

Lightweight Neural Networks for Human Activity Recognition

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

M.Sc. Yexu Zhou

Tag der mündlichen Prüfung: 25. November 2024

1. Referent: Prof. Dr. Michael Beigl
2. Referent: Prof. Dr. Kristof Van Laerhoven



This document is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0): <https://creativecommons.org/licenses/by-sa/4.0/deed.en>

Acknowledgements

I would like to express my deepest and most sincere gratitude to my primary advisor, Prof. Dr. Michael Beigl. His unwavering support, insightful guidance, and patient mentorship have been the cornerstones of my journey through this dissertation. From the very beginning, he listened attentively to my ideas, providing direction when I needed it most, yet always granting me the freedom to explore my own path. His trust in my abilities allowed me to grow not only as a researcher but also as an individual. I cannot imagine having had a better mentor and advisor during my PhD journey, and for that, I am forever grateful.

My heartfelt thanks also go to my second advisor, Prof. Dr. Kristof van Laerhoven of Siegen University, for his steady companionship throughout this process. His research insights sparked countless ideas that inspired the direction of my work.

I owe special thanks to Dr. Till Riedel, whose intellectual curiosity and vibrant discussions fueled much of my scientific exploration. It has been a privilege to work alongside him on numerous projects, where I not only honed my research skills but also gained invaluable insights into practical applications that extend beyond academia.

I am equally grateful to my colleagues at KIT – Dr. Haibin Zhao, Yiran Huang, Dr. Tobias Röddiger, Chaofan Li, Dr. Michael Hefenbrock, Dr. Matthias Budde, Tobias King, Alexander Studt, Yunus Bulut, and Dr. Paul Tremper. Our collaborations, thoughtful discussions, and moments of shared work and rest made my time at TECO truly unforgettable. The friendships and experiences we built together will always remain close to my heart.

To Melissa Alpman, Zina Tsiouma, Denise Hillmann, and Helga Scherer, thank you for your tireless support in all administrative matters. Your assistance has been invaluable and greatly appreciated.

I would also like to extend my gratitude to Dr. Markus Scholz, Vincent

Schnitzbauer, Sascha Rudolph, and Adnan Khan for the opportunity to work together on an exciting vision-based project. The journey was both enriching and rewarding, and I am thankful for our collaboration.

Lastly, and most importantly, I want to thank my beloved wife, Mengting Lu, and my parents, Shijie Zhou and Heli Ye. Their boundless love, unwavering support, and unshakable belief in me have carried me through every challenge and triumph. Without their constant encouragement, none of this would have been possible. To my wife, who stood by me through every late night and every moment of doubt – words cannot express how much your presence has meant to me.

Abstract

Wearable Human Activity Recognition (HAR) offers numerous advantages, such as real-time functionality, enhanced data privacy, and reduced dependency on environmental infrastructure. With the rapid advancement of sensor technology, various sensors—including accelerometers, gyroscopes, magnetometers, oxygen saturation sensors, heart rate monitors, and electrooculography (EOG)—have been integrated into devices like smartphones, earphones, and smart glasses. These technological developments have significantly expanded the applications of wearable HAR systems, including healthcare monitoring, fitness coaching, gaming, gesture control, and more.

The core of a wearable HAR system is the machine learning model that classifies activity based on sensor data. The performance of such systems is heavily reliant on the quality of these models. While deep learning models have demonstrated superior performance in HAR tasks, they are often characterized by large memory footprints and high computational demands, making them unsuitable for real-time deployment on resource-constrained devices. As of 2021, research on lightweight deep learning models for HAR remains limited, and many wearable applications struggle to deploy advanced models on edge devices. Some efforts have aimed to reduce model size but frequently at the cost of classification performance.

Motivated by these challenges, this dissertation focuses on developing lightweight neural network models for wearable HAR systems, aiming to bridge the gap between high-performance models and practical deployment on edge devices such as micro-controller units (MCUs), without sacrificing performance or with minimal degradation.

The development of a wearable HAR system, from data collection to model deployment, typically follows four key stages: data preparation, model design, model optimization, and quantization for deployment. After analyzing the

challenges and characteristics of HAR tasks, as well as the limitations of neural networks—particularly in capturing frequency-domain information—this dissertation explores various methods to reduce model size while maintaining high performance.

In the **data preparation** stage, we addressed challenges related to inter-class similarity, intra-class variability, and the multi-modal nature of HAR tasks. Data augmentation techniques were employed to enrich the dataset, improving model generalization and mitigating the performance degradation typically associated with smaller models. Additionally, we experimented with transforming time-series data into frequency-domain representations. To manage the additional hyper-parameters and computational complexity introduced by this approach, we proposed a learnable wavelet layer combined with pruning techniques, improving the model’s ability to capture frequency-domain features while reducing its size.

In the **model design** stage, we established five key principles for designing lightweight HAR models based on the characteristics of HAR tasks and neural network operators. Following these principles, we developed **TinyHAR**, a landmark model. Recognizing the difficulty of neural networks in capturing high-frequency information, we further proposed a dual-branch model, **Cross-Atten**, which leverages both time-series and spectrogram representations. While both models achieved state-of-the-art (SOTA) performance with lightweight architectures, they did not fully address real-time requirements and hardware constraints for edge devices. To overcome these limitations, we developed a fully connected layer-based model, **MLP-HAR**, which achieved excellent classification performance, fast inference times, and minimal memory consumption, positioning it as one of the most efficient lightweight HAR models to date.

Manual model design requires significant expert knowledge, posing a barrier for non-expert users in optimizing models for specific application scenarios. To address this, we integrated automated machine learning (AutoML) techniques into the model optimization process with the **MicroNAS** framework. This framework optimizes not only the classification performance but also accounts for hardware constraints and real-time inference requirements. Addi-

tionally, we introduced a novel framework, **NAS meets Pruning (SFTNAS)**, which not only optimizes model structure but also automatically selects the most critical sensor channels. SFTNAS has demonstrated remarkable ability in reducing model complexity while offering valuable insights into the design of manually constructed models.

Through extensive experimentation on benchmark datasets, the proposed methods consistently yielded HAR models that achieved SOTA performance while maintaining low memory usage and fast inference times, making them suitable for real-time applications. These models were successfully deployed on micro-controllers and smartwatches, where they exhibited both SOTA classification accuracy and fast inference capabilities.

In summary, this dissertation addresses the critical challenge of developing lightweight yet high-performance neural network models for wearable HAR systems. By thoroughly examining the unique characteristics of HAR tasks and the limitations of current deep learning models, we propose a series of innovations. Our work demonstrates that it is possible to significantly reduce model size without sacrificing classification accuracy by employing techniques such as automated data augmentation, frequency-domain transformations, and novel architectures like **TinyHAR**, **Cross-Atten**, and **MLP-HAR**. Furthermore, by introducing AutoML techniques, we automate the search for optimal architectures while considering hardware constraints and real-time performance requirements. This dissertation advances the field of wearable HAR, offering practical solutions for deploying efficient, high-performance models on resource-constrained devices.

Zusammenfassung

Tragbare Human Activity Recognition (HAR) bietet zahlreiche Vorteile, wie z.B. Echtzeit-Funktionalität, verbesserten Datenschutz und eine geringere Abhängigkeit von der Infrastruktur der Umgebung. Mit dem raschen Fortschritt in der Sensortechnologie wurden verschiedene Sensoren – darunter Beschleunigungsmesser, Gyroskope, Magnetometer, Sauerstoffsättigungssensoren, Herzfrequenzmonitore und Elektrookulografie (EOG) – in Geräte wie Smartphones, Ohrhörer und Smart Glasses integriert. Diese technologischen Entwicklungen haben die Anwendungsbereiche von tragbaren HAR-Systemen erheblich erweitert, einschließlich Gesundheitsüberwachung, Fitness-Coaching, Gaming, Gestensteuerung und mehr.

Das Herzstück eines tragbaren HAR-Systems ist das maschinelle Lernmodell, das Aktivitäten basierend auf Sensordaten klassifiziert. Die Leistungsfähigkeit solcher Systeme hängt stark von der Qualität dieser Modelle ab. Obwohl Deep-Learning-Modelle bei HAR-Aufgaben überlegene Leistungen gezeigt haben, zeichnen sie sich oft durch einen großen Speicherbedarf und hohe Rechenanforderungen aus, was sie für den Echtzeit-Einsatz auf ressourcenbeschränkten Geräten ungeeignet macht. Seit 2021 ist die Forschung zu leichtgewichtigen Deep-Learning-Modellen für HAR begrenzt, und viele tragbare Anwendungen kämpfen mit der Implementierung fortschrittlicher Modelle auf Edge-Geräten. Einige Ansätze zielen darauf ab, die Modellgröße zu reduzieren, jedoch häufig auf Kosten der Klassifikationsgenauigkeit.

Angesichts dieser Herausforderungen konzentriert sich diese Dissertation auf die Entwicklung leichtgewichtiger neuronaler Netzmodelle für tragbare HAR-Systeme, mit dem Ziel, die Lücke zwischen leistungsstarken Modellen und deren praktischer Implementierung auf Edge-Geräten wie Mikrocontrollern (MCUs) zu schließen, ohne dabei auf Leistung zu verzichten oder mit minimaler Leistungseinbuße.

Die Entwicklung eines tragbaren HAR-Systems, von der Datenerfassung bis zur Modellimplementierung, folgt typischerweise vier Schüsselschritten: Datenaufbereitung, Modellentwurf, Modelloptimierung und Quantisierung zur Implementierung. Nach der Analyse der Herausforderungen und Eigenschaften von HAR-Aufgaben sowie der Einschränkungen neuronaler Netze – insbesondere bei der Erfassung von Frequenzbereichsinformationen – untersucht diese Dissertation verschiedene Methoden zur Reduzierung der Modellgröße bei gleichbleibend hoher Leistung.

Im **Datenaufbereitungsprozess** haben wir uns mit Herausforderungen wie interklassenähnlichkeit, intraklassenvariabilität und der multimodalen Natur von HAR-Aufgaben auseinandergesetzt. Zur Verbesserung der Generalisierungsfähigkeit des Modells und zur Minderung der typischen Leistungsverschlechterung bei kleineren Modellen wurden Datenaugmentationstechniken eingesetzt. Darüber hinaus experimentierten wir mit der Transformation von Zeitreihendaten in Frequenzbereichsdarstellungen. Um die durch diesen Ansatz eingeführte zusätzliche Hyperparameter- und Rechenkomplexität zu bewältigen, haben wir eine lernbare Wavelet-Schicht in Kombination mit Pruning-Techniken vorgeschlagen, die die Fähigkeit des Modells, Frequenzbereichsmerkmale zu erfassen, verbessert und gleichzeitig die Größe des Modells reduziert.

Im **Modellentwurfsprozess** haben wir fünf Schlüsselprinzipien für den Entwurf leichtgewichtiger HAR-Modelle auf der Grundlage der Eigenschaften von HAR-Aufgaben und neuronalen Netzoperatoren festgelegt. Nach diesen Prinzipien entwickelten wir **TinyHAR**, ein wegweisendes Modell. Angesichts der Schwierigkeit neuronaler Netze, Hochfrequenzinformationen zu erfassen, schlugen wir zusätzlich ein Dual-Branch-Modell, **Cross-Atten**, vor, das sowohl Zeitreihen- als auch Spektrogramm-Darstellungen nutzt. Während beide Modelle mit leichtgewichtigen Architekturen eine State-of-the-Art (SOTA)-Leistung erzielten, erfüllten sie nicht vollständig die Echtzeitanforderungen und Hardwareeinschränkungen für Edge-Geräte. Um diese Einschränkungen zu überwinden, entwickelten wir ein Modell auf Basis vollständig vernetzter Schichten, **MLP-HAR**, das ausgezeichnete Klassifikationsleistungen, schnelle Inferenzzeiten und einen minimalen Speicherverbrauch erzielte und somit als eines der effizientesten leichtgewichtigen HAR-Modelle gilt.

Das manuelle Design von Modellen erfordert erhebliche Fachkenntnisse und stellt eine Hürde für nicht-Experten bei der Optimierung von Modellen für spezifische Anwendungsfälle dar. Um dem entgegenzuwirken, haben wir automatisierte maschinelle Lerntechniken (AutoML) in den Modelloptimierungsprozess integriert, insbesondere mit dem **MicroNAS**-Framework. Dieses Framework optimiert nicht nur die Klassifikationsleistung, sondern berücksichtigt auch Hardwareeinschränkungen und Echtzeitanforderungen. Zusätzlich führten wir ein neuartiges Framework, **NAS meets Pruning (SFTNAS)**, ein, das nicht nur die Modellstruktur optimiert, sondern auch automatisch die kritischsten Sensor-Kanäle auswählt. SFTNAS hat eine bemerkenswerte Fähigkeit zur Reduzierung der Modellkomplexität gezeigt und gleichzeitig wertvolle Einblicke in das Design manuell konstruierter Modelle geliefert.

Durch umfangreiche Experimente an Benchmark-Datensätzen zeigten die vorgeschlagenen Methoden durchweg HAR-Modelle, die SOTA-Leistung erzielten, während sie gleichzeitig einen geringen Speicherverbrauch und schnelle Inferenzzeiten aufwiesen, was sie für Echtzeitanwendungen geeignet macht. Diese Modelle wurden erfolgreich auf Mikrocontrollern und Smartwatches implementiert, wo sie sowohl SOTA-Klassifikationsgenauigkeit als auch schnelle Inferenzfähigkeiten zeigten.

Zusammenfassend behandelt diese Dissertation die entscheidende Herausforderung, leichtgewichtige und dennoch leistungsstarke neuronale Netzmodelle für tragbare HAR-Systeme zu entwickeln. Durch die gründliche Untersuchung der einzigartigen Eigenschaften von HAR-Aufgaben und der Einschränkungen aktueller Deep-Learning-Modelle schlagen wir eine Reihe von Innovationen vor. Unsere Arbeit zeigt, dass es möglich ist, die Modellgröße erheblich zu reduzieren, ohne die Klassifikationsgenauigkeit zu beeinträchtigen, indem Techniken wie automatisierte Datenaugmentation, Frequenzbereichstransformationen und neuartige Architekturen wie **TinyHAR**, **Cross-Atten** und **MLP-HAR** eingesetzt werden. Darüber hinaus automatisieren wir durch die Einführung von AutoML-Techniken die Suche nach optimalen Architekturen unter Berücksichtigung von Hardwareeinschränkungen und Echtzeitleistungsanforderungen. Diese Dissertation trägt zur Weiterentwicklung des tragbaren HAR bei und bietet praktische Lösungen zur Implementierung effizien-

ter, leistungsstarker Modelle auf ressourcenbeschränkten Geräten.

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem and Motivation	2
1.3	Aim and Objectives	3
1.4	Research Hypothesis and Contributions	4
1.4.1	HAR Data Preparation	4
1.4.2	HAR Network Architecture Design	6
1.4.3	Automatically Network Architecture Optimization	7
1.4.4	Hardware-Aware Model Optimization	7
1.5	Structure of This Thesis	8
1.6	List of Publications	11
2	Fundamentals	15
2.1	Problem Definition	15
2.2	Challenges and Characteristics of HAR Task	16
2.2.1	Challenges	16
2.2.2	Characteristics	17
2.2.3	Discussion	18
2.3	Taxonomy of Human Activity Recognition Models	19
2.3.1	Convolutional Neural Networks	19
2.3.2	Recurrent Neural Networks	21
2.3.3	Attention-based Networks	22
2.3.4	Hybrid Networks	23
2.3.5	Lightweight HAR Models	24
2.3.6	Discussion	25
2.4	Experimental Setup	25
2.4.1	Datasets	25

2.4.2	Evaluation Protocol	29
2.4.3	Training Procedures	31
3	Data Preparation	33
3.1	Related Works	33
3.1.1	Data Augmentation	34
3.1.1.1	Traditional Approaches	35
3.1.1.2	Advanced Approaches	36
3.1.1.3	Challenges	37
3.1.2	Data Transformation	38
3.1.2.1	Short-Time Fourier Transform	39
3.1.2.2	Wavelet Transformation	40
3.1.2.3	Challenges	42
3.2	Data Augmentation AutoAugHAR	43
3.2.1	Preliminaries	43
3.2.1.1	Background	43
3.2.1.2	Naive Approach	44
3.2.1.3	Data Augmentation Optimization	44
3.2.2	Methodology	45
3.2.2.1	Overview of AutoAugHAR	45
3.2.2.2	Gradient Based Optimization	49
3.2.2.3	Candidate Operations And Search Space	53
3.2.3	Experiments and Discussions	58
3.2.3.1	Experiment Setup	58
3.2.3.2	Comparison to State-of-the-art	60
3.2.3.3	Model Compression	65
3.2.3.4	Ablation Study	66
3.2.3.5	Training Overhead	70
3.2.4	Discussion	71
3.3	Learnable Data Transformation	72
3.3.1	Methodology	73
3.3.1.1	Wavelet Filters	73
3.3.1.2	Learnable Wavelets	76
3.3.1.3	Filter Pruning	77

3.3.2	Experiments and Discussions	78
3.3.2.1	Experiment Setup	78
3.3.2.2	Comparison to State-of-the-art	78
3.3.3	Discussion	80
3.4	Summary	81
4	Model Architecture Design	83
4.1	Related Work	83
4.2	TinyHAR	85
4.2.1	Practical Guidelines for Efficient HAR Model Design .	85
4.2.2	Methodology	87
4.2.2.1	Individual Convolutional Subnet	87
4.2.2.2	Cross-Channel Info Interaction	87
4.2.2.3	Cross-Channel Info Fusion	89
4.2.2.4	Global Temporal Info Extraction	90
4.2.2.5	Global Temporal Info Enhancement	91
4.2.3	Experiments and Discussions	92
4.2.3.1	Experiment Setup	92
4.2.3.2	Comparison to State-of-the-art	92
4.2.4	Discussion	95
4.3	Cross-Attention with Multi-Representation	96
4.3.1	Methodology	96
4.3.1.1	Data Transformation	98
4.3.1.2	Individual Embedding Module	98
4.3.1.3	Attention Module	99
4.3.1.4	Prediction Module	101
4.3.2	Experiments and Discussions	102
4.3.2.1	Experimental Setup	102
4.3.2.2	Comparison to State-of-the-art	103
4.3.2.3	Ablation Study	108
4.3.2.4	Generalization Performance on UEA Datasets	110
4.3.3	Discussion	111

4.4	MLP-HAR	111
4.4.1	Methodology	112
4.4.1.1	Data Embedding Module	112
4.4.1.2	Mixer Module	114
4.4.1.3	Prediction Module	115
4.4.2	Experiments and Discussions	115
4.4.2.1	Experiment Setup	115
4.4.2.2	Comparison to State-of-the-art	117
4.4.2.3	Post-Processing Experiment	120
4.4.2.4	Ablation Study and Parameter Analysis	121
4.4.2.5	Deployment on Hardware	123
4.4.3	Discussion	125
4.5	Summary	125
5	Model Architecture Optimization	127
5.1	Related Work	127
5.2	AutoML Framework : ECLSTM	128
5.2.1	Methodology	128
5.2.1.1	Embedded Convolutional LSTM	128
5.2.1.2	Automatic Prediction Framework	133
5.2.1.3	Temporal Feature Extraction Stage	134
5.2.1.4	Prediction Stage	134
5.2.1.5	Hyper-parameter Optimization	135
5.2.2	Experiments and Discussions	135
5.2.2.1	Experiment Setup	135
5.2.2.2	Comparison to State-of-the-art	136
5.3	Summary	137
6	Hardware-aware Model Architecture Optimization	139
6.1	Related Works	140
6.1.1	Neural Architecture Search	140
6.1.2	Hardware-aware Neural Architecture Search	141
6.1.3	Neural Architecture Search for HAR	142

6.2	MicroNAS	143
6.2.1	Methodology	143
6.2.1.1	Search Space	144
6.2.1.2	Dynamic Convolutions	148
6.2.1.3	Latency & Peak Memory Estimation	149
6.2.1.4	Optimization	151
6.2.2	Experiments and Discussions	153
6.2.2.1	Experiment Setup	153
6.2.2.2	Latency Prediction Experiment	154
6.2.2.3	Latency vs. Performance	155
6.2.2.4	Peak Memory vs. Performance	156
6.2.2.5	Comparison to Random Search	158
6.2.3	Discussion	159
6.3	NAS Meets Pruning	160
6.3.1	HAR Model Complexity and Size	160
6.3.2	Methodology	164
6.3.2.1	Sensor and Convolutional Channel Pruning	164
6.3.2.2	Search Space	165
6.3.2.3	Optimization	167
6.3.3	Experiments and Discussions	168
6.3.3.1	Experiment Setup	168
6.3.3.2	Comparison to State-of-the-art	169
6.3.3.3	Performance on Hardware	170
6.3.4	Discussion	172
6.4	Summary	173
7	Conclusion	175

List of Figures

1.1	An illustration of sensor-based HAR system.	2
1.2	Structure of the dissertation focusing on lightweight neural networks for HAR tasks, with Chapter 1 (Introduction) excluded. Abbreviations: AutoAugHAR refers to an automated augmentation framework for HAR; MLP-HAR stands for Multi-Layer Perceptron for HAR; AutoML denotes automated machine learning; ECLSTM represents Embedded Convolutional Long Short-Term Memory; NAS stands for Neural Architecture Search. . .	9
2.1	This figure provides an overview of wearable HAR systems, with a focus on designing lightweight deep learning models for HAR tasks.	16
2.2	1D convolution vs 2D convolution	19
2.3	This figure illustrates various CNN-based fusion strategies. The differently colored branches represent distinct weights, indicating that there is no weight sharing between them.	21
2.4	This figure shows the sensor placement locations for each dataset. In this case, sensor modality is categorized by placement. The body is divided into four regions: head, torso, upper limbs, and lower limbs, each represented by a different color.	30
3.1	New signal channels are generated using the Differencing Time Series (DTS) method and the Separating Movement and Gravity Components (SMGC) method.	34

3.2	Data augmentation is a process that generates additional data points in alignment with the original training data distribution. However, in HAR datasets, a significant train-test distribution discrepancy often exists. Therefore, it is crucial to identify augmentation policies that generate data points more representative of both (train and test) data distribution, thereby improving the model’s generalization ability.	38
3.3	The diagram illustrates the time and frequency resolutions of different representations.	39
3.4	STFT with different interval length τ	40
3.5	This figure illustrates six different representative mother wavelet functions. The results of transforming the same sensor readings using these mother wavelet functions are visualized to the right of each corresponding wavelet. It is evident that the resulting representations vary significantly.	41
3.6	Categorical distribution of 10 augmentation sub-policies. Red bars indicate sub-policies with a negative influence, while green bars represent those with a positive impact.	44
3.7	An overview of the proposed AutoAugHARbasic. Every channel across all modalities undergoes the same augmentation sub-policy. During the data propagation phase, only one augmentation sub-policy is chosen and applied. The selection of this sub-policy depends on the probability p^i associated with each path. The objective of the optimization is to ensure that sub-policies which improve performance have a higher probability compared to those that affect performance.	46
3.8	Overview of the proposed AutoAugHAR framework. Different colors represent data from various modalities. In the first stage, data from all modalities are transformed using the same selected operator. In the second stage, data from each modality is individually transformed by different operators and subsequently integrated.	48

3.9	Examples of candidate augmentation operators on the HAPT dataset.	55
3.10	Comparison of the classification performance between the proposed AutoAugHAR and other SOTA augmentation methods across five datasets, each containing multiple modalities. . . .	61
3.11	Comparison of the classification performance between the proposed AutoAugHAR and other SOTA augmentation methods across three datasets, each containing only a single modality. . .	62
3.12	Comparison of the classification performance between the proposed AutoAugHAR and several of its variants across five multi-modal datasets. This comparison validates the contributions of AutoAugHAR’s design.	67
3.13	Evolution of augmentation sub-policy probabilities over training epochs during the AutoAugHAR training process. These examples are derived from training the Attend model.	69
3.14	The training time for one iteration of the LOSO-CV process.	70
3.15	Raw signals from the accelerometer’s x-axis (middle column) are convolved with Symlets wavelets at different scales (left column).	74
3.16	Raw signals from the accelerometer’s x-axis (middle column) are convolved with Coiflet wavelets at different scales (left column).	74
3.17	This figure shows the six mother wavelet functions representing the cluster centroids from the k-means clustering.	75
3.18	Overview of the proposed <i>learnable sparse wavelet layer</i> , which can be integrated into any HAR model. Figure (a) represents the training process, where the learnable layer learns which wavelet filters are important. Figure (b) shows the deployment process, where unimportant wavelet filters are pruned. C = number of channels, L = length of sliding window, F = number of initially selected filters, F' = number of filters after pruning.	76

3.19	Result of the experiment. Different colors indicate different HAR models (green for DeepConvLSTM, purple for SA-HAR, and blue for MCNN). Different linetypes denote the macro F_1 -scores from different setups (dash lines for baselines, solid lines for <i>learnable sparse wavelet layers</i> , and dot lines for learnable wavelet layers without pruning). The bars with different intensities refer to the number of floating point operations required by different setups, namely light colors for baselines, normal colors for <i>learnable sparse wavelet layers</i> , and dark colors show learnable wavelets without pruning.	79
4.1	Overview of the proposed TinyHAR model.	86
4.2	Architecture of transformer block.	88
4.3	The self-attention is employed to facilitate cross-channel interactions, enabling each channel to dynamically integrate information from all other channels.	89
4.4	Attention-based fusion (a) vs fully-connected layer-based fusion (b).	90
4.5	LOSO-CV performance comparison between TinyHAR and the DCL model with varying model sizes. For each dataset, the averaged $F1_M$, the corresponding model size (number of trainable parameters) and number of FLOPs are shown separately.	93
4.6	An overview of the proposed model, Cross-Atten , which consists of two branches. The branches are color-coded: blue for the time-series branch and green for the spectrogram branch. The notation of the feature map and the parameters within each branch are denoted by different symbols, hat and bar, respectively.	97
4.7	Cross-attention block for two branches. Each feature vector serves as a query to interact with all feature vectors from the opposite branch. Both branches adhere to the same procedure.	100
4.8	Performance comparison on four HAR datasets between the proposed Cross-Atten model and other SOTA HAR models (I).	105

4.9	Performance comparison on four HAR datasets between the proposed Cross-Atten model and other SOTA HAR models (II).	106
4.10	Performance comparison on four HAR datasets between the proposed Cross-Atten model and other SOTA HAR models (III).	107
4.11	Performance comparison on six HAR datasets between the reduced Cross-Atten model and the IF-Conv model (I).	108
4.12	Performance comparison on six HAR datasets between the reduced Cross-Atten model and the IF-Conv model (II).	109
4.13	This figure presents an overview of the architecture of the proposed MLP-HAR model.	112
4.14	Figure (a) shows the specific structure of the data embedding module. Figure (b) shows the specific structure of the mixer module. Different colors in the figure represent readings of different sensor channels, and different shades of the same color represent different intervals. Mixed color shows fused information from FC layer.	113
4.15	Performance comparison on three HAR benchmark datasets between the MLP-HAR model and other SOTA HAR model (I).	117
4.16	Performance comparison on three HAR benchmark datasets between the MLP-HAR model and other SOTA HAR model (II).	118
4.17	(a) Ablation study validating the contributions of different representations and skip connections, along with an impact analysis of parameter N in the MLP-HAR model. (b) Impact analysis of parameters f and N in the MLP-HAR model.	122
4.18	Comparison of inference time between the MLP-HAR and Tiny-HAR models on two devices. The Y-axis is logarithmically scaled (base 10).	124
5.1	Sensor values at each time step or within an interval are sequentially fed into the LSTM.	129

5.2	Multiple features (represented as shapes) at various sampling times (represented as colors) within a single interval must be flattened into a 1D vector for LSTM input, disrupting the natural sequence. Alternatively, a convolutional kernel can be applied to aggregate local information while preserving the original structure.	130
5.3	(a) In hybrid fusion convolution, the kernel height remains 1, but the kernel slides in two directions: along both the temporal axis and the sensor channel axis. Thanks to weight sharing, this approach significantly reduces the number of parameters. (b) In late fusion convolution, the kernel height is set to 1, with each sensor channel assigned its own convolution kernel, which also slides along the temporal axis. (c) In early fusion convolution, the kernel height is fixed to match the number of sensor channels, and the convolution kernel slides along the temporal axis.	132
5.4	Overview of the search space for the ECLSTM framework. . .	134
6.1	Illustration of manual network design and optimization based on neural architecture search.	140
6.2	Before using the MicroNAS framework, the user needs to specify the dataset to be used, the target micro-controller (MCU_t), and the maximum allowable hardware utilization in terms of execution latency (Lat_t) and peak memory consumption (Mem_t). The output of the system is a corresponding neural network in the TF-Lite format.	144
6.3	High-level overview of the search space for MicroNAS framework.	145
6.4	(a) Search space of Time-Reduce cell. It contains two decision groups. α to choose a convolution operator and α_{ch} to search for the number of filters for each operator. (b) Dynamic convolution with searchable filter masks.	145

6.5	Search space of Sensor-Fusion cell. It contains six decision groups. Each decision group is denoted as α in dashed box. Five of them are for the five pathways and one for the filter masks.	147
6.6	Execution latency of whole architectures from our search space. Left: Our lookup-table latency approach. MAE: 1.59 ms, R^2 : 99.97%. Right: Flops based estimate: MAE: 15.57 ms, R^2 : 96.78%.	155
6.7	Correlation between predicted execution latency and actual execution latency for several architectures from the search space.	156
6.8	Classification performance and latency trade-offs on the UCI-HAR dataset. Left: Accuracy, Right: F1-Score (Macro). . . .	157
6.9	Classification performance and latency trade-offs on the SkodaR dataset. Left: Accuracy, Right: F1-Score (Macro).	157
6.10	Trade-off between peak memory consumption and Accuracy / F1-Score (Macro). Comparison on the SkodaR dataset.	157
6.11	MicroNAS compared against random search on the UCI-HAR dataset with the Nucleo-L552ZE-Q.	158
6.12	MicroNAS compared against random search on the SkodaR dataset with the Nucleo-L552ZE-Q.	159
6.13	Overview of the architecture design of HAR models.	161
6.14	Impact of the number of sensor channels on model complexity (a) and on number of trainable parameters (b).	162
6.15	DSC layer for the input layer and convolutional layer.	164
6.16	Search space of each layer for SFTNAS framework.	166
6.17	(a) Trade-off between inference time and performance from SFTNAS framework. (b) Distribution of optimized filter numbers from SFTNAS framework.	173

List of Tables

2.1	Statistical Summary of Selected Datasets.	26
3.1	Candidate operators and the settings of hyper-parameters.	53
3.2	Comparison between TinyHAR without AutoAugHAR and the compressed TinyHAR model with AutoAugHAR.	66
4.1	Parameters related to FFT transformation for each dataset.	102
4.2	The structure for the time-series branch.	104
4.3	The structure for the spectrogram branch.	104
4.4	Ablation study on seven datasets	110
4.5	Performance on 10 datasets from the UEA multivariate time series classification archive.	110
4.6	Hyper-parameters related to the Short-Time Fourier Transform for each dataset.	116
4.7	Average ranking of all models across six datasets. The smaller the rank, the better the performance.	119
4.8	Comparison of performance before and after post-processing. The model name + P stands for post-processing.	121
5.1	Comparison of classification performance based on averaged F1-scores on holdout test sequences between the ECLSTM model and other HAR models.	136
6.1	Performance comparison between the SFTNAS framework and other pruning methods on the MotionSense dataset.	170
6.2	Performance comparison between the SFTNAS framework and other pruning methods on the Skoda(r) dataset.	171

6.3	Performance comparison between the SFTNAS framework and other pruning methods on the UCI-HAR dataset.	172
-----	---	-----

1 Introduction

1.1 Background

Human Activity Recognition (HAR) utilizes sensor technology and data analysis to detect and monitor the physical activities of individuals [24]. By collecting data from various sensors, HAR systems can identify a range of activities, such as walking, running, or sitting [33], as well as physiological states like sleep patterns and stress levels [50]. These insights are crucial for understanding human behavior and health, and they serve as a driving force for the application of HAR systems across various fields. These fields include gaming, human-robot interaction, e-commerce, security, rehabilitation, sign language recognition, sports, health monitoring, smart homes, and advanced manufacturing [165].

