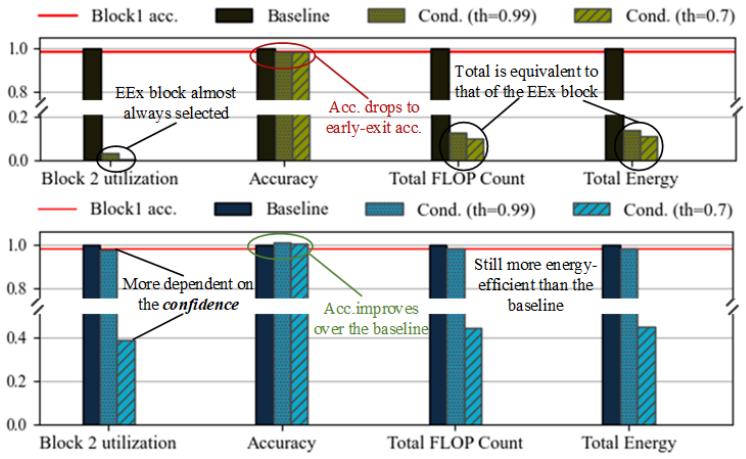


Figure A.9: Comparisons between conditional models at different confidence thresholds and their baselines for uncalibrated *(top)* and calibrated *(bot)* cases on the MI dataset. Blocks 1 and 2 are the consecutive inference blocks from Figure A.8.

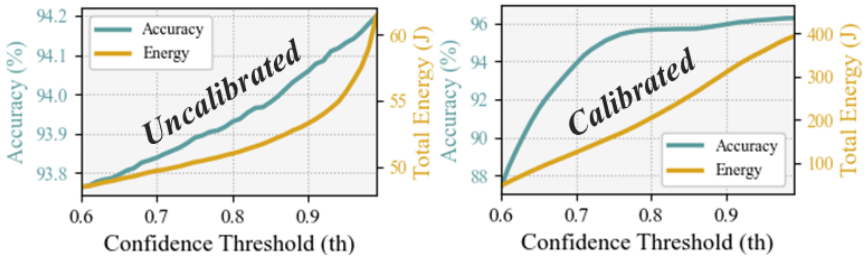


Figure A.10: Parametric sweeps across the confidence threshold for the uncalibrated *(left)* and calibrated *(right)* models

A.3.1 EExNAS Design Methodology

Figure A.11 illustrates an overview of EExNAS methodology. We go through the main components of the methodology in the following subsections.

A.3.2 Neural Architecture Search

The purpose of the NAS within EExNAS is not only to identify the models with the best accuracy evaluations but also the ones that efficiently utilize the limited resources of the target wearable device. Thus, the problem becomes a multi-objective optimization one incorporating performance objectives as well. Due to the conflicting nature between the

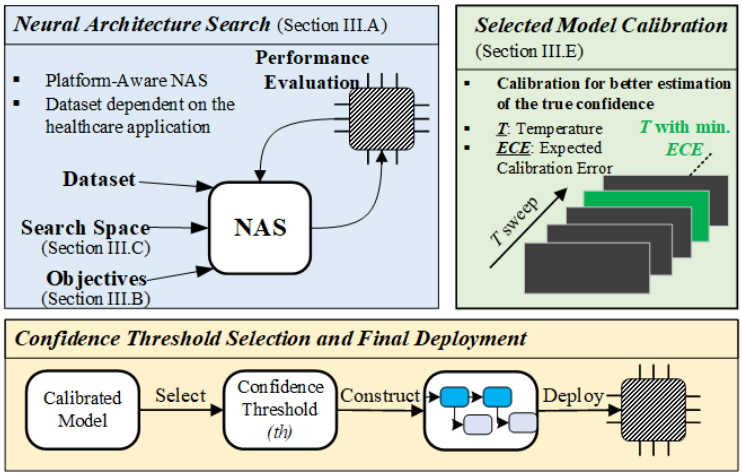


Figure A.11: EExNAS Design Methodology Overview

accuracy and performance objectives, the problem would not have a single solution, but a set of Pareto optimal ones instead. These Pareto-optimal solutions dominate all other explored solutions except each other, where formally in a minimization context, an architecture $x^{\star}$ would belong to the Pareto set if: $f_k(x^{\star}) \leq f_k(x) \forall k, x$ and $\exists j : f_j(x^{\star}) < f_j(x) \forall x \neq x^{\star}$. where f_k represents the k^{th} objective function.

When searching for the Pareto optimal architectures, a NAS controller each iteration needs to: sample architectural candidates from the search space, evaluate their respective objective functions, and update its search strategy based on these evaluations. In this work, the search strategy employed by the EExNAS controller is implemented using Multi-Objective Bayesian Optimization (MOBO) [212]. Other strategies like RL could've been implemented as well without any loss of generality. MOBO approximates each objective function with a surrogate Gaussian Process (GP) model. Thus, previous evaluations f_{kn} of the k^{th} objective function at iteration n are assumed to be jointly Gaussian with mean m and co-variance κ, *i.e.*, $\mathrm{f}_{kn}|x_{1:n} \sim N(m, \kappa)$, making the GP model a probabilistic distribution over possible functions of the associated objective function. From these GP models, an acquisition function is constructed and solved analytically to identify the next query point, whose true objective evaluations are determined and used to update the GP models.

Multi-Objective Formulation

To solve the multi-objective optimization problem, we apply linear scalarization across the multiple objective function estimates to identify the next query point through:

$$x_n = \arg\max_x \sum_i w_i.f_i \tag{A.2}$$

where our aim each iteration n is to identify the sample x_n which maximizes a reward associated with the objective functions. Note that f_i and w_i represent the i^{th} function estimate sampled from the respective objective's GP model and its associated user-assigned weight, respectively.

To maximize the effect of the function weights, true evaluations of each objective function F_i need to be normalized with respect to their max. and min. attainable values as follows:

$$F_{i_{norm.}} = \frac{F_i - F_{i_{min}}}{F_{i_{max}} - F_{i_{min}}} \tag{A.3}$$

Hence, each weight w_i becomes the sole determiner of the extent of impact each objective function can have on the search process. In our experiments, 4 objective functions were defined as shown in Figure A.8, where the max. and min. values for the performance objectives were obtained through evaluating the largest and smallest possible architectures in the search space, respectively. Whereas for the accuracy objectives max. and min. were estimates from well- and poorly- trained models.

Search Space

Our search space encompasses a backbone macro-architecture of 1D convolutional blocks with an additional EEx block. Variable architectural parameters from each convolution block are

used for the search space. These parameters encompass the number of output filters in addition to the kernel and strides for each of the convolution and pooling layers, respectively (Note that pooling layers are optional for each block). Therefore, each architecture in the search space can be characterized by a string x defined as follows:

$$x = (n_{F_1}, k_{c_1}, s_{c_1}, k_{p_1}, s_{p_1}, ...k_{p_N}, s_{p_N}, fc)$$

where k and s are the kernel and stride for each successive convolutional c and pooling p layer. N is the number of blocks and fc is an extra optional fully-connected layer. Although the dimensionality increases with the number of blocks in the search space, we found that utilizing 2-3 blocks is enough for multiple healthcare applications with time-series data, keeping the dimensionality relatively low. Even so, our MOBO-based solution is built on dragonfly [213], which provides techniques to handle high-dimensionality problems if needed.

Algorithm

We show in Algorithm 4 the pseudo-code for the MOBO-based NAS in EExNAS over a predefined search space X. In ***lines 1-4***, random samples are obtained according to pre-configured capital C_{init} (usually 5% of the total iteration number N_{iter}). These samples, alongside their corresponding evaluations, are used to initialize the GP models of the K objective functions. Then each iteration, one function estimate from each objective function's posterior is sampled in ***line 8***. Then, the sampled function estimates are used to identify the next query point using the acquisition function in ***line 9***. This query point is then evaluated using the real objective functions and used to update the sets of the queried points D and the Pareto frontier X^* in ***lines 12-13*** . After the last iteration N_{iter}, the Pareto set in ***line 14*** is returned as the final solution.

EEx Confidence Calibration

Once the final models are settled on from the NAS process, they need to be calibrated for their classification probabilities to reflect true confidence values, as was discussed in the motivational case study earlier. Firstly, the classification probability estimates arising from the softmax activation function at the EEx's block last layer are defined as:

$$\sigma(z_i)^k = \frac{exp(z_i^k)}{\sum_{j=1}^{K} exp(z_i^j)} \tag{A.4}$$

where K is the total number of classes and z_i^k is the softmax's function input for the k^{th} class. We follow the temperature scaling technique in [216] to calibrate the probability estimates. The technique involves dividing the probability estimates into M interval bins, where the accuracy and average confidence for each bin B_m can be defined as:

$$acc(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{I}(\hat{y}_i == y_i) \tag{A.5}$$

$$conf(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i \tag{A.6}$$

where $\hat{y}_i$ and y_i, represent the predicted and true class labels while $\hat{p}_i$ and $|B_m|$ are the prediction's probability estimate and samples' count in the m^{th} bin. To calibrate the probability estimates, the Expected Calibration Error (ECE) between accuracy and confidence defined below needs to be minimized:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)| \tag{A.7}$$

where n is the total number of samples. One way to minimize ECE is through scaling the inputs to the softmax by a temperature factor, where new confidence prediction becomes:

$$\hat{q}_i = \max_k \sigma(z_i/T)^{(k)} \tag{A.8}$$

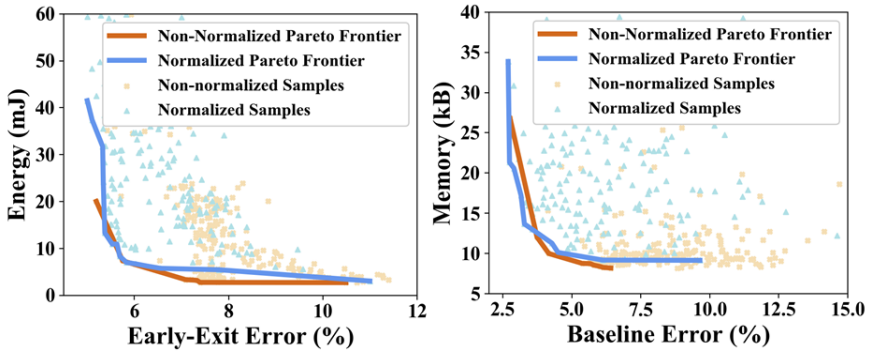


Figure A.12: Sampled architectures and Pareto frontiers in non-normalized (*brown*) and normalized (*blue*) search approaches.