HAR systems are generally categorized into two types: video-based and sensor-based [66; 165]. Video-based systems use cameras to capture visual data for behavior recognition, while sensor-based systems rely on wearable sensors to track and log activity. Compared to video-based HAR systems, wearable sensor-based systems offer several distinct advantages:

- **Outdoor Usability:** These systems can operate effectively in outdoor environments without the need for special camera equipment installation.
- **Enhanced Privacy and Ubiquity:** Given the privacy concerns associated with camera use in personal spaces, sensor-based systems are preferable for daily activity monitoring. Additionally, sensors are widely available and can be easily integrated into everyday environments.
- **On-device Processing:** HAR systems can be implemented directly on

the device, allowing for on-site activity classification without the need to transmit data to external cloud servers.

- **Comprehensive Data Collection:** Sensor-based systems provide more comprehensive data, including metrics such as blood oxygen levels, heart rate, and body temperature, which are beyond the capabilities of camera-based systems.

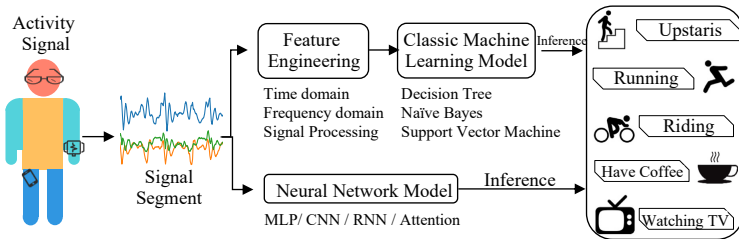


Figure 1.1: An illustration of sensor-based HAR system.

Therefore, **this dissertation focuses on wearable HAR systems**. The primary task of a wearable HAR system is to classify user activities based on segments of data collected from wearable sensors, typically utilizing Machine Learning (ML) or Deep Learning (DL) algorithms. As illustrated in Figure 1.1, a standard HAR process involves segmenting the collected sensor signals into fixed-size windows, which are then input into a classifier model. For real-time monitoring and immediate feedback on the user’s state, the classifier should be deployed on edge devices such as smartphones or smartwatches. To minimize costs, more lightweight devices like smart bands or Micro-controller Units (MCUs) may also be considered for deployment.

1.2 Problem and Motivation

However, implementing wearable HAR systems on-device presents significant challenges. The classifier must be lightweight and efficient enough to operate within the stringent memory constraints and relatively low clock speeds of

ultra-low-power computational platforms, such as MCUs. Due to these limitations, most current on-device HAR systems employ simple, traditional ML algorithms, particularly tree-based models like Decision Trees (DTs) and Random Forests (RFs) [32]. These models are favored for their low inference complexity and ease of software implementation, enabling real-time classification with minimal memory usage.

Nevertheless, traditional ML approaches for HAR require the careful design and selection of relevant features [54; 24], a process that demands substantial human intervention and expert knowledge. Simplistic features may fail to deliver optimal performance, while complex features, although potentially more effective, can significantly burden computational resources.

Over the past decade, neural network models have undergone rapid development and have demonstrated exceptional performance in HAR tasks. Unlike traditional ML algorithms, neural networks excel by automatically learning features that are highly relevant to the target task, eliminating the need for manual feature design and selection [184]. To further improve the performance of neural network models in HAR, various advanced network architectures have been proposed, as discussed in Section 2.3. Although these studies highlight the effectiveness of neural networks models in achieving superior classification results for HAR, their deployment on-device remains limited. This limitation is primarily due to the substantial memory requirements and the large computational overhead needed for inference. **There remains a significant gap between research on HAR neural network models and their practical deployment on lightweight devices.**

1.3 Aim and Objectives

The primary aim of this research is to bridge the gap between the development of HAR neural networks models and their practical deployment on edge devices by creating lightweight neural networks models. A "lightweight model" is defined as one that possesses fewer parameters, simpler structures, and lower overall complexity, which in turn enables faster inference on devices with limited computational resources. To achieve this aim, the dissertation aims to overcome the current limitations of existing HAR systems through the follow-

ing key objectives:

1. **Thorough Literature Review:** Conduct an in-depth review of existing deep learning models for HAR to thoroughly understand their challenges and limitations.

2. **Identification of Novel Opportunities:** Identify and explore novel opportunities and under-explored areas in the development of lightweight HAR models.

3. **Design and Implementation of Innovative Algorithms:** Design and implement new algorithms that address the identified opportunities, specifically optimizing them for deployment on edge devices.

4. **Comprehensive Evaluation:** Perform a rigorous evaluation of the developed solutions in terms of their performance, model size, complexity, and inference time on lightweight devices. This evaluation will be based on extensive experiments using diverse open-source datasets to ensure the generalizability of the models' performance.

By achieving these objectives, this dissertation aims to advance the field of DL-based HAR systems, making a significant contribution to the development of efficient, high-performance models that are feasible for deployment on lightweight edge devices.

1.4 Research Hypothesis and Contributions

Aligned with the aims and objectives, this dissertation presents a series of contributions that provide solutions on designing lightweight DL models for wearable HAR systems.

1.4.1 HAR Data Preparation

HAR presents several unique challenges. First, the amount of labeled data is typically limited, and the data often originates from a small number of subjects, which leads to poor representativeness and informativeness [188]. Additionally, HAR data exhibits significant intra-class variability and inter-class similarity [24; 1], further complicating the classification task. Moreover, signals acquired through wearable sensors often display multi-frequency, non-periodic,

and fluctuating characteristics. Frequency features are critical for distinguishing between different human activities. While frequency information can be implicitly modeled within DL HAR models using time-series representations, these models often struggle to optimally learn and generalize high-frequency data. Consequently, researchers frequently resort to using larger or more complex models due to their enhanced feature extraction capabilities.

Question 1: How can HAR data be optimally pre-processed to improve its quality, better highlight generalizable patterns, and reduce the complexity of extracting intricate time-series information, allowing models with fewer parameters or simpler structures to maintain SOTA performance?

To address this question, we implemented two approaches: data augmentation and data transformation, with the latter involving the conversion of time-series data into alternative representations, such as spectral representations. Both methods face a common challenge: the need for method and hyperparameter optimization. Given the diverse scenarios in HAR tasks, different data augmentation techniques or representation transformations can yield varying results, sometimes positive and other times negative. Our innovation lies in leveraging the characteristics of HAR tasks in combination with automated machine learning techniques. When applying these methods, both the techniques and hyperparameters are automatically optimized and trained alongside the model weights according to the specific task.

Contributions: Extensive experiments have validated our approach. By properly preparing the data, we can enhance the performance of SOTA HAR models without increasing their size. Additionally, it is feasible to reduce the size of existing models using shrinking strategies [142] (e.g., depth and width scaling factors) while still maintaining SOTA performance. Moreover, the training overhead does not significantly increase and remains comparable to that of previous methods.

1.4.2 HAR Network Architecture Design

Another factor contributing to the large and complex nature of HAR models is the necessity to extract a diverse range of information from sensor data to accurately predict human activities. This includes extracting local context information, capturing long-term temporal dependencies, distinguishing the contributions of different modalities, and understanding the cooperative relationships between these modalities. The design of neural networks model architectures is essentially a process of information extraction. Simply increasing the model size without considering the characteristics of HAR data does not effectively extract the critical information needed for accurate predictions.

Question 2: How can we design a model that effectively considers the various types of information required for HAR tasks, enabling the model to maintain strong information extraction capabilities despite having a small size, and thereby achieving high classification performance?

Taking these considerations into account, we established five key principles for constructing lightweight HAR models. Based on these principles, we initially developed the TinyHAR model by carefully selecting deep learning operators to efficiently extract the necessary information. Building upon the foundation of the TinyHAR structure, we extended it to create a cross-attention HAR model with hybrid representations, incorporating the data preparation techniques discussed earlier. Finally, drawing inspiration from both the TinyHAR and cross-attention models, and considering the specific constraints of MCU hardware, we designed the MLP-HAR model, which leverages fully connected layers exclusively to achieve efficient performance.

Contribution 2: Through extensive experiments, we demonstrated that the proposed models, despite having fewer parameters, can achieve and even surpass the performance of existing SOTA HAR models. Furthermore, we successfully deployed these models on several edge devices, where they exhibited significantly faster inference times compared to other HAR models, enabling real-time monitoring.

1.4.3 Automatically Network Architecture Optimization

In addition to designing the overall network architecture, defining parameters such as the number of layers, kernel sizes, and the number of filters is crucial. In most current HAR models, these parameters are predefined, which can lead to either oversized models or suboptimal performance due to the risk of overfitting. For instance, in one study, reducing the number of LSTM layers in the benchmark model DeepConvLSTM improved its performance. However, optimizing these structural parameters is a labor-intensive and time-consuming process.

Question 3: Is it possible to quickly and automatically optimize these model parameters, thereby reducing the difficulty of manual parameter tuning and resulting in a more compact model?

To address this challenge, we propose leveraging automated machine learning (AutoML) techniques and model pruning methods. Building on the five design principles for HAR models discussed earlier, we define the search space for HAR model architecture. Through techniques such as gradient descent or Bayesian optimization, we simultaneously optimize the model structure during the training process.

Contribution 3: By employing AutoML and neural architecture search (NAS) techniques, we achieved further improvements in HAR model performance without the need for manual parameter tuning.

1.4.4 Hardware-Aware Model Optimization

In real-world deployment, models must not only be compact but also meet specific task-related requirements, such as inference speed. For instance, in fall detection scenarios, a model's response time must be extremely fast to trigger protective mechanisms in time to prevent injury. Therefore, optimizing lightweight models with hardware constraints in mind is crucial to ensure efficient and effective deployment.

However, the inference time needed for different tasks can vary significantly. Additionally, different hardware platforms support various operators to differing degrees and have varying memory capacities and computational efficiencies. As a result, a model with the same architecture may exhibit different inference times across different hardware platforms. This highlights the importance of considering hardware constraints when selecting models and designing network architectures for deployment. The process of designing within these multi-dimensional constraints is challenging, complex, and time-consuming, as it requires balancing trade-offs between performance, memory usage, and computational efficiency.

Question 4: How can we incorporate hardware constraints into the network architecture optimization process to automatically generate models that maintain SOTA performance while also meeting deployment requirements?

To address this challenge, we first conduct a detailed profiling of the target hardware to understand its memory capacity, computational capabilities, and supported operators. This information guides the design of the search space. Next, we incorporate hardware constraints, such as memory consumption and inference time, into the loss function. By employing NAS techniques, it aims to balance model performance with computational efficiency and memory size.

Contribution 4: We successfully automated the optimization of models based on specific task requirements, such as setting a target inference time on the selected MCU. These optimized models not only meet the specified conditions but also maintain SOTA-level classification performance.

1.5 Structure of This Thesis

The dissertation is organized into seven logically structured sections that delve into various research aspects related to the development of lightweight neural networks models for HAR. Figure 1.2 illustrates the structure of the thesis,

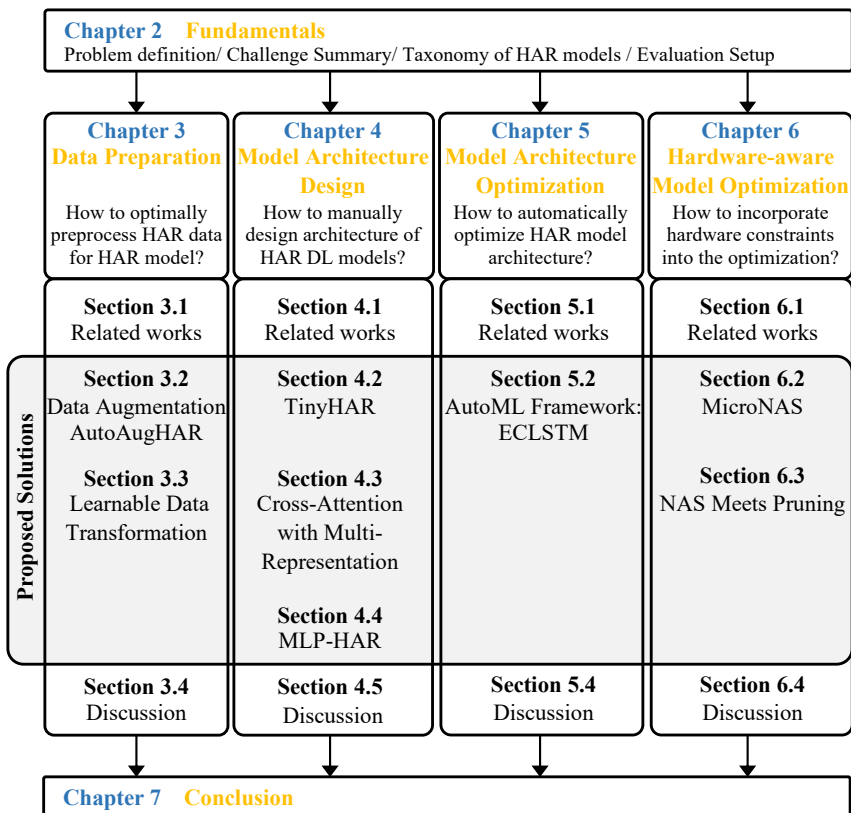


Figure 1.2: Structure of the dissertation focusing on lightweight neural networks for HAR tasks, with Chapter 1 (Introduction) excluded. Abbreviations: AutoAugHAR refers to an automated augmentation framework for HAR; MLP-HAR stands for Multi-Layer Perceptron for HAR; AutoML denotes automated machine learning; ECLSTM represents Embedded Convolutional Long Short-Term Memory; NAS stands for Neural Architecture Search.

providing an overview of the relationships between the different sections and subsections. Figure 1.2 allows readers to quickly understand how the various parts of the dissertation are interconnected and how they collectively contribute to the overarching goal of designing lightweight neural networks models for HAR tasks.

Chapter 2 provides a comprehensive review and summary of the current SOTA HAR models, critically assessing their strengths and limitations with a particular emphasis on their architectures and how these relate to the unique characteristics and challenges of HAR data. The insights derived from this analysis form the foundation for the methodologies and approaches developed in subsequent chapters. Additionally, this chapter details the experimental setup used throughout the dissertation, helping to avoid redundancy in later discussions, as most experiments follow a consistent configuration. Any deviations from this setup are explicitly noted in the corresponding chapters. The experimental setup description includes details on the datasets, data preparation methods, model training configurations, and evaluation metrics used throughout the dissertation.

Chapters 3 to 6 follow the typical workflow of a ML project—data preparation, model design, model training, and model deployment. Each chapter focuses on specific methods aimed at achieving model lightweighting at various critical stages of this process.

Chapter 3 explores strategies for making existing DL HAR models more lightweight through **data preparation** techniques. Two approaches are proposed: a modality-aware automated data augmentation framework and a learnable wavelet layer.

Chapter 4 focuses on the **design of neural networks architectures** that combine strong feature extraction capabilities with a lightweight structure. We propose five guiding principles and introduce three models based on these principles: TinyHAR, Cross-Attention with Multi-Representation, and Multi-layer Perceptron HAR (MLP-HAR).

Chapter 5 addresses the **automated optimization of model structural parameters**. This chapter defines the search space specific to HAR models and introduces AutoML Framework ECLSTM designed to optimize models for HAR tasks.

Chapter 6 builds upon the concepts introduced in Chapter 6 by exploring how **hardware constraints and task-specific requirements** can be incorporated into the optimization process. This chapter focuses on optimizing models to meet practical deployment needs, such as inference time and memory usage.

Chapter 7 concludes the dissertation by summarizing the key findings and discussing future directions for the development of lightweight neural networks models for HAR.

1.6 List of Publications

The following list gives an comprehensive overview of all scientific papers published by the author that are relevant for this dissertation. Significant parts of this dissertation (across all chapters) were copied from the relevant earable papers listed below and assembled into a coherent monograph structure.

Yexu Zhou, Michael Hefenbrock, Yiran Huang, Till Riedel and Michael Beigl. "Automatic remaining useful life estimation framework with embedded convolutional LSTM as the backbone". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*. September 14–18, 2020, Proceedings, Part IV.

Yexu Zhou, Haibin Zhao, Yiran Huang, Till Riedel, Michael Hefenbrock and Michael Beigl. "Tinyhar: A lightweight deep learning model designed for human activity recognition". In: *Proceedings of the 2022 ACM International Symposium on Wearable Computers*. 2022, pages 89-93.

Yexu Zhou, Haibin Zhao, Till Riedel, Michael Hefenbrock and Michael Beigl. "Improving Human Activity Recognition Models by Learnable Sparse

Wavelet Layer". In: *Proceedings of the 2022 ACM International Symposium on Wearable Computers*. 2022, pages 84-88.

Nils Schwabe, **Yexu Zhou**, Leon Hielscher, Tobias Röddiger, Till Riedel and Sebastian Reiter. "Tools and methods for Edge-AI-systems". In: *at Automatisierungstechnik*. 2022, pages 767-776.

Tobias King, **Yexu Zhou**, Tobias Röddiger and Michael Beigl. "MicroNAS: Memory and Latency Constrained Hardware-Aware Neural Architecture Search for Time Series Classification on Microcontrollers". In: *arXiv preprint arXiv: 2310.18384*. 2023.

Yiran Huang, **Yexu Zhou**, Michael Hefenbrock, Till Riedel, Likun Fang, Michael Beigl. "Automatic feature engineering through Monte Carlo tree search". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*. 2023, pages 581-598.

Yexu Zhou, Haibin Zhao, Yiran Huang, Tobias Röddiger, Till Riedel and Michael Beigl. "Cross-Attention Transformer with Multi-Representation for Human Activity Recognition Using Wearable Sensors". In: *KITOPEN*. 2023.

Yexu Zhou, Haibin Zhao, Michael Hefenbrock, Siyan Li, Yiran Huang and Michael Beigl. "Deep Neural Network Pruning with Progressive Regularizer". In: *IEEE International Joint Conference on Neural Network (IJCNN)*. July 2024.

Yexu Zhou, Tobias King, Yiran Huang, Haibin Zhao, Till Riedel, Tobias Röddiger and Michael Beigl. "Enhancing Efficiency in HAR Models: NAS Meets Pruning". In: *IEEE International Conference on Pervasive Computing and Communications Workshops*. 2024, pages 33-38.

Yiran Huang, **Yexu Zhou**, Haibin Zhao, Till Riedel and Michael Beigl. "Optimizing AutoML for Tiny Edge Systems: A Baldwin-effect Inspired Genetic Algorithm". In: *IEEE International Conference on Pervasive Computing and Communications Workshops*. 2024, pages 160-165.

Yexu Zhou, Haibin Zhao, Yiran Huang, Tobias Röddiger, Murat Kurnaz, Till Riedel and Michael Beigl. "AutoAugHAR: Automated Data Augmentation for Sensor-based Human Activity Recognition". In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*. 2024, pages 1-27.

Yexu Zhou, Tobias King, Haibin Zhao, Yiran Huang, Till Riedel and Michael Beigl. "MLP-HAR: Boosting Performance and Efficiency of HAR Models on Edge Devices with Purely Fully Connected Layers". In: *Proceedings of the 2024 ACM International Symposium on Wearable Computers*. 2024.

2 Fundamentals

This chapter provides the foundational knowledge and background information necessary for the research presented in this dissertation. In Section 2.1, we define the problem of wearable HAR, establishing the core focus and objectives of this study. Section 2.2 then elaborates on the characteristics and challenges inherent in HAR tasks, which are critical considerations for the design and training of effective models. In Section 2.3, we conduct a systematic and comprehensive review of the current SOTA HAR models, analyzing their strengths and weaknesses to guide the research direction of this work. Finally, Section 2.4 presents the experimental setup employed throughout this dissertation, detailing data processing methods, model training configurations, and evaluation metrics to ensure the consistency and reproducibility of the results.

2.1 Problem Definition

This dissertation focuses on sensor-based HAR. Devices equipped with Inertial Measurement Units (IMUs) are placed on the human body and synchronized to collect data. An IMU typically consists of built-in sensors such as accelerometers, gyroscopes, and magnetometers, each capable of generating a signal vector along the x-axis, y-axis, and z-axis simultaneously. The data collected over time from these sensors can be represented as a multi-channel time series $[\mathbf{x}_1, \dots, \mathbf{x}_t, \dots]$, where $\mathbf{x}_t = [x_{t1}, x_{t2}, \dots, x_{tC}]$. Here, x_{tC} denotes the sensing value from the C -th sensor channel at time t , and C is the total number of sensor channels used.

Given a recorded multi-channel signal recordings, the task is to detect a series of human activities (e.g., walking, running, biking) over a specific duration. Specifically, long signal recordings is segmented into segments $\mathbf{X} \in \mathcal{R}^{L \times C}$ using a sliding window method. Each segment $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L]$ is represented

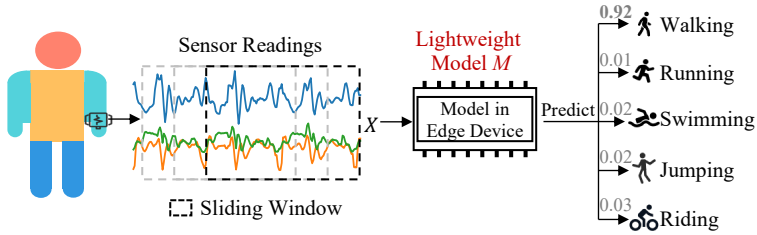


Figure 2.1: This figure provides an overview of wearable HAR systems, with a focus on designing lightweight deep learning models for HAR tasks.

as a 2D matrix, where L denotes the window length. An activity label y is assigned to each segment, forming a data sample (\mathbf{X}, y) . Subsequently, each segments is fed into a classification model M , which maps the segment \mathbf{X} to a predefined activity label. The activity recognition problem can therefore be formally described as follows, $\hat{y} = M(\mathbf{X})$. The process described above is illustrated in Figure 2.1.

This dissertation aims to develop models M that achieve high prediction accuracy while also maintaining a compact size, fewer learnable parameters, and faster inference times on edge devices.

2.2 Challenges and Characteristics of HAR Task

To ensure the effectiveness of a model, it is essential to first understand the challenges and characteristics associated with the HAR task.

2.2.1 Challenges

This section outlines several challenges unique to sensor-based activity recognition that the HAR research community must address:

Limited Annotated Dataset: Sensor-based HAR datasets are typically limited in size [1], primarily due to the challenges associated with data collection and annotation, as well as the high costs involved in conducting user studies and recruiting volunteers. These limitations reduce the representativeness and

informativeness of the data [188]. While DL models have shown promising results in HAR tasks, their ability to generalize across different subjects remains a significant concern.

Inter-Class Similarity Challenge: HAR datasets often exhibit significant similarities between different activities [71; 24]. For instance, activities such as walking and running may share overlapping characteristics, making them difficult to distinguish. Additionally, sensor data is inherently noisy due to the imperfections of the sensors themselves, further complicating the extraction of distinguishable features that uniquely represent each activity.

Intra-Class and Inter-Subject Variability Challenge: Activity patterns are highly individual-dependent, leading to diverse activity styles among different users [24]. Moreover, the same individual’s activity patterns may vary over time. Hardware-related factors, such as variations in sensor placement, device tightness, and the use of different devices, also contribute to this variability. These factors create substantial variability within activity classes and across different subjects, leading to discrepancies between the distributions of training and test data. Addressing these distribution discrepancies is crucial for improving model performance.

2.2.2 Characteristics

Below are some unique characteristics of HAR compared to fields like Natural Language Processing (NLP) and computer vision:

Data Format: Unlike NLP, where data is represented as 1D sequential words, HAR data consists of multivariate time series, with each modality conveying distinct information. In comparison to image data, where both dimensions (height and width) hold similar significance, HAR data involves two fundamentally different dimensions: sensor channel dimension and temporal dimension. Moreover, the temporal dimension is often significantly larger than the sensor channel dimension, and the interpretations of these two dimensions differ greatly.

Local Context: A single timestamp in a sensor signal sequence provides limited semantic information [1; 119; 155], unlike a word in a sentence. Most activities are composite, consisting of a sequence of atomic actions. Therefore,

understanding the context surrounding each timestamp is crucial for accurately predicting the overall activity.

Global Temporal Extraction: Certain timesteps within a sequence may exhibit more salient patterns than others [102]. For accurate activity recognition, it is essential for the model to capture global temporal dependencies throughout the entire sequence.

Different Contribution Across Multi-Modality: Different sensor modalities originate from various domains, and merging them without accounting for their unique contributions can limit the model's effectiveness [23]. For example, accelerometer data might be more significant for distinguishing between "walking" and "biking," whereas gyroscope data might be crucial for differentiating between "turning left" and "turning right." Treating all modalities equally could undermine the overall performance of activity classification.

Collective Interaction within Multi-Modality: Human activities often involve the coordinated motions of multiple body parts. For instance, running can be viewed as a combination of arm and leg movements. Therefore, it is important to consider the interactions between different modalities to accurately represent and classify activities [1; 23].

2.2.3 Discussion

Unlike fields such as computer vision and NLP, where datasets are often more structured and abundant, HAR presents distinct challenges that necessitate specific architectural considerations. Applying techniques directly from NLP or computer vision while disregarding these unique challenges is likely to result in suboptimal model performance. This highlights the need for specialized approaches tailored to HAR tasks, which differ significantly from those used in other fields. By addressing these unique challenges head-on, we can enhance the model's learning capabilities, ultimately resulting in more efficient and lightweight models optimized for HAR applications.

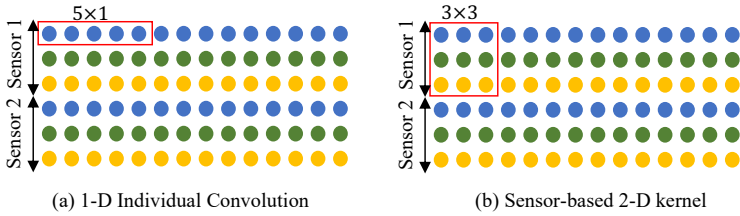


Figure 2.2: 1D convolution vs 2D convolution

2.3 Taxonomy of Human Activity Recognition Models

In line with the goal of designing lightweight DL models for HAR, Sections 2.3.1, 2.3.2, 2.3.3 and 2.3.4 will first review existing research on DL-based HAR models, categorized by the types of operators they employ. Following this, Section 2.3.5 will examine relevant work focused on lightweight HAR models. Finally, in Section 2.3.6, we will discuss the key insights gained from these reviews and how they can inform the design of more efficient and effective models for HAR tasks.

2.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are among the most widely used models for HAR tasks in the field of ubiquitous computing [55; 24], primarily due to their exceptional ability to extract local dependencies [160] and their inherent support for parallel processing [139].

The study in [166] demonstrated the potential of CNNs in HAR tasks by utilizing individual convolutional kernels that traverse the temporal dimension, effectively extracting hierarchical temporal features. As illustrated in Figure 2.2 (a), these individual convolutions employ 1D kernels along the temporal axis, allowing each channel to be processed separately, which accounts for the varying contributions of different modalities. However, this approach does not sufficiently capture the dependencies between sensor channels.

Furthermore, CNNs typically utilize fixed kernel sizes, which restricts their ability to capture fluctuations across different temporal ranges. To overcome

this limitation, the study in [88] adopted an inception-style [68] convolution block, combining multiple CNN structures with varying kernel sizes to extract temporal features across multiple time scales. However, the use of multiple kernels increases computational complexity and results multi-branch structure, which are not good for inference time, because it increases the memory access cost [89].

Another challenge with CNNs is their inherently small receptive field, which limits their ability to capture long-term temporal dependencies. One approach to address this limitation is to insert pooling operations between CNN layers, which can effectively reduce the length of the time series [24]. However, this approach can lead to significant information loss [24]. An alternative approach is to apply deep dilated CNNs [161]. While dilated convolutions avoid information loss and do not introduce additional computational costs, this operator is often not supported on many edge devices [84].

In addition to considering various temporal scales and long-term dependencies, the different contributions of sensing modalities and their interactions are also critical factors in HAR tasks. The aforementioned CNN models typically treat different modalities uniformly, which can limit their effectiveness. To address this limitation, the study in [58] introduced 2D kernels in the convolutional layers to capture both local temporal dependencies and spatial dependencies across sensors, as illustrated in Figure 2.2 (b). Despite this enhancement, CNNs still struggle to explore correlations between non-adjacent modalities, as they tend to focus primarily on neighboring modalities.

To effectively learn modality-specific features and efficiently fuse information from different sensor modalities, several studies have proposed multi-branch CNN architectures. These approaches explore various fusion mechanisms, such as sensor-based late fusion [125; 57] and channel-based late fusion [172; 40], as illustrated in the Figure 2.3. In these architectures, independent CNN branches are assigned to each sensor modality or sensor channels, and a cross-sensor fully connected layer integrates these branches to uncover inter-modality information.

Overall, the strength of CNN models lies in their ability to extract local context and their strong parallel processing capabilities, making them well-suited

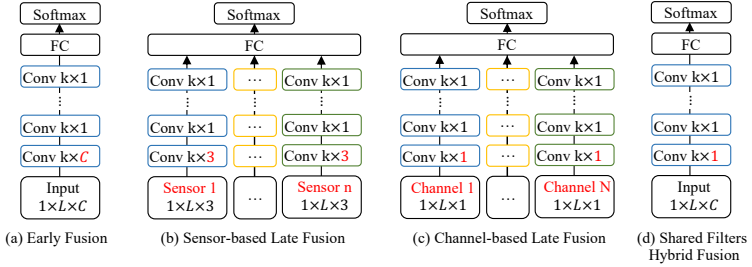


Figure 2.3: This figure illustrates various CNN-based fusion strategies. The differently colored branches represent distinct weights, indicating that there is no weight sharing between them.

for hardware deployment. However, due to the characteristics of HAR tasks, which require the extraction of diverse types of information, the capabilities of CNNs may not be fully sufficient. Although various methods have been proposed to mitigate these limitations, multi-branch CNN models compromise memory utilization efficiency, as they require storing the outputs of each branch until they are combined through addition or concatenation [89].

2.3.2 Recurrent Neural Networks

To better capture temporal dependencies, Long Short-Term Memory (LSTM) [25] and Gated Recurrent Unit (GRU) [133] networks have been introduced into HAR tasks. Beyond the basic LSTM network, ongoing research in the HAR field has explored various RNN variants. For example, the study in [182] employed a Bidirectional LSTM (Bi-LSTM) architecture, which utilizes two conventional LSTM layers to extract temporal dynamics from both forward and backward directions. Additionally, some studies have attempted to design specific cell structures within RNNs [24]. However, some studies [53] have shown that these alternative RNN cells do not provide notably superior performance compared to conventional LSTM cells, and these custom RNN cells often lack robust support on hardware platforms [84]. Furthermore, EnsembleLSTM [55] advanced LSTM-based models by combining multiple sets of LSTM learners and developing modified training procedures to address the challenge of unbal-

anced data. However, this approach, which requires the parallel execution of multiple models, is impractical for deployment on edge devices.

Overall, LSTMs have demonstrated superior capability over CNNs in capturing temporal information. However, similar to the development trajectory observed with CNNs, the pursuit of more powerful feature extraction capabilities has resulted in increasingly complex model architectures. Moreover, RNN variants have following limitations. Their cells consist of different gated branches, leading to complex multi-branch structures that increase memory access costs [153; 89]. Furthermore, RNNs propagate information iteratively along the time axis, which hinders parallel processing and slow down inference time. Additionally, RNNs may entirely overlook the extraction of local context and the differentiation and integration of modality-specific information [184]. These limitations contribute to the relatively infrequent use of pure RNN models in HAR applications.

2.3.3 Attention-based Networks

In recent years, self-attention-based models [183; 150] have demonstrated superior performance over RNN-based models in capturing long-term temporal dependencies, particularly in NLP tasks. The utility of this attention mechanism for HAR has also been explored. For example, studies [104; 148] have employed self-attention mechanisms in place of CNNs or LSTMs to extract temporal information as well as information across channels, showcasing the flexibility of self-attention. This flexibility allows for the exchange and processing of information across various dimensions. Unlike RNNs, which require input data points to maintain a sequential order, self-attention mechanisms do not rely on such order constraints. Furthermore, self-attention enables efficient operations on long sequence inputs, reducing the maximum length of network signal paths to the theoretical minimum of $O(1)$ and avoiding the limitations of recurrent structures in RNN [183].

However, self-attention is highly memory-intensive, with memory consumption growing quadratically with the input length [81; 15; 34]. Even when the model size is relatively small, the memory demands can pose significant challenges during deployment. Moreover, experimental results have shown that

using self-attention alone may not yield optimal performance [178]. One key issue is that the self-attention mechanism’s emphasis on global information exchange can result in the neglect of local context and the temporal sequence of events [150].

2.3.4 Hybrid Networks

The three types of HAR models discussed earlier each have their own strengths and weaknesses, leading to the natural idea of combining them to leverage their respective advantages.