Algorithm 4: EExNAS Architectural Search

Input: Configuration Variables: $\{C_{init}, N_{iter}\}$, Search Space: X
Output: Pareto Optimal Set: X^*

$D = \phi, X^* = \phi$
// Random initialization starts
for $i = 1$ ***to*** C_{init} **do**
  Randomly sample x_i and store its fn. evaluations $F_i(x_i)$ in Y_i
  $D = D \cup (x_i, Y_i)$ // Update queried points
end
$X^* = \text{Pareto\_init}(D)$ // Initial Pareto frontier
// MOBO starts
for $n = 1$ ***to*** N_{iter} **do**
  for $k = 1$ ***to*** K **do**
    $f_k = \text{GP}_k(D)$ // Sample fn. from posterior
  end
  $x_n = \arg\max_{x \in X} \sum_i w_i.f_i$ // Next Query
  for $k = 1$ ***to*** K **do**
    $Y_n[k] = F_k(x_n)$ // Evaluate objective fns.
  end
  $D = D \cup (x_n, Y_n)$ // Update queried points
  $X^* = \text{Pareto\_update}(x_n, Y_n, X^*)$ // Update frontier
end
return X^* // Final Pareto set

this works because the temperature factor T raises the output entropy of the softmax if $T > 1$. Thus through a simple sweep operation, the temperature value which gives the minimal ECE can be determined. At runtime based on the selected th value, if $\hat{q}_i > th$, then EEx would be invoked for that segment.

A.3.3 Experimental Setup

The MOBO-based NAS runs on a desktop machine and is built on top of Dragonfly [213]. Each search run takes 200 iterations where wrapper scripts are implemented around the objective functions to automate the evaluation. Sampled architectures' from the search have their accuracy evaluations estimated after training for 30 epochs. We followed the same Dataset preprocessing steps like filtering and segmentation in [51, 52, 111]. We also follow the training procedure in [52] for the MI ECG dataset in which the *"MI"* labeled segments are divided first into 7 groups, and then the normal segments are repeated across the 7 groups to handle the class imbalance within the dataset. The overall accuracy is then the average across the 10 fold cross-validations from all groups.

In terms of the target device, the EFM32 Giant Gecko [117] is selected as in [50, 52] for its specifications emulate those of wearable devices. Mainly, it runs on an ARM Cortex-M3 with a max operating frequency of 48 Mhz and a 128 kB RAM size. During the search process, a C code version of each sampled architecture is automatically generated and flashed onto the device to retrieve its energy and memory measurements.

A.3.4 Experiments and Results

The search strategy is first evaluated then the final models are bench-marked against other works as follows:

Search Process Assessment

First, we illustrate the effectiveness of normalizing the objective functions in Figure A.12, in which two searches, non-normalized and normalized, are run for 200 iterations each. In the non-normalized setting, it can be observed that the sample distribution is more skewed

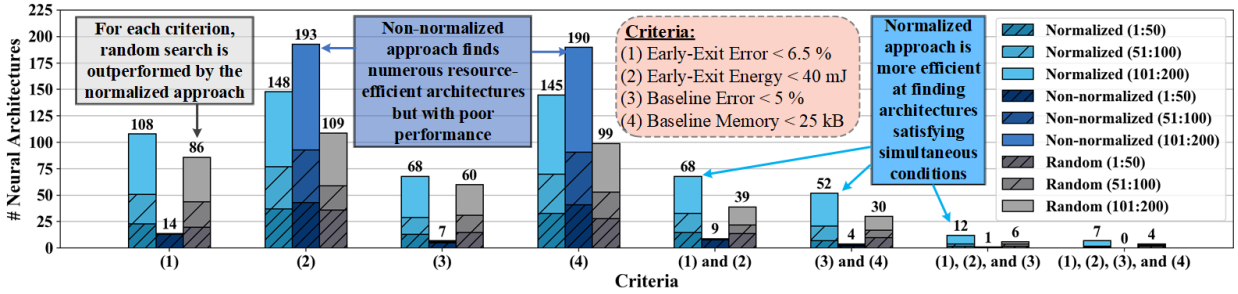
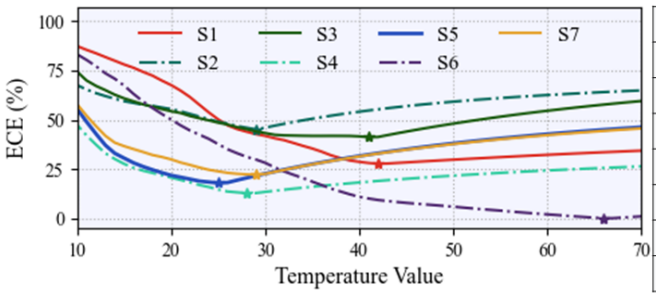


Figure A.13: Comparison between normalized, non-normalized, and random search approaches in terms of the number of architectures sampled that satisfy various criteria of the objective functions over 200 iterations of each.

towards improving upon the performance objectives. This is due to the large variations in the energy and memory values across different models in comparison to accuracy values, making the reward estimate more reliant on these objectives. On the other hand, the normalized version remedies this through normalizing variations across all objectives between 0 and 1. In this setting, the accuracy objectives were assigned 10× more the weights of the performance objectives. Consequently, it can be seen from the samples' distribution that the search has become more biased towards minimizing the accuracy-related objectives.

Next, we compare the non-normalized, normalized, and random search approaches through the quality of their sampled architectures. The results are shown in Figure A.13 where 4 criteria are defined to reflect the models' quality with regard to the various objectives. From the figure, the first observation is that the normalized approach always outperforms the random search in identifying architectures that meet the criteria. This is attributed to the more balanced exploitation-exploration nature of the normalized search. The second observation is that although the non-normalized search finds the most resource-efficient architectures, most of these architectures are trivial and do not satisfy the accuracy-related criteria. More importantly, the non-normalized search presents the lowest number of architectures capable of satisfying simultaneous criteria, pointing up once again the necessity of normalization.



Temp.	ECE (%)
41	27.77
28	45.14
40	41.22
27	12.89
24	18
65	0.13
28	22.23

Figure A.14: Selected Temperature for each split from the MI ECG dataset is the one that provides the minimal ECE

MI ECG Dataset Benchmarking

From the search process, two models, *a* and *b*, are chosen from the normalized Pareto frontier and retrained over 100 epochs for our final benchmarking. First, we illustrate how the temperature value is determined for *model a* in Figure A.14, where the ECE values are swept across the temperature using *model a*'s evaluations for each of the 7 training splits. The temperature which gives the minimum ECE value provides the optimal temperature value for that split. Exact values for each split are also shown in which a larger ECE value indicates a larger mismatch between the prediction estimates and the true confidence values, highlighting once more the importance of calibration to scale down the confidence represented by the prediction estimates.

We compare our models' accuracy against those from other works in Table A.6. We also provide the sensitivity and specificity evaluations as supplementary results. Although the k-NN offers the best accuracy evaluation, it is not built for wearable devices as it requires all the training data on the device. Our model *b* with EEx option at *th=0.99* offers the best accuracy results for a wearable-based solution at 98.54%. We can also observe how the calibrated conditional models can offer better accuracy than their respective baselines. Note that model *a* is the one we used for our motivational case study earlier.

In Table A.7, we compare the performance of our models against the CNN-based models

Table A.6: Performance Benchmarking on MI ECG Dataset

Work	Type	Acc. (%)	Sen. (%)	Spe. (%)
k-NN [48]	-	98.8	99.45	96.27
SVM [49]	Baseline	95	x	x
	Cond.	90	x	x
RF [50]	Baseline	83.26	87.95	78.82
	Cond.	80.32	81.02	79.63
CNN [51]	-	95.22	95.49	94.19
BCNN [52]	-	90.29	90.41	90.16
Model *a* (Ours)	Baseline	95.61	93.44	97.85
	Cond. (*th=0.7*)	**95.66**	**95.55**	**95.77**
	Cond. (*th=0.99*)	**96.5**	**95.44**	**97.59**
Model *b* (Ours)	Baseline	98.03	97.26	98.82
	Cond. (*th=0.7*)	**97.21**	**96.6**	**97.84**
	Cond. (*th=0.99*)	**98.54**	**97.66**	**99.44**

[51, 52]. We implemented their models in C-code and flashed them onto the EFM32 device for a fair comparison. The energy calculation for the conditional models is obtained through the summation of products of the ratios of segments classified and the total energy consumption at each exit point. Although the binarized nature of the BCNN [52] deems it the most resource-efficient solution with 13.03 mJ for each inference, the more accurate *model a (th=0.7)* is not far behind with 16.34 mJ. Note that more efficient versions of *model a* with $th < 0.7$ outperformed the BCNN with respect to both accuracy and performance. However, we focus on providing more generic model versions. Moreover, we notice *model b (th=0.99)* with the best accuracy is more efficient than its *model a (th=0.99)* counterpart. Because, unlike *model a*, the complexity of *model b*'s architecture is more evenly distributed between the two convolution blocks, making it the most suited overall candidate whenever high confidence is demanded.

w-HAR Benchmarking

To demonstrate how the methodology adapts to other applications, we showcase the benchmarking results on the w-HAR dataset for HAR. After NAS, *model c* is selected and retrained

Table A.7: Measurements on the EFM32 for MI models

Work	Type	RAM Occ. (kB)	Ergy/Inf. (mJ)
CNN [51]	-	101.380	97.651
BCNN [52]	-	3.556	13.033
Model *a* (Ours)	Baseline	**15.66**	36.394
	Cond. (*th=0.7*)		**16.344**
	Cond. (*th=0.99*)		**35.978**
Model *b* (Ours)	Baseline	**15.972**	28.32
	Cond. (*th=0.7*)		**22.187**
	Cond. (*th=0.99*)		**28.189**

Table A.8: Performance Benchmarking on wHAR dataset

Work	Type	Acc. (%)	Weigh. F1 (%)
Baseline [109]	-	94.87	94.96
Act.-aware [109]	-	97.34	97.37
Model *c* (Ours)	Baseline	95.59	95.4
	Cond. (*th=0.9*)	**96.203**	**95.995**
	Cond. (*th=0.99*)	**96.772**	**96.627**

for 300 epochs for the final evaluation. Its optimal temperature value was found at 5.1 with an ECE of 2.29%. The relatively small temperature factor means that the initial estimates were a relatively good indication of the true confidence, and the estimate values would not need to be scaled down aggressively. *Model c* is then compared against the ones in [109] in terms of both accuracy and performance. The WF1 score, obtained from the confusion matrix of each classification class, is also provided as a supplementary result. As displayed in Tables A.8 and A.9, the activity-aware implementation still achieves the best accuracy readings. However, *model c (th=0.99)* incurs a 0.584% drop in accuracy for 78.985% and 47.076% gains in memory and energy efficiency, respectively.

Table A.9: Measurements on EFM32 for HAR models

Work	Type	RAM Occ. (kB)	Ergy/Inf. (mJ)
[109]	Baseline	10.012	1.037
	Act.-aware		1.368
Model *c* (Ours)	Base.	2.104	0.931
	Cond. *(th=0.9)*		0.637
	Cond. *(th=0.99)*		0.724

A.3.5 Summary

In this work, we have introduced EExNAS, a design methodology to render 1D CNN-based wearable device solutions with EEx capability. Because the EEx decision depends on the classification confidence at the EEx block, final models rendered through the NAS are calibrated using temperature scaling to remedy the mismatch between the prediction estimates and the true confidence. We've shown that calibrated models with EEx are not only more resource-efficient but also can outperform their baselines in terms of accuracy evaluations. We demonstrated the efficiency of our methodology over MI and HAR applications, where our *model b (th=0.99)* achieved state-of-the-art accuracy of 98.54% for a wearable device solution on the MI PTB ECG dataset. While our *model c (th=0.99)* a 0.584% drop in accuracy on the w-HAR dataset, but is 47.076% more energy-efficient.

A.4 Feature Augmented Hybrid CNN for Stress Recognition

Stress is a physiological state that hampers mental health and has serious consequences to physical health. Moreover, the COVID-19 pandemic has increased stress levels among people across the globe. Therefore, continuous monitoring and detection of stress are necessary. The recent advances in wearable devices have allowed the monitoring of several physiological signals related to stress. Among them, wrist-worn wearable devices like smartwatches are most popular due to their convenient usage. And the photoplethysmography (PPG) sensor is the most prevalent sensor in almost all consumer-grade wrist-worn smartwatches. Therefore, this work [110] focuses on using a wrist-based PPG sensor that collects Blood Volume Pulse (BVP) signals to detect stress which may be applicable for consumer-grade wristwatches. Moreover, state-of-the-art works have used either classical machine learning algorithms to detect stress using hand-crafted features or have used deep learning algorithms like Convolutional Neural Network (CNN) which automatically extracts features. This work proposes a novel hybrid CNN (H-CNN) classifier that uses both the hand-crafted features and the automatically extracted features by CNN to detect stress using the BVP signal. Evaluation on the benchmark WESAD dataset shows that, for 3-class classification (Baseline vs. Stress vs. Amusement), our proposed H-CNN outperforms traditional classifiers and normal CNN by ≈5% and ≈7% accuracy, and ≈10% and ≈7% macro F1 score, respectively. Also for 2-class classification (Stress vs. Non-stress), our proposed H-CNN outperforms traditional classifiers and normal CNN by ≈3% and ≈5% accuracy, and ≈3% and ≈7% macro F1 score, respectively.

The novel contributions of this work are as follows:

- Propose a novel hybrid CNN (H-CNN) classifier for stress detection using wrist-based

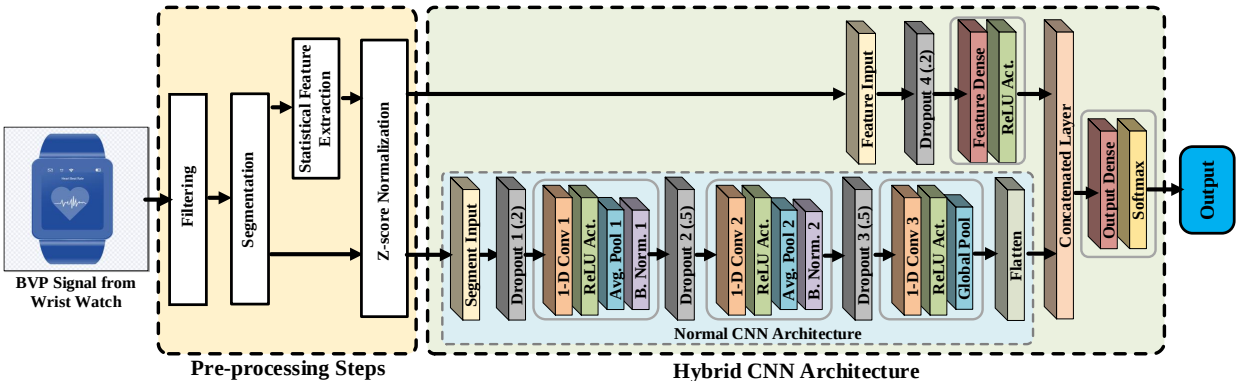


Figure A.15: Overview of Our Proposed Methodology

PPG sensor. It uses both handcrafted features and automatically extracted features by CNN to detect stress.

- Validation of our proposed approach using BVP signal from WESAD [**?**] dataset collected through wrist-based PPG.
- Evaluation on the benchmark WESAD dataset shows that, for 3-class classification (Baseline vs. Stress vs. Amusement), our proposed H-CNN outperforms traditional classifiers and normal CNN by ≈5% and ≈7% accuracy, and ≈10% and ≈7% macro F1 score, respectively. Also for 2-class classification (Stress vs. Non-stress), our proposed H-CNN outperforms traditional classifiers and normal CNN by ≈3% and ≈5% accuracy, and ≈3% and ≈7% macro F1 score, respectively.

A.4.1 Methodology

Pre-processing Steps

Filtering: As shown in Figure A.15, the pre-processing steps start with filtering the raw BVP signal. We filter the raw BVP signal by a butter-worth bandpass filter of order 3 with cutoff frequencies (f_1=.7 Hz and f_2=3.7 Hz). We take into account the heart rate at rest (≈40 BPM) or high heart rate due to exercise scenarios or tachycardia (≈220 BPM)

Table A.10: List of Extracted Features

Feature Symbol	Feature Names
μ_{HR}, σ_{HR}	Mean and Standard Devaiation of HR
μ_{HRV}, σ_{HRV}	Mean and Standard Devaiation of HRV
$NN50$, $pNN50$	Number and percentage of HRV intervals differing more than 50 ms
rms_{HRV}	Root mean square of the HRV
f^{x}_{HRV} $x \in ULF, LF, HF, UHF$	Energy in different frequency component of the HRV
$f^{LF/HF}_{HRV}$	Ratio of LF and HF component
$\sum_{x}^{f}$ $x \in ULF, LF, HF, UHF$	$\sum$ of the frequnecy components in ULF-HF
rel^{f}_{x}	Relative power of freq. components
LF_{norm}, HF_{norm}	Normalised LF and HF component

Heart Rate (HR), Heart Rate Variability (HRV)

following the method mentioned in [217].

Segmentation: The filtered signal is segmented by a window of 60 seconds of data following the paper that introduced the WESAD dataset [174]. We use a sliding length of 5 seconds in between the segments. Each segment contains 3840 samples as the sampling rate of the BVP signal is 64 Hz.

Feature extraction: The first step of the feature extraction is the detection of heart-beats. Once the peaks are detected, different time domain and frequency domain features are extracted based on the location of the peaks. We extract the time and frequency domain features as in [174] to ensure a fair comparison of our H-CNN classifier against the traditional machine learning classifiers used in the WESAD paper. We use the same frequency bands - ultra-low (ULF: 0.01-0.04 Hz), low (LF: 0.04-0.15 Hz), high (HF: 0.15-0.4 Hz) and ultra-high (UHF: 0.4-1.0 Hz) band as in [174] to calculate different frequency domain features. The list of extracted features is given in Table A.10.

Z-score normalization: Z-score normalization is performed before passing the segments and extracted features to the H-CNN architecture.

Hybrid CNN (H-CNN) Architecture

The normalized BVP segments and the corresponding features for each segment are passed to our H-CNN architecture as shown in Figure A.15. The H-CNN architecture has two input layers- Segment and feature input. The segment input layer is followed by a dropout layer (with a 20% dropout rate) which is then followed by 3 convolution blocks. The first and second convolution blocks have - convolution, *ReLU* activation, average pooling, and batch normalization layers. Both first and second convolution block is followed by dropout layers with 50% dropout rate which are added to reduce overfitting. The third convolution block has one convolution layer followed by a global average pooling layer which is also used to reduce the overfitting of the CNN. For the normal CNN architecture, the output of the global average pooling layer is directly fed to the output dense layer followed by a *Softmax* activation. However, for the H-CNN architecture, the output of the global average pooling layer is concatenated with the feature dense layer. Finally, the concatenated layer is fed to the output dense layer that is followed by the *Softmax* activation.The details of our H-CNN architecture are shown in Table A.11. As shown in Table A.11, the total number of parameters required to classify a segment is 6846+(13*n_c), where n_c is the number of output classes. In this paper, we perform both 2-class (Stress vs. Non-stress) and 3-class (Baseline vs. Stress vs. Amusement) classification from the WESAD dataset.