DeepConvLSTM [119] initially used a CNN subnet to extract local features from different sensor modalities and then fed these features into a LSTM-based block to capture long-term temporal dependencies. To further enhance information extraction across different dimensions, self-attention mechanisms have been integrated into HAR models due to their flexibility. For example, to address the "forgetting" limitations of LSTMs [78], DeepConvLSTM-Attention (DCL-A)[113] augments DeepConvLSTM with temporal attention, facilitating global temporal information exchange in a single forward pass. Multi-agent HAR[23] advances DeepConvLSTM with a spatio-temporal attention module to intelligently select informative modalities. The Attend model [1] incorporates self-attention mechanisms to learn interactions between different sensor channels. The ALAE-TAE model [3] is engineered with an attention encoder, based on the squeeze and excitation method [62], with the objective of exploiting relationships among latent features. The IF-ConvT model [178] designs a CNN block that implements the function of the complementary filter [106] and pre-concatenates this block to a CNN-self-attention hybrid model.

Additionally, some HAR models, considering the importance of frequency information for classification [23], first transform data inputs into frequency representations before feeding them into the model. For instance, DeepSense [167] applies FFT transformation to input data and then processes it with a hybrid multi-branch CNN-GRU architecture. GlobalFusion [99] builds on DeepSense by adding two global self-attention modules to efficiently fuse features from various locations and sensor modalities.

However, many of these efforts, while pushing for optimal performance,

have totally overlooked deployment efficiency and real-time processing capabilities [32].

2.3.5 Lightweight HAR Models

Only a limited number of studies have focused on designing lightweight models specifically for HAR tasks. In January 2021, the Layer-Wise CNN [144] proposed a lightweight CNN using "Lego" filters for HAR. These filters, which are lower-dimensional, are stacked like Lego bricks to form conventional filters. The authors claim that this is the first paper to propose a lightweight CNN for HAR in the ubiquitous and wearable computing arena. Following this, Yves Luduvico Coelho et al. [28] combined decision trees with CNNs to create a dynamic HAR system that can be deployed on MCUs. The Distributed CNN [65] introduced a multi-branch structure that, during deployment, uses a distributed set of computing nodes collocated with sensors in specific regions, rather than centrally collecting and processing all sensor measurements with a monolithic neural network. However, this approach not only requires numerous hardware nodes but also faces performance risks due to potential data asynchrony caused by communication delays between nodes. Quantized and Adaptive DNN [32] utilized hyper-parameter optimization with sub-byte and mixed-precision quantization to optimize 1-dimensional CNNs, aiming to find a balance between classification accuracy and memory usage.

Overall, the aforementioned studies are all based on CNNs. The limitations of CNN-based models have already been discussed in Section 2.3.1. This is further evidenced by the experimental results from these lightweight CNN models mentioned above, which fail to surpass earlier SOTA CNN models such as [166; 58; 31]. Consequently, these models fall short of the performance levels achieved by the more advanced hybrid HAR models discussed previously.

There have been a few attempts to create lightweight hybrid HAR models, but these efforts have employed relatively simple methods. Their approach involves manually simplifying the architectures of some SOTA HAR models. For instance, [7] manually reduced the model depth and the number of filters to make the model suitable for deployment on edge devices. Similarly, Improve-DeepConvLSTM [19] merely reduced the number of LSTM layers.

2.3.6 Discussion

To date, research on lightweight neural network HAR models suitable for deployment on edge devices remains scarce. Existing models often sacrifice performance to achieve deployability. Consequently, the primary objective of this dissertation is to develop HAR models that preserve SOTA performance while being deployable on edge devices. Several methods exist for reducing model complexity, including pruning [26], quantization [51], architecture design, and neural architecture search [45]. Although these techniques have been widely studied in other domains, their application in the HAR domain is still limited. Therefore, we aim to explore the integration of these techniques to address the specific challenges and characteristics of HAR tasks and seek to generalize these techniques for optimizing lightweight HAR models.

2.4 Experimental Setup

As discussed earlier, this dissertation introduces several methods aimed at reducing the size of neural network models. To evaluate the performance of these models, extensive experiments were conducted using open-source datasets. This section provides a detailed overview of the evaluation setup, including the datasets utilized, the training procedures applied to HAR models, and the evaluation metrics used to assess their performance.

2.4.1 Datasets

This section provides a detailed overview of the data utilized. Table 2.1 presents a statistical summary of the data as an overview. Within the table, the column labeled "# classes" denotes the number of activity types, while "sensor types" specifies the utilized sensor signal, using the abbreviations: "Acc" for accelerometer, "Gry" for gyroscope, "Mag" for magnetometer, "Grav" for gravimeter and "PRS" for piezoresistive sensor. The column "# subjects" indicates the number of subjects involved in data collection. Each dataset is then described individually in detail as follows.

Daphnet Gait (Daphnet) [11]: The Daphnet Gait dataset was specifically

Table 2.1: Statistical Summary of Selected Datasets.

Datasets	Sampling Frequency	#subjects	#Classes	#Sensor Channels	Sensor Type	Sliding Window Size	# Modalities
Daphnet [11]	64 Hz	10	2	9	Acc	1s	2
PAMAP2 [127]	100 Hz	9	18	18	Acc, Gyro	2s	3
OPPO [21]	30 Hz	4	18	42	Acc, Gyro	1s	3
RW [141]	50 Hz	15	8	21	Acc	2.56s	4
DSADS [14]	25 Hz	8	19	30	Acc, Gyro	5s	3
WISDM [86]	20 Hz	36	6	3	Acc	5.12s	1
HAPT [128]	50 Hz	30	12	6	Acc, Gyro	2.56s	1
GestHome [115]	25 Hz	20	18	9	Acc, Gyro, Mag	2s	1
Mhealth [13]	50 Hz	10	12	12	Acc, Gyro	2.56s	2
MotionSense [107]	50 Hz	24	6	12	Acc, Gyro, Mag, Grav	2.56s	1
Skoda (r) [170]	98 Hz	1	10	30	Acc	2s	10
L-sign [123]	25 Hz	16	36	16	Acc, Gyro, PRS	3s	1

designed to serve as a benchmark for methods that automatically detect freezing of gait using wearable accelerometers placed on the legs and pelvis. This is a binary classification problem (i.e., freeze or no freeze). The dataset includes accelerometer data from three body locations: the ankle, upper limb, and trunk. Data were collected from 10 patients diagnosed with Parkinson’s disease, with a sampling rate of 64 Hz. For model input, we selected 2-second segments of this data.

PAMAP2 Physical Activity Monitoring Dataset (PAMAP2) [127]: The PAMAP2 dataset comprises data collected from 9 subjects performing 18 different physical activities, such as walking, cycling, and playing soccer, using three Inertial Measurement Units (IMUs). Each IMU captures readings from a tri-axial accelerometer, gyroscope, and magnetometer. The sampling rate for all sensors is 100 Hz. In our experiments, we utilized only the signals from the tri-axial accelerometers and gyroscopes, resulting in a total of 18 sensor channels. The data were segmented into 2-second intervals for analysis.

Opportunity Dataset(OPPO) [21]: The OPPO dataset was collected from 4 participants, each equipped with multiple wearable sensors while performing natural kitchen activities. The activities included 17 sporadic gestures and one null class. Signals from tri-axial accelerometers, gyroscopes, magnetometers, and several other internal sensors were recorded at a constant rate of 30 Hz. In our experiments, we utilized only the signals from the tri-axial accelerometers and gyroscopes located on the participants’ back, arms, and shoes. These signals correspond to 7 different locations, resulting in 42 sensor channels. For activity prediction, we used 1-second segments of the data.

RealWorld Human Activity Recognition (RW) [141]: The RealWorld-HAR dataset includes data from 15 subjects performing 8 different activities, such as climbing stairs, jumping, lying down, standing, sitting, running/jogging, and walking. Data were collected using sensors placed at 7 different body locations, including accelerometers, GPS, light, magnetic field, and sound level sensors. For our study, we only utilized accelerometer readings from all 7 body locations (i.e., head, chest, forearm, waist, shin, thigh, and upper arm). GPS and sound level readings were excluded due to their low sampling rates—0.08 Hz for GPS and 2 Hz for sound level—while the remaining sensors operated

at approximately 50 Hz. For prediction tasks, we used 2.56 seconds data segments.

Daily and Sports Activities Dataset (DSADS) [14]: The DSADS dataset was collected from 8 participants (4 females and 4 males). Each participant wore wearable devices at 5 different body locations while performing 19 types of activities. Signals from tri-axial accelerometers, gyroscopes, and magnetometers were recorded at a constant rate of 25 Hz. In our experiments, we utilized only the signals from the tri-axial accelerometers and gyroscopes, resulting in a total of 30 sensor channels.

Wireless Sensor Data Mining Dataset (WISDM) [83]: The WISDM dataset consists of data collected from 36 individuals who carried a cell phone while performing 6 everyday activities: walking, jogging, sitting, standing, going upstairs, and going downstairs. The accelerometer embedded in the cell phone recorded tri-axial data (x, y, and z directions) at a sampling rate of 20 Hz. For prediction tasks, the data were segmented into 5.12-second intervals.

Human Activities and Postural Transitions Dataset (HAPT) [128]: The HAPT dataset was collected from 30 volunteers. Each participant wore a smartphone on their waist while performing 12 daily activities, including 6 basic activities and 6 postural transitions. Signals from the tri-axial accelerometer and gyroscope were recorded at a constant rate of 50 Hz. All sensor signals were segmented using a sliding window containing 128 readings, corresponding to a 2.56-second interval.

GesHome Dataset [115]: The GesHome dataset consists of 18 hand gestures performed by 20 non-professional subjects of various ages and occupations. Each participant performed each gesture 50 times over a period of 5 days, resulting in a total of 18,000 gesture samples. The data were recorded using an embedded accelerometer, magnetometer, and gyroscope at a frequency of 25 Hz, resulting in a total of 9 sensor channels. The data have been segmented into 2-second intervals for analysis.

Mobile HEALTH Dataset (MHEALTH) [13]: The MHEALTH dataset comprises body motion and vital signs recordings from 10 volunteers of diverse profiles while performing 12 physical activities. Sensors, including accelerometers, gyroscopes, and magnetometers, were placed on the subjects' right wrist

and left ankle to measure motion, resulting in a total of 18 sensor channels. Additionally, a sensor positioned on the chest provided 2-lead ECG measurements, which were not used in the development of the recognition model. For prediction tasks, the data were segmented into 2.56-second intervals.

Motion Sense Dataset: The Motion Sense dataset was collected from 24 volunteers. Each participant performed 6 daily activities across 15 trials under consistent conditions, with a phone placed in the front pocket. Signals from a tri-axial gravimeter, linear accelerometer, gyroscope, and magnetometer were recorded at a sampling rate of 50 Hz. In our experiments, the sensor signals were segmented using a sliding window of 128 readings, corresponding to 2.56 seconds.

Skoda Dataset [170]: The Skoda dataset addresses the problem of recognizing 10 manipulating gestures performed by assembly-line workers in a manufacturing scenario. Each worker’s left and right arms were equipped with 10 accelerometers, with data recorded at a sampling rate of 98 Hz. Following the approach in [reference], we used the data collected from the right arm, resulting in a total of 30 sensor channels. The model is designed to classify the 10 manipulating gestures based on 2-second data segments. L-sign

Letters of Polish Sign Language Alphabet Dataset (L-sign) [123]: The L-sign dataset contains data obtained by measuring hand movements while performing the letters of the Polish Sign Language alphabet. The dataset includes data from 16 users, each performing all 36 letters ten times. Data collection was conducted using 10-channel finger piezoresistive sensor readings, a 3-channel gyroscope sensor, and a 3-channel accelerometer sensor, resulting in a total of 16 channels. The sampling rate is 25 Hz. Each gesture execution is recorded in 75 samples, meaning the data have been segmented into 3-second intervals.

The sensor placement locations and modality counts for each dataset are visualized in Figure 2.4.

2.4.2 Evaluation Protocol

The performance of all models is evaluated using the Leave-One-Subject-Out (LOSO) Cross-Validation (CV) methodology. In each iteration of the CV pro-

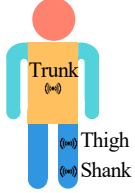
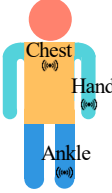
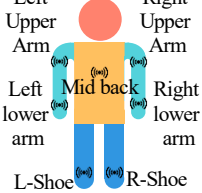
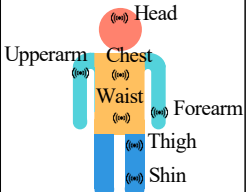
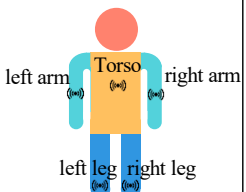






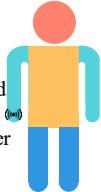
 <p>Trunk (=)</p> <p>Thigh (=)</p> <p>Shank (=)</p> <p>DG</p>	 <p>Chest (=)</p> <p>Hand (=)</p> <p>Ankle (=)</p> <p>PAMAP2</p>	 <p>Left Upper Arm (=)</p> <p>Right Upper Arm (=)</p> <p>Left lower arm (=)</p> <p>Right lower arm (=)</p> <p>Mid back (=)</p> <p>L-Shoe (=)</p> <p>R-Shoe (=)</p> <p>OPPO</p>
 <p>Head (=)</p> <p>Upperarm (=)</p> <p>Chest (=)</p> <p>Waist (=)</p> <p>Forearm (=)</p> <p>Thigh (=)</p> <p>Shin (=)</p> <p>RW</p>	 <p>left arm (=)</p> <p>right arm (=)</p> <p>left leg (=)</p> <p>right leg (=)</p> <p>DSADS</p>	 <p>leg pocket (=)</p> <p>WISDM</p>
 <p>Waist (=)</p> <p>HAPT</p>	 <p>Smartwatch (=)</p> <p>GesHome</p>	 <p>10x IMU (=)</p> <p>Skoda(r)</p>
 <p>Front Pocket (=)</p> <p>MotionSense</p>	 <p>Right Wrist (=)</p> <p>Left Ankle (=)</p> <p>MHealth</p>	 <p>Hand (=)</p> <p>+ (=)</p> <p>Finger (=)</p> <p>L-Sign</p>

Figure 2.4: This figure shows the sensor placement locations for each dataset. In this case, sensor modality is categorized by placement. The body is divided into four regions: head, torso, upper limbs, and lower limbs, each represented by a different color.

cess, data from one subject is designated as the test set, while data from all remaining subjects constitute the training/validation set. The model is trained on the training set, and the validation set is used to identify the best-performing model, which is then tested on the test set. A fixed 9:1 ratio is maintained between the training and validation sets. This CV process is repeated until data from every subject has been utilized as test data.

For the segmentation of the training set, we employed a Semi-Non-Overlapping-Window strategy [73]. Specifically, the training data were split using a sliding window with a 50% overlap between adjacent windows. In contrast, for the test data, the window was slid forward by one time step [185]. It is important to note that before segmentation, the data from each channel were normalized using the z-score method.

Given the imbalanced nature of the HAR datasets, the macro-average F1 score ($F1_M$) is selected as the evaluation metric. At the conclusion of the LOSO-CV process, the mean $F1_M$ is calculated across all subjects. This entire LOSO-CV procedure is repeated five times for each dataset with random seeds of 1, 2, 3, 4, and 5. The mean and standard deviation of the mean $F1_M$ across these five repetitions are calculated and reported as the final performance.

An exception is the Skoda dataset, which contains data from only a single subject. In this case, a hold-out evaluation is performed, following [3], where the first 80% of the data is used for training, the next 10% for validation, and the final 10% for testing.

2.4.3 Training Procedures

All models are implemented using PyTorch [121]. Training is conducted using the Adam optimizer [80], with its default parameters as defined in PyTorch 1.9.1, starting with a learning rate of $\xi = 10^{-4}$. The learning rate is reduced by a factor of 0.9 if the validation loss does not improve after a patience threshold of 10 epochs. The maximum number of training epochs is set to $epoch_{\max} = 200$, with a fixed batch size of 256. All models were trained on a single NVIDIA A100 40G GPU. Early stopping is employed, halting the training if the validation loss does not improve for 15 consecutive epochs.

3 Data Preparation

In this chapter, we discuss how to achieve lightweight models through data processing approaches. In Section 3.1, we review related work on HAR data processing, highlighting the challenges associated with data processing in HAR tasks. In Section 3.2, we introduce methods to enhance data quality through data augmentation, demonstrating how, with the help of data augmentation, a model can achieve SOTA performance even when it has a simple structure or its size is manually reduced. In Section 3.3, we explore how data transformation can be leveraged to develop lightweight models, proposing a learnable wavelet layer as a key component. Finally, in Section 3.4, we provide a summary of this chapter.

3.1 Related Works

In most studies, HAR data is typically standardized using z-score normalization before being directly fed into the model. Additionally, some approaches involve further processing steps [178]. Firstly, high-frequency noise is suppressed by passing the original signal through a 3rd-order low-pass Butterworth filter [129] with a cutoff frequency of 20 Hz, which has been considered adequate for capturing human body motion [77]. Subsequently, additional signals are generated using two methods: the Differencing Time Series (DTS) method and the Separating Movement and Gravity Components (SMGC) method [6]. Figure 3.1 presents an example of generating additional signals from the raw x-axis accelerometer signal after applying both processing methods. The DTS method involves calculating the differences between two consecutive temporal observations, effectively eliminating stationary components and accentuating temporal dependencies and fluctuations. This approach reduces bias and produces more informative features related to activity

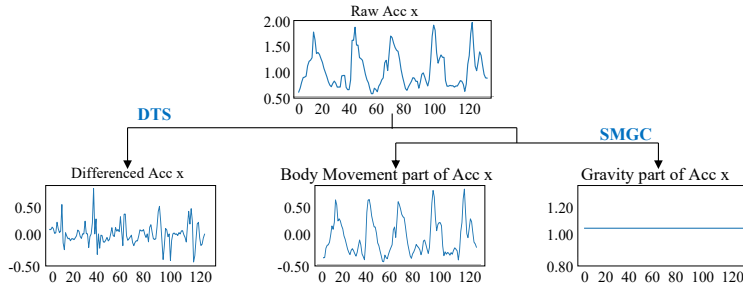


Figure 3.1: New signal channels are generated using the Differencing Time Series (DTS) method and the Separating Movement and Gravity Components (SMGC) method.

transitions and dynamic patterns. For acceleration signals captured by sensors—which include both movement due to human activities and static acceleration due to gravity—the SMGC method is employed to separate these components. Following the approach in [6], a Butterworth low-pass filter with a cutoff frequency of 0.3 Hz is used. The movement component contains information related to the intensity, frequency, and changes in movement, while the gravity component provides insights into body posture or orientation.

However, the aforementioned procedures represent relatively simple data processing methods. In contrast, data augmentation and transformation frameworks have been extensively studied and are routinely used in NLP [47] and computer vision model training [135]. These methods have become standard practices in those fields, yet they are rarely applied in HAR tasks [69; 156; 3]. In the following sections, we will review the related work on data augmentation and transformation in HAR tasks.

3.1.1 Data Augmentation

To improve data quality and enhance the generalization performance of models, data augmentation has emerged as a promising solution. data augmentation involves enriching datasets by creating virtual training samples through various transformations of the original data. Existing data augmentation methods

for HAR tasks can be broadly classified into two categories: traditional and advanced [156].

3.1.1.1 Traditional Approaches. Traditional data augmentation techniques for HAR tasks typically involve random signal transformations such as adding noise, window slicing, magnitude scaling, and random warping [27; 4; 75]. However, these methods are not specifically designed for HAR tasks and often lack awareness of the target task and data characteristics [156]. Consequently, the implementation of inappropriate data augmentation methods can distort the original data characteristics, potentially altering the label semantic information [71] and negatively impacting model performance [69]. These observations underscore the importance of selecting appropriate data augmentation methods that minimize the distortion of the original data characteristics.

To address this, w-augment [49] proposed sample-adaptive automatic weighting schemes that learn the contribution of each random transformation, enabling the exclusion of excessive or redundant methods. However, this approach was not specifically assessed on HAR data. Additionally, while it learns the importance of individual random transformations, it does not explore their combination. Research has shown that combining multiple augmentation methods can lead to improved performance [149; 156; 71]. However, these studies typically involve manually predefined combinations of methods. Identifying the optimal set of augmentation methods is a combinatorial problem, which is NP-hard and requires substantial computational resources.

To mitigate the issue of label semantics distortion arising from random transformations, Abedin et al. [1] proposed the MixUp data augmentation method, which randomly linearly mixes two data samples and their labels to generate virtual data. Another approach, ALAE-TAE-CutMix [3], employs a different data mixing technique by randomly replacing sub-segments in one data sample with corresponding sub-segments from another sample. However, these methods face challenges in generating a diverse range of augmented data, especially considering variations in subjects, activities, sensor placements, and sensor elasticity [143]. The substantial intra-class and inter-subject variability prevalent in HAR datasets implies that relying solely on training set augmen-

tation through mixing techniques may not adequately bridge the distributional gap between training and test sets. As a result, the exploration of supplementary data augmentation techniques capable of increasing diversity and mirroring real-world scenario variability becomes indispensable.

3.1.1.2 Advanced Approaches. Advanced approaches for data augmentation primarily involve synthesizing data using generative models. For instance, Goubeaud et al. [52] trained a variational autoencoder (VAE) on the original data and subsequently generated new samples to augment the training set. Similarly, ActivityGAN [90] employed a generative adversarial network (GAN)-based approach to generate new training samples.

Another emerging class of generative models is the diffusion model, increasingly applied in the image domain and recently adapted for HAR. This includes applications in WiFi channel state information-based HAR [64], and wearable-based HAR [191; 134]. Shao et al. [134] reconfigured the U-net architecture for the denoising model to assess the efficacy of diffusion-based models, while Zuo et al. [191] conditioned the diffusion model on statistical information to generate diverse synthetic sensor data.

However, a noteworthy limitation of these approaches is that they did not train the virtual data generator in an end-to-end manner alongside the HAR model, potentially resulting in sub-optimal performance. To address this issue, the Sample Fusion Network (SFN) [111] cascaded a long short-term memory (LSTM) autoencoder (AE) network to the HAR network, employing a data mixing style to create a combined network that can be trained in an end-to-end manner.

While these advanced approaches offer data- and task-dependent advantages, it is important to acknowledge that they tend to be computationally expensive [69; 156]. Successfully applying these advanced techniques often requires a high level of expertise in model structure design and training configuration to ensure the quality of synthetic samples. To mitigate the need for such expert knowledge, fields like image processing have extensively explored automated optimization frameworks [29; 59]. However, these frameworks have not been explicitly applied to the HAR field. Furthermore, unlike im-

age data, HAR data involves more complex modalities and is more sensitive to perturbations, factors that have not been adequately addressed in previous research [29; 59].

3.1.1.3 Challenges. Based on the review of data augmentation techniques in HAR tasks, we identify four primary challenges in applying data augmentation to the HAR domain:

Inter-Class Similarity Challenge: HAR datasets often exhibit significant similarities between different activities [24], making it difficult to apply random transformations without altering activity labels. For example, Jeong et al. [71] demonstrated how augmenting a "walking" segment could lead to a misclassification as "jogging." Aggressive or inappropriate augmentations can bias the data and diminish model performance[71; 69; 149]. The key challenge is selecting data augmentation methods that preserve the data's characteristics and the semantic nuances of activities.

Data Augmentation Combination Challenge: Research [75; 149; 156] has shown that combining multiple data augmentation methods can significantly improve HAR performance. However, the large number of available methods [69] makes exhaustive experimentation with all combinations impractical. Automating the selection and integration of data augmentation algorithms for HAR tasks remains an unresolved issue that requires further investigation.

Multi-Modality Challenge: Data captured from different body locations using various sensors (e.g., accelerometers, gyroscopes) provide unique insights into activities. Applying transformations indiscriminately across modalities can compromise their effectiveness.

Intra-Class and Inter-Subject Variability Challenge: The variability in data leads to a distribution discrepancy between training and test datasets, causing HAR model performance to decline when applied to new subjects. Although many data augmentation methods aim to generate diverse synthetic data that approximates the original data distribution, they often do not account for the differences between training and test distributions, potentially failing to capture the variations encountered in test scenarios, as illustrated in the Figure 3.2.

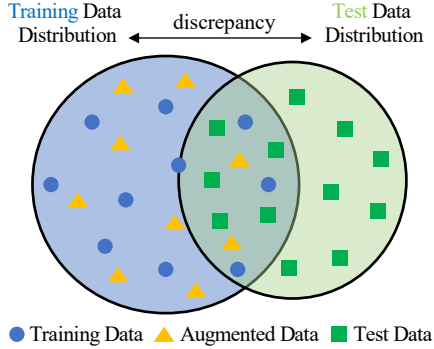


Figure 3.2: Data augmentation is a process that generates additional data points in alignment with the original training data distribution. However, in HAR datasets, a significant train-test distribution discrepancy often exists. Therefore, it is crucial to identify augmentation policies that generate data points more representative of both (train and test) data distribution, thereby improving the model’s generalization ability.

In response to these challenges, we propose AutoaugHAR, an automated data augmentation optimization framework, which will be introduced in Section 3.2.

3.1.2 Data Transformation

Signals acquired through wearable sensors often exhibit multi-frequency, non-periodic, and fluctuating characteristics [24]. Therefore, frequency features are imperative for differentiating human activities. Although frequency information can be implicitly modeled within DL models using time-series representations [164], according to the frequency principle [164], DL models face challenges in effectively learning and generalizing from high-frequency data.

Directly utilizing frequency representations offers richer frequency features that may be imperceptible in the time domain. This approach enables the model to learn the evolution of frequency amplitude over time. Given these considerations, frequency representations have become increasingly popular in research

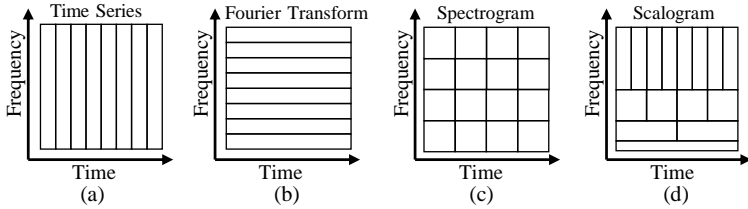


Figure 3.3: The diagram illustrates the time and frequency resolutions of different representations.

efforts focused on HAR tasks [99; 167; 102; 23]. Two widely adopted methods for converting time-domain representations to frequency-domain representations are the use of Short-Time Fourier Transform (STFT) to create spectrograms, and Wavelet Transform to generate scalograms.

3.1.2.1 Short-Time Fourier Transform. The time and frequency resolutions of different representations are illustrated in Figure 3.3. The size and orientation of the blocks indicate the granularity of features that can be distinguished in the time and frequency domains. As observed from the Figure 3.3 (a), the time-series representation provides high resolution in the time domain but offers no information in the frequency domain. However, when the time series is transformed into a frequency representation using Fourier Transform, the frequency representation loses temporal information while gaining high resolution in the frequency domain, as observed from the Figure 3.3 (b). This indicates that the Fourier Transform eliminates the time-dependency of frequency information.

The general rule is that this approach, utilizing the Fourier Transform, performs effectively when the frequency spectrum is stationary, meaning that the frequencies present in the signal remain constant over time. However, HAR data are often non-stationary or dynamic signals. Consequently, many HAR researchers have adopted the STFT to address this issue. In this approach, the original signal is divided into T intervals of equal length τ , without overlap [99; 167]. If the length of the time series is L , then the relationship between T and τ is given by $T = L/\tau$. Each interval then undergoes an FFT transforma-

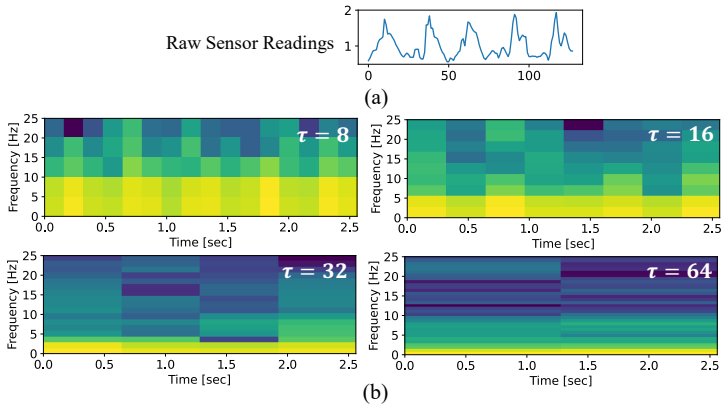


Figure 3.4: STFT with different interval length τ .

tion, with f amplitude and phase spectral pairs computed from each interval. Typically, the value of f is equal to τ . This process yields a representation known as a spectrogram. As shown in Figure 3.3 (c), the resolution in both time and frequency domains depends on the parameters T and τ . Figure 3.4, based on data from an x -axis accelerometer, illustrates the effect of varying parameter settings in the STFT on the resulting representations. When the interval length τ is small, the resolution in the frequency dimension is low, but the resolution in the time dimension is high. As τ increases, finer details are captured in the frequency dimension, while the time-dependent information becomes increasingly blurred.

3.1.2.2 Wavelet Transformation. The Fourier Transform uses a series of sine waves with different frequencies to analyze a signal, representing it as a linear combination of these sine waves. In contrast, the Wavelet Transform employs wavelet functions to capture frequency information [174]. Wavelet functions are scaled—either compressed or stretched—using a scale parameter to capture frequency information across different ranges. Small scales correspond to high frequencies, while large scales correspond to low frequencies. The representation obtained through the Wavelet Transform is known as a scalogram.

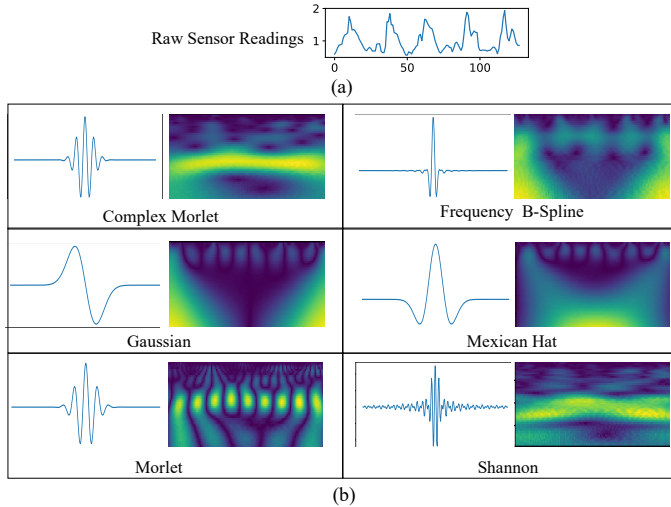


Figure 3.5: This figure illustrates six different representative mother wavelet functions. The results of transforming the same sensor readings using these mother wavelet functions are visualized to the right of each corresponding wavelet. It is evident that the resulting representations vary significantly.

As illustrated in Figure 3.3 (d), the scalogram differs from the spectrogram in that it provides high resolution in the frequency domain and low resolution in the time domain for small frequency values. Conversely, for large frequency values, it offers low resolution in the frequency domain and high resolution in the time domain. This reflects a trade-off inherent in the Wavelet Transform.

To effectively utilize the Wavelet Transform, various families (types) of wavelets are available. Some representative wavelets are shown in Figure 3.5. These wavelet families differ from each other based on the specific trade-offs made regarding the compactness and smoothness of the wavelet. Each family optimizes different aspects of the wavelet's shape to suit particular analysis needs.

3.1.2.3 Challenges. A challenge in implementing spectrogram or scalogram representations lies in defining the appropriate hyper-parameters. For instance, when calculating the spectrogram representation, the Fourier Transform is executed to calculate the frequency over a short time interval with length τ . According to the uncertainty principle [124], determining the optimal interval length τ is complex: A smaller window provides more precise information about the timing of frequency occurrences in the signal but offers less accuracy regarding the frequency value itself, potentially losing information about long temporal dependencies. Conversely, a larger interval size provides more accurate information about the frequency value but less about its exact timing. Activities such as falling or stumbling, which exhibit abrupt and sharp changes in time-series representation, may become blurred or even erased in spectrogram representations. In previous HAR studies that utilized spectrograms as inputs, the specifics of how these hyperparameters were defined are often not introduced [167; 99; 102; 23].