A.4.2 Experimental Evaluation

Dataset

WESAD dataset is used for the validation of our proposed methodology as it is the only publicly available dataset that contains wrist-based PPG sensor data for stress and affect detection. Although the dataset contains data for a total of 15 subjects from both chest

Table A.11: Hybrid CNN Architecture Details

Layer Name	**Kernel Size**	**Stride Size**	**Act. Func.**	**Output Shape**	**# of Param.**
Seg. Inp.	-	-	-	3840x1	0
D.O. 1	-	-	-	3840x1	0
Conv 1	64	4	ReLU	945x8	520
Pool 1	4	4	-	236x8	0
B.N. 1	-	-	-	236x8	32
D.O. 2	-	-	-	236x8	0
Conv 2	32	2	ReLU	103x16	4112
Pool 2	4	4	-	25x16	0
B.N. 2	-	-	-	25x16	64
D.O. 3	-	-	-	25x16	0
Conv 3	16	1	ReLU	10x8	2056
G. Pool	4	4	-	8	0
Flatten	-	-	-	8	0
Feat. Inp.	-	-	-	19	0
D.O. 4	-	-	-	19	0
Feat. Den.	-	-	ReLU	4	80
Concate	-	-	-	12	0
Out. Den.	-	-	SM	n_c	13*n_c
Total Number of Parameters					**6846+(13*n_c)**

Segment Input (Seg. Inp.), Dropout (D.O.), Batch Normalization (B.N.), Global Average Pooling (G. Pool), Feature Input (Feat. Inp.), Feature Dense (Feat. Den.), Output Dense (Out. Den.), Softmax (S.M.)

(RespiBAN) and wrist (Empatica E4) worn sensors, we are only interested in using the wrist-based BVP signal collected through the PPG sensor. The dataset is labeled for 3 types of classes - baseline (neutral), amusement, stress.

Model Training and Evaluation

We train our normal CNN and H-CNN classifiers with a batch size of 500. The models are trained for 200 epochs with an early stopping mechanism having a patience value of 70. We monitor the validation recall value to select the best model from the epochs. To ensure

proper training for the imbalance dataset, we assign class weights to each class using the following formula in Eq. A.9.

$$w_i = \frac{1}{N_i} * \frac{N}{n_c} \tag{A.9}$$

Here, w_i, and N_i represent the class weight and the number of segments belonging to class i, respectively. N is the total number of segments from all classes and n_c is the number of output classes. The *CategoricalCrossentropy* is used as the loss function. We use the *Adam* optimizer with a learning rate of .001. To demonstrate the generalization property of our trained model and to ensure a fair comparison with the traditional classifiers in [174], we also perform Leave One Subject Out (LOSO) validation. As shown in Figure A.16, the Linear Discriminant Analysis (LDA) classifier in [174] outperforms other classical algorithms for 3-class classification with an accuracy of 70.17% and macro F1 score of 54.72%. Our normal CNN achieves slightly less accuracy of 68.52% compared to LDA but outperforms in macro F1 score with 57.67%. Our H-CNN classifier outperforms both LDA and our normal CNN with an accuracy of 75.21% and macro F1 score of 64.15%. Thus, our H-CNN improves the accuracy by ≈5% and ≈7% compared to LDA and normal CNN, respectively. For macro F1 score, our H-CNN shows higher improvement of ≈10% and ≈7% compared to LDA and normal CNN, respectively. For 2-class (Stress vs. Non-stress) classification, baseline and amusement are considered as the non-stress class. As shown in Figure A.17, for 2-class classification also, our H-CNN improves the accuracy by ≈3% and ≈5% compared to LDA classifier and normal CNN, respectively. Similarly, for macro F1 score, our H-CNN improves the performance by ≈3% and ≈7% compared to LDA and normal CNN, respectively.

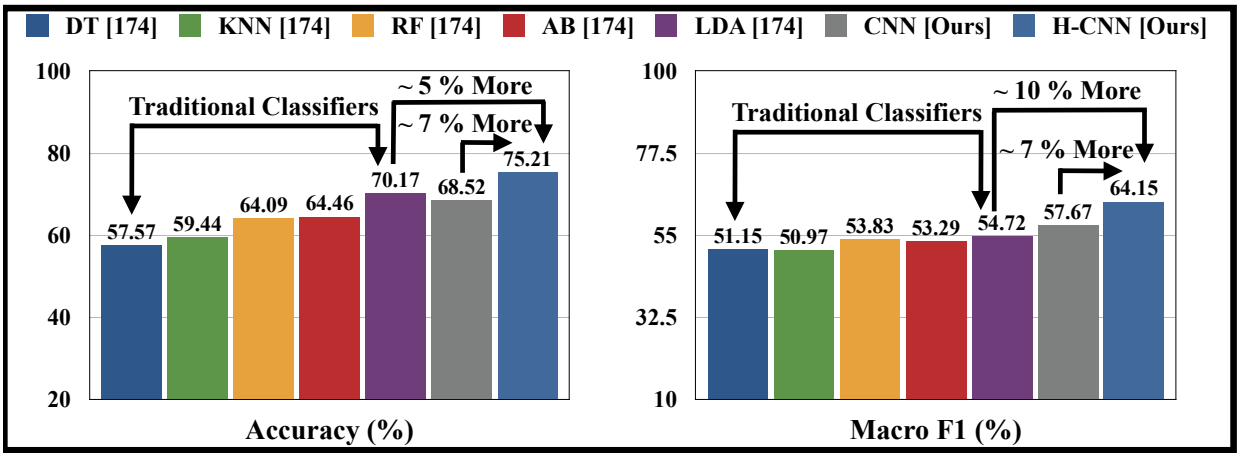


Figure A.16: Performance Comparison on 3-Class (Baseline vs Stress vs Amusement) Classification

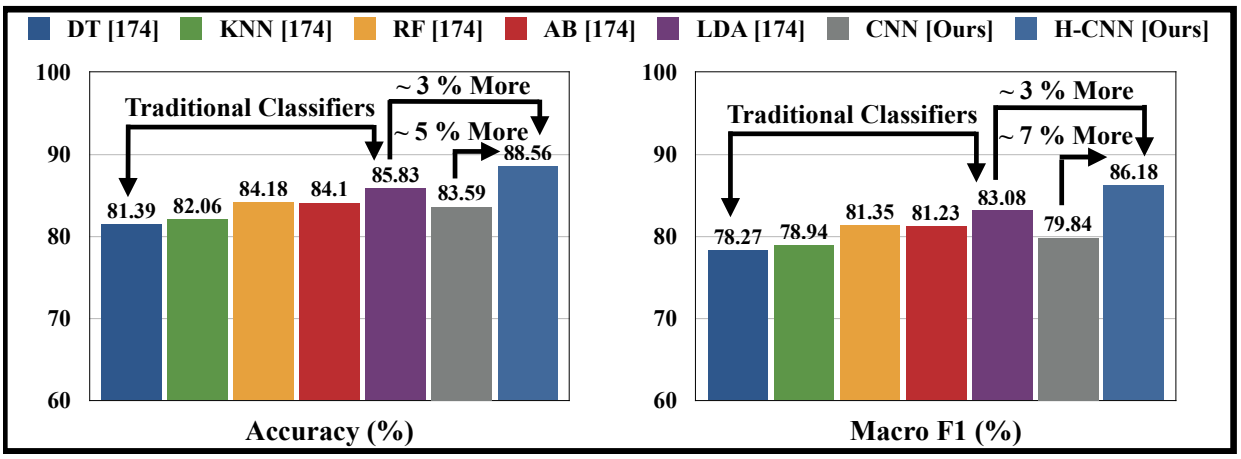


Figure A.17: Performance Comparison on 2-Class (Stress vs Non-stress) Classification

A.4.3 Summary

This paper proposes a novel hybrid CNN (H-CNN) classifier to detect stress using a wrist-based PPG sensor focusing on consumer-grade wristwatches. Our H-CNN uses both the hand-crafted features and the automatically extracted features by CNN to detect stress using the BVP signal. Evaluation on the benchmark WESAD dataset shows that, for 3-class classification (Baseline vs. Stress vs. Amusement), our proposed H-CNN outperforms traditional classifiers and normal CNN by ≈5% and ≈7% accuracy, and ≈10% and ≈7% macro F1 score, respectively. Also for 2-class classification (Stress vs. Non-stress), our

proposed H-CNN outperforms traditional classifiers and normal CNN by ≈3% and ≈5% accuracy, and ≈3% and ≈7% macro F1 score, respectively. To the best of our knowledge, our H-CNN shows the highest performance for both 3-class and 2 -class classification using the BVP signal from the WESAD dataset while performing LOSO validation.

Bibliography

[1] Mike Gianfagna. What is moore's law?, 2021. URL https://www.synopsys.com/glossary/what-is-moores-law.html#:~:text=Definition,doubles%20about%20every%20two%20years.

[2] Juhong Feng and Kam Yu. Moore's law and price trends of digital products: the case of smartphones. *Economics of Innovation and New Technology*, 29(4):349–368, 2020.

[3] Marco Giordani, Michele Polese, Marco Mezzavilla, Sundeep Rangan, and Michele Zorzi. Toward 6g networks: Use cases and technologies. *IEEE Communications Magazine*, 58(3):55–61, 2020.

[4] Zakria Qadir, Khoa N Le, Nasir Saeed, and Hafiz Suliman Munawar. Towards 6g internet of things: Recent advances, use cases, and open challenges. *ICT Express*, 2022.

[5] Feng Xia, Laurence T Yang, Lizhe Wang, Alexey Vinel, et al. Internet of things. *International journal of communication systems*, 25(9):1101, 2012.

[6] Apek Mulay. *Sustaining moore's law: uncertainty leading to a certainty of iot revolution*. Morgan & Claypool Publishers, 2015.

[7] Massimo Alioto. *Enabling the Internet of Things: from integrated circuits to integrated systems*. Springer, 2017.

[8] Mohammad Hasan. State of iot 2022, May 2022. URL https://iot-analytics.com/number-connected-iot-devices/.

[9] Global iot market. URL https://www.marketsandmarkets.com/Market-Reports/internet-of-things-market-573.html#:~:text=The%20global%20IoT%20market%20size,16.7%25%20during%20the%20forecast%20period.

[10] T Poongodi, Rajalakshmi Krishnamurthi, R Indrakumari, P Suresh, and Balamurugan Balusamy. Wearable devices and iot. *A handbook of Internet of Things in biomedical and cyber physical system*, pages 245–273, 2020.

[11] Nishank Jain, Alka Chaudhary, Nidhi Sindhwani, and Ajay Rana. Applications of wearable devices in iot. In *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pages 1–4. IEEE, 2021.

[12] Heetae Yang, Jieun Yu, Hangjung Zo, and Munkee Choi. User acceptance of wearable devices: An extended perspective of perceived value. *Telematics and Informatics*, 33 (2):256–269, 2016.

[13] Eunil Park. User acceptance of smart wearable devices: An expectation-confirmation model approach. *Telematics and Informatics*, 47:101318, 2020.

[14] Suranga Seneviratne, Yining Hu, Tham Nguyen, Guohao Lan, Sara Khalifa, Kanchana Thilakarathna, Mahbub Hassan, and Aruna Seneviratne. A survey of wearable devices and challenges. *IEEE Communications Surveys & Tutorials*, 19(4):2573–2620, 2017.

[15] Global wearable technology market. URL `https://www.grandviewresearch.com/industry-analysis/wearable-technology-market`.

[16] Sheikh MA Iqbal, Imadeldin Mahgoub, E Du, Mary Ann Leavitt, and Waseem Asghar. Advances in healthcare wearable devices. *NPJ Flexible Electronics*, 5(1):9, 2021.

[17] Yuemeng Cheng, Kan Wang, Hao Xu, Tangan Li, Qinghui Jin, and Daxiang Cui. Recent developments in sensors for wearable device applications. *Analytical and bioanalytical chemistry*, 413(24):6037–6057, 2021.

[18] Casey Erdmier, Jason Hatcher, and Michael Lee. Wearable device implications in the healthcare industry. *Journal of medical engineering & technology*, 40(4):141–148, 2016.

[19] Lin Lu, Jiayao Zhang, Yi Xie, Fei Gao, Song Xu, Xinghuo Wu, Zhewei Ye, et al. Wearable health devices in health care: narrative systematic review. *JMIR mHealth and uHealth*, 8(11):e18907, 2020.

[20] Global digital health market. URL `https://www.grandviewresearch.com/industry-analysis/digital-health-market`.

[21] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

[22] Paolo Bellavista, Javier Berrocal, Antonio Corradi, Sajal K Das, Luca Foschini, and Alessandro Zanni. A survey on fog computing for the internet of things. *Pervasive and mobile computing*, 52:71–99, 2019.

[23] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. Ieee, 2010.

[24] Gustavo Caiza, Morelva Saeteros, William Oñate, and Marcelo V Garcia. Fog computing at industrial level, architecture, latency, energy, and security: A review. *Heliyon*, 6(4):e03706, 2020.

[25] Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. *ACM Computing Surveys (CSUR)*, 51(6):1–29, 2019.

[26] Jianbing Ni, Kuan Zhang, Xiaodong Lin, and Xuemin Shen. Securing fog computing for internet of things applications: Challenges and solutions. *IEEE Communications Surveys & Tutorials*, 20(1):601–628, 2017.

[27] Syrine Sahmim and Hamza Gharsellaoui. Privacy and security in internet-based computing: cloud computing, internet of things, cloud of things: a review. *Procedia computer science*, 112:1516–1522, 2017.

[28] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE access*, 8:85714–85728, 2020.

[29] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[30] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49 (5):78–81, 2016.

[31] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[32] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE international conference on smart cloud (SmartCloud)*, pages 20–26. IEEE, 2016.

[33] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE transactions on knowledge and data engineering*, 31(12):2346–2363, 2018.

[34] Indrė Žliobaitė, Mykola Pechenizkiy, and Joao Gama. An overview of concept drift applications. *Big data analysis: new algorithms for a new society*, pages 91–114, 2016.

[35] Samuel Ackerman, Eitan Farchi, Orna Raz, Marcel Zalmanovici, and Parijat Dube. Detection of data drift and outliers affecting machine learning model performance over time. *arXiv preprint arXiv:2012.09258*, 2020.

[36] Keyvan Rahmani, Rahul Thapa, Peiling Tsou, Satish Casie Chetty, Gina Barnes, Carson Lam, and Chak Foon Tso. Assessing the effects of data drift on the performance of machine learning models used in clinical sepsis prediction. *International Journal of Medical Informatics*, page 104930, 2022.

[37] Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *Neurocomputing*, 459:249–289, 2021.

[38] Raffaele Gravina, Parastoo Alinia, Hassan Ghasemzadeh, and Giancarlo Fortino. Multi-sensor fusion in body sensor networks: State-of-the-art and research challenges. *Information Fusion*, 35:68–80, 2017.

[39] Philip Schmidt, Robert Dürichen, Attila Reiss, Kristof Van Laerhoven, and Thomas Plötz. Multi-target affect detection in the wild: an exploratory study. In *Proc. of the 23rd Int Symposium on Wearable Computers*, pages 211–219, 2019.

[40] Nafiul Rashid, Berken Utku Demirel, Mohanad Odema, and Mohammad Abdullah Al Faruque. Template matching based early exit cnn for energy-efficient myocardial infarction detection on low-power wearable devices. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 6(2), jul 2022. doi: 10.1145/3534580. URL `https://doi.org/10.1145/3534580`.

[41] Heart disease facts, December 2020. URL `https://www.cdc.gov/heartdisease/facts.htm`.

[42] WHO. Who — prevention of recurrences of myocardial infarction and stroke study, 2020. URL `https://www.who.int/cardiovascular_diseases/priorities/secondary_prevention/country/en/index1.html`.

[43] GUSTO Angiographic Investigators. The effects of tissue plasminogen activator, streptokinase, or both on coronary-artery patency, ventricular function, and survival after acute myocardial infarction. *New England Journal of Medicine*, 329(22):1615–1622, 1993.

[44] N. Rashid, M. Dautta, P. Tseng, and M. A. Al Faruque. Hear: Fog-enabled energy-aware online human eating activity recognition. *IEEE Internet of Things Journal*, 8 (2):860–868, 2021. doi: 10.1109/JIOT.2020.3008842.

[45] Hossein Mamaghanian, Nadia Khaled, David Atienza, and Pierre Vandergheynst. Compressed sensing for real-time energy-efficient ecg compression on wireless body sensor nodes. *IEEE Transactions on Biomedical Engineering*, 58(9):2456–2466, 2011.

[46] Kalle Tammemäe, Axel Jantsch, Alar Kuusik, Jürgo-Sören Preden, and Enn Õunapuu. Self-aware fog computing in private and secure spheres. In *Fog Computing in the Internet of Things*, pages 71–99. Springer, 2018.

[47] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.

[48] U Rajendra Acharya, Hamido Fujita, Vidya K Sudarshan, Shu Lih Oh, Muhammad Adam, Joel EW Koh, Jen Hong Tan, Dhanjoo N Ghista, Roshan Joy Martis, Chua K Chua, et al. Automated detection and localization of myocardial infarction using electrocardiogram: a comparative study of different leads. *Knowledge-Based Systems*, 99:146–156, 2016.

[49] Dionisije Sopic, Amin Aminifar, Amir Aminifar, and David Atienza. Real-time classification technique for early detection and prevention of myocardial infarction on wearable devices. In *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4. IEEE, 2017.

[50] Dionisije Sopic, Amin Aminifar, Amir Aminifar, and David Atienza. Real-time event-driven classification technique for early detection and prevention of myocardial infarction on wearable systems. *IEEE transactions on biomedical circuits and systems*, 12 (5):982–992, 2018.

[51] U Rajendra Acharya, Hamido Fujita, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, and Muhammad Adam. Application of deep convolutional neural network for automated detection of myocardial infarction using ecg signals. *Information Sciences*, 415:190–198, 2017.

[52] Nafiul Rashid and Mohammad Abdullah Al Faruque. Energy-efficient real-time myocardial infarction detection on wearable devices. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC)*, pages 4648–4651, 2020. doi: 10.1109/EMBC44109.2020.9175232.

[53] Mohanad Odema, Nafiul Rashid, and Mohammad Abdullah Al Faruque. Eexnas: Early-exit neural architecture search solutions for low-power wearable devices. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2021.

[54] Nils Strodthoff, Patrick Wagner, Tobias Schaeffter, and Wojciech Samek. Deep learning for ecg analysis: Benchmarks and insights from ptb-xl. *IEEE Journal of Biomedical and Health Informatics*, 25(5):1519–1528, 2021. doi: 10.1109/JBHI.2020.3022989.

[55] Harold Martin, Ulyana Morar, Walter Izquierdo, Mercedes Cabrerizo, Anastasio Cabrera, and Malek Adjouadi. Real-time frequency-independent single-lead and single-beat myocardial infarction detection. *Artificial Intelligence in Medicine*, 121:102179, 2021.

[56] R Bousseljot, D Kreiseler, and A Schnabel. Nutzung der ekg-signaldatenbank cardiodat der ptb über das internet. *Biomedizinische Technik/Biomedical Engineering*, 40(s1): 317–318, 1995.