Similarly, when calculating scalogram representations, the choice of the mother wavelet function is critical. As shown in the Figure 3.5, there is a significant difference in the visualizations of the results after applying different wavelet transforms to the data. Experimental findings in [114] indicate that the selection of the mother wavelet significantly impacts the results. To address this issue, several works [175; 109] have utilized multiple wavelets in combination with CNN for HAR tasks. However, these approaches typically employ only a limited number of pre-selected wavelets (one wavelet in [175] and seven wavelets in [109]), and importantly, the wavelets in these works are not learnable. A framework for learnable "wavelet" filters was proposed in [130], but it only preserves the form of wavelet transformation (i.e., correlation and down-sampling). The necessary properties of wavelets, such as bi-orthogonality and energy conservation, are not guaranteed in this framework.

To mitigate the impact of these hyperparameter choices on model performance, we propose a learnable wavelet layer for HAR models, which is introduced in Section 3.3.

3.2 Data Augmentation AutoAugHAR

This section explores the impact of data augmentation on the performance of HAR models. Considering the various challenges associated with applying data augmentation to HAR tasks, as discussed in Section 3.1.1.3, we propose a framework for the automated optimization of data augmentation policies, named AutoAugHAR [188].

3.2.1 Preliminaries

Before introducing the AutoAugHAR framework, the foundational concepts essential for the implementation of a self-optimizing generalizable data augmentation are presented. In particular, we introduce the concept of data augmentation sub-policies and formally define the corresponding optimization problem.

3.2.1.1 Background. Let \mathcal{O} represent a set comprising m candidate time series processing operations. Each operation o within \mathcal{O} represents a function capable of transforming the time series samples, given by $\tilde{x} = o(x)$. An augmentation sub-policy, denoted as s and composed of n consecutive transformations, is expressed as:

$$\tilde{x} = s(x) = o_n(\cdots o_2(o_1(x))) \quad (3.1)$$

In this formulation, each operation is sequentially applied to the time series sample x . To implement an augmentation sub-policy, it is necessary to determine the number of consecutive operations n and identify the appropriate operation from the set \mathcal{O} for each transformation step. For instance, Chung et al. [27] utilized a single transformation method jittering, which means the number of consecutive operations $n = 1$ and the $op = \text{"jittering"}$. And Um et al. [149] utilized an augmentation sub-policy encompassing three consecutive operations ($n = 3$), which are $o_1 = \text{rotation}$, $o_2 = \text{permutation}$, and $o_3 = \text{time-warping}$, respectively.

Let \mathcal{S} symbolize the set of all possible augmentation sub-policies. Given m candidate operations and each augmentation sub-policy comprising n consecu-

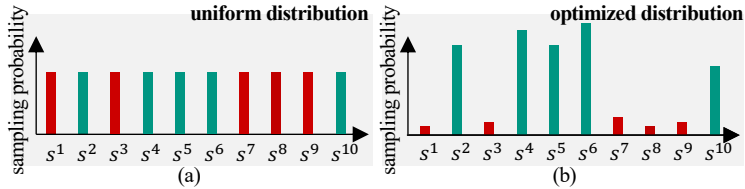


Figure 3.6: Categorical distribution of 10 augmentation sub-policies. Red bars indicate sub-policies with a negative influence, while green bars represent those with a positive impact.

tive transformations, the total number of sub-policies is $L_s = |\mathcal{S}| = P_m^n$, where $\mathcal{S} = (s^1, s^2, \dots, s^{L_s})$. Each augmentation sub-policy possesses a distinct combination of the candidate operations. However, a brute-force approach to explore all various augmentation sub-policies demands significant experimentation and computational resources, rendering it inefficient and sub-optimal.

3.2.1.2 Naive Approach. To address the challenge of efficiently selecting effective augmentation sub-policies, we use a baseline approach, that is commonly used data augmentation mechanism in computer vision model [30]. Instead of using only a singular, predefined augmentation sub-policy, the entire set of possible sub-policies \mathcal{S} is employed in a stochastic manner during the training phase. During the augmentation phase of each mini-batch iteration, a single augmentation sub-policy is randomly selected from \mathcal{S} and subsequently applied to the mini-batch data.

Let the categorical distribution p represent the likelihood of each sub-policy being sampled. In this context, $p = [p^1, p^2, \dots, p^{L_s}] = [\frac{1}{L_s}, \frac{1}{L_s}, \dots, \frac{1}{L_s}]$ follows a uniform probability distribution, providing an equal chance for all augmentation sub-policies to be utilized. Thus, the entirety of possible augmentation sub-policies is leveraged, offering a significantly increased diversity of the augmented data. Furthermore, this approach mitigates potential model degradation that could arise from the utilization of predetermined, sub-optimal augmentation sub-policies.

3.2.1.3 Data Augmentation Optimization. The naive approach adopted a uniform probability distribution, treating all augmentation sub-policies equally without considering their potential positive or negative impacts, as demonstrated in Figure 3.6 (a). Ideally, augmentation sub-policies that exert positive influences should be assigned higher probabilities. Therefore, the primary objective of this study is to automatically optimize the categorical distribution for all augmentation sub-policies, as illustrated in Figure 3.6 (b). Moreover, this optimization process should be executed without incurring excessive training overheads or necessitating model modifications.

3.2.2 Methodology

To effectively optimize the categorical distribution (combinatory space) of sub-policies, we present a two-stage gradient-based framework, termed AutoAugHAR. This framework takes into account multi-modality characteristics inherent in HAR tasks. Optimization of the categorical distribution and weights of the HAR model is performed end-to-end (see Section 3.2.2.1) using gradient descent (Section 3.2.2.2). We have designed a HAR specific search space of data augmentation operators that constitute the augmentation sub-policies (Section 3.2.2.3).

3.2.2.1 Overview of AutoAugHAR. During the training phase, data samples are subjected to data augmentation transformations before being input into the HAR models (see augmentation layer preceding the HAR model in Figure 3.7). Contrary to prior studies, this framework incorporates a total of L_s distinct augmentation sub-policies. For each mini-batch iteration, only a sub-policy is selected and executed, ensuring that memory consumption aligns with traditional HAR model training paradigms. The selection is determined by sampling the sub-policy in accordance with the categorical distribution $p = [p^1, p^2, \dots, p^{L_s}]$. Each element p^i represents the probability of selecting the i -th sub-policy in the sampling process:

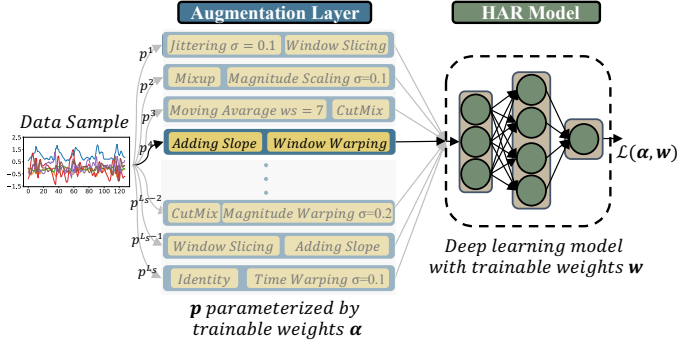


Figure 3.7: An overview of the proposed AutoAugHARbasic. Every channel across all modalities undergoes the same augmentation sub-policy. During the data propagation phase, only one augmentation sub-policy is chosen and applied. The selection of this sub-policy depends on the probability p^i associated with each path. The objective of the optimization is to ensure that sub-policies which improve performance have a higher probability compared to those that affect performance.

$$c = \text{sample} \left(p^1, p^2, \dots, p^{L_s} \right) = \begin{cases} [1, 0, \dots, 0] \text{ with probability } p^1 \\ \dots \\ [0, 0, \dots, 1] \text{ with probability } p^{L_s} \end{cases} \quad (3.2)$$

Upon sampling, the path within the augmentation layer is represented as a one-hot encoded vector, denoted as c , contingent on the associated probability:

$$\tilde{x} = \sum_{i=1}^{L_s} c^i s^i(x) = \begin{cases} s^1(x) \text{ with probability } p^1 \\ \dots \\ s^L(x) \text{ with probability } p_s^L \end{cases} \quad (3.3)$$

At any given instance, only a single augmentation sub-policy path is activated. If the categorical distribution of sub-policies p is uniformly distributed and remains unoptimized, this procedure mirrors the baseline approach de-

tailed in section 3.2.1.2. For clarity in subsequent experimental comparisons, we refer to this framework as **AutoAugHARrandom**.

Categorical distribution p is typically parameterized by a learnable vector $\alpha = [\alpha^1, \alpha^2, \dots, \alpha^{L_s}]$. Subsequently, the distribution p is derived by applying the softmax function over α leading to the following probability of selecting the i -th sub-policy:

$$p^i = p_\alpha(s = s^i) = \text{softmax}(\alpha^i; \alpha) = \frac{\exp(\alpha^i)}{\sum_{j=1}^{L_s} \exp(\alpha^j)} \quad (3.4)$$

α_i signifies the importance attributed to the i -th sub-policy: a relatively large value of α_i indicates a higher likelihood for selecting the corresponding i -th sub-policy. We define the resulting optimization problem as follows:

$$\min_{(\alpha, w)} \mathcal{L}(\alpha, w) \quad (3.5)$$

Both α and w are subjected to end-to-end training, minimizing the loss function. The "sample" operation in equation 3.2 introduces a discontinuity, thereby inhibiting the propagation of gradients to the weights α (solution please refer to section 3.2.2.2). While this optimization strategy enables dynamic weights optimization for augmentation sub-policies during the model's training phase, it still overlooks the distinct attributes of the candidate data augmentation operations within set \mathcal{O} and the multi-modality characteristics of HAR tasks. (We refer to this framework as **AutoAugHARbasic**.)

Given two distinct categories of candidate data augmentation operations, each with varying capacities to preserve label semantics (refer to Section 3.2.2.3), and aiming to optimize augmentation sub-policies for each modality, we revised the structure. The updated framework, named **AutoAugHAR**, is depicted in Figure 3.8. **AutoAugHAR** operates in two stages. The first mixing stage transforms the data using label-preserving sample-pair-based mixing techniques like MixUp [1] and CutMix [3]. The second stage, which has a slightly higher risk of compromising label semantics, incorporates random data augmentation augmentations to further enrich the diversity of the data.

In contrast to **AutoAugHARbasic**, which optimizes p universally across all modalities, **AutoAugHAR** tailors the distribution of augmentation sub-policies

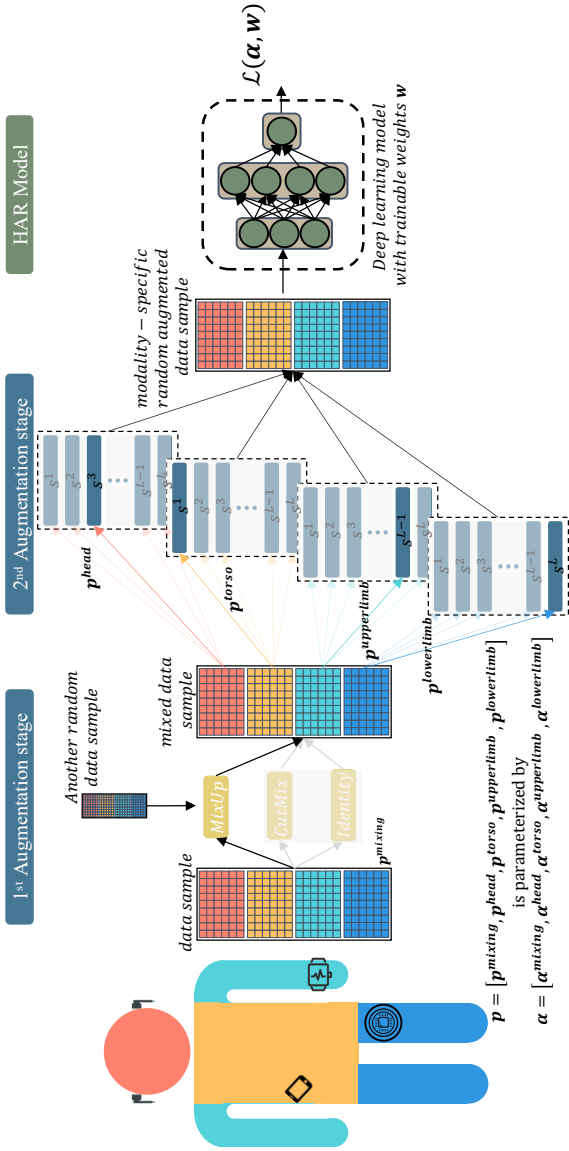


Figure 3.8: Overview of the proposed AutoAugHAR framework. Different colors represent data from various modalities. In the first stage, data from all modalities are transformed using the same selected operator. In the second stage, data from each modality is individually transformed by different operators and subsequently integrated.

for each modality in the second stage individually. Modalities are classified based on sensor placement, e.g. head, upper limb, lower limb, and torso. During the forward pass of the second stage, data samples are partitioned according to the modality. For each, a path is sampled independently, with the corresponding augmentation sub-policy applied. Subsequently, data from all modalities are integrated and fed to the HAR model. We did not perform individual modality-wise optimization in the first stage. This is attributed to the incompatibility of sample-pair-based mixing techniques for such a purpose (see section 3.2.2.3).

For optimization, separate categorical distributions are initialized for the first mixing stage and each modality of the second stage. These separate categorical distributions are denoted as $(p^{mixing}, p^{head}, p^{upperlimb}, p^{lowerlimb}, p^{torso})$ with learnable vectors $(\alpha^{mixing}, \alpha^{head}, \alpha^{upperlimb}, \alpha^{lowerlimb}, \alpha^{torso})$. The size and search space of these distributions are detailed in section 3.2.2.3. During the forward pass, within each categorical distribution, a path will be sampled using equation 3.2. The corresponding sub-policy is then applied to the data according to equation 3.3. Upon optimization, the first mixing stage as well as all individual modalities of the second stage, will acquire a specifically optimized distribution for applying data augmentation sub-policies. By leveraging this two-stage structure, **AutoAugHAR** seeks to preserve the label semantic information of the data while generating optimized categorical distribution of augmentation policies intrinsic to each modality, thus improving the overall performance.

3.2.2.2 Gradient Based Optimization. The loss function in equation 3.5 is differentiable with respect to the model weights w , allowing optimization via stochastic gradient descent. However, the loss is not directly differentiable with respect to the sampling parameter α , because the discrete "sample" operation introduces non-differentiable points in the network. This section will describe how to optimize the weights α using a gradient descent approach. Because the forward process and gradient back propagation process of all categorical distributions parameterized by α are same, we decide to omit the use of superscripts (head, upperlimb, lowerlimb and torso) to explain these processes.

In order to facilitate back-propagation through these non-differentiable operations, Straight-Through Estimators (STE) [16] are employed. The basic idea behind STE is to provide a way to back-propagate gradients through these non-differentiable operations while maintaining their original behavior during forward pass and avoiding any gradient vanishing or exploding issues during the backward pass. In order to obtain a differentiable approximation, we apply the Straight-Through Gumbel-Softmax Estimator [70]. Compared to sampling the path with equation 3.2, the Gumbel-Max trick [56; 103] provides a different way to sample the path (sub-policy):

$$c = \text{Onehot_Encoding} \left(\underset{i}{\operatorname{argmax}} \left(g^i + \log \left(\alpha^i \right) \right) \right) \quad (3.6)$$

where g^i are independent samples drawn from a standard Gumbel distribution $g^i \sim \text{Gumbel}(0, 1)$. The reparameterization trick refractors the sampling of c into a deterministic distribution function using α and independent noise g from a fixed distribution, maintaining an identical sampling procedure using equation 3.2. This technique avoids having to back-propagate through the stochastic node g and instead only back-propagate into the deterministic distribution function, updating the parameters α .

During the gradient back-propagation, *argmax* operation is still not differentiable. To address this issue, a differentiable approximation of *argmax* is needed. Gumbel *softmax* [70] offers a differentiable approximation to *argmax*, as utilized in various works [159; 162; 41; 91]:

$$p^i = \text{GumbelSoftmax} \left(\alpha^i; \alpha \right) = \frac{\exp \left((\log \left(\alpha^i \right) + g^i) / \theta \right)}{\sum_{j=1}^L \exp \left((\log \left(\alpha^j \right) + g^j) / \theta \right)} \quad (3.7)$$

θ is the temperature parameter that controls the fidelity of the approximation to discrete one-hot vectors. Consequently, this allows the model to be trained with discrete operations, using equations 3.6 and 3.3 for the forward pass and the differentiable equation 3.7 for gradient back-propagation.

Following the exploration of the differentiable optimization problem, we now present the entire optimization process. To evaluate whether a categorical distribution for augmentation sub-policies is good or not good, it is needed

to train the HAR model to converge to obtain the optimal model weights, $w^*(\alpha) = \operatorname{argmin}_w \mathcal{L}_{train}(w, \alpha)$. The optimal weight w^* is affected by the categorical distribution parameterized by α , if α changes, the corresponding optimal w^* will also change. This implies a typical bi-level optimization problem [171; 97; 5] with α as the upper-level variable and the model weight w as the lower-level variable, mathematically defined as follows.

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) \text{ s.t. } w^*(\alpha) = \operatorname{argmin}_w \mathcal{L}_{train}(w, \alpha) \quad (3.8)$$

where \mathcal{L}_{train} and \mathcal{L}_{val} denote the training and validation loss, respectively. The objective is to determine the categorical distribution parameterized by α , which minimizes the validation loss, where the weights of the HAR model w are obtained by minimizing the training loss. Using the performance of the validation set as a reward for updating upper-level variable is a common practice [189; 97; 190; 96]. In this case, minimizing the validation loss through α encourages the optimized categorical distribution for augmentation sub-policies preserve the data semantic information and also fill the gap between seen (training) and unseen (validation) data. This mitigates the risk of generating augmentation sub-policies that might over-fit the training set.

During the training process, w and α are alternately fine-tuned through gradient descent. The training protocol is explained in Algorithm 1. Initially, an augmentation sub-policy is sampled for the initial mini-batch data loading in accordance with equation 3.6 (line 2). Following this step, α undergoes an update through gradient calculations (line 4). Subsequently, the weights w of the model are updated on the basis of the updated α (line 5). Conclusively, the augmentation sub-policy is re-sampled for the forthcoming mini-batch data loading (line 6). This alternating optimization procedure is repeated until the maximum optimization epoch is reached.

However, during the α update step, the calculation of the gradient of α requires a computationally intensive internal optimization. w^* are derived by minimizing training loss. To avoid extensive optimization, a one-step optimization technique is employed to approximate w^* , as outlined below.

Algorithm 1 Training Procedure

Variables:
 α - Categorical distribution for augmentation sub-policies

$$\alpha = (\alpha^{mixing}, \alpha^{head}, \alpha^{upperlimb}, \alpha^{lowerlimb}, \alpha^{torso})$$

 w - weights of the model

 ξ_w - Learning rate for updating w
 ξ_α - Learning rate for updating α
 $epoch_{search}$ - Number of epoch for optimization the data augmentation sub-policies

- 1: **for** $i=1$ to $epoch_{search}$ **do**
 - 2: Augmentation sub-policy sampling
 - 3: **for** Sample a mini-batch of data **do**
 - 4: Update distribution α : $\alpha = \alpha - \xi_\alpha \nabla_\alpha \mathcal{L}_{val}(w^*, \alpha)$
 - 5: Update model weights w : $w = w - \xi_w \nabla_w \mathcal{L}_{train}(w, \alpha)$
 - 6: Augmentation sub-policy sampling
 - 7: **end for**
 - 8: **end for**
-

$$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \tag{3.9}$$

$$\approx \nabla_\alpha \mathcal{L}_{val}(w - \xi_w \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha) \tag{3.10}$$

$$= \nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi_w \nabla_{\alpha, w}^2 \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \tag{3.11}$$

here, $w^* \approx w' = w - \xi_w \nabla_w \mathcal{L}_{train}(w, \alpha)$ is approximated using a single virtual gradient step over the training set. By applying the chain rule for derivatives, equation 3.11 is derived. However, the second term in the equation 3.11 contains an expensive matrix-vector product with a computational complexity of $O(|\alpha||w|)$. Fortunately, the complexity can be reduced using a finite difference approximation. Let ε be a small scalar, and $w^\pm = w \pm \varepsilon \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$, then:

$$\nabla_{\alpha, w}^2 \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\varepsilon} \tag{3.12}$$

The evaluation of the finite difference requires only two forward passes for the weights and two backward passes for α . following the settings in [97; 91], we let $\varepsilon = 0.01 / \left\| \nabla_{w', \alpha} \mathcal{L}_{val}(w', \alpha) \right\|_2$.

Table 3.1: Candidate operators and the settings of hyper-parameters.

<i>Method</i>	<i>Parameter</i>	<i>value/range</i>
Jittering	σ	0.05
Jittering	σ	0.10
Jittering	σ	0.15
Moving Average	ws	3
Moving Average	ws	5
Moving Average	ws	7
Magnitude Scaling	σ	0.1
Magnitude Scaling	σ	0.2
Magnitude Warping	σ	0.2
Magnitude Warping	σ	0.4
Window Slicing	λ	[0.7, 0.9]
Slope-Like Trend	<i>slope</i>	[-0.1, 0.1]
Time Warping	σ	0.1
Time Warping	σ	0.2
Mixup	α	0.3
CutMix	α	0.8

3.2.2.3 Candidate Operations And Search Space. In this section, we present an exhaustive overview of considered candidate augmentation operators that constitute the augmentation sub-policies. Drawing from a thorough review of the relevant literature, we have identified a set of 17 operators, which are both diverse and computationally efficient. All considered candidate data augmentation operators are graphically depicted in Figure 3.9 and their hyper-parameter settings are shown in the Table 3.1. The settings of these hyper-parameters are summarized from related works [1; 3; 49; 69]. It is important to note that the same operators but with different hyper-parameters are regarded as distinct and unique entities. The candidate operators can be systematically categorized into two primary categories: label-preserving augmentation oper-

ators and random transformation operators. In the following sections, we will go into the details of each of these two categories of operators.

Label-preserving Operators. Label-preserving transformation operators aim to produce virtual data that retain the intrinsic characteristics of the original data. A widely applied technique in this regard is the use of sample-pair-based methods, which involves mixing signals and labels from two input data samples.

In the **MixUp** approach, for two given samples (x_1, y_1) and (x_2, y_2) , a virtual training sample (\tilde{x}, \tilde{y}) is created through linear interpolation between the input sample pair as follows: $\tilde{x} = \lambda x_1 + (1 - \lambda)x_2$ and $\tilde{y} = \lambda y_1 + (1 - \lambda)y_2$. The mixing ratio λ is a stochastic value drawn from the beta distribution $\mathcal{B}(\theta, \theta)$, determining the mixing intensity. Adhering to the configuration presented in [1], the parameter θ is specified as 0.3.

In the **CutMix** methodology, segments from two samples are swapped to produce a new virtual sample. Given two samples, x_1 and x_2 , each comprising T time steps. A randomly selected region, spanning a length of λT , is delineated within x_1 , from which the respective sub-segment is cropped. This cropped sub-segment subsequently replaces the counterpart in sample x_2 , leading to the formation of a mixed virtual sample. Unlike the MixUp mechanism, CutMix generates virtual samples without altering the raw data values. The samples generated by CutMix have steep transitions between activities. The label associated with this augmented data is derived from the formula, $\tilde{y} = \lambda y_1 + (1 - \lambda)y_2$. Here, the coefficient λ , which controls the degree of mixing, is also drawn from the beta distribution $\mathcal{B}(\theta, \theta)$. As referenced in [3], the parameter θ is set at 0.8.

In addition to those two operators, the identity operator is also incorporated. Within the first stage, the count of conservative operations n_{1st} is set to 1. Thus, the size of the categorical distribution for this stage is $p^{mixing} \in \mathbb{R}^3$. Modality-wise optimization is not adopted, because the mixing approach does not merely amalgamate data values, but also integrates their corresponding labels, which would become semantically ambiguous.

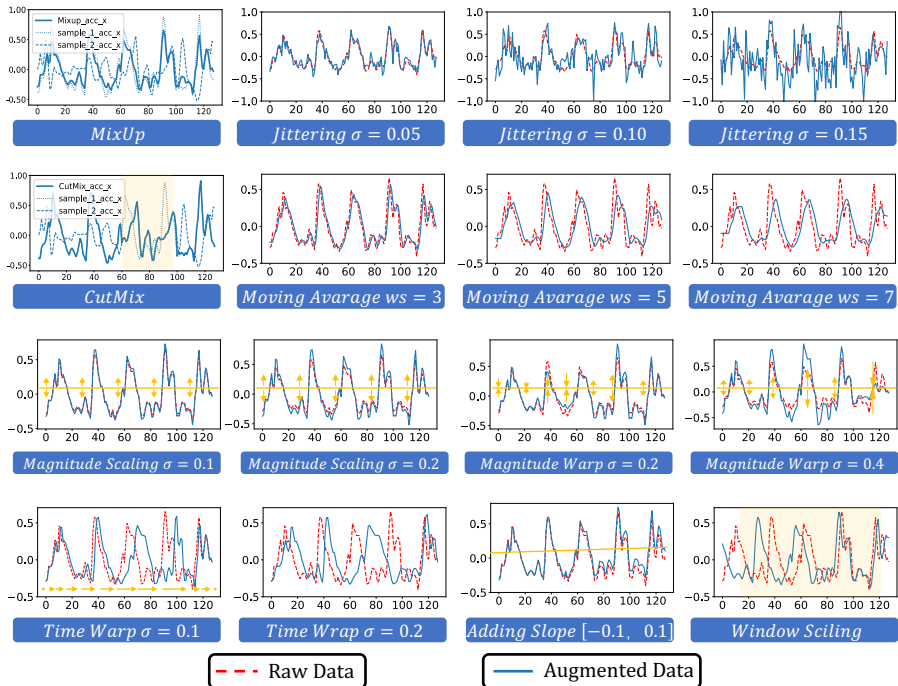


Figure 3.9: Examples of candidate augmentation operators on the HAPT dataset.

Random Transformation Operators. Given that random transformations run the risk of corrupting the original semantic information of the data, in total 14 such candidate operators are employed in the second stage of **AutoAugHAR**.

Adding Noise to data samples, also referred to as jittering, is specifically designed to simulate sensor noise. To implement this method, controlled amounts of random noise are added to the raw data, producing a new representation denoted as $\tilde{x} = x + \varepsilon$, with ε symbolizing the random drawn noise vector from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$. Unlike prior studies, we introduced multiple distinct values of σ , namely 0.05, 0.1, and 0.15, with each corresponding to different noise intensities.

Moving Average involves calculating the average of a sliding window of sensor data over time. It is effective in mitigating the impact of outliers and noise present in the sensor data, but might introduce lag, especially with larger window sizes (ws). The moving average’s inherent smoothing effect can dampen abrupt changes. Therefore, the selection of an appropriate ws becomes critical. We have incorporated three ws : 3, 5, and 7.

Magnitude Scaling is a technique that alters the magnitude of a signal by applying a stochastic scaling factor to simulate variations in the intensity of physical activity observed in real-world scenarios, defined as $\tilde{x} = x \times sf$, where the scaling factor sf is a stochastic variable drawn from a Gaussian distribution $\mathcal{N}(1, \sigma^2)$. We have introduced two scaling ranges, $\sigma = 0.1$ and $\sigma = 0.2$.

Magnitude Warping [10]. In contrast to magnitude scaling, which uniformly scales all values within the signal using the same factor, magnitude warping involves distorting the magnitude of the signal by applying a smoothed curve generated through cubic spline interpolation with K knots. As a result, Magnitude Warping can produce more realistic variations in the intensity of time-series data. To implement magnitude warping, K knots (reference points) are selected along the time-series data, dividing the time-series data into $K - 1$ equal segments. For each knot, a random scaling factor is sampled from a Gaussian distribution $\mathcal{N}(1, \sigma^2)$. These knots, along with their corresponding scaling factors, serve as control points for the subsequent cubic spline interpolation. The original time-series data is then warped by applying the values of the interpolated curve at corresponding time points t . We defined two scaling

ranges, $\sigma = 0.2$ and $\sigma = 0.4$ to avoid unrealistic distortions.

Window Slicing [86], also known as cropping, involves randomly selecting a segments of random length from the original signal. Given a data sample x with L time steps, mathematically, window slicing can be expressed as $\bar{x} = x[t : t + \lambda L]$ at the starting point t with a random segment length λL . We chose to sample segments between 70% and 90% of the original length of the input signal, $\lambda \in [0.7, 0.9]$.

Time Warping [69] involves warping the time steps based on a cubic spline interpolation with K knots, that are first selected along the time-series data, dividing the time-series data into $K - 1$ equal segments. These knots are randomly perturbed by multiplying them by a random factor which is sampled from a Gaussian distribution $\mathcal{N}(1, \sigma^2)$, where sigma controls the strength of perturbations (we use $\sigma = 0.1$ and $\sigma = 0.2$). The original time steps are replaced with warped time steps from the new spline. Values corresponding to new time steps are obtained through interpolation.

Incorporating a Slope-Like Trend involves adding a linear trend to the time series to represent patterns in specific scenarios such as a drift in accelerometer data. The slope is selected randomly from a predetermined range, $[-0.1, 0.1]$.

If operators change the length of the original time series, we interpolate the transformed time series back to the original length. In addition, the identity operator is also incorporated in this step.

In this second stage of **AutoAugHAR**, we set the conservative operation count n_{2nd} to 2. Therefore, there are in total $P_{14+1}^2 - 14 \cdot 3 \times 2 \cdot 3 \times 2 \cdot 2 \cdot 2 = 178$ augmentation sub-policies. Among all possible augmentation sub-policies, we eliminate identical ones. For example, 'identity + jittering' is the same as 'jittering + identity'. We also removed sub-policies with the same operators, such as 'jittering $\sigma = 0.05$ + jittering $\sigma = 0.10$ '. As a result, each modality's categorical distribution size is as follows: $p^{head} \in \mathbb{R}^{178}$, $p^{upperlimb} \in \mathbb{R}^{178}$, $p^{lowerlimb} \in \mathbb{R}^{178}$, $p^{torso} \in \mathbb{R}^{178}$. In this setting, the total number of conservative operations $n = 3 = n_{1st} + n_{2nd}$.

3.2.3 Experiments and Discussions

We hypothesize that AutoAugHAR is more effective than manually selecting existing data augmentation approaches without additional expert-based optimizations. Thus, AutoAugHAR must not perform worse than any of the existing techniques given a specific sensor-based HAR task using DL techniques. To validate the effectiveness and universality of the proposed **AutoAugHAR**, we conducted extensive evaluations.

3.2.3.1 Experiment Setup.

Datasets and HAR Models. Eight datasets are selected to represent a broad spectrum of sensing modalities, sampling frequencies, and activity classifications. These datasets include: **HAPT** [128], **PAMAP2** [127], **OPPO** [21], **RW** [141], **DSADS** [14], **WISDM** [83], **DG** [11] and **GesHome** [115]. Sensors are grouped into different modalities based on their mounting locations. In order to demonstrate the generalizability, we considered five diverse HAR models in the experiments, Multi-branch CNN (**MCNN**) [112], hybrid model DeepConvLSTM (**DCL**) [119], DeepConvLSTM-Attention (**DCL-A**) [113], Attend-Discriminate (**Attend**) [1] and **TinyHAR** [185].

Compared Data Augmentation Approaches. We compare the proposed **AutoAugHAR** against the following seven data augmentation techniques in addition to a baseline (training without DA). These techniques can be categorized into three groups: traditional data augmentation techniques (MixUp, CutMix), generative models (SFN [111], ActivityGAN, SF-DM), and data augmentation optimization framework (*w*-augment). The brief description of each technique is as follows.

SDA [71] introduces a data augmentation pipeline that incorporates time-warping and data masking strategy, drawing inspiration from the SpecAugment method [120] used in language processing. Unlike the original SpecAugment approach, SDA proposes different data masking strategies. Based on their experimental findings, we implemented the random masking strategy.

MixUp [1] and **CutMix** [3] are also included in the candidate operators, please refer to section 4.4.1.2 for more details.

SFN [111] draws inspiration from sample-pair-based augmentation strategies. Instead of using hand crafted techniques like MixUp and CutMix, SFN generates virtual samples using a 4-layer LSTM autoencoder (AE). This LSTM AE is cascaded with the HAR model through a MixUp fusion style to form a unified network that can be trained end-to-end.

ActivityGAN [90]: This method presents a GAN-based framework for generating synthetic sensor-based data. The framework consists of a generator model and a discriminator model. The generator model uses a stack of 1D-convolution and 1D-transposed convolution layers to generate synthetic sensor data, and the discriminator model employs 2D-convolution networks to distinguish between real and synthetic data. After training this GAN-based framework, the generator model is unitized for DA. In the experiments, the configuration of ActivityGAN aligns with the original study’s design as described in [90].