[57] Priyadarshini Panda, Abhronil Sengupta, and Kaushik Roy. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 475–480. IEEE, 2016.

[58] Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. Why should we add early exits to neural networks? *arXiv preprint arXiv:2004.12814*, 2020.

[59] Nitthilan Kanappan Jayakodi, Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Design and optimization of energy-accuracy tradeoff networks for mobile platforms via pretrained deep models. *ACM Transactions on Embedded Computing Systems (TECS)*, 19(1):1–24, 2020.

[60] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In *International Conference on Machine Learning*, pages 527–536. PMLR, 2017.

[61] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Dieter Kreiseler, Fatima I Lunze, Wojciech Samek, and Tobias Schaeffter. Ptb-xl, a large publicly available electrocardiography dataset. *Scientific data*, 7(1):1–15, 2020.

[62] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

[63] Giorgio Roffo, Simone Melzi, Umberto Castellani, and Alessandro Vinciarelli. Infinite latent feature selection: A probabilistic latent graph-based ranking approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1398–1406, 2017.

[64] Bram van Ginneken. Fifty years of computer analysis in chest imaging: rule-based, machine learning, deep learning. *Radiological physics and technology*, 10(1):23–32, 2017.

[65] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[66] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[67] Qing Lu, Weiwen Jiang, Xiaowei Xu, Yiyu Shi, and Jingtong Hu. On neural architecture search for resource-constrained hardware platforms. *arXiv preprint arXiv:1911.00105*, 2019.

[68] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[69] Lile Cai, Anne-Maelle Barneche, Arthur Herbout, Chuan Sheng Foo, Jie Lin, Vijay Ramaseshan Chandrasekhar, and Mohamed M Sabry Aly. Tea-dnn: the quest for time-energy-accuracy co-optimized deep neural networks. In *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2019.

[70] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. In *Advances in Neural Information Processing Systems*, pages 4978–4990, 2019.

[71] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pages 759–770, 2019.

[72] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[73] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL `https://arxiv.org/pdf/1908.09791.pdf`.

[74] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[75] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8612–8620, 2019.

[76] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme fixed-point compression. *arXiv preprint arXiv:2004.07320*, 2020.

[77] Fragoulis Nikolaos, Ilias Theodorakopoulos, Vasileios Pothos, and Evangelos Vassalos. Dynamic pruning of cnn networks. In *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pages 1–5, 2019. doi: 10.1109/IISA.2019.8900711.

[78] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *arXiv preprint arXiv:1812.08928*, 2018.

[79] TensorFlow. Post-training dynamic range quantization, 2022. URL `https://www.tensorflow.org/lite/performance/post_training_quant`.

[80] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits*, 52(1):127–138, 2016.

[81] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, 2019.

[82] Jiapu Pan and Willis J Tompkins. A real-time qrs detection algorithm. *IEEE transactions on biomedical engineering*, (3):230–236, 1985.

[83] HJ Landau. Sampling, data transmission, and the nyquist rate. *Proceedings of the IEEE*, 55(10):1701–1706, 1967.

[84] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[85] Smartcardia, August 2020. URL `https://smartcardia.com/`.

[86] Srinivasan Murali, Francisco Rincon, and David Atienza. A wearable device for physical and emotional health monitoring. In *2015 Computing in Cardiology Conference (CinC)*, pages 121–124. IEEE, 2015.

[87] Silicon Labs. Silicon labs energy profiler, 2022. URL `https://docs.silabs.com/simplicity-studio-5-users-guide/1.0/using-the-tools/energy-profiler/#:~:text=Simplicity%20Studio%C2%AE%205%20(SSv5,power%20performance%20of%20these%20systems.`

[88] Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. Ahar: Adaptive cnn for energy-efficient human activity recognition in low-power edge devices. *IEEE Internet of Things Journal*, 2022.

[89] Judit Bort-Roig, Nicholas D Gilson, Anna Puig-Ribera, Ruth S Contreras, and Stewart G Trost. Measuring and influencing physical activity with smartphone technology: a systematic review. *Sports medicine*, 44(5):671–686, 2014.

[90] Alan K Bourke and Gerald M Lyons. A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. *Medical engineering & physics*, 30(1):84–90, 2008.

[91] Walter Maetzler, Jochen Klucken, and Malcolm Horne. A clinical view on the development of technology-based tools in managing parkinson's disease. *Movement Disorders*, 31(9):1263–1271, 2016.

[92] M. Dautta, A. Jimenez, K. K. H. Dia, N. Rashid, M. A. A. Faruque, and P. Tseng. Wireless qi-powered, multinodal and multisensory body area network for mobile health. *IEEE Internet of Things Journal*, pages 1–1, 2020. doi: 10.1109/JIOT.2020.3040713.

[93] M. Abdel-Basset, H. Hawash, V. Chang, R. K. Chakrabortty, and M. Ryan. Deep learning for heterogeneous human activity recognition in complex iot applications. *IEEE Internet of Things Journal*, pages 1–1, 2020. doi: 10.1109/JIOT.2020.3038416.

[94] Muhammad Shoaib, Stephan Bosch, Ozlem Durmaz Incel, Hans Scholten, and Paul JM Havinga. A survey of online activity recognition using mobile phones. *Sensors*, 15(1): 2059–2085, 2015.

[95] Oscar D Lara and Miguel A Labrador. A mobile platform for real-time human activity recognition. In *2012 IEEE consumer communications and networking conference (CCNC)*, pages 667–671. IEEE, 2012.

[96] Luis Miguel Soria Morillo, Luis Gonzalez-Abril, Juan Antonio Ortega Ramirez, De la Concepcion, and Miguel Angel Alvarez. Low energy physical activity recognition system on smartphones. *Sensors*, 15(3):5163–5196, 2015.

[97] Oscar D Lara and Miguel A Labrador. A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials*, 15(3):1192–1209, 2012.

[98] Adil Mehmood Khan, Young-Koo Lee, Sungyoung Y Lee, and Tae-Seong Kim. A triaxial accelerometer-based physical-activity recognition via augmented-signal features and a hierarchical recognizer. *IEEE transactions on information technology in biomedicine*, 14(5):1166–1172, 2010.

[99] Jiadong Zhu, Rubén San-Segundo, and José M Pardo. Feature extraction for robust physical activity recognition. *Human-centric Computing and Information Sciences*, 7 (1):16, 2017.

[100] Elliott Fullerton, Ben Heller, and Mario Munoz-Organero. Recognizing human activity in free-living using multiple body-worn accelerometers. *IEEE Sensors Journal*, 17(16): 5290–5297, 2017.

[101] Wen Qi, Hang Su, Chenguang Yang, Giancarlo Ferrigno, Elena De Momi, and Andrea Aliverti. A fast and robust deep convolutional neural networks for complex human activity recognition using smartphone. *Sensors*, 19(17):3731, 2019.

[102] Wenchao Jiang and Zhaozheng Yin. Human activity recognition using wearable sensors by deep convolutional neural networks. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1307–1310, 2015.

[103] Charissa Ann Ronao and Sung-Bae Cho. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert systems with applications*, 59:235–244, 2016.

[104] Francisco Javier Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, 2016.

[105] Emilio Sansano, Raúl Montoliu, and Óscar Belmonte Fernández. A study of deep neural networks for human activity recognition. *Computational Intelligence*, 2020.

[106] Eunji Kim. Interpretable and accurate convolutional neural networks for human activity recognition. *IEEE Transactions on Industrial Informatics*, 2020.

[107] F. Samie, L. Bauer, and J. Henkel. Hierarchical classification for constrained iot devices: A case study on human activity recognition. *IEEE Internet of Things Journal*, 7(9):8287–8295, 2020. doi: 10.1109/JIOT.2020.2989053.

[108] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.

[109] Ganapati Bhat, Yigit Tuncel, Sizhe An, Hyung Gyu Lee, and Umit Y Ogras. An ultra-low energy human activity recognition accelerator for wearable health applications. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22, 2019.

[110] Nafiul Rashid, Luke Chen, Manik Dautta, Abel Jimenez, Peter Tseng, and Mohammad Abdullah Al Faruque. Feature augmented hybrid cnn for stress recognition using wrist-based photoplethysmography sensor. In *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 2374–2377. IEEE, 2021.

[111] Ganapati Bhat, Nicholas Tran, Holly Shill, and Umit Y Ogras. w-har: An activity recognition dataset and framework using low-power wearable devices. *Sensors*, 20(18): 5356, 2020.

[112] Daniel Roggen, Alberto Calatroni, Mirco Rossi, Thomas Holleczek, Kilian Förster, Gerhard Tröster, Paul Lukowicz, David Bannach, Gerald Pirkl, Alois Ferscha, et al. Collecting complex activity datasets in highly rich networked sensor environments. In

2010 Seventh international conference on networked sensing systems (INSS), pages 233–240. IEEE, 2010.

[113] Mohanad Odema, Nafiul Rashid, and Mohammad Abdullah Al Faruque. Energy-aware design methodology for myocardial infarction detection on low-power wearable devices. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 621–626. IEEE, 2021.

[114] Mohanad Odema, Nafiul Rashid, and Mohammad Abdullah Al Faruque. Eexnas: Early-exit neural architecture search solutions for low-power wearable devices. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2021. doi: 10.1109/ISLPED52811.2021.9502503.

[115] Mohanad Odema, Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. Lens: Layer distribution enabled neural architecture search in edge-cloud hierarchies. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021.

[116] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[117] Silicon Labs. Efm32™ giant gecko 32-bit microcontroller, 2021. URL `https://www.silabs.com/mcu/32-bit/efm32-giant-gecko`.

[118] Eric A Finkelstein, Justin G Trogdon, Joel W Cohen, and William Dietz. Annual medical spending attributable to obesity: Payer-and service-specific estimates: Amid calls for health reform, real cost savings are more likely to be achieved through reducing obesity and related risk factors. *Health affairs*, 28(Suppl1):w822–w831, 2009.