SF-DM [191] proposes an unsupervised statistical feature-guided diffusion model for sensor-based HAR. By conditioning the diffusion model on statistical information, SF-DM can generate diverse and representative synthetic sensor data. The structure of the diffusion model and training setup are consistent with the original paper.

w-augment [49] is very similar to our proposed framework: both are specifically designed to learn the optimal weight of each data augmentation sub-policy during the training phase. In *w*-augment, for all data augmentation sub-policies, a weight vector with a dimension equals to the number of sub-policies is initialized with equal weights. During the optimization process, the training loss is utilized to update the weights of each data augmentation sub-policy. *w*-augment aims to prioritize sub-policies by assigning them larger weights. Following the settings in the original paper, only one-step sub-policies are considered. In this experiment, the included one-step sub-policies for *w*-augment are the data augmentation methods summarized in Table 3.1.

It is worth noting that for all data augmentation techniques, original samples are incorporated into the training process.

3.2.3.2 Comparison to State-of-the-art. The results presented in Figures 3.10 and 3.11 provide an exhaustive comparison of various data augmentation techniques across multiple datasets and models. The bars represent the mean of the macro F1 score ($F1_M$), with the standard deviation indicated above each bar. Each row in the figure corresponds to the performance on a specific dataset. Furthermore, each row is divided into five groups, each representing the performance of one HAR model under different data augmentation algorithms. To determine the statistical difference in performance between the two data augmentation algorithms, we employed the Mann-Whitney U test [157]. Bold items mark the statistically significant best result with a p-value less than 0.05.

Across all datasets and models, applying data augmentation techniques leads to an improvement of performance compared to the baseline that does not incorporate DA. Among the data augmentation techniques examined, AutoAugHAR stood out, achieving the best results in 38 out of 40 comparison experiments. SFN also exhibits commendable performance, followed by SF-DM, MixUp and CutMix. SFN, MixUp and CutMix are sample-pair-based approaches. They generate virtual samples while preserving label semantic information, underscoring the importance of label-preserving data augmentation strategies in HAR tasks.

Before comparing AutoAugHAR with other data augmentation frameworks, we first examine how AutoAugHAR enhances the performance of lightweight HAR models. We begin with the MCNN model, a purely CNN-based model that is highly deployment-friendly. However, due to its simple architecture and the inherent limitations of CNNs in capturing long-term dependencies, its performance significantly lags behind SOTA models such as Attend. This disparity is clearly illustrated in Figure 3.10 and Figure 3.11, where the green bars in the columns representing the MCNN and Attend models show their performance without data augmentation. It is evident that the green bar for Attend is much higher than that of MCNN.

However, after applying the AutoAugHAR framework to the MCNN model (indicated by the pink bars), its performance surpasses that of the Attend model (without data augmentation) on 3 out of 8 datasets (DSADS, RW, and GesHome). Moreover, on the WISDM and DG datasets, the performance gap between

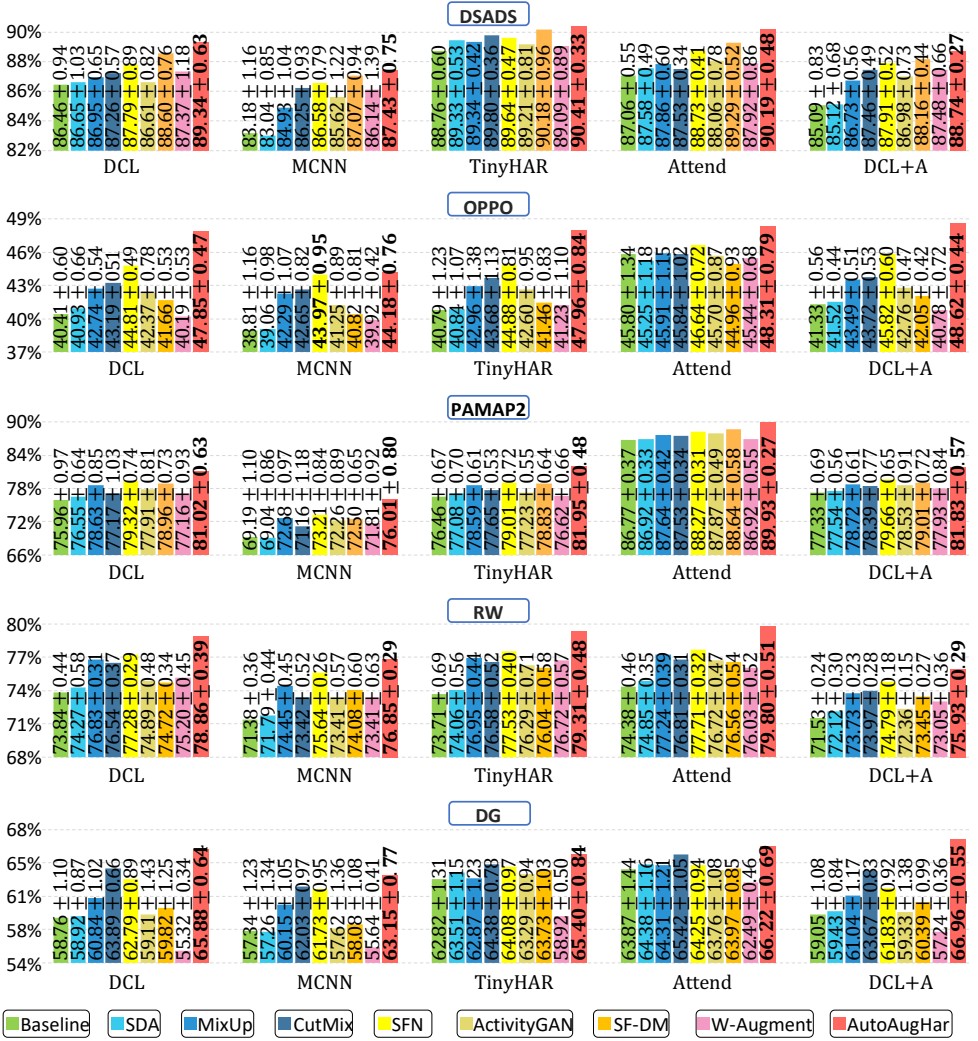


Figure 3.10: Comparison of the classification performance between the proposed AutoAugHar and other SOTA augmentation methods across five datasets, each containing multiple modalities.

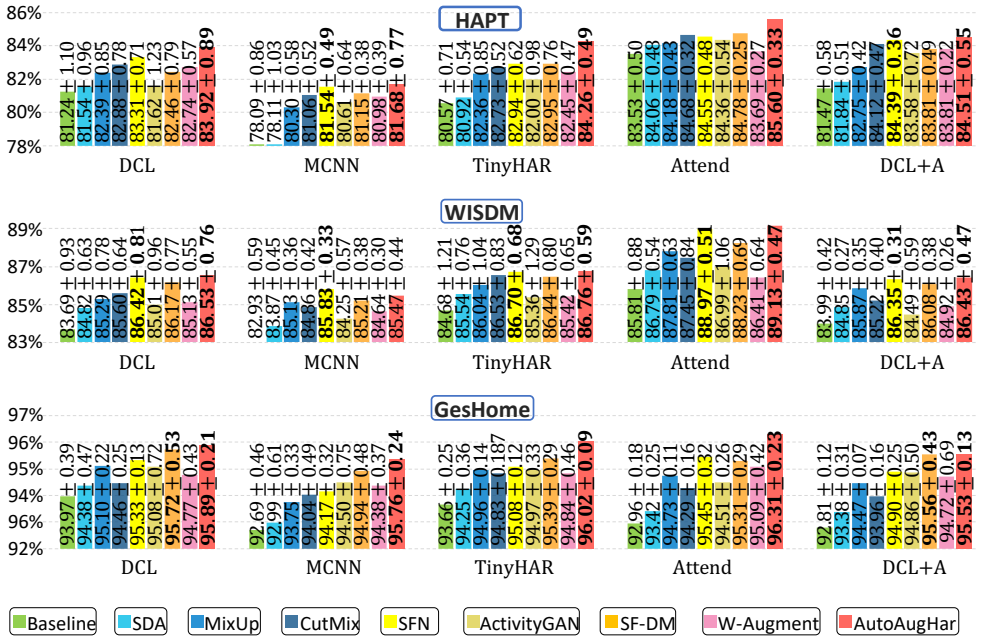


Figure 3.11: Comparison of the classification performance between the proposed AutoAugHar and other SOTA augmentation methods across three datasets, each containing only a single modality.

MCNN (with AutoAugHAR) and Attend (without AutoAugHAR) narrows to less than 1%. This demonstrates that AutoAugHAR can significantly enhance the performance of lightweight models, compensating effectively for their inherent limitations due to smaller model sizes.

A similar trend is observed with the TinyHAR model. Without data augmentation, TinyHAR underperforms compared to Attend on the OPPO, RW, DG, HAPT, and WISDM datasets. However, with the application of the AutoAugHAR framework, TinyHAR outperforms Attend without data augmentation. This underscores AutoAugHAR’s ability to improve model performance, particularly in simple or lightweight models where inherent limitations can be mitigated through the proposed framework. In Section 3.2.3.3, we further reduce the size of the TinyHAR model to investigate the impact of AutoAugHAR on its performance.

Next, we will discuss the performance comparison between AutoAugHAR and other data augmentation frameworks.

The w -augment shows inconsistent results in all datasets, performing particularly poorly on the DG and OPPO datasets. While w -augment and AutoAugHAR both aim to allocate more weight to beneficial augmentation sub-policies during training, they diverge in their augmentation sub-policy weight update procedures. Specifically, w -augment employs training loss for augmentation sub-policy weight optimization, while the proposed AutoAugHAR leverages validation loss. This difference makes w -augment more prone to overfitting, potentially favoring "easy-to-learn" augmentation sub-policies. "Easy-to-learn" augmentation sub-policies might significantly reduce training loss, but they often fail to bridge the distribution gap between training and testing datasets. This shortcoming is especially evident on the OPPO and DG datasets, characterized by challenges such as limited subjects and intra-class variability.

ActivityGAN improves the performance of the model, but it does not reach the extent achieved by AutoAugHAR, SFN, SF-DM, MixUp or CutMix. This can be attributed to the separate training procedures of the generator and the HAR model, potentially leading to sub-optimal solutions. The generator within ActivityGAN is trained to produce virtual samples that match the original data distribution, often neglecting the need to bridge the gap with data not previ-

ously encountered. These virtual samples may not align optimally with the downstream HAR model’s task requirements. In contrast, the standout performance of AutoAugHAR and SFN can be attributed to their end-to-end training approach. This ensures that the virtual data both mirrors the original distribution and improves the model’s performance on unfamiliar data.

Compared to ActivityGAN, another generative model, SF-DM has shown superior performance, especially notable in its outperformance over SFN across three datasets: DSADS, HAPT, and GesHome. However, it still lags behind the proposed model, AutoAugHar. In datasets collected from a relatively larger number of subjects, SF-DM demonstrates its impressive ability to generate diverse and complex data. However, it encounters challenges with certain datasets, particularly the OPPO dataset. This could be attributed to two primary factors. First, the OPPO dataset, which consists of data from only four subjects, exhibits a highly varied data distribution. Similar to ActivityGAN, the separate training of the data generator and the HAR model hinders the use of validation loss as guidance for data generation. Second, the simple denoising model employed by SF-DM fails to deal with datasets such as OPPO, which feature 42 channels and 18 classes. The structure of the model does not adequately address the characteristics of HAR datasets. Literatures [116; 39] suggest that the success of a diffusion-based approach depends greatly on the model’s design and its denoising configuration. Therefore, there is considerable room for improvement in the performance of diffusion-based techniques.

Handcrafted algorithms, MixUp and CutMix, produce consistent results across all datasets. However, their performance varies among different datasets and models. For example, MixUp outperforms CutMix on the RW and PAMAP2 datasets, whereas CutMix excels over MixUp on the OPPO and DSADS datasets. This variability highlights the importance and need for automated data augmentation techniques, such as AutoAugHAR, which autonomously determine the best techniques or combine them.

The effectiveness of the SDN technique, is generally less impressive compared to MixUp and CutMix. SDN achieves only modest improvements on most datasets and can sometimes even reduce model performance. This underperformance is mainly due to SDN’s dependence on a predefined data augmen-

tation policy, which might not be suitable for all scenarios.

AutoAugHAR surpasses SFN on most datasets and models, though it shows comparable performance on the WISDM dataset. We attribute this observation to two main reasons: 1) SFN employs a MixUp-style approach to fuse original samples with virtual samples generated by LSTM AutoEncoder (AE). These though MixUp-style generated samples may appear completely different from the original sequences and become meaningless from a human perspective [3]. As past comparisons between CutMix and MixUp indicate, each sample-pair-based technique has its unique advantages. The adoption of the MixUp-style fusion approach in SFN could potentially reduce its efficacy in specific scenarios. 2) Sample-pair-based algorithms are constrained in their capacity to explore more diverse data domains, being entirely reliant on data sample fusion. In contrast, the proposed AutoAugHAR can effectively select from various sample-pair-based methods and combine them with random transformation methods to produce more diverse data.

In the context of the WISDM and HAPT datasets, AutoAugHAR and SFN exhibit comparable performance. On both datasets, AutoAugHAR and SFN obtained the best performance 6 times without significant differences. These datasets possess shared characteristics, including the utilization of a singular sensor, the categorization of everyday activities, and data sourced from over 30 subjects. These factors make the datasets sufficiently representative for their tasks, eliminating the need for random augmentation to enhance data diversity. Given the singular modality in these datasets, AutoAugHAR can't optimize on a modality basis to boost performance. However, it's worth noting that SFN's integration with an additional LSTM AE encoder into the HAR model enlarges the model's size and necessitates expert knowledge for its design. In contrast, AutoAugHAR doesn't alter the HAR model's structure or increase its size.

3.2.3.3 Model Compression. There are various methods for model compression, including pruning, quantization, and knowledge distillation. A more straightforward approach is to start with a strong baseline model and reduce its size by decreasing the number of layers or filters, similar to the width multiplier or filter multiplier strategies employed in EfficientNet [142]. This method

Table 3.2: Comparison between TinyHAR without AutoAugHAR and the compressed TinyHAR model with AutoAugHAR.

Dataset	TinyHAR without DA	TinyHAR_0.9 with DA	TinyHAR_0.8 with DA
DSADS	88.76± 0.60	89.68± 0.81	88.91± 0.77
OPPO	40.79 ± 1.23	44.32 ± 0.94	42.68 ± 1.08
PAMAP2	76.46 ± 0.67	80.95 ± 0.58	78.14 ± 0.92
RW	73.71± 0.69	78.77± 0.43	75.31± 0.36
DG	62.82 ± 1.31	65.15 ± 1.16	63.82 ± 0.75
HAPT	80.57± 0.71	83.10± 0.67	80.17± 0.49
WISDM	84.68 ± 1.21	85.52 ± 0.83	85.29 ± 1.07
GesHome	93.66 ± 0.25	95.74 ± 0.31	94.43 ± 0.20

requires less expert knowledge compared to other techniques. However, performance degradation is a major concern when compressing models. All compression techniques must aim to strike an optimal balance between efficiency and accuracy. Our proposed AutoAugHAR can be easily integrated with compressed models to mitigate performance degradation.

To demonstrate this, we conducted a series of experiments. Specifically, we selected the TinyHAR model, which is already lightweight, as our baseline. We further reduced the model size by decreasing the number of filters using a scaling factor, resulting in the TinyHAR_0.9 and TinyHAR_0.8 models. Compared to the original TinyHAR, TinyHAR_0.9 retains approximately 82% of the trainable parameters, while TinyHAR_0.8 retains around 64%. If these compressed models can achieve performance levels close to the original, it would validate the effectiveness of the data augmentation strategy in training lightweight models.

Their performance is listed in Table 3.2. As shown, when utilizing data augmentation, the compressed models perform similarly to the original TinyHAR. In most cases, even TinyHAR_0.8 surpasses the performance of the original model. This clearly demonstrates the crucial role of the proposed framework in maintaining high performance, even when the model size is significantly reduced.

3.2.3.4 Ablation Study. In order to assess the contribution of the design underlying the proposed AutoAugHAR, three variants of AutoAugHAR were

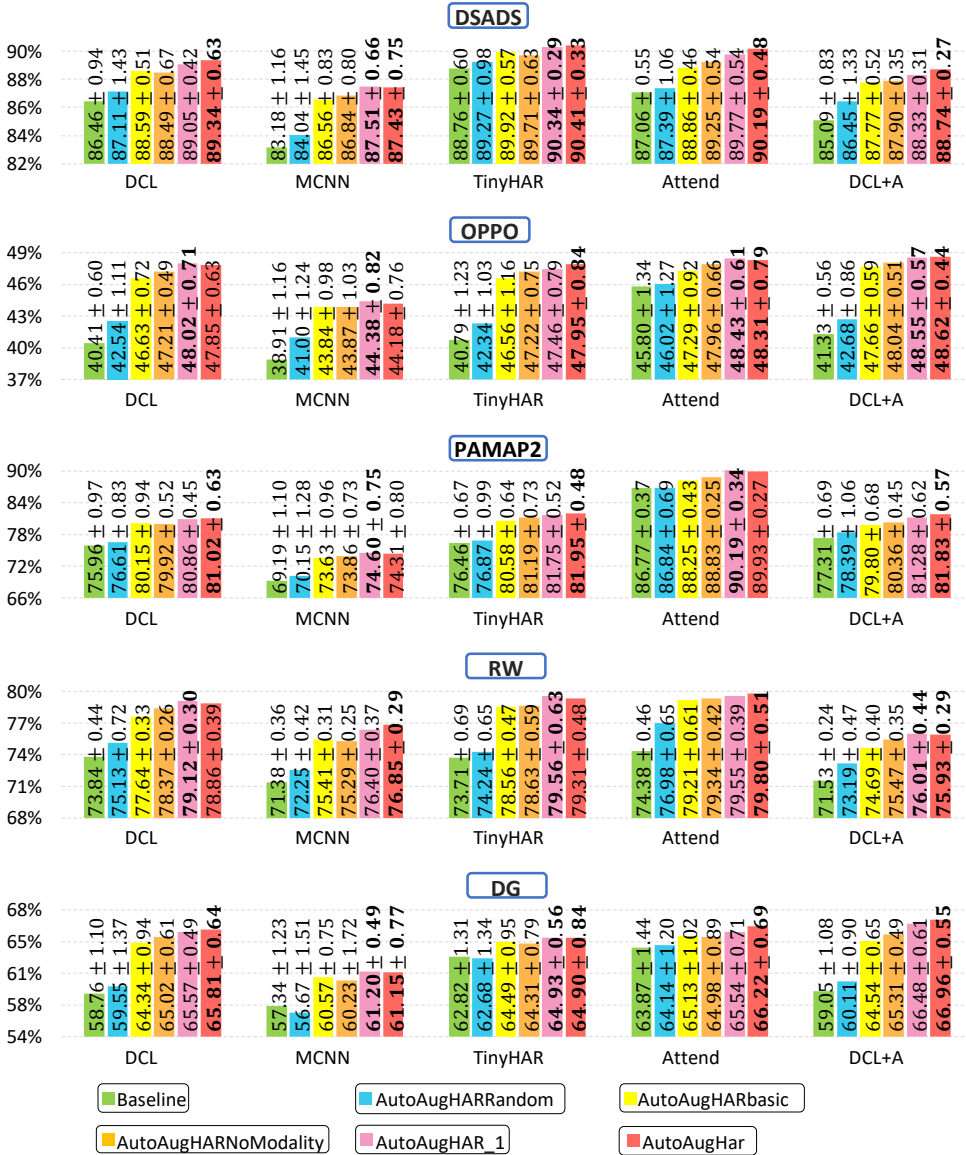


Figure 3.12: Comparison of the classification performance between the proposed AutoAugHAR and several of its variants across five multi-modal datasets. This comparison validates the contributions of AutoAugHAR’s design.

subjected to comparative analysis: AutoAugHARrandom, AutoAugHARbasic, and AutoAugHARNoModality.

AutoAugHARrandom, as elaborated in section 3.2.1.2, does not optimize the categorical distribution for data augmentation sub-policies. It adopts a uniform distribution for the application of these data augmentation sub-policies.

AutoAugHARbasic, detailed in section 3.2.2.1, optimizes the categorical distribution for data augmentation sub-policies. However, it doesn't differentiate between sample-pair-based data augmentation methods and random transformation methods, and it overlooks multi-modality considerations.

AutoAugHARNoModality follows a two-stage structure like the proposed AutoAugHAR but doesn't account for multi-modality.

When the total conservative operation count n is set to 3, the total number of augmentation sub-policies for AutoAugHARrandom and AutoAugHARbasic surpasses 3000, even after eliminating redundant ones. Given the expansive search space, the performance for both AutoAugHARrandom and AutoAugHARbasic is poor. In order to conduct an effective ablation study, we reduced n to 2. With $n = 2$, after removing redundant and identical augmentation sub-policies, the number of augmentation sub-policies for AutoAugHARrandom and AutoAugHARbasic becomes $P_{17}^{16} - 16 - 3 \times 2 - 3 \times 2 - 2 - 2 - 2 = 238$. For a fair comparison, we set n_{1st} and n_{2nd} to 1 for each stage in AutoAugHARNoModality. Furthermore, we also re-trained AutoAugHAR with $n_{2nd} = 1$ in the second stage, denoted as AutoAugHAR_1. Experiments were specifically conducted on datasets characterized by the presence of multiple modalities, thereby validating the contribution of multi-modality optimization in the proposed AutoAugHAR. The results are visually represented in Figure 3.12.

It can be observed that AutoAugHARrandom's performance is unstable. For instance, on the DG dataset, it had a detrimental impact on the MCNN model's performance. Furthermore, its improvements are marginal compared to the performance of CutMix and MixUp, as illustrated in Figures 3.10 and 3.11. Although the stochastic application of data augmentation techniques like AutoAugHARrandom is a conventional procedure in computer vision tasks, it proves unsuitable for HAR tasks. The inherent nature of HAR data, which are more sensitive to perturbations compared to image data, can lead to distortions

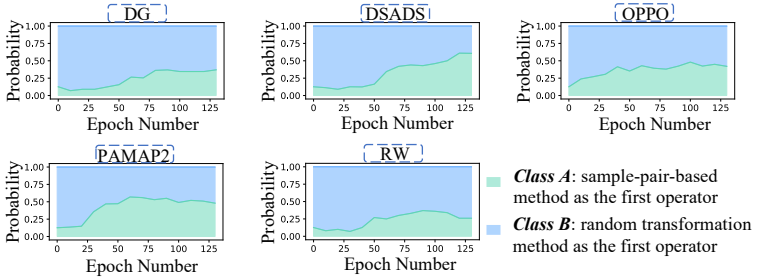


Figure 3.13: Evolution of augmentation sub-policy probabilities over training epochs during the AutoAugHAR training process. These examples are derived from training the Attend model.

in label semantic information due to excessive perturbations.

By employing the AutoAugHARbasic algorithm, after the optimization of weights for augmentation sub-policies, a significant improvement in performance compared to AutoAugHARRandom was noted. This observation validates the benefits of automatic optimization of data augmentation sub-policies.

When compared to AutoAugHARbasic, AutoAugHARNoModality mostly demonstrated marginally superior performance. Although the forced ordering of these two operator categories might limit the diversity of augmentation sub-policies, it effectively reduces the search space, which helps optimization. We believe that augmentation sub-policies, when arranged in this specific order, garner more attention during training. To validate this hypothesis, we further examined the evolution of the probability distribution during the AutoAugHARbasic optimization process. Figure 3.13 illustrates this evolution while training the Attend model on five datasets. For clarity, the 238 augmentation sub-policies were grouped into two classes. Class A consists of sub-policies where sample-pair-based methods are applied as the first operator, totaling 30 sub-policies. Class B includes sub-policies where random augmentation is the initial operator, with 208 sub-policies in this class. The probability assigned to each class is the sum of its constituent probabilities. Given the limited number of sub-policies in Class A, its initial sampling probability is relatively low. However, as optimization continues, there’s a noticeable in-

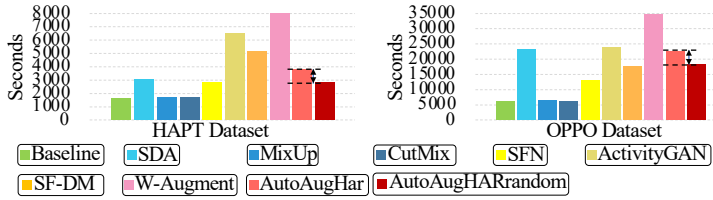


Figure 3.14: The training time for one iteration of the LOSO-CV process.

crease in this class’s probability. Even though Class A accounts for a small portion of the total sub-policies, its probability exceeds 40% by the end of the optimization process. Remarkably, for the DSADS dataset, this probability reaches 60%. We observed that sub-policies that prioritize the sample-pair-based method as the primary operation tend to receive higher weights, further affirming the rationale behind AutoAugHAR’s design.

Compared to AutoAugHARNoModality, AutoAugHAR_1 consistently outperformed AutoAugHARNoModality, highlighting the benefits of considering the multi-modal nature of HAR task. This finding underscores the contribution of optimizing each modality separately.

To understand the influence of the number of operations N on performance, we included the results of AutoAugHAR (with $N_{2nd} = 2$ in the second stage) from previous experiments in Figure 3.12. Our analysis revealed that AutoAugHAR achieved optimal results in 18 instances, while AutoAugHAR_1 did so in 13 instances. Although AutoAugHAR often had the edge over AutoAugHAR_1, the difference in performance between the two was marginal. An increase in the number of operations indeed offers a more diverse augmentation policy. However, this benefit is offset by the challenges of a larger search space and the potential for excessive transformations due to the increased operational steps.

3.2.3.5 Training Overhead. Figure 3.14 depicts the training time required for a single iteration of the LOSO-CV process on two datasets, primarily differentiated by their number of sensor channels. Among the three traditional data augmentation algorithms, MixUp and CutMix lead to a slight increase in

training time. In contrast, the SDA algorithm significantly extends training time due to its default application of the 'time-warp' operation, which requires re-interpolation for each sequence. This impact is more pronounced on the OPPO dataset, where the training time is further extended due to the increased number of channels.

In terms of generative algorithms, SFN is the most time-efficient, followed by ActivityGAN and SF-DM. The longer training durations for ActivityGAN and SF-DM can be attributed to their separate training processes, in which the generator/denoising model is first trained and then incorporated into the dataloader for HAR model training. SF-DM's intelligent design uses unlabeled data as input instead of random noise, resulting in shorter training times compared to ActivityGAN.

The training time required for AutoAugHAR is lower than that of most generative augmentations but higher than SFN. The training time primarily consists of three components: HAR model training, data transformation using data augmentation policies, and data augmentation policy optimization. The extra training time is largely attributable to data transformations using data augmentation policies. As shown in Figure 3.14, even AutoAugHARrandom without policy optimization significantly increases training time, particularly due to time-intensive operations like 'time-warp' and 'magnitude-warp'. The additional training time required to update the data augmentation policy weights is indicated by the arrows in Figure 3.14. The reason why the training time for data augmentation policy optimization is acceptable is that the weights for data augmentation policies are updated using gradients obtained through a backward process in the Adam algorithm. The training time of w -augment is substantially higher than AutoAugHAR because all available data augmentation operators are applied simultaneously to each data sample.

3.2.4 Discussion

The consistent and significant improvements observed across a wide range of target datasets and tasks indicate that automated data augmentation techniques, such as AutoAugHAR, have the potential to become a standard tool for many HAR applications. This technique not only enhances the performance of SOTA

models but also addresses the limited learning capacity of lightweight models, particularly in resource-constrained settings. Numerous studies [43] have demonstrated that data quality plays a critical role in determining model performance. As outlined in Section 2.2, HAR data presents several unique challenges. By utilizing our end-to-end approach, the learned data augmentation policies are precisely tailored to the model’s learning requirements, resulting in enhanced data quality and, consequently, improved overall model performance, even with reduced model size.

3.3 Learnable Data Transformation

In section 3.1.2, we reviewed two data transformation methods commonly used for HAR tasks: one based on wavelet transforms and the other on short-time Fourier transforms (STFT). A key challenge with both methods is how to set their hyperparameters, as these can significantly affect the performance of HAR models. To address this issue, in this section, we introduce our proposed learnable sparse wavelet layer [181].

We chose the wavelet transform because wavelets are based on well-established mathematical principles and do not require learning from data. In contrast to FFT, which combines raw signals using sine functions, wavelets offer a broader range of mother wavelets that can better interpret the original sequence. Additionally, wavelets possess essential properties for signal filtering, such as biorthogonality. Therefore, we hypothesize that when integrated into a learning framework, wavelets can provide superior performance.

To test this hypothesis, we extend SOTA HAR models by incorporating a learnable sparse wavelet layer as a feature extraction component. The learnable sparse wavelet layer functions like a convolutional layer, composed of several learnable wavelet primitives. By employing multiple wavelet types, we avoid the challenge of selecting a single mother wavelet. At the same time, to maintain the sparsity of this layer, non-informative wavelet filters are automatically identified and pruned during training.

Our experiments show that this approach enhances the performance of HAR models, particularly when model size is constrained. This advantage is critical for real-time HAR applications and supports the deployment of HAR models

on wearable computing devices with limited computational resources.

3.3.1 Methodology

In the following sections, we will first provide a detailed explanation of the generation, selection, and pre-processing of the wavelets in section 3.3.1.1. Next, we will describe the implementation of the learnable components of the wavelets in section 3.3.1.2. Finally, in section 3.3.1.3, we will introduce the filter pruning technique, which is designed to reduce computational costs.

3.3.1.1 Wavelet Filters. There are various mother wavelets, each representing different underlying information. For example, the Shannon wavelet functions as an ideal band-pass filter [74], while the Morlet wavelet behaves more like a low-pass filter closely related to human perception [35; 108]. Some wavelets, such as the Daubechies wavelet [151], extend beyond frequency-domain filtering. By utilizing these wavelets, robust and diverse features can generally be extracted for HAR tasks [146].

Unlike the approaches in [175] and [109], we do not pre-specify the wavelets to be used. Instead, we begin by applying all 127 discrete mother wavelets provided by PyWavelets¹. These mother wavelets are then sampled to match the length of the sliding window L used for activity recognition. To capture information across different frequency ranges, each mother wavelet primitive undergoes temporal scaling through down-sampling by power-of-two scaling factors. For example, when L is 128, a mother wavelet is downsampled into various sizes through a series of scaling factors represented by the vector $\mathbf{sc} = [1, 2, 4, 8, 16, 32]$. The size of the scaling factor vector \mathbf{sc} is $\log_2 L - 1$. As illustrated in Figure 3.15 and Figure 3.16, a raw signal was convolved with wavelet functions at different scales, resulting in various filtered signals that capture the dynamics of the original signal from different perspectives.

When the 127 mother wavelets are transformed through temporal scaling with different scaling factors, they result in thousands of wavelets. To reduce the number of mother wavelets while preserving their expressiveness, we first use a clustering approach to select the most representative mother wavelet

¹<https://pywavelets.readthedocs.io>

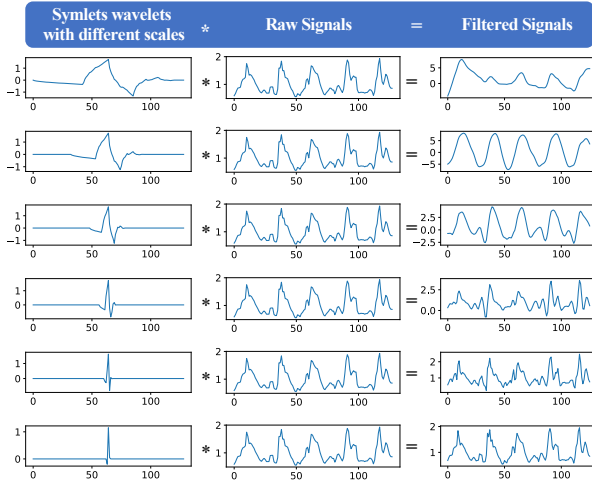


Figure 3.15: Raw signals from the accelerometer's x-axis (middle column) are convolved with **Symlets wavelets** at different scales (left column).

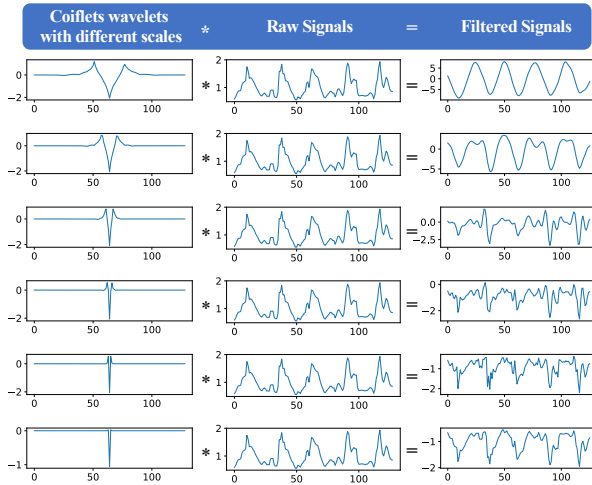


Figure 3.16: Raw signals from the accelerometer's x-axis (middle column) are convolved with **Coiflet wavelets** at different scales (left column).

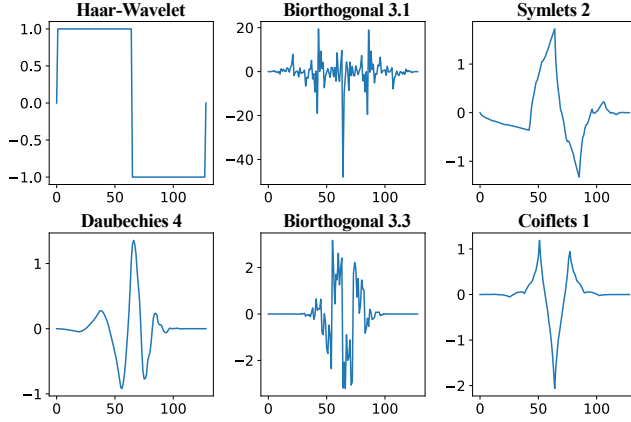


Figure 3.17: This figure shows the six mother wavelet functions representing the cluster centroids from the k-means clustering.

primitives. Specifically, we apply K-means clustering based on both temporal and frequency domains of the 127 different mother wavelets. The distance between the i -th and j -th mother wavelet is defined as $\|f_i^* - f_j^*\|_2 + \|\mathcal{F}_i - \mathcal{F}_j\|_2$ where f_i^* and \mathcal{F}_i represent the i -th wavelet and its Fourier transformation, respectively. To ensure the conservation of energy in each filtering, we normalize the wavelets by

$$\tilde{f}_i^* = \begin{cases} f_i^*, & E_i \leq 1, \\ f_i^*/E_i, & E_i > 1, \end{cases} \quad E_i = \left| \sum_t f_i^*[t] \right|, \quad (1)$$

where $f_i^*[t]$ denotes the value of the t -th element in the i -th wavelet. After clustering, we select the centroid of each cluster as the representative wavelet for that cluster. The optimal number of clusters, $n_{cluster}$, is determined using the silhouette coefficient [131]. Figure 3.17 presents six automatically selected representative wavelet functions. These $n_{cluster}$ representative mother wavelets, after undergoing temporal scaling, result in a total number of $n_{cluster} \times \log_2 L$ filters applied to each sensor channel. It is important to note that this nor-

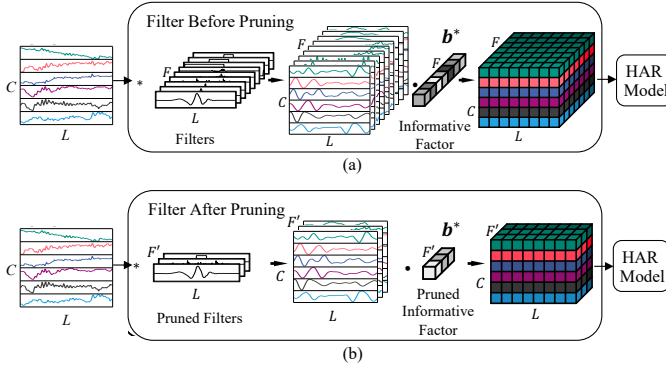


Figure 3.18: Overview of the proposed *learnable sparse wavelet layer*, which can be integrated into any HAR model. Figure (a) represents the training process, where the learnable layer learns which wavelet filters are important. Figure (b) shows the deployment process, where unimportant wavelet filters are pruned. C = number of channels, L = length of sliding window, F = number of initially selected filters, F' = number of filters after pruning.

malization in equation is applied to all mother wavelets, including those after temporal scaling.

Even after the selection process, the number of mother wavelets remains substantial following temporal scaling, and they are convolved with each sensor channel. This results in an increase in the input dimension from the original sensor count C to $F = C \times n_{cluster} \times (\log_2 L - 1)$. The increase in sensor channels also leads to greater computational demands for the model. The current challenge, therefore, is how to further reduce the number of input sensor channels F .

3.3.1.2 Learnable Wavelets. To enable more flexible and task-oriented selection of important wavelet filters without losing their functional properties, we introduce a learnable parameter $\mathbf{b}^* \in \mathcal{R}^F$ that indicates the informativeness of each filtered signal in the HAR model. As illustrated in Figure 3.18 (a), the informative indicator \mathbf{b}^* is multiplied by the corresponding filtered signals, with each element in \mathbf{b}^* directly corresponding to a filtered signal. A larger

element in \mathbf{b}^* indicates a more important signal, while a smaller element indicates a less important one. Due to the presence of the informative factor \mathbf{b}^* , we can remove non-informative filtered signals based on their corresponding indicators.

3.3.1.3 Filter Pruning. The current task is to identify the non-informative signals, which involves learning the weights of k and automatically optimizing which elements of the \mathbf{b}^* vector should be large and which should be small. Similar problems have been studied in the field of neural network pruning. Here, we adopt a strategy similar to that used in [101] and [97]. Specifically, we train the HAR model along with the informative factors \mathbf{b}^* . Additionally, we add a penalty term to the objective function to encourage the informative factors to approach zero. Ideally, this penalty term would be $\ell_0(\mathbf{b}^*)$, indicating the number of non-zero elements in \mathbf{b}^* , also known as the *sparsity* of \mathbf{b}^* . However, $\ell_0(\cdot)$ is ill-conditioned [180] and cannot be solved using gradient-based optimization. Fortunately, it has been shown that $\ell_0(\mathbf{b}^*)$ can often be approximated by the sum of the absolute values of all elements in the vector \mathbf{b}^* [42], also known as the ℓ_1 norm, i.e., $\ell_1(\mathbf{b}^*) = \sum_i |\mathbf{b}_i^*|$. The ℓ_1 regularization serves as a trade-off between the sparsity of the informative factor and the performance of the model.

During training, while maintaining the classification performance of the model, elements in \mathbf{b}^* corresponding to unimportant signals will be compressed to very small values. After training, we remove all informative factors lower than a threshold, along with their corresponding filtered signals. Figure 4.13 (a) and (b) shows a comparison of the filtering process before and after pruning. After removing the unimportant filters, F is reduced to F' , where F' is significantly smaller than F . Since the indicator of the pruned signals is not exactly zero, the performance of the model typically degrades after pruning. Therefore, the model is fine-tuned to adapt to the pruned input. We remove the ℓ_1 norm during fine-tuning.

3.3.2 Experiments and Discussions

3.3.2.1 Experiment Setup. To verify the improvement provided by the *learnable sparse wavelets layer* for HAR models, we conducted experiments on six benchmark datasets: Opportunity, Skoda, PAMAP2, DSADS, Daphnet, and WISDM. We integrated the learnable wavelet layer into three SOTA HAR models: DeepConvLSTM [119], Multibranch CNN (MCNN) [112], and Self-Attention HAR (SA-HAR) [105]. These three models were selected because they represent distinct architectural approaches in HAR modeling, thereby allowing us to demonstrate the generality of our proposed method. Specifically, MCNN is a purely convolution-based model that employs late-fusion techniques to optimally combine multimodal sensor data. DeepConvLSTM is a hybrid model that leverages the strengths of both CNN and LSTM. SA-HAR is a purely self-attention-based HAR model without any recurrent structures.

Furthermore, to demonstrate that our proposed layer can help maintain SOTA performance even when the model size is reduced, we evaluated the three baseline models with different model sizes using the width scaling method [61]. By applying the model width scaling factor γ , the number of filters in each layer of the model is reduced. For example, when $\gamma = 0.5$, the number of convolutional filters in each layer is halved, resulting in a model size that is quadratically reduced to approximately $\gamma^2 = 0.25$ of the original model size.

3.3.2.2 Comparison to State-of-the-art. The experimental results are presented in Figure 3.19, where different colors indicate different HAR models: green for DeepConvLSTM, purple for SA-HAR, and blue for MCNN. Different line types represent the macro F_1 -scores from various setups: dashed lines for baselines, solid lines for models with pruned learnable sparse wavelet layers, and dotted lines for models with all learnable sparse wavelet layers. The bars with varying intensities indicate the number of floating-point operations required by different setups: light colors for baselines, normal colors for models with pruned learnable sparse wavelet layers, and dark colors for models with learnable wavelets without pruning.

From Figure 3.19, we can observe that the learnable sparse wavelet layer improves the overall performance of the baseline HAR models. This is evident

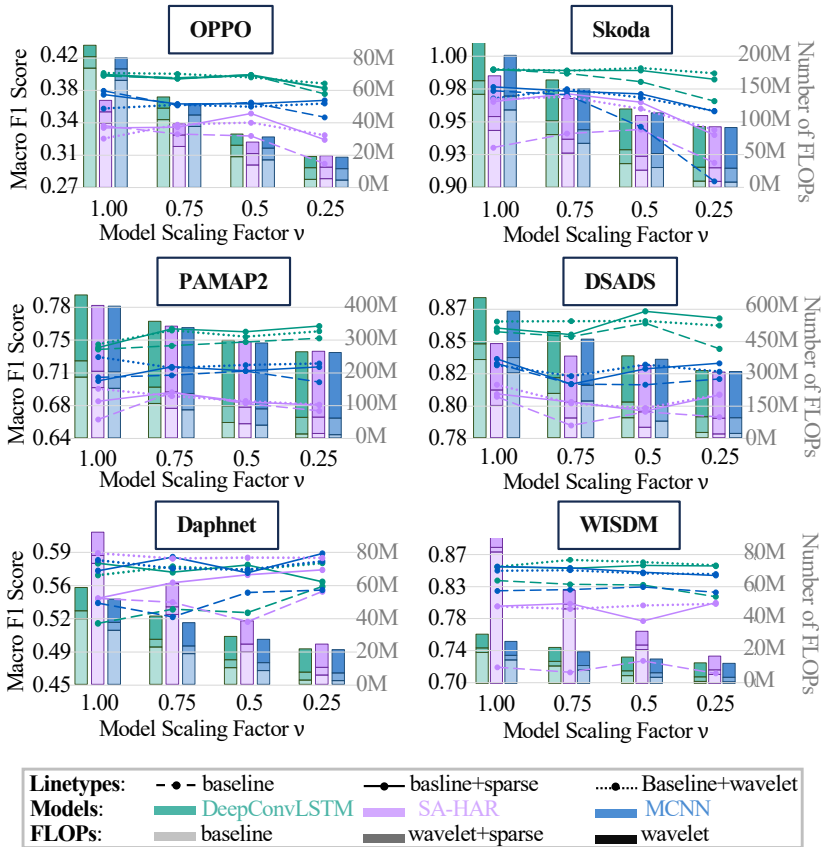


Figure 3.19: Result of the experiment. Different colors indicate different HAR models (green for DeepConvLSTM, purple for SA-HAR, and blue for MCNN). Different linetypes denote the macro F_1 -scores from different setups (dash lines for baselines, solid lines for *learnable sparse wavelet layers*, and dot lines for *learnable wavelet layers without pruning*). The bars with different intensities refer to the number of floating point operations required by different setups, namely light colors for baselines, normal colors for *learnable sparse wavelet layers*, and dark colors show *learnable wavelets without pruning*.

as both the solid and dotted lines are positioned above the dashed line in most cases. This observation is particularly pronounced in the *Daphnet* and *WISDM* datasets, where adding the learnable wavelet layer to the model significantly enhances performance. Upon closer inspection of the differences between the solid and dotted lines, we can see that the differences are minimal, with models containing all learnable sparse wavelet layers performing slightly better than those with pruned learnable sparse wavelet layers. This suggests that the filters retained after pruning are indeed crucial. Moreover, as observed from the bar plot, the introduction of pruning significantly reduces the computational cost without compromising performance, and in some instances, even improves it.

Additionally, we notice that for the datasets **Opportunity**, **Skoda**, **DSADS**, and **WISDM**, the performance of the baseline models deteriorates as the model sizes decrease. This suggests that less information can be learned when the models are smaller. However, in these cases, the information extracted by the learnable wavelets compensates significantly for the reduced model size. Notably, the contribution of the learnable wavelets increases as the model size scales down. Conversely, when the model approaches its saturation size, the learnable wavelets may no longer provide additional information to enhance the model’s performance (e.g., **Opportunity** with $v = 1, 0.75, 0.5$).

Regarding the **Daphnet** dataset, we speculate that the baseline models may overfit, meaning that as model sizes increase, generalizability decreases. As hypothesized in Section 3.3, learnable wavelets, due to their non-data-oriented nature, are less prone to over-fitting the training data. Consequently, they may help improve the generalizability of the models to some extent.

3.3.3 Discussion

In this work, we proposed the learnable sparse wavelets layer by leveraging the superior properties of wavelets. To make the wavelets capable of learning without losing the ability to extract generally useful features and necessary properties for general signal-filtering, we designed the temporal scaling factors k and informative factor w as learnable parameters. Our hypothesis is supported by our experiment that, the learnable sparse wavelets layer extracts rich and general information for the subsequent HAR model, and thus, the performance

can be improved. Furthermore, the proposed layer is more pronounced when the model is the smaller, this facilitates the deployment of the HAR models on wearable devices.

3.4 Summary

In this chapter, we demonstrate that both data augmentation and data transformation techniques are effective methods for maintaining model performance while reducing model size. By leveraging these two approaches, we show that it is possible to create more efficient models without compromising their accuracy or generalization ability. The use of data augmentation helps to enrich the training data, improving the model's capacity to learn robust patterns, while data transformation techniques optimize the input data in ways that make models more compact and computationally efficient. Together, these methods offer a promising solution for achieving lightweight models suitable for resource-constrained environments, without sacrificing performance. This highlights the potential of data-centric strategies in the development of efficient HAR models.

4 Model Architecture Design

Both data augmentation and transformation techniques have been applied SOTA HAR models. While these methods have contributed to enhancing the performance of lightweight HAR models, they are not always the most direct solution. For example, when these techniques are applied to models that are inherently too large, they may reduce the model size to some extent, but the resulting models may still remain oversized due to the complexity of the original architecture. Therefore, the most effective approach is to focus on model design from the outset. In this chapter, we explore manual design strategies for creating lightweight HAR models.

First, by comprehensively considering the unique characteristics of HAR tasks, we designed the TinyHAR model, which will be discussed in Section 4.2. Building upon the TinyHAR architecture and addressing the inherent limitations of neural networks, we propose a dual-branch model that processes two types of input representations: one as a raw time-series representation and the other as a spectrogram, as introduced in Section 4.3. Finally, based on the insights gained from TinyHAR and the dual-branch model, and considering hardware constraints, we developed a fully-connected layer-based model, MLP-HAR, which is detailed in Section 4.4. In the concluding Section 4.5, we summarize the work presented in this chapter.

4.1 Related Work

In Section 2.3, we reviewed the structure of current SOTA HAR models and the relevant work on lightweight HAR models. However, there is limited research within the HAR domain specifically focused on developing deployable lightweight models [144]. In contrast, significant advancements have been made in lightweight model design within other fields, particularly in computer

vision [153; 89; 61; 132; 168; 177; 142]. In this section, we draw upon key insights from computer vision by reviewing notable developments in lightweight model design.

Two primary strategies have emerged for lightweight model design in computer vision tasks: (1) structural design aimed at enhancing information extraction [153; 89; 168; 142; 62] and (2) efficient operator design to reduce computational complexity [61; 132; 177]. Structural designs, such as those in the YOLO series [153] and SSD [100], simplify object detection by reformulating it as a single-stage process. YOLO, for example, frames detection as a regression problem to accelerate speed, while SSD uses multi-scale feature maps to detect objects of different sizes in a single pass. The hourglass network architecture [163] further enhances multi-scale feature extraction through a symmetrical encoder-decoder design, making it particularly effective for tasks like pose estimation. DenseNet [63] facilitates feature reuse and propagation by densely connecting layers, enabling models to reduce depth without sacrificing performance. Similarly, Feature Pyramid Networks (FPN) [94] enhance object detection by aggregating features across multiple scales, supporting detection across varied resolutions. These structural innovations are specifically tailored to address the demands of computer vision, where tasks require both detailed local features and global structural information, prompting the development of these advanced feature extraction mechanisms. This raises an important question: What type of information is necessary for accurate HAR, and how can these requirements be effectively integrated into model design?

In contrast, efficient operator design focuses on reducing computational overhead. MobileNet [61], for instance, employs depthwise separable convolutions to minimize parameters and computational costs while maintaining robust performance. ShuffleNet [177] builds upon this by introducing group convolutions and channel shuffling, further lowering computational demands, particularly for mobile and low-power devices. EfficientNet [142] tackles the problem from a scaling perspective, employing a compound scaling approach to balance network depth, width, and resolution, optimizing performance while keeping the model compact.

The key distinction between these approaches lies in their focus: structural

designs, such as those in YOLO and SSD, prioritize improved feature extraction and handling of multi-scale information, whereas operator designs, like MobileNet and ShuffleNet, emphasize computational efficiency through specialized operations. However, as noted in YOLOv6 [89], while depthwise separable convolutions reduce parameters and FLOPs, their higher memory access costs can sometimes result in slower computations compared to standard convolutions. Additionally, custom operators, like those in ShuffleNet [177], may not be supported by inference libraries on edge devices, limiting their deployability in practical applications.

In summary, these insights guide our approach, emphasizing the design of model architectures tailored to the specific needs of HAR tasks and the information required for accurate classification. While custom operators can provide certain advantages, their deployment on resource-constrained edge devices is often hindered by the lack of support from inference libraries.

4.2 TinyHAR

In this section, we present our proposed model, TinyHAR [185]. An overview of the model architecture is shown in Figure 4.1. Before delving into the details of TinyHAR’s structure, we outline the design principles informed by our review of SOTA HAR models in Section 4.2.1. Based on these principles, the methodology behind TinyHAR is introduced in Section 4.2.2.

4.2.1 Practical Guidelines for Efficient HAR Model Design

Designing an optimal, lightweight DL model requires careful consideration of the characteristics of target tasks and the factors that could reduce inference time and the number of operations. Based on these considerations, we developed the following guidelines for designing lightweight HAR models:

- **G1: Enhance the extraction of local temporal context.** Unlike NLP tasks, where each word carries meaning in a sequence, the values at a single point in a time series offer limited information [1; 119].
- **G2: Unequal treatment of different sensor modalities.** These modal-

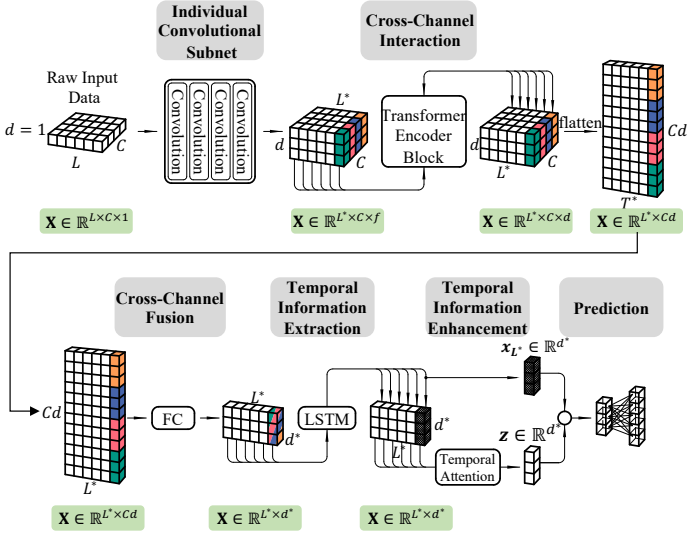


Figure 4.1: Overview of the proposed TinyHAR model.

ities include different sensor types and wearing positions. Only some modalities are informative for recognizing certain activities, while others may contain patterns irrelevant to the activity [167; 102]. Irrelevant modalities can negatively impact recognition and undermine performance [23].

- **G3: Multi-modal fusion.** Activities involve the collective movement of various body parts. Extracting features without considering the interaction between different modalities may limit the model’s performance [99].
- **G4: Global temporal information extraction.** Human activities are embedded as sequential, transient information in sensor readings. Information at certain time steps may exhibit more salient patterns [119] than others. Accurate classification depends on a thorough understanding of the dynamic changes across the entire sequence.
- **G5: Appropriately reduce the temporal dimension.** Compared to im-

age data, HAR data typically has a much larger temporal dimension relative to its spatial dimension. An excessively long temporal dimension can lead to high model complexity and hinder the effective extraction of global temporal dependencies [78]. Reducing the temporal dimension can alleviate this issue and reduce computational cost.

In essence, HAR models developed prior to the publication of these guidelines have often overlooked some of these principles.

4.2.2 Methodology

Following the guidelines above, we designed TinyHAR, which consists of five parts, as illustrated in Figure 4.1. The input data to the model is denoted as $X \in \mathbb{R}^{L \times C \times d}$, where L represents the temporal sliding window size, C is the number of sensor channels, and d indicates the number of filters ($d = 1$ for raw data input that has not been processed).

4.2.2.1 Individual Convolutional Subnet. To enhance the local context, we implemented a convolutional subnet to extract and integrate local features from the raw data (following **G1**). Given the varying contributions of different modalities, each channel is processed independently through four distinct convolutional layers (following **G2**). Each convolutional layer employs ReLU activation functions [2] and batch normalization [67]. The term "individual convolution" refers to kernels that are 1D along the temporal axis, with a kernel size of 5×1 . To reduce the temporal dimension (in line with **G5**), the stride in each layer is set to 2. All four convolutional layers share the same number of filters, denoted by d . Consequently, the output shape of this convolutional subnet is $\mathbb{R}^{L^* \times C \times d}$, where L^* represents the reduced temporal length. Since padding is not applied, the size of L^* is approximately $\frac{1}{16}$ of the original input length L . Notably, the kernel size of 5, being larger than the stride of 2, prevents information loss that could occur when the stride exceeds 1.

4.2.2.2 Cross-Channel Info Interaction. After extracting local context information from each sensor channel, the next step is to enable the learning of

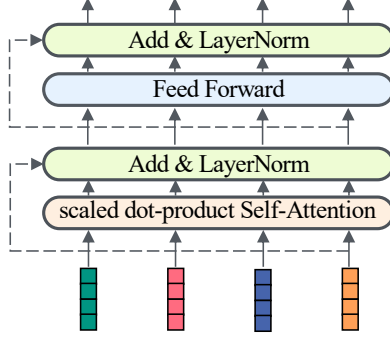


Figure 4.2: Architecture of transformer block.

interactions between them. Building on the success of the self-attention mechanism in learning collaborations between sensor channels, as demonstrated in [1], we implemented a transformer encoder block [150] to capture these interactions. The transformer encoder block comprises a scaled dot-product self-attention layer and a two-layer fully connected (FC) feed-forward network, as illustrated in Figure 4.2.

This process is applied across the sensor channel dimension (following **G2**) at each time step. The scaled dot-product self-attention layer determines the relative importance of each sensor channel by assessing its similarity to all other sensor channels. As illustrated in Figure 4.3, the extracted features at each time step, represented by a 2-dimensional matrix $X_t \in \mathbb{R}^{C \times d}$, are processed sequentially and separately using the self-attention layer. During the self-attention process, the feature vector of each sensor channel in X_t undergoes a mapping by three linear transformations: $Q = X_t W_q$ (query), $K = X_t W_k$ (key), and $V = X_t W_v$ (value). Here, W_q , W_k , and $W_v \in \mathbb{R}^{d \times d}$ are learnable weight matrices. Normalized correlation scores across all pairs of feature vectors for each sensor channel are computed by applying a scaled dot product, $\frac{QK^T}{\sqrt{d}}$, followed by a softmax function. These correlation scores dictate the extent to which the feature vector of each sensor channel should interact with the other feature vectors. A feature vector with higher correlation scores will exert a larger influence. The correlation scores are utilized to compute a weighted

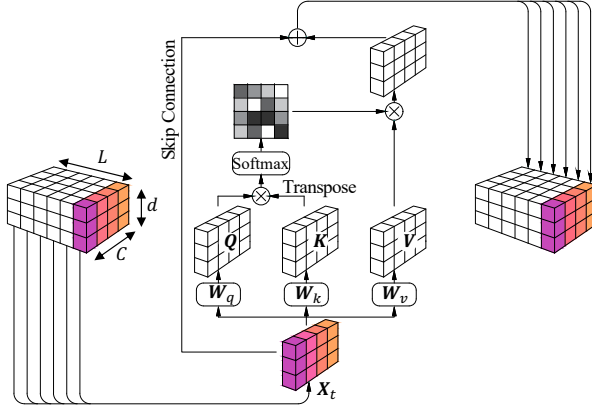


Figure 4.3: The self-attention is employed to facilitate cross-channel interactions, enabling each channel to dynamically integrate information from all other channels.

sum of the value matrix V , generating a new feature vector for each sensor channel that aggregates information from all other sensor channels. Subsequently, the new feature vectors are reintegrated with the initial feature vectors via a residual connection. This approach enables each sensor channel to adaptively and flexibly from interactions among other sensor channels. The process can be mathematically formulated as

$$X_t = \text{softmax} \left(\frac{QK^T}{\sqrt{d}} \right) V + X_t.$$

Following this, as illustrated in Figure 4.2, two feed-forward layers are applied to each sensor channel, further refining the aggregated features. At this stage, the features of each channel are contextualized with the underlying cross-channel interactions. The skip connections are employed to preserve each sensor channel's unique information, even as key information from other channels is integrated and fused.

4.2.2.3 Cross-Channel Info Fusion. Up to this point, information exchange between sensor channels has been facilitated, but a comprehensive fusion of all

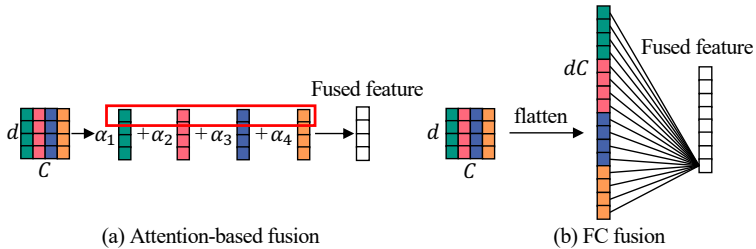


Figure 4.4: Attention-based fusion (a) vs fully-connected layer-based fusion (b).

the information has not yet been performed. Many models, such as Attend [1] and ALAE [3], overlook this step. They opt to vectorize the 2D feature representations at each time step into a 1D vector and directly feed this into subsequent RNN models. This approach not only bypasses feature fusion but also increases the input size to $d \times C$, leading to a significant increase in the number of learnable parameters in the subsequent models.

As illustrated in Figure 4.4 (b), in TinyHAR, we similarly first vectorize these representations at each time step, transforming $X \in \mathbb{R}^{L^* \times C \times d}$ to $X \in \mathbb{R}^{L^* \times Cd}$. To effectively fuse the learned features from all sensor channels (in line with **G3**), we apply a fully connected layer that performs a weighted summation of all features. Unlike the attention mechanism used in [102], as shown in Figure 4.4 (a), where features from the same sensor channel share the same weights, the FC layer allows different features within the same sensor channel to have distinct weights. This flexibility enables more comprehensive feature fusion. Additionally, this FC layer acts as a bottleneck in TinyHAR, reducing the feature dimension to d^* . As a result, the input size to the RNN is reduced from Cd to d^* . In our design, we set $d^* = 2d$, which is typically much smaller than the original number of sensor channels, C . Compressing the dimensions not only enhances feature fusion but also effectively reduces the size and computational complexity of the subsequent RNN layers.

4.2.2.4 Global Temporal Info Extraction. After fusing the features across the sensor and filter dimensions, we obtain a sequence of refined feature vec-

tors in $\mathbb{R}^{L^* \times d^*}$ that are ready for sequence modeling. At this stage, we apply a single LSTM layer to learn the global temporal dependencies. One layer is sufficient since the temporal dimension has already been significantly reduced. Additionally, further global temporal enhancement will be addressed in subsequent stages.

4.2.2.5 Global Temporal Info Enhancement. Given that not all time steps contribute equally to the recognition of ongoing activities, it is crucial to evaluate the significance of features at each time step within the sequence. Following the methodology proposed by AttenSense [102], we construct a global contextual representation $\mathbf{z} \in \mathbb{R}^{d^*}$ by calculating a weighted average of the hidden states (features) across past time steps. The weights β are derived through a temporal self-attention mechanism, which initially transforms the context of each time step x_i to \tilde{x}_i and then calculates their relative importance based on these transformed contexts \tilde{x}_i . Since the feature at the final time step $x_{L^*} \in \mathbb{R}^{d^*}$ encapsulates information from the entire sequence, the generated global representation \mathbf{z} is subsequently added to x_{L^*} . To allow the model to dynamically decide whether to incorporate or disregard the generated global representation \mathbf{z} , we introduce a trainable scaling parameter γ . This process can be mathematically formulated as follows:

$$\begin{aligned}
 \text{past hidden context} &= [x_1, x_2, \dots, x_{L^*-1}] \\
 \text{last context} &= x_{L^*} \\
 \text{transformed context } \tilde{x}_i &= \tanh\left(\mathbf{W}_\beta^1 x_i + b_\beta^1\right) \\
 \text{weight } \beta_i &= \frac{\exp\left(\mathbf{W}_\beta^2 \tilde{x}_i\right)}{\sum_{j=1}^{L^*-1} \exp\left(\mathbf{W}_\beta^2 \tilde{x}_j\right)} \\
 \text{global representation } \mathbf{z} &= \sum_{i=1}^{L^*-1} \beta_i x_i \\
 \text{output} &= x_{L^*} + \gamma \mathbf{z}
 \end{aligned}$$

In this context, \mathbf{W}_β^1 and \mathbf{W}_β^2 represent the weights of two fully connected layers, which transform the past context into a scalar value. The relative weights β are then obtained by applying a softmax function to these scalar values.

4.2.3 Experiments and Discussions

4.2.3.1 Experiment Setup. In order to validate the effectiveness of the proposed TinyHAR, we evaluate it on six widely used HAR benchmark datasets, they are DSADS [14], SKODA(r) [170], WISDM [83], Daphnet [11], PAMAP2 [127] and OPPO [21]. The datasets were selected to exhibit a great diversity in terms of the sensing modalities used, installation locations, sampling frequency, data collection scenarios and activities to be recognized.

We compare TinyHAR with the improved DeepConvLSTM (DCL) [19] model, an enhanced version of the widely used benchmark model DeepConvLSTM [119] frequently referenced in various works. The improved DeepConvLSTM exhibits a reduced model size while achieving better performance. The specifications of all layers in DCL remain consistent with the settings in the original work [19]. Furthermore, we introduce a model shrinking hyper-parameter width multiplier ν [142] to thin DCL uniformly at each layer. By setting $\nu = \{0.25, 0.5, 0.75\}$, the model size will be reduced to approximately $\frac{1}{2}$, $\frac{1}{4}$ and $\frac{1}{16}$ respectively (referred as DCL_0_75, DCL_0_50 and DCL_0_25). We adjust the filter number d of TinyHAR, so that has a comparable number of parameters to DCL_0_25.

4.2.3.2 Comparison to State-of-the-art. Figure 4.5 presents the experimental results. Each column in the table represents the results for a specific dataset. The first row in each column shows the macro F1 score $F1_M$, the second row displays the model size in terms of the number of trainable parameters, and the third row indicates the model complexity in terms of FLOPs. Observing the second row, it is evident that TinyHAR, by design, has a significantly smaller size compared to the baseline models and their scaled-down variants. However, when examining the first row, it is clear that TinyHAR achieves superior performance on three datasets (PAMAP2, DSADS, and Daphnet) compared to its competitors. On these datasets, TinyHAR outperforms the baseline models in their original size, with improvements in $F1_M$ of 1.8%, 3.1% and 9.78%, respectively.