[119] URL `https://www.stateofobesity.org/obesity-rates-trends-overview/`.

[120] Diana Sonntag, Shehzad Ali, and Freia De Bock. Lifetime indirect cost of childhood overweight and obesity: A decision analytic model. *Obesity*, 24(1):200–206, 2016.

[121] Tri Vu, Feng Lin, Nabil Alshurafa, and Wenyao Xu. Wearable food intake monitoring technologies: A comprehensive review. *Computers*, 6(1):4, 2017.

[122] Haik Kalantarian, Nabil Alshurafa, and Majid Sarrafzadeh. A survey of diet monitoring technology. *IEEE Pervasive Computing*, 16(1):57–65, 2017.

[123] Oliver Amft, Martin Kusserow, and Gerhard Troster. Bite weight prediction from acoustic recognition of chewing. *IEEE Transactions on Biomedical Engineering*, 56(6): 1663–1672, 2009.

[124] Oliver Amft and Gerhard Troster. Methods for detection and classification of normal swallowing from muscle activation and sound. In *2006 Pervasive Health Conference and Workshops*, pages 1–10. IEEE, 2006.

[125] Tauhidur Rahman, Alexander Travis Adams, Mi Zhang, Erin Cherry, Bobby Zhou, Huaishu Peng, and Tanzeem Choudhury. Bodybeat: a mobile system for sensing non-speech body sounds. In *MobiSys*, volume 14, pages 2–594, 2014.

[126] Edward S Sazonov and Juan M Fontana. A sensor system for automatic detection of food intake through non-invasive monitoring of chewing. *IEEE sensors journal*, 12(5): 1340–1348, 2011.

[127] Juan M Fontana, Muhammad Farooq, and Edward Sazonov. Automatic ingestion monitor: a novel wearable device for monitoring of ingestive behavior. *IEEE Transactions on Biomedical Engineering*, 61(6):1772–1779, 2014.

[128] Haik Kalantarian, Nabil Alshurafa, and Majid Sarrafzadeh. A wearable nutrition monitoring system. In *2014 11th international conference on wearable and implantable body sensor networks*, pages 75–80. IEEE, 2014.

[129] Haik Kalantarian, Nabil Alshurafa, Tuan Le, and Majid Sarrafzadeh. Monitoring eating habits using a piezoelectric sensor-based necklace. *Computers in biology and medicine*, 58:46–55, 2015.

[130] Nabil Alshurafa, Haik Kalantarian, Mohammad Pourhomayoun, Jason J Liu, Shruti Sarin, Behnam Shahbazi, and Majid Sarrafzadeh. Recognition of nutrition intake using time-frequency decomposition in a wearable necklace using a piezoelectric sensor. *IEEE sensors journal*, 15(7):3909–3916, 2015.

[131] Abdelkareem Bedri, Apoorva Verlekar, Edison Thomaz, Valerie Avva, and Thad Starner. A wearable system for detecting eating activities with proximity sensors in the outer ear. In *Proceedings of the 2015 ACM International Symposium on Wearable Computers*, pages 91–92, 2015.

[132] Oliver Amft and Gerhard Troster. On-body sensing solutions for automatic dietary monitoring. *IEEE pervasive computing*, 8(2):62–70, 2009.

[133] Shibo Zhang, William Stogin, and Nabil Alshurafa. I sense overeating: Motif-based machine learning framework to detect overeating using wrist-worn sensing. *Information Fusion*, 41:37–47, 2018.

[134] Edward Sazonov, Stephanie Schuckers, Paulo Lopez-Meyer, Oleksandr Makeyev, Nadezhda Sazonova, Edward L Melanson, and Michael Neuman. Non-invasive monitoring of chewing and swallowing for objective quantification of ingestive behavior. *Physiological measurement*, 29(5):525, 2008.

[135] Eli Cohen, William Stogin, Haik Kalantarian, Angela F Pfammatter, Bonnie Spring, and Nabil Alshurafa. Smartnecklace: Designing a wearable multi-sensor system for smart eating detection. In *Proceedings of the 11th EAI International Conference on Body Area Networks*, pages 33–37, 2016.

[136] Abdelkareem Bedri, Richard Li, Malcolm Haynes, Raj Prateek Kosaraju, Ishaan Grover, Temiloluwa Prioleau, Min Yan Beh, Mayank Goel, Thad Starner, and Gregory Abowd. Earbit: using wearable sensors to detect eating episodes in unconstrained environments. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, 1(3):1–20, 2017.

[137] Shibo Zhang, Dzung Nguyen, Zachary King, Jishnu Pradeep, and Nabil Alshurafa. Habits necklace: A neck-worn sensor that captures eating related behavior and more. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, pages 484–487, 2018.

[138] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16, 2012.

[139] Mohammad Abdullah Al Faruque and Korosh Vatanparvar. Energy management-as-a-service over fog computing platform. *IEEE internet of things journal*, 3(2):161–169, 2015.

[140] Mung Chiang and Tao Zhang. Fog and iot: An overview of research opportunities. *IEEE Internet of things journal*, 3(6):854–864, 2016.

[141] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

[142] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23:69–101, 1996.

[143] Jyrki Kivinen, Alexander J Smola, and Robert C Williamson. Online learning with kernels. *IEEE transactions on signal processing*, 52(8):2165–2176, 2004.

[144] Ganapati Bhat, Ranadeep Deb, Vatika Vardhan Chaurasia, Holly Shill, and Umit Y Ogras. Online human activity recognition using low-power wearable devices. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. ACM, 2018.

[145] Rawan Alharbi, Tammy Stump, Nilofar Vafaie, Angela Pfammatter, Bonnie Spring, and Nabil Alshurafa. I can't be myself: effects of wearable cameras on the capture of authentic behavior in the wild. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, 2(3):1–40, 2018.

[146] Mark A Hall and Lloyd A Smith. Feature selection for machine learning: comparing a correlation-based filter approach to the wrapper. In *FLAIRS conference*, volume 1999, pages 235–239, 1999.

[147] Maximilian Christ, Andreas W Kempa-Liehr, and Michael Feindt. Distributed and parallel time series feature extraction for industrial big data applications. *arXiv preprint arXiv:1610.07717*, 2016.

[148] Geoffrey E Hinton. Connectionist learning procedures. In *Machine learning*, pages 555–610. Elsevier, 1990.

[149] Russell Stuart and P Novig. Artificial intelligence: A modern approach, (2016). *doi*, 10:363.

[150] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[151] J. Ross Quinlan. Simplifying decision trees. *International journal of man-machine studies*, 27(3):221–234, 1987.

[152] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[153] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[154] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[155] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20: 273–297, 1995.

[156] James M Keller, Michael R Gray, and James A Givens. A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4):580–585, 1985.

[157] Erin M Wilson, Jordan R Green, and Gary Weismer. A kinematic description of the temporal characteristics of jaw motion for early chewing: Preliminary findings. 2012.

[158] TAT Hughes and CM Wiles. Clinical measurement of swallowing in health and in neurogenic dysphagia. *QJM: An International Journal of Medicine*, 89(2):109–116, 1996.

[159] Nafiul Rashid, Trier Mortlock, and Mohammad Abdullah Al Faruque. Self-care: Selective fusion with context-aware low-power edge computing for stress detection. In *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 49–52. IEEE, 2022.

[160] Nafiul Rashid, Trier Mortlock, and Mohammad Abdullah Al Faruque. Stress detection using context-aware sensor fusion from wearable devices. *arXiv preprint*, 2023.

[161] Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. Ahar: Adaptive cnn for energy-efficient human activity recognition in low-power edge devices. *IEEE Internet of Things Journal*, pages 1–1, 2022. doi: 10.1109/JIOT.2022.3140465.

[162] Nafiul Rashid, Manik Dautta, Peter Tseng, and Mohammad Abdullah Al Faruque. Hear: Fog-enabled energy-aware online human eating activity recognition. *IEEE Internet of Things Journal*, 8(2):860–868, 2021. doi: 10.1109/JIOT.2020.3008842.

[163] American Psychological Association. Stress in America 2020: a national mental health crisis, 2020. URL `https://www.apa.org/news/press/releases/stress/2020/sia-mental-health-crisis.pdf`.

[164] David S Goldstein. Adrenal responses to stress. *Cellular and molecular neurobiology*, 30(8):1433–1440, 2010.

[165] Kenneth Lai, Svetlana N Yanushkevich, and Vlad P Shmerko. Intelligent stress monitoring assistant for first responders. *IEEE Access*, 9:25314–25329, 2021.

[166] Rosalind W Picard. *Affective computing*. MIT press, 2000.

[167] Markets and Markets. Affective computing market, 2020. URL `https://www.marketsandmarkets.com/Market-Reports/affective-computing-market-130730395.html`.

[168] Jonghwa Kim and Elisabeth André. Emotion recognition based on physiological changes in music listening. *IEEE transactions on pattern analysis and machine intelligence*, 30(12):2067–2083, 2008.

[169] Martin Gjoreski, Hristijan Gjoreski, Mitja Luštrek, and Matjaž Gams. Continuous stress detection using a wrist device: in laboratory and real life. In *proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing: Adjunct*, pages 1185–1193, 2016.

[170] Karen Hovsepian, Mustafa Al'Absi, Emre Ertin, Thomas Kamarck, Motohiro Nakajima, and Santosh Kumar. cstress: towards a gold standard for continuous stress assessment in the mobile environment. In *Proc. of the ACM Int. joint conference on pervasive and ubiquitous computing*, pages 493–504, 2015.

[171] Jennifer A Healey and Rosalind W Picard. Detecting stress during real-world driving tasks using physiological sensors. *IEEE Trans. on intelligent transportation systems*, 6(2):156–166, 2005.

[172] Rosalind W. Picard, Elias Vyzas, and Jennifer Healey. Toward machine emotional intelligence: Analysis of affective physiological state. *IEEE Trans. on pattern analysis and machine intelligence*, 23(10):1175–1191, 2001.