We observed an overfitting trend in the DCL variants for the PAMAP2 and Daphnet datasets, where the model’s performance improved as the model be-

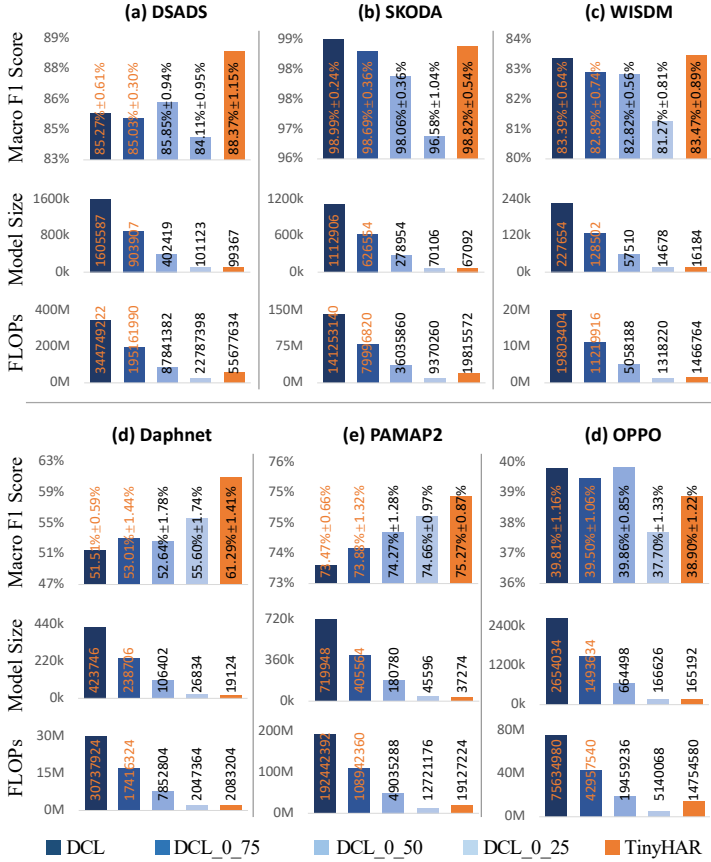


Figure 4.5: LOSO-CV performance comparison between TinyHAR and the DCL model with varying model sizes. For each dataset, the averaged $F1_M$, the corresponding model size (number of trainable parameters) and number of FLOPs are shown separately.

coming smaller with the decreased shrink parameter ν . To further investigate this behavior, we continued reducing the size of the DCL model by decreasing the shrink parameter ν . For the Daphnet dataset, we set $\nu = 0.21$, resulting in a model size of 17,786 parameters, making it smaller than TinyHAR. Given the risk of overfitting on this dataset, the reduced DCL achieved an $F1_M$ of $55.93\% \pm 1.75\%$. Although the performance improved, it was still 5.36% lower than that of TinyHAR. For the PAMAP2 dataset, we reduced ν to 0.23, yielding a model size of 36,254 parameters. In this case, we observed a degradation in performance ($74.36\% \pm 1.02\%$). Even when the parameter count was reduced to a level comparable to TinyHAR, TinyHAR consistently outperformed the DCL variants, particularly on datasets prone to overfitting. This superior performance is attributed to TinyHAR’s design, which fully considers the unique characteristics of HAR tasks.

Compared to the baseline, TinyHAR obtains slightly lower $F1_M$ on Skoda dataset and slightly higher $F1_M$ on WISDM dataset. Significant tests show that there is no statistically significant difference in the performance between both models. TinyHAR is, however, much smaller than the comparison model and performs significantly better than DCL_0_25 with a similar size. We speculate that the reason for the positive correlation of performance with model size for the baseline is that, Skoda data was collected from only one subject, so it does not penalize overfitting as the statistical independence of training and test samples within the cross validation is smaller. Similarly, the WISDM dataset was collected from 36 subjects through controlled, laboratory conditions, which reduces the effect of generalizability. Even under this condition, the TinyHAR still reaches the same performance with around 6% model size. The Performance on Daphnet, PAMAP2, SKODA, WISDM, and DSADS suggests that TinyHAR has a great capability to capture temporal-spatial patterns in multi-modal sensing.

However, TinyHAR obtains lower $F1_M$ score on Opportunity. Compared to other datasets, dataset Opportunity has much more sensor channels. We assume that the poor performance is owing to the fact that the model is too small to effectively extract and fuse information from so many channels. Thus, we increased the filter number d from 28 to 42 (model size 370082). The model’s

$F1_M$ performance of $41.22\% \pm 1.19\%$ in this case (as predicted) exceeds the performance of all DCL variants.

When evaluating computational complexity in terms of FLOPs, as shown in Figure 4.5 (third row), the TinyHAR model exhibits significantly fewer FLOPs than the DCL model and its variants, DCL_0_75 and DCL_0_50. However, TinyHAR has a higher number of FLOPs compared to the DCL_0_25 model. This increase in complexity is attributed to TinyHAR’s more advanced architecture, which includes additional modules such as the cross-channel interaction module, feature fusion module, and temporal information enhancement module. Nevertheless, despite its higher computational cost relative to DCL_0_25, TinyHAR consistently delivers superior performance across all datasets.

4.2.4 Discussion

Through a HAR-specific design, we were able to develop a lightweight but highly competitive DL model. Particularly, when taking the model size into account, which is of great importance for wearable computing, the model clearly outperforms the DeepConvLSTM as an example of a state of the art HAR model in our experiments.

Instead of only adapting an architecture from another domain and letting the optimizer do its magic, different saliency of multi modalities, multimodal collaboration and temporal information extraction were specifically translated into a network architecture to achieve this performance.

We believe there is still significant potential for improving the design of lightweight HAR models, particularly for deployment on wearable computing devices. For example, the structural diversity of TinyHAR’s components can result in relatively high computational complexity, especially when the filter size d is small. Moreover, certain operators, such as RNN-based layers, are less favorable in time-constrained environments due to their sequential nature, which limits parallel processing and hinders efficient inference.

4.3 Cross-Attention with Multi-Representation

Despite TinyHAR’s compact size and its ability to outperform the benchmark model DeepConvLSTM, there remains a performance gap compared to other SOTA models. In this chapter, we introduce our proposed Cross-Attention with Multi-Representation (Cross-Atten) model, which surpasses the performance of all existing SOTA HAR models.

4.3.1 Methodology

To more effectively exploit the complementarity of representations during the learning process, we devised a dual-branch model. The overall structure of this model, depicted in Figure 4.6, includes a time-series branch (Branch-ts) and a spectrogram branch (Branch-sc). Within this framework, the data processing flow is divided into four stages. (1) Initially, the raw sensory time-series representation is transformed into a spectrogram representation through the Fourier transformation. (2) Subsequently, both the spectrogram representation and the time-series representation are processed separately through the local embedding module corresponding to their respective branches. Notably, although these two embedding modules follow the same design, they do not share their parameters. (3) In the ensuing phase, the extracted features are passed through the attention module, which is designed to extract global temporal dependencies. Instead of directly concatenating these features, the interaction between the features acquired by each branch is enhanced through the incorporation of a cross-attention block. This cross-attention block allows the features of each branch to adaptively gather relevant and complementary information from each other, thus facilitating joint robust feature learning. (4) In the final stage, the features derived from both branches are concatenated to generate the final prediction. Subsequent sections will provide a detailed introduction of these modules, presented in sequential order.

Throughout this work, we use notations with different symbols to differentiate between parameters and feature maps associated with the two branches within the proposed framework. Notations pertaining to the time-series branch will have a hat symbol, whereas those associated with the spectrogram branch

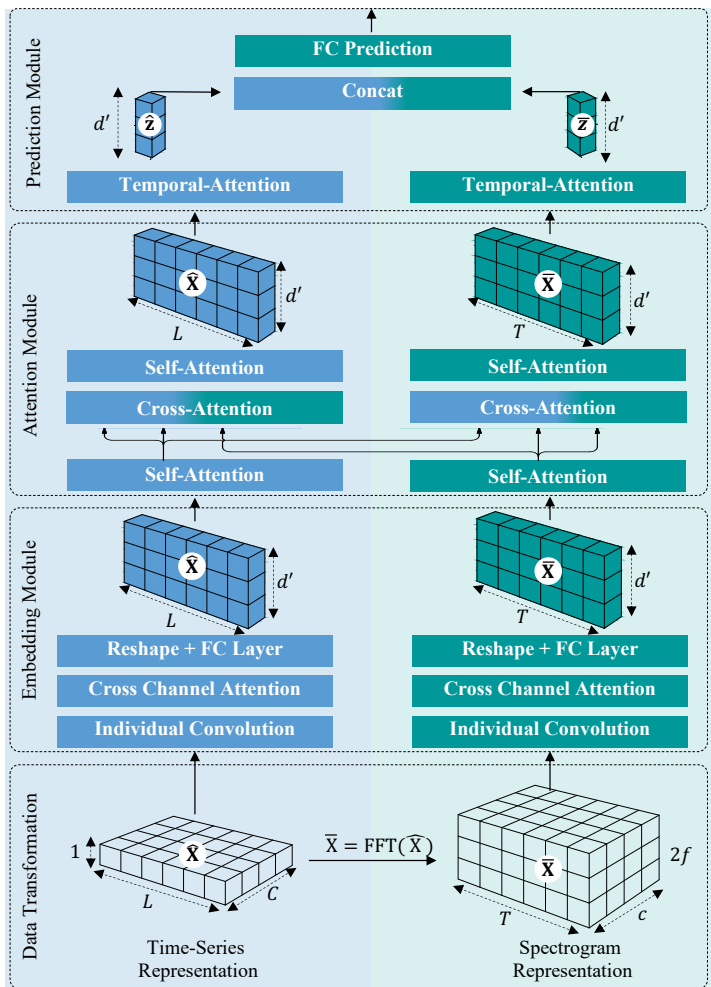


Figure 4.6: An overview of the proposed model, **Cross-Atten**, which consists of two branches. The branches are color-coded: blue for the time-series branch and green for the spectrogram branch. The notation of the feature map and the parameters within each branch are denoted by different symbols, hat and bar, respectively.

will bear a bar symbol. If a notation does not have a symbol in the following discussions, it implies that the process is equivalent in both branches.

4.3.1.1 Data Transformation. Given the readings of multidimensional sensors, the initial step involves segmenting these readings into uniform lengths using a sliding window. Following this, a classification model maps each data segment to predefined activity classes. The segment directed to the time-series branch is represented as a 3-dimensional matrix, $\hat{X} \in \mathbb{R}^{L \times C \times d}$, where L represents the size of the sliding window (the length of the segment), and C denotes the number of sensor channels. Since the raw data are not processed, the feature dimension equals to $d = 1$. Meanwhile, the input directed to the spectrogram branch is also a 3-dimensional matrix, symbolized as $\bar{X} \in \mathbb{R}^{T \times C \times 2f}$, having been transformed from \hat{X} . To compute the spectrogram \bar{X} , we follow methodologies employed in prior works [99; 167; 102]. The time-series data from each sensor channel is first segmented along the temporal dimension into T equal-length intervals, with each interval having a length of τ . The FFT is then applied to each interval, producing spectral features that consist of f pairs of magnitude and phase, where $f = \tau$. Consequently, the feature dimension of \bar{X} is established as $2f$.

4.3.1.2 Individual Embedding Module. Following the data transformation, the two representations are fed into the embedding module of each respective branch to independently extract features, with the embedding module comprising three parts. The structural design of this module follows the principles outlined in TinyHAR [185].

Initially, to enhance the local context, an individual convolutional subnet is applied to extract and fuse local features from the raw data. Given the varying contributions of different modalities, each sensor channel is processed separately through four individual convolutional layers. Individual convolution implies that the kernels have a 1D structure along the temporal axis, with a kernel size of $(k \times 1)$. Consequently, features are extracted solely along the temporal axis, without interaction between sensor channels. For each convolutional layer, ReLU nonlinearities [2] and batch normalization [67] are employed. Notably, the feature dimensionality of the convolutional kernel, denoted as d , re-

mains consistent across all four convolutional layers.

To facilitate learning of the collaboration between sensor channels, cross-channel attention is applied following individual convolution [1]. As illustrated in Figure 4.3, the extracted features at each time step are processed sequentially and separately using the self-attention block. During the self-attention process, the feature vector of each sensor channel in X_t undergoes a mapping by three linear transformations: $Q = X_t W_q$ (query), $K = X_t W_k$ (key), and $V = X_t W_v$ (value). Normalized correlation scores across all pairs of feature vectors for each sensor channel are computed by applying a scaled dot product, $\frac{QK^T}{\sqrt{d}}$, followed by a softmax function. The correlation scores are utilized to compute a weighted sum of the value matrix V , generating a new feature vector for each sensor channel that aggregates information from all other sensor channels. Subsequently, the new feature vectors are reintegrated with the initial feature vectors via a residual connection. The process can be mathematically formulated as

$$X_t = \text{softmax} \left(\frac{QK^T}{\sqrt{d}} \right) V + X_t.$$

Upon learning the interaction and to fuse the learned features across all sensor channels, the features at each time step are vectorized in the final step of this embedding module. Feature maps from both branches are converted into sequential feature vectors $\hat{X} \in \mathbb{R}^{l \times cd}$ and $\bar{X} \in \mathbb{R}^{n \times cd}$. Subsequently, a fully connected layer is employed to perform a weighted mapping of the vectorized feature vectors. In our work, we define the number of features will be mapped from cd to $d' = 2d$. This fully-connected layer not only facilitates the fusion of features across all sensor channels but also plays a role in reducing the feature size. This effectively decreases the computational load of the subsequent model structure as well as the number of learnable parameters. Regardless of the number of sensor channels C , the number of features is compressed to $d' = 2d$.

4.3.1.3 Attention Module. Up to this point, the extracted features from the embedding module of both branches have only covered the local context. At this stage, the two feature vector sequences $\hat{X} \in \mathbb{R}^{l \times d'}$ and $\bar{X} \in \mathbb{R}^{n \times d'}$ will be introduced to the attention block. This block is tasked with learning the global

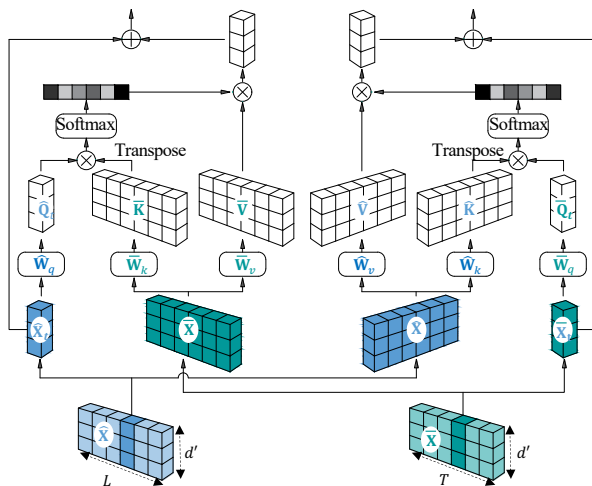


Figure 4.7: Cross-attention block for two branches. Each feature vector serves as a query to interact with all feature vectors from the opposite branch. Both branches adhere to the same procedure.

temporal dependencies through two self-attention blocks and also facilitating interaction between the two branches via the cross-attention block.

Initially, the attention module employs a self-attention block to extract global temporal dependencies within each branch independently. The self-attention mechanism utilized here is the same as that described in Sec 4.3.1.2 with the only difference: the self-attention in the previous section learned the interaction across sensor channels, here, it is executed along the temporal dimension.

After the feature vector at each time step has exchanged information with the feature vectors at other time points within the same sequence, the attention module employs the cross-attention block to enable information exchange across the feature vector sequences of the two branches. The process of cross-attention is illustrated in Figure 4.7. The concept of cross-attention extends the mechanism of self-attention: rather than each feature vector in a sequence attending solely to feature vectors from the same sequence (as in self-attention), in cross-attention, the feature vector from one sequence computes correlation

scores with feature vectors from another sequence.

Taking the time-series branch as an example, when it is producing its output sequence, the feature vector at one time step serves as a query to interact with all feature vectors from the spectrogram branch. After the linear transformation, the scaled dot product of the matrix query and key determine the relative correlation between the feature vector from the time-series branch and all feature vectors from the spectrogram branch. After normalizing the correlation through the softmax function, the mapped feature vectors from the spectrogram branch are summed with weights, according to the normalized correlation scores, to form a new feature vector, which blends information from the spectrogram branch. This new feature vector is then added back to the feature vector of the time-series branch. In this manner, communication is facilitated between the two branches, enabling them to give attention to each other and leverage information from each other during learning. This process is mirrored in the spectrogram branch. The cross-attention mechanism for the two branches can be formulated as follows:

$$\begin{aligned}\hat{X} &= \text{softmax} \left(\frac{\hat{Q}\bar{K}^\top}{\sqrt{d'}} \right) \bar{V} + \hat{X}, \\ \bar{X} &= \text{softmax} \left(\frac{\bar{Q}\hat{K}^\top}{\sqrt{d'}} \right) \hat{V} + \bar{X},\end{aligned}$$

where $\bar{Q} = \bar{X}\bar{W}_q$, $\bar{K} = \bar{X}\bar{W}_k$, $\bar{V} = \bar{X}\bar{W}_v$, $\hat{Q} = \hat{X}\hat{W}_q$, $\hat{K} = \hat{X}\hat{W}_k$ and $\hat{V} = \hat{X}\hat{W}_v$, with trainable weight matrices \hat{W}_q , \hat{W}_k and \hat{W}_v for the time-series branch and trainable weight matrices \bar{W}_q , \bar{W}_k and \bar{W}_v for the spectrogram branch.

After the cross-attention block, the attention module employs a second self-attention block, allowing the features in each respective branch to further refine the temporal dependency, based on the fused complementary information from the opposite branch.

4.3.1.4 Prediction Module. The output of the attention module in each branch remains a sequence of feature vectors. Considering that not all time steps contribute equally to the recognition of activities, the prediction module employs temporal attention to generate a representative feature vector z for

each branch by taking a weighted average sum of the feature vectors across all time steps, given by $\hat{z} = \sum_{i=1}^l \hat{\beta}_i \hat{x}_i$ and $\bar{z} = \sum_{i=1}^n \bar{\beta}_i \bar{x}_i$. Here, β_i represents the computed attention weight for the feature vector at each time step. Technically, attention weights β_i are obtained by first mapping each feature vector to a single score with two Fully Connected layers and then normalizing these scores across time steps using a softmax function. Subsequently, the learned representations from both branches are concatenated to form a joint representation, denoted as $\text{concat}(\hat{z}, \bar{z})$. This joint representation is then fed into an FC layer to generate predictions for pre-defined activities.

4.3.2 Experiments and Discussions

4.3.2.1 Experimental Setup. To systematically validate the efficacy of the proposed model, an extensive performance evaluation was conducted using twelve widely used datasets. These datasets exhibit substantial diversity, showing variations in activity types, sensor installation locations, and sampling rates. These datasets include: PAMAP2 [127], DSADS [14], Daphnet [11], RealWorld HAR (RW) [141], HAPT [128], Motion Sense [107], Skoda(r) [170], Ges-Home [115], EMG-G [87], Letters of Sign Language dataset (L-Sign) [123] and Heterogeneity Activity Recognition (HHAR) [136].

Table 4.1: Parameters related to FFT transformation for each dataset.

<i>Data</i>	<i>L Window Size</i>	<i>T # Intervals</i>	<i># f</i>
PAMAP2	1.28 s	80	16
DSADS	5 s	5	25
Daphnet	1 s	4	16
RW	2 s	10	10
HAPT	2.56 s	8	16
MotionSense	2.56 s	8	16
Skoda	2 s	10	20
GesHome	2 s	7	20
L-Sign	3 s	6	25
EMG-G	1 s	10	20
HHAR	2 s	10	20
MHealth	2.56 s	8	16

Table 4.1 provides information about the parameters related to data preparation/transformation. The column "window size" L denotes the size of the

sliding window. The signals of each sensor are z -normalized separately before the FFT transformation. For generating the spectrogram representation, "# interval" T denotes the specific number of intervals into which the data segments are subdivided. Furthermore, "# f " represents the quantity of amplitude and phase spectral pairs computed from each interval. This data preparation setup adhered to the setup delineated in relevant works [167; 99; 178] without any tuning adjustments.

In the context of this research, the proposed model is analytically compared with six SOTA HAR models. **DeepConvLSTM** [119] is a widely used baseline model. The other five models are SOTA HAR models. **DeepSense** [167] and **GlobalFusion** [99] are representative of models that utilize spectrogram representations as input. The models **Attend-Discriminate (Attend)** [1], **ALAE-TAE** [3], and **IF-ConvTransformer (IF-ConvT)** [178] represent models that use time-series representations as input and exhibit diverse neural architecture designs.

The structure of the proposed Cross-Atten model is detailed in Tables 4.2 and 4.3, with Table 4.2 presenting the architecture of the time-series branch and Table 4.3 showing the structure of the spectrogram branch. Following the practices of models like DeepSense, Attend, Globalfusion, and ALAE-TAE, we set the total number of filters to 64 in our model. However, given its dual-branch design, we reduced the filter number to 32 for each branch's individual embedding block to maintain balance ($d = 32$). This ensures that the total number of filters across both branches remains consistent with the comparison models. It is important to note that in this experiment, the parameter $d = 32$ was not optimized but was simply set to evenly split the 64 filters between the two branches, in line with the settings of the benchmark HAR models.

4.3.2.2 Comparison to State-of-the-art. Figure 4.8, Figure 4.9 and Figure 4.10 illustrate the comparative performance of the models across twelve datasets. Each figure presents the model's performance on four datasets, with each dataset represented by a column. Each column is divided into three rows: the first row shows the classification performance, the second row displays the computational complexity, and the third row reports the number of learnable

Table 4.2: The structure for the time-series branch.

Individual	kernel_size = (5,1), padding=0, d=32, stride=(2,1)
Convolutional	kernel_size = (5,1), padding=0, d=32, stride=(1,1)
Layers	kernel_size = (5,1), padding=0, d=32, stride=(2,1)
	kernel_size = (5,1), padding=0, d=32, stride=(1,1)
Cross Channel Attention	d=32
Fully Connected Layer	$d'=64$
Self-Attention	$d'=64$
Cross-Attention	$d'=64$
Self-Attention	$d'=64$

Table 4.3: The structure for the spectrogram branch.

Individual	kernel_size = (3,1), padding=1, d=32, stride=(1,1)
Convolutional	kernel_size = (3,1), padding=1, d=32, stride=(1,1)
Layers	kernel_size = (3,1), padding=1, d=32, stride=(1,1)
	kernel_size = (3,1), padding=1, d=32, stride=(1,1)
Cross Channel Attention	d=32
Fully Connected Layer	$d'=64$
Self-Attention	$d'=64$
Cross-Attention	$d'=64$
Self-Attention	$d'=64$

parameters in the model. To assess the statistical difference in performance between the models, we employed the Mann-Whitney U test [157], considering a difference statistically significant if the p-value is less than 0.05. The best performances are highlighted in bold.

An overall observation reveals that the proposed model, Cross-Atten, consistently surpasses its counterparts, achieving the best averaged macro F1 Score $F1_M$ on 10 of 12 datasets. For the datasets where the Cross-Atten did not secure the top performance, it achieved the second-best. This observation underscores the robustness and efficacy of Cross-Atten in navigating various scenarios.

A more granular exploration of the data representations employed by the models unveils additional insights. Notably, the GlobalFusion model, which utilizes a spectrogram representation, offers superior results on the EEG-G, HHAR, GesHome and L-sign datasets compared to models leveraging time-series representation, as illustrated in Figure 4.9. In contrast, on the PAMAP2,

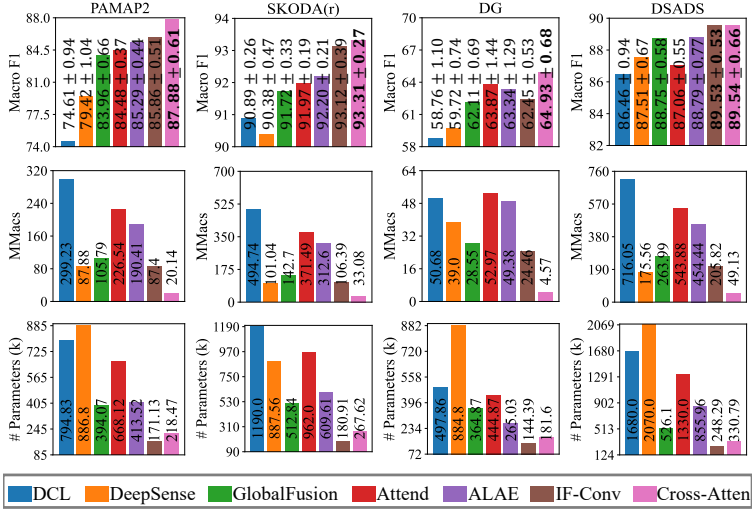


Figure 4.8: Performance comparison on four HAR datasets between the proposed Cross-Atten model and other SOTA HAR models (I).

DG, DSADS and SKODA(r) datasets, as illustrated in Figure 4.8, this dominance is reversed; model Attend utilizing time-series data representation clearly demonstrates better performance. This phenomenon implicitly shines light on the strengths and weaknesses inherent within respective data representations, which subsequently influence the models’ proficiency in feature extraction. However, Cross-Atten demonstrates excellent performance on most datasets, thereby confirming the hypothesis that utilizing both representations simultaneously can benefit from their respective strengths.

A detailed analysis of the performance on the PAMAP2 dataset (Figure 4.8) and the EMG-G dataset (Figure 4.9) underscores the effectiveness of the Cross-Atten model. Notably, Cross-Atten surpasses the IF-Conv model on the PAMAP2 dataset by a significant margin of 2.02%, and outperforms the GlobalFusion model on the EMG-G dataset by 1.42%. These statistically significant improvements demonstrate the superior ability of Cross-Atten to leverage the complementarity between the two representations. By simultaneously utiliz-

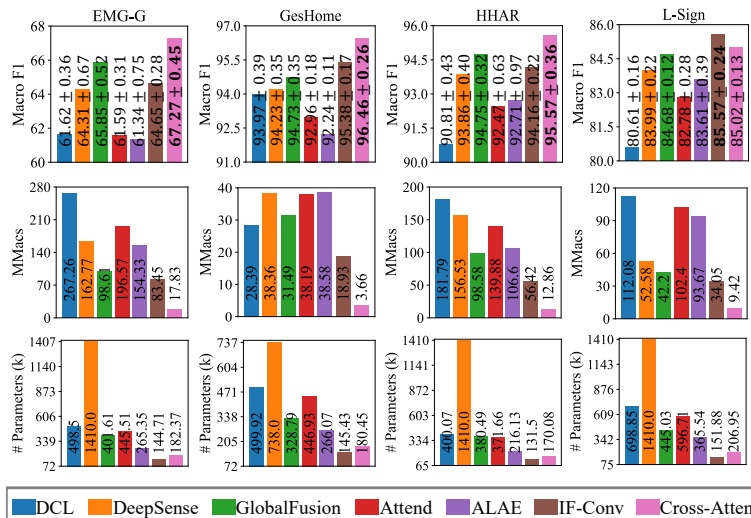


Figure 4.9: Performance comparison on four HAR datasets for proposed Cross-Attend model and other SOTA HAR models (II).

ing both representations for feature extraction, and facilitating their interaction through the cross-attention block, the model generates enhanced features enriched by complementary information from the opposite branch. This, in turn, boosts the model’s overall predictive performance.

When closely examining the second row in Figure 4.8, Figure 4.9, and Figure 4.10, it becomes evident that the proposed Cross-Attend model stands out for its minimal complexity compared to the other evaluated models. Why is the complexity of this dual-branch model still lower than that of single-branch models? This is because the model’s complexity is quadratically related to the number of filters. In our two-branch model, each branch uses 32 filters, which significantly reduces the computational complexity compared to a single-branch model with 64 filters. Furthermore, following the guidelines from the TinyHAR work, Cross-Attend minimizes the temporal dimensions wherever possible. For the spectrogram representation, since the STFT is applied, the raw sequence is split into T intervals, where the temporal dimen-

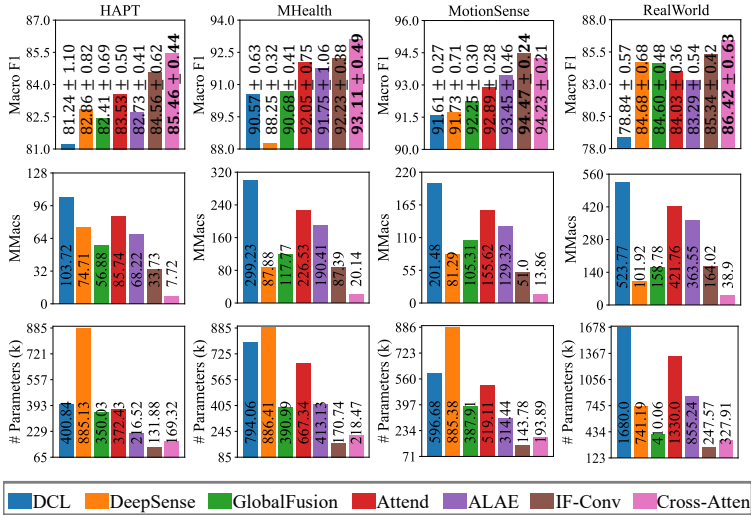


Figure 4.10: Performance comparison on four HAR datasets between the proposed Cross-Atten model and other SOTA HAR models (III).

sion T is considerably smaller than the original sequence length L .

When closely examining the third row in Figure 4.8, Figure 4.9, and Figure 4.10, it is clear that the proposed Cross-Atten model ranks second in terms of having the fewest trainable parameters. The IF-Conv model holds the fewest trainable parameters among all the compared models. To further explore the potential for model compression when utilizing two representations, we conducted additional experiments. Specifically, we reduced the filter count d from 32 to 24 of the proposed Cross-Atten model. This adjustment aims to further decrease the model’s computational complexity and the total number of trainable parameters, which are lower than those of the IF-Conv model.

The results for Cross-Atten with $d = 24$ are presented in Figure 4.11 and Figure 4.12. As shown in the figures, reducing d from 32 to 24 decreases the model’s size, making it smaller than the IF-Conv model and further reducing its complexity. Despite the reduction in size, the performance gap between Cross-Atten with $d = 32$ and Cross-Atten with $d = 24$ remains minimal. In nine out

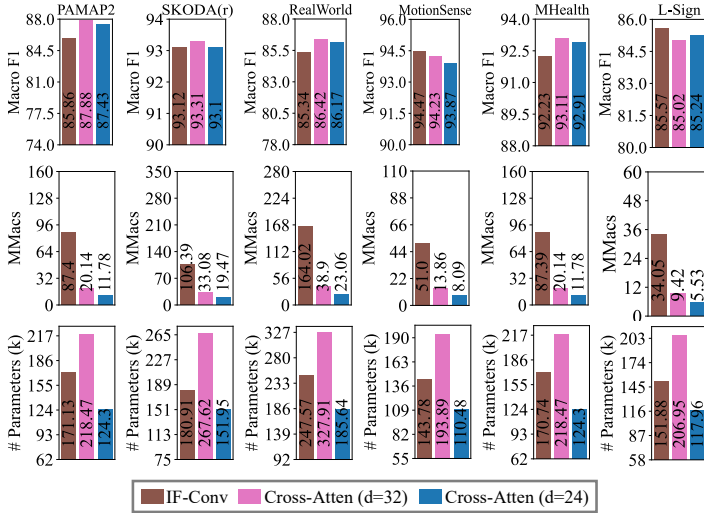


Figure 4.11: Performance comparison on six HAR datasets between the reduced Cross-Atten model and the IF-Conv model (I).

of twelve datasets, the reduced model, Cross-Atten ($d = 24$), still outperforms the IF-Conv model. On the Skoda(r) dataset, both the reduced Cross-Atten ($d = 24$) and the IF-Conv model achieved comparable performance. Only on the MotionSense and L-Sign datasets did the IF-Conv model outperform the reduced Cross-Atten ($d = 24$). However, it is important to emphasize that both the size and complexity of the IF-Conv model are significantly greater than those of the reduced Cross-Atten ($d = 24$).

4.3.2.3 Ablation Study. In order to verify the contributions of the cross-attention block, we performed a detailed ablation study on seven datasets. In addition to the proposed models, we consider here three more variants as follows: **V1** Branch-TS that perform the classification only by operating on time-series representation. **V2** Branch-SC that predicts the activity by only operating on spectrogram representations. **V3**, dual-branch model that operates both representations but without cross-attention. The architecture of all three model variants follow the structure design as described in section 4.4.1,

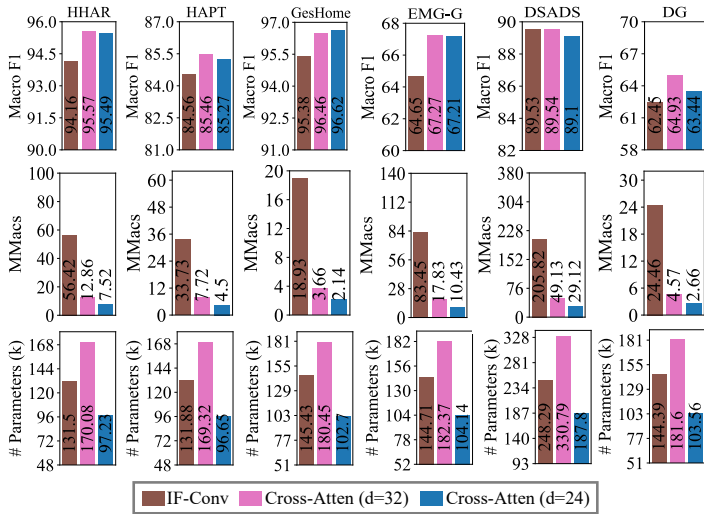


Figure 4.12: Performance comparison on six HAR datasets between the reduced Cross-Atten model and the IF-Conv model (II).

only the cross-attention block is replaced as self-attention block. For a fair comparison, by varying the parameter d , all model variants have a number of parameters comparable to the proposed model Cross-Atten.

The comparative results are listed in Table 4.4. By comparing model variants V1 and V2, they perform differently on different datasets, sometimes V1 is better, sometimes V2 is better. The model structure of V1 and V2 is exactly the same, the only difference is the input representation type. Again, the effect of representation on the results is highlighted. V3 has gained momentum by taking the features extracted from the two representations and concatenating them directly. Model V3 even outperformed the SOTA model on some datasets. For example, V3 outperforms HAR models GlobalFusion and IF-Conv on DSADS, RealWorld, HAPT and Skoda(r) datasets. When cross-attention is utilized, which enables the information of the two branches to be further exchanged during the feature learning process, so the proposed model Cross-Atten further improves the performance of V3 in all cases except on

Table 4.4: Ablation study on seven datasets

	PAMAP2	DSADS	DG	RealWorld
V1 (TS)	86.29 ± 0.44	86.98 ± 0.63	61.35 ± 0.92	83.96 ± 0.62
V2 (SC)	85.17 ± 0.68	86.24 ± 0.55	63.73 ± 1.26	84.43 ± 0.54
V3 (TS+SC)	86.76 ± 0.53	88.47 ± 0.49	63.70 ± 0.43	85.59 ± 0.37
Cross-Atten	87.88 ± 0.61	89.54 ± 0.66	64.93 ± 0.68	86.42 ± 0.63
	HAPT	MotionSense	SKODA(r)	
V1 (TS)	82.29 ± 0.65	92.82 ± 0.43	92.17 ± 0.31	
V2 (SC)	83.92 ± 0.68	93.54 ± 0.29	90.65 ± 0.48	
V3 (TS+SC)	84.57 ± 0.40	93.97 ± 0.27	93.49 ± 0.26	
Cross-Atten	85.46 ± 0.44	94.23 ± 0.21	93.31 ± 0.27	

Table 4.5: Performance on 10 datasets from the UEA multivariate time series classification archive.

	MiniRocket	TST	TimeNet	WHEN	Cross-Atten
EthanolConcentration	0.475	0.326	0.357	0.422	0.341
FaceDetection	0.620	0.689	0.686	0.658	0.696
Handwriting	0.511	0.359	0.321	0.561	0.388
Heartbeat	0.766	0.776	0.780	0.780	0.785
JapaneseVowels	0.984	0.997	0.984	0.995	0.995
PEMS-SF	0.827	0.896	0.896	0.925	0.919
SelfRegulationSCP1	0.904	0.922	0.918	0.908	0.934
SelfRegulationSCP2	0.506	0.604	0.572	0.589	0.612
SpokenArabicDigits	0.991	0.998	0.997	0.990	0.998
UWaveGestureLibrary	0.925	0.913	0.919	0.853	0.919

the Skoda(r) dataset. These observations validate our hypothesis and also the design of the model design.

4.3.2.4 Generalization Performance on UEA Datasets. The HAR task is fundamentally a multivariate time series analysis. To illustrate the adaptability and generalizability of the proposed model, ten datasets are selected from the UEA Time Series Classification Archive [12] for empirical evaluation. The selection criteria for these datasets are based on prior research [173; 160]. These datasets encompass a broad spectrum of applications, including audio recognition, medical diagnostics, heartbeat monitoring, and other real-world applications. For comparative analysis, the baseline model is Mini Rocket [38], which is a representative method of feature-based models with a ridge regression classifier. Moreover, this evaluation incorporates three SOTA models specif-

ically designed for multivariate time-series analysis tasks; they are Time Series Transformer (TST) [173], TimesNet [160] and WHEN [154].

As illustrated by the Table 4.5, the proposed model attained accuracy comparable to other models under comparison. It secured first place in five out of ten datasets. In the remaining five datasets, it achieved second place in three datasets. This outstanding and robust performance underscores the advantage of utilizing both representations simultaneously.

4.3.3 Discussion

This work highlighted the importance and advantages of simultaneously utilizing both time-series and spectrogram representations for HAR tasks. The careful selection and application of appropriate input representations can enhance model performance. When combining these two representations through the proposed two-branch attention-based model, it is not necessary to rigorously optimize each representation and its parameters. Comprehensive experiments on twelve HAR benchmark datasets demonstrated the superior performance of the proposed model over other contemporary SOTA HAR models. The ablation study further validated the contribution of integrating the cross-attention block. Due to the strong feature extraction capability of the proposed dual-branch architecture, the number of parameters used is significantly smaller compared to other models, while still surpassing their performance. However, this model can only be deployed on edge devices like smartwatches and is not suitable for deployment on micro-controllers.

4.4 MLP-HAR

The proposed TinyHAR and Cross-Atten models demonstrate strong feature extraction capabilities despite their small size. However, both approaches overlook hardware constraints, particularly in highly resource-constrained edge devices like micro-controllers. Specifically, TinyHAR integrates the unique characteristics of HAR tasks into the model's structural design, while Cross-Atten compensates for inherent limitations in neural networks by leveraging multi-representations. Nevertheless, neither model accounts for hardware limitations

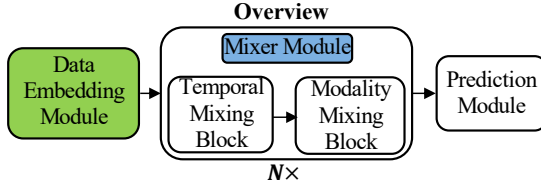


Figure 4.13: This figure presents an overview of the architecture of the proposed MLP-HAR model.

or deployment requirements. For instance, the dual-branch structure imposes significant memory access demands, and the RNN architecture in TinyHAR, due to its inability to parallelize, negatively impacts inference time. Taking these considerations into account, this section addresses such hardware limitations by designing a model that acknowledges the specific characteristics and constraints of resource-constrained devices. Consequently, we propose the MLP-HAR model [187].

4.4.1 Methodology

In this section, we elucidate the proposed model MLP-HAR, illustrated in Figure 4.13, which comprises three main modules: the Data Embedding Module, the Information Mixing Module, and the Prediction Module. The Data Embedding Module extracts local temporal features from raw data. The Information Mixing Module employs a repeated alternating structure to facilitate information exchange and integration across temporal and sensor channel dimensions. Finally, the Prediction Module condenses the extracted features to make the final prediction. All modules consist solely of fully connected (FC) layers.

4.4.1.1 Data Embedding Module. The detailed structure of the Data Embedding Module is illustrated in Figure 4.14 (a). The design is inspired by the previously introduced Cross-Atten model, leveraging two representations: the time-series representation and the spectrogram representation. Unlike the Cross-Atten model, in the Data Embedding Module, features extracted from the two representations are concatenated early to avoid the prolonged dual-branch structure. Specifically, in this module, the raw input segment $\mathbf{X} \in \mathbb{R}^{L \times C}$

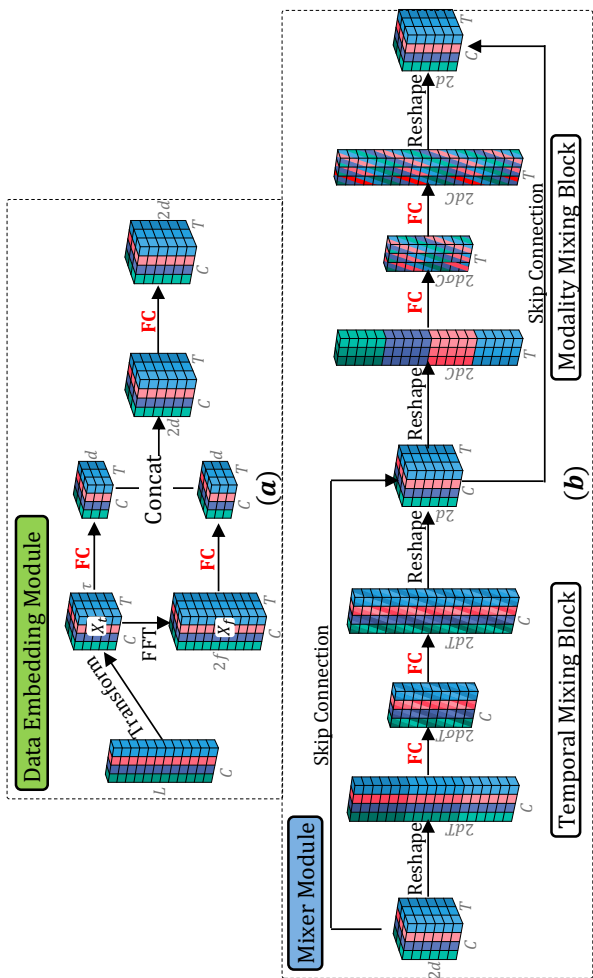


Figure 4.14: Figure (a) shows the specific structure of the data embedding module. Figure (b) shows the specific structure of the mixer module. Different colors in the figure represent readings of different sensor channels, and different shades of the same color represent different intervals. Mixed color shows fused information from FC layer.

(where L is the sliding window size and C is the number of sensor channels) is divided into intervals of length τ , resulting in T intervals, denoted as $\mathbf{X}_t \in \mathbb{R}^{T \times C \times \tau}$. Each interval then undergoes an FFT transformation to extract frequency-domain representations, yielding $\mathbf{X}_f \in \mathbb{R}^{T \times C \times 2f}$, where f represents the frequency magnitude and phase pairs.

The benefits of this process are twofold: (1) Interval segmentation reduces the raw data’s temporal length, lowering computational demands in subsequent steps. (2) Frequency features are crucial for differentiating human activities, as signals acquired through wearable sensors often exhibit multi-frequency characteristics [102]. Extracting frequency information directly via FFT simplifies the model’s task of frequency feature extraction [164]. Both the time and frequency representations are then processed through separate FC layers, each followed by layer normalization [9] and a ReLU activation. The output size of the FC layers is d . The feature maps extracted from both representations are concatenated to form a combined feature with dimensions $T \times C \times 2d$, which is then further fused by an additional FC layer to maintain consistent dimensions.

4.4.1.2 Mixer Module. After embedding the local information from each sensor channel, the Mixing Module fuses this information across both temporal and sensor channel dimensions. As shown in Figure 4.13 (b), this module consists of two components: the temporal mixing block and the modality mixing block.

Temporal Mixing Block. The temporal mixing block fuses information within each sensor channel along the temporal dimension. It begins by flattening the feature map from $T \times C \times 2d$ to $C \times 2dT$, followed by layer normalization to standardize features within each channel. The feature map then passes through two FC layers. To reduce the parameter count, the first FC layer decreases the feature size using a shrink ratio v , while the second FC layer restores it to $2d$. A ReLU function is applied after the first FC layer as activation. These two FC layers enable efficient feature exchange across different temporal intervals. A skip connection reintegrates the fused features back into the original feature space, allowing each interval to incorporate information from others. The process concludes with another layer normalization before reshaping the data back

to $T \times C \times 2d$.

Modality Mixing Block. The Modality Mixing Block enables the exchange and fusion of information between intervals at the same time step but in different modalities. It omits layer normalization since it was applied in the temporal mixing block. Initially, the features are flattened based on the modality dimensions, transforming the feature map from $T \times C \times 2d$ to $T \times 2dC$. Subsequently, two FC layers facilitate interactions among features of different modalities. Finally, the fused features across modalities are reintegrated into the original feature set via a skip connection.

Discussion. This design enables rapid information propagation across different time steps and modalities. Unlike traditional HAR models, which follow a fixed sequence of feature extraction—first channel interaction, then temporal information—our model adopts a more flexible approach. By stacking this mixer model N times, the iterative process allows multiple integrations of information across both dimensions for each interval, enhancing the model’s ability to extract powerful features.

4.4.1.3 Prediction Module. This module includes two FC layers. The first FC layer fuses the features from each sensor channel into a single vector. The second FC layer then uses these fused vectors from all sensor channels to predict the final activity.

4.4.2 Experiments and Discussions

4.4.2.1 Experiment Setup. To validate the performance of our model, we conducted experiments using six open-source datasets: PAMAP2 [127], HAPT [128], DSADS [14], Daphnet [11], MotionSense [107], and MHealth [13]. For certain models, including ours, the segmented raw data undergoes an FFT transformation to generate spectrograms. The parameters for this transformation, such as the number of intervals (T), interval length (τ), and the number of amplitude and phase spectral pairs (f), are provided in Table 4.6. The FFT transformation is implemented directly within the model using the `torch.fft` function, with

Table 4.6: Hyper-parameters related to the Short-Time Fourier Transform for each dataset.

Dataset # <i>SW</i>	# <i>T</i>	# τ	# <i>f</i>
PAMAP2 [127]	1.28 s	8	16
HAPT [128]	2.56 s	8	16
DSADS [14]	5 s	5	25
Daphnet [11]	1 s	4	16
MotionSense [107]	2.56 s	8	16
Mhealth [13]	2.56 s	8	16

the model accepting only raw time-series data as input. In subsequent evaluations, the computational time and complexity of the FFT transformation are accounted for in the measurement of inference time and overall model complexity.

In this experiment, we compared our approach against following ten comparative models. **DCNN** [166], **DCL** [119], **DCL-Attn** [113], **Attend** [1] **IF-ConvT** [178], **ALAE-TAE** [3], **DeepSense** [167], **GlobalFusion** [99]: **TinyHAR** [185], **MLP** [117]: which has a similar model architecture to the proposed MLP-HAR model. The key difference is that the MLP model treats the input time series as an image and directly applies the MLP architecture designed for vision tasks [147] to the HAR task. It is important to note that, except for the MLP model [117], the configurations of all the aforementioned models strictly adhere to their descriptions and source code as presented in the referenced literature. The MLP model’s configuration in the original work [117] is much larger than the other comparison models and cannot run on the watch due to exceeding the watch’s memory. The specifications used for the MLP model in this experiment are: 5 layers, a patch-embedding size of 256, a patch dimension of 64, and a channel dimension of 256. For our proposed MLP-HAR model, we fix the number of mixer modules N at 2 and the filter number d at 6.

In addition to reporting the mean and variance of F1-scores from the LOSO-CV experiments, we also provide the model size in terms of the number of trainable parameters (in thousands) and computational complexity in Million MACs (MMACs). Furthermore, the models are deployed on a Samsung Galaxy

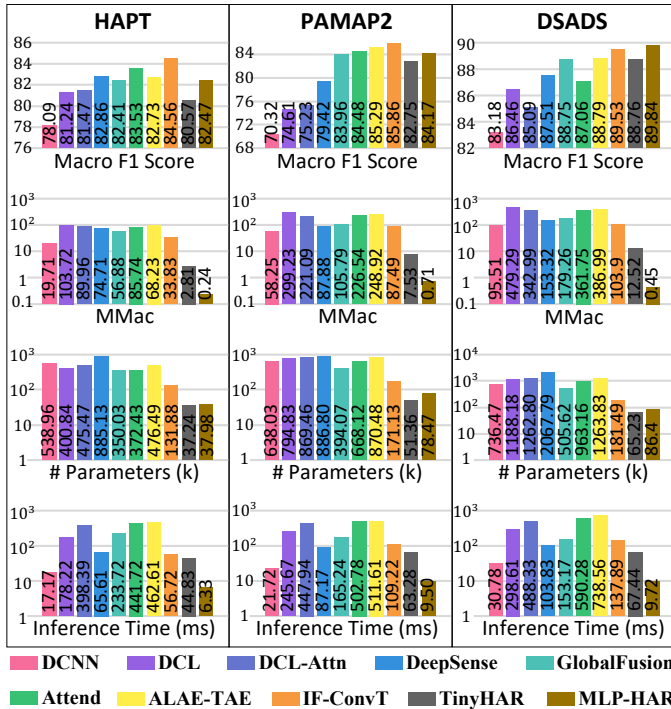


Figure 4.15: Performance comparison on three HAR benchmark datasets between the MLP-HAR model and other SOTA HAR model (I).

Watch 5 Pro¹ to evaluate their real-world performance. We report the average inference time for each model to process 10,000 input samples on the watch.

4.4.2.2 Comparison to State-of-the-art. Figures 4.15 and 4.16 present the performance of all models across six datasets, visualizing the averaged macro F1 score, model complexity, model size, and inference time for each dataset in column blocks. Since our model and TinyHAR exhibit significantly lower computational complexity and fewer learnable parameters compared to the other models, the y-axes for model complexity and size are logarithmically

¹For deployment, we used post-training int8 quantization from TensorFlow and the inference framework is TensorFlow Lite Micro.

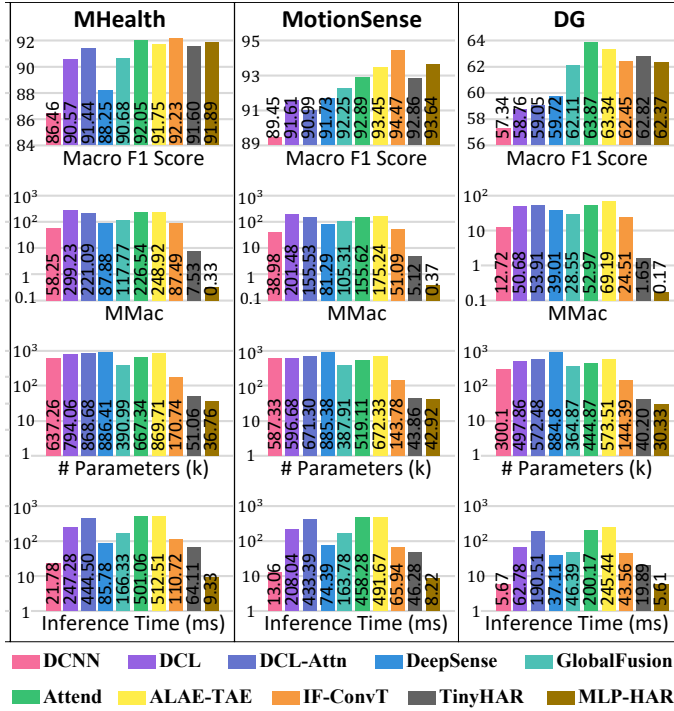


Figure 4.16: Performance comparison on three HAR benchmark datasets between the MLP-HAR model and other SOTA HAR model (II).

Table 4.7: Average ranking of all models across six datasets. The smaller the rank, the better the performance.

IF-ConvT	ALAE-TAE	Attend	MLP-HAR	TinyHAR	GlobalFusion
1.83	3.17	3.33	3.33	5.67	6.00
MLP	DeepSense	DCL-Attn	DCL	DCNN	
6.17	7.33	8.83	9.33	11.00	

scaled with a base of 10.

Our proposed MLP-HAR model achieves superior performance on the DSADS dataset and comparable performance on the PAMAP2, Mhealth, and Motion-sense datasets. Notably, to achieve such performance, our model requires only significantly fewer learnable parameters and lower complexity. For a comprehensive comparison, we conducted a pairwise average rank comparison using the Wilcoxon signed-rank test with Holm’s alpha correction at 5% [157; 60]. The average ranking across all datasets, listed in Table 4.7, shows the IF-ConvT model achieving the best performance, followed by the Attend, ALAE-TAE, and proposed MLP-HAR models, which form a closely ranked cluster with no significant performance differences. Compared to the best-performing model IF-ConvT, our MLP-HAR can provide $10\times$ speed up in inference time while lose only 0.74% lower F1 score on average across six datasets. This slight accuracy drop can be further mitigated by the post-process in practical application, see Section 4.4.2.3.

Compared to the model TinyHAR, our MLP-HAR, despite having a similar number of learnable parameters, demonstrates significantly ($10\times$) lower computational complexity. This reduced complexity also reflects in the inference time on the device, with our model running about $6\times$ faster on average compared to TinyHAR.

Compared to the benchmark model DCL, MLP-HAR significantly outperforms it. The DCL model has much greater complexity and substantially slower inference times. For instance, when processing the DSADS dataset, DCL requires 298.61 ms, whereas our model operates in just 10.61 ms. Enhancements to the DCL framework, such as DCL-Attn, Attend, and ALAE-TAE models, have shown effectiveness in our experiments. However, these models are even slower than DCL. Interestingly, on most datasets, these three

models exhibit generally lower complexity than DCL. Theoretically, the DCL model, which incorporates only one LSTM layer, should exhibit lower complexity than these three models, which use two LSTM layers. However, the large kernel size of 11 in DCL's convolutional layers significantly increases its complexity. This observation highlights two key points: convolutional layers processing raw data contribute substantially to the model's overall complexity, and inference time is influenced not only by computational complexity but also by architectural design. The excessive use of LSTM layers and self-attention in the DCL-Attn, Attend, and ALAE-TAE models, due to the sequential calculations inherent in LSTM layers which lack efficient parallelism, further slows down their inference times.

This observation that structural complexity can slow down inference is underscored by comparing the DeepSense and GlobalFusion models. Despite having lower model complexity per dataset, GlobalFusion exhibits significantly slower inference times. This slowdown is due to its complex multi-branch structure, which incorporates self-attention layers for global information fusion. This example highlights the importance of our proposed model's design philosophy: its plain topology is deployment-friendly, emphasizing efficiency and simpler architectural choices that facilitate faster processing speeds.

Among all models examined, the DCNN model is the least effective. Despite its higher computational complexity and greater number of learnable parameters, it achieves the second-fastest inference time on the device. For example, when processing the DG dataset on a smartwatch, its inference time is comparable to that of the proposed MLP-HAR, even with a higher parameter count and complexity. This underscores that a plain topology, combined with the use of CNN and FC layers, significantly aids in the efficient deployment of models. This observation is further supported by the performance of the MLP model, which is fast despite having many parameters and high computational complexity due to its configuration. However, since the MLP model completely ignores the characteristics of the HAR task, its performance is much worse than that of the proposed MLP-HAR model.

Table 4.8: Comparison of performance before and after post-processing. The model name + P stands for post-processing.

	HAPT	PAMAP2	DSADS	MHealth	MotionSense	DG
IF-ConvT	84.56	85.86	89.53	92.23	94.47	62.45
IF-ConvT+P	87.28	87.42	92.49	97.45	98.96	63.04
MLP-HAR	82.28	84.17	89.90	92.03	93.64	62.63
MLP-HAR+P	86.93	87.50	92.81	97.29	98.73	62.11

4.4.2.3 Post-Processing Experiment. In the deployment of HAR models, transitions between different activities are typically gradual. To enhance the robustness and accuracy of predictions, a post-processing technique known as majority voting is frequently utilized. To assess the performance of IF-ConvT and our proposed MLP-HAR model during deployment, we applied post-processing to all their predictions. The device operates with a double buffering mechanism, where one thread collects and buffers data, and another thread manages model inference. The window size for majority voting was set to 10. Table 4.8 illustrates the results of both models before and after post-processing. The results indicate that post-processing generally improves model performance, especially when the models already demonstrate good initial performance. For example, on the MotionSense and MHealth datasets, both models showed significant improvements. On the HAPT dataset, the performance gap between IF-ConvT and MLP-HAR narrowed from 2.28% to 0.35%. Notably, models with faster inference times hold a distinct advantage in actual deployment. A model with shorter inference times can update its predictions more frequently, allowing it to process more data windows within the same amount of time when using a majority voting strategy, thereby solidifying the results. Conversely, if a model’s inference time is long, a large voting window can lead to response delays.

4.4.2.4 Ablation Study and Parameter Analysis. To validate the impact of different representations in the data embedding module and the effect of skip connections in the mixer module, we conducted an extensive ablation study. We evaluated the model using only *frequency* representation, only *temporal* representation, and *both* representations in the data embedding module. Addi-

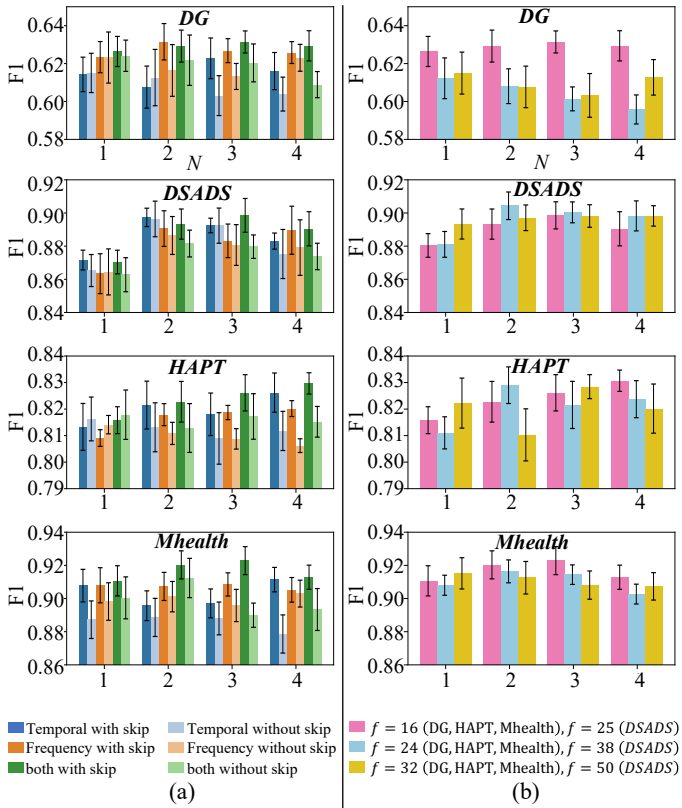


Figure 4.17: (a) Ablation study validating the contributions of different representations and skip connections, along with an impact analysis of parameter N in the MLP-HAR model. (b) Impact analysis of parameters f and N in the MLP-HAR model.

tionally, we examined the impact of including (*True*) or excluding (*False*) skip connections in the mixer module. This resulted in six different model configurations. We also investigated the effect of the number of blocks N on performance, training and evaluating the six model configurations with N set to 1, 2, 3, and 4. The results are presented in Figure 4.17 (a). The results show that using skip connections generally improves model performance. The performance of using only frequency or temporal representation varies across datasets. For example, on the DG and MHealth datasets, models using frequency representation outperform those using temporal representation, whereas on the DSADS and HAPT datasets, the opposite is true. Using both representations leverages the strengths of each, leading to improved performance. The model’s performance is also influenced by changes in N , varying by dataset. For instance, on the HAPT dataset, more blocks lead to better performance, while on the DSADS dataset, $N = 2$ achieves the best results. Increasing N raises model complexity and the number of parameters, introducing different over-fitting risks depending on the dataset. However, models with $N \geq 2$ consistently outperform those with $N = 1$. We speculate this is because $N \geq 2$ allows the model to perform multiple fusions across temporal and sensor channel dimensions.

After confirming the benefits of using both representations and skip connections, we examined the impact of parameter variations in the FFT transformation. Specifically, we varied the size of f and the number of blocks N while employing both representations and skip connections. Figure 4.17(b) shows the range of f and N values and their corresponding model performance. The results indicate that there is no universal combination of f and N that consistently improves performance across all datasets. However, optimizing these parameters for individual datasets does enhance performance. For instance, on the DSADS and HAPT datasets, adjusting f and N values resulted in significant performance improvements.

4.4.2.5 Deployment on Hardware. In this section, we explore the deployment of our model on the Arduino Portenta H7 LITE, a board with more limited computing capabilities. As previously shown, because the TinyHAR model demonstrates the lowest inference time and computational complexity aside

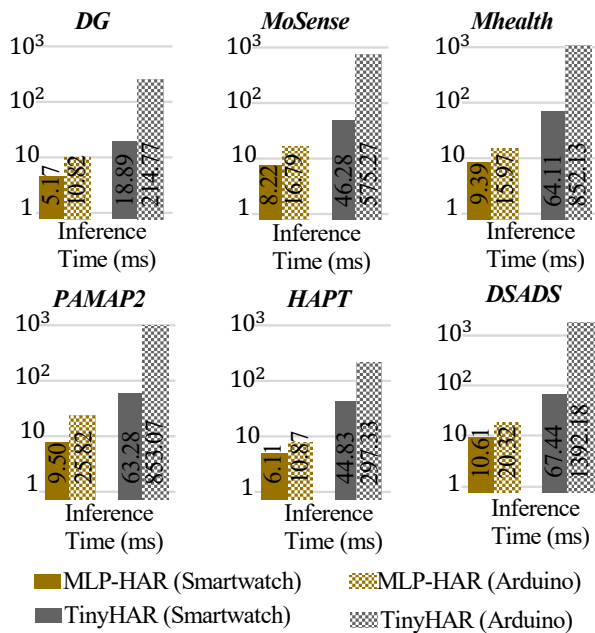


Figure 4.18: Comparison of inference time between the MLP-HAR and TinyHAR models on two devices. The Y-axis is logarithmically scaled (base 10).

from our proposed MLP-HAR model, we report only the inference times for TinyHAR and MLP-HAR on the Arduino Portenta H7 LITE. The results are illustrated in Figure 4.18. From the figure, it is evident that due to the reduced computational power of the device, the inference times of both models are longer compared to those on the smartwatch. However, it is noteworthy that the increase in inference time for TinyHAR is substantial, slowing down by at least $10\times$. In contrast, the increase in inference time for the proposed MLP-HAR model is approximately $2\times$ to $3\times$ between the two devices. This outcome underscores the superiority of our proposed model when deployed on devices with more restricted computing capabilities, which can be attributed to MLP-HAR’s plain topology composed solely of FC layers.

4.4.3 Discussion

In this work, we introduced a purely FC model architecture, thoughtfully designed not only to leverage the different saliencies of multi-modalities and temporal information extraction, but also to facilitate efficient deployment on edge devices. Experimental results demonstrate that compared to current SOTA HAR models, our model delivers comparable performance while boasting the smallest model size, minimal computational complexity, and the fastest inference time. When deployed on edge devices with limited computational capacity, the proposed model’s superior capabilities were further showcased, highlighting its potential for practical real-world applications where computational resources are at a premium.

4.5 Summary

In this chapter, we introduce three lightweight models for HAR tasks, starting with TinyHAR, which serves as the foundation for the other two models. TinyHAR demonstrates that by carefully considering the unique characteristics of HAR, a small model can still exhibit strong learning capabilities. However, despite its potential, TinyHAR remains structurally complex and falls short of achieving SOTA performance on some datasets.

Building on the design principles of TinyHAR, we developed two addi-

tional models: Cross-Atten and MLP-HAR. Cross-Atten is a semi-lightweight model that, while unsuitable for micro-controller deployment due to its dual-branch structure, performs exceptionally well on various datasets and is ideal for smartwatch deployment, thanks to its robust feature extraction capabilities. However, its high memory consumption limits its use in highly resource-constrained environments.

To overcome this, we designed MLP-HAR, a model optimized for micro-controller deployment. MLP-HAR matches the performance of SOTA models while delivering faster inference times, making it highly suitable for scenarios with stringent hardware limitations.

In summary, the primary contribution of this chapter is offering two models tailored for different edge computing environments: Cross-Atten for devices with moderate resources, and MLP-HAR for highly constrained devices. Both models are grounded in the design principles established by TinyHAR, which we believe will continue to guide future HAR model development.

5 Model Architecture Optimization

In the previous chapters, we introduced models that were manually designed, heavily relying on expert knowledge. In this chapter, we explore how to leverage automated machine learning (AutoML) techniques to automatically optimize model architectures, thereby reducing the dependency on expert knowledge.

5.1 Related Work

AutoML has garnered significant attention in recent years, primarily due to its ability to automate model selection and hyper-parameter