[173] Sander Koelstra, Christian Muhl, Mohammad Soleymani, Jong-Seok Lee, Ashkan Yazdani, Touradj Ebrahimi, Thierry Pun, Anton Nijholt, and Ioannis Patras. Deap: A database for emotion analysis; using physiological signals. *IEEE transactions on affective computing*, 3(1):18–31, 2011.

[174] Philip Schmidt, Attila Reiss, Robert Duerichen, Claus Marberger, and Kristof Van Laerhoven. Introducing wesad, a multimodal dataset for wearable stress and affect detection. In *Proceedings of the 20th ACM international conference on multimodal interaction*, pages 400–408, 2018.

[175] Sirat Samyoun, Abu Sayeed Mondol, and John A Stankovic. Stress detection via sensor translation. In *2020 16th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 19–26. IEEE, 2020.

[176] Lam Huynh, Tri Nguyen, Thu Nguyen, Susanna Pirttikangas, and Pekka Siirtola. Stressnas: Affect state and stress detection using neural architecture search. In *Adj. Proc. of the 2021 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proc. of the 2021 ACM International Symposium on Wearable Computers*, pages 121–125, 2021.

[177] Abhijith Ragav, Nanda Harishankar Krishna, Naveen Narayanan, Kevin Thelly, and Vineeth Vijayaraghavan. Scalable deep learning for stress and affect detection on resource-constrained devices. In *2019 18th IEEE Int. Conference On Machine Learning And Applications (ICMLA)*, pages 1585–1592. IEEE, 2019.

[178] Emad Kasaeyan Naeini, Sina Shahhosseini, Anil Kanduri, Pasi Liljeberg, Amir M Rahmani, and Nikil Dutt. Amser: Adaptive multi-modal sensing for energy efficient and resilient ehealth systems. *arXiv preprint arXiv:2112.08176*, 2021.

[179] Patrícia Bota, Chen Wang, Ana Fred, and Hugo Silva. Emotion assessment using feature fusion and decision fusion classification based on physiological data: Are we there yet? *Sensors*, 20(17):4723, 2020.

[180] Arnav Vaibhav Malawade, Trier Mortlock, and Mohammad Abdullah Al Faruque. Hydrafusion: Context-aware selective sensor fusion for robust and efficient autonomous vehicle perception. *arXiv preprint arXiv:2201.06644*, 2022.

[181] Jionghao Lin, Shirui Pan, Cheng Siong Lee, and Sharon Oviatt. An explainable deep fusion network for affect recognition using physiological signals. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2069–2072, 2019.

[182] Nazanin Fouladgar, Marjan Alirezaie, and Kary Främling. Cn-waterfall: a deep convolutional neural network for multimodal physiological affect detection. *Neural Computing and Applications*, pages 1–20, 2021.

[183] Sharon Oviatt, Björn Schuller, Philip Cohen, Daniel Sonntag, Gerasimos Potamianos, and Antonio Krüger. *The handbook of multimodal-multisensor interfaces, Volume 2: Signal processing, architectures, and detection of emotion and cognition*. Morgan & Claypool, 2018.

[184] MaoSong Yan, Zhen Deng, BingWei He, ChengSheng Zou, Jie Wu, and ZhaoJu Zhu. Emotion classification with multichannel physiological signals using hybrid feature and adaptive decision fusion. *Biomedical Signal Processing and Control*, 71:103235, 2022.

[185] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[186] Arjun Pakrashi and Brian Mac Namee. Kalman filter-based heuristic ensemble (kfhe): A new perspective on multi-class ensemble classification using kalman filters. *Information Sciences*, 485:456–485, 2019.

[187] Seyed Salehizadeh, Duy Dao, Jeffrey Bolkhovsky, Chae Cho, Yitzhak Mendelson, and Ki H Chon. A novel time-varying spectral filtering algorithm for reconstruction of motion artifact corrupted heart rate signals during intense physical activities using a wearable photoplethysmogram sensor. *Sensors*, 16(1):10, 2016.

[188] Berken Utku Demirel, Ivan Skelin, Haoxin Zhang, Jack J Lin, and Mohammad Abdullah Al Faruque. Single-channel eeg based arousal level estimation using multitaper spectrum estimation at low-power wearable devices. In *2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 542–545. IEEE, 2021.

[189] Wenrui Lin, Berken Utku Demirel, Mohammad Abdullah Al Faruque, and GP Li. Energy-efficient blood pressure monitoring based on single-site photoplethysmogram on wearable devices. In *2021 43rd annual international conference of the IEEE engineering in medicine & biology society (EMBC)*, pages 504–507. IEEE, 2021.

[190] Berken Utku Demirel, Luke Chen, and Mohammad Al Faruque. Neural contextual bandits based dynamic sensor selection for low-power body-area networks. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 1–6, 2022.

[191] Floranne Tavailau Ellington, Berken Demirel, Daniel Jilani, MAA Faruque, and Hung Cao. Edge-based real-time fetal electrocardiography monitoring in the home setting. *Computing in cardiology*, 2022.

[192] Mojtaba Taherisadr, Mohammad Abdullah Al Faruque, and Salma Elmalaki. Erudite: Human-in-the-loop iot for an adaptive personalized learning system. *arXiv preprint arXiv:2303.04292*, 2023.

[193] Rozhin Yasaei, Felix Hernandez, and Mohammad Abdullah Al Faruque. Iot-cad: Context-aware adaptive anomaly detection in iot systems through sensor association. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–9, 2020.

[194] Arnav Malawade, Mohanad Odema, Sebastien Lajeunesse-DeGroot, and Mohammad Abdullah Al Faruque. Sage: A split-architecture methodology for efficient end-to-end autonomous vehicle control. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s):1–22, 2021.

[195] Arnav Vaibhav Malawade, Trier Mortlock, and Mohammad Abdullah Al Faruque. Ecofusion: Energy-aware adaptive sensor fusion for efficient autonomous vehicle perception. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 481–486, 2022.

[196] Luke Chen, Mohanad Odema, and Mohammad Abdullah Al Faruque. Romanus: Robust task offloading in modular multi-sensor autonomous driving systems. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–8, 2022.

[197] Mohanad Odema, Luke Chen, Marco Levorato, and Mohammad Abdullah Al Faruque. Testudo: Collaborative intelligence for latency-critical autonomous systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[198] Mohanad Odema, James Ferlez, Yasser Shoukry, and Mohammad Abdullah Al Faruque. Seo: Safety-aware energy optimization framework for multi-sensor neural controllers at the edge. *arXiv preprint arXiv:2302.12493*, 2023.

[199] Halima Bouzidi, Mohanad Odema, Hamza Ouarnoughi, Smail Niar, and Mohammad Abdullah Al Faruque. Map-and-conquer: Energy-efficient mapping of dynamic neural nets onto heterogeneous mpsocs. *arXiv preprint arXiv:2302.12926*, 2023.

[200] Halima Bouzidi, Mohanad Odema, Hamza Ouarnoughi, Mohammad Abdullah Al Faruque, and Smail Niar. Hadas: Hardware-aware dynamic neural architecture search for edge performance scaling. *arXiv preprint arXiv:2212.03354*, 2022.

[201] Mohanad Odema, James Ferlez, Goli Vaisi, Yasser Shoukry, and Mohammad Abdullah Al Faruque. Energyshield: Provably-safe offloading of neural network controllers for energy efficiency. *arXiv preprint arXiv:2302.06572*, 2023.

[202] Sujit Rokka Chhetri, Nafiul Rashid, Sina Faezi, and Mohammad Abdullah Al Faruque. Security trends and advances in manufacturing systems in the era of industry 4.0. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1039–1046. IEEE, 2017.

[203] Sujit Rokka Chhetri, Sina Faezi, Nafiul Rashid, and Mohammad Abdullah Al Faruque. Manufacturing supply chain and product lifecycle security in the era of industry 4.0. *Journal of Hardware and Systems Security*, 2:51–68, 2018.

[204] Nafiul Rashid, Jiang Wan, Gustavo Quiros, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Modeling and simulation of cyberattacks for resilient cyber-physical systems. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 988–993. IEEE, 2017.

[205] Nafiul Rashid, Gustavo Quirós, and Mohammad Abdullah Al Faruque. A survivability-aware cyber-physical systems design methodology. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 848–853. IEEE, 2019.

[206] Manik Dautta, Abel Jimenez, Kazi Khurshidi Haque Dia, Nafiul Rashid, Mohammad Abdullah Al Faruque, and Peter Tseng. Wireless qi-powered, multinodal and multisensory body area network for mobile health. *IEEE internet of things journal*, 8 (9):7600–7609, 2020.

[207] Jiang Wan, Nafiul Rashid, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Concept design: Modeling and synthesis from requirements to functional models and simulation. *Design Automation of Cyber-Physical Systems*, pages 3–20, 2019.

[208] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[209] Bradley McDanel, Surat Teerapittayanon, and HT Kung. Embedded binarized neural networks. *arXiv preprint arXiv:1709.02260*, 2017.

[210] Srinivasan Murali, Francisco Rincon, and David Atienza. A wearable device for physical and emotional health monitoring. In *2015 Computing in Cardiology Conference (CinC)*, pages 121–124. IEEE, 2015.

[211] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20:55:1–55:21, 2019.

[212] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[213] Kirthevasan Kandasamy et al. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *Journal of Machine Learning Research*, 21(81):1–27, 2020.

[214] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

[215] Efm32 leopard gecko—silicon labs, 2020. URL `https://www.silabs.com/products/mcu/32-bit/efm32-leopard-gecko`.

[216] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

[217] Abdulrahman Alarifi, AbdulMalik Al-Salman, Mansour Alsaleh, Ahmad Alnafessah, Suheer Al-Hadhrami, Mai A Al-Ammar, and Hend S Al-Khalifa. Ultra wideband indoor positioning technologies: Analysis and recent advances. *Sensors*, 16(5):707, 2016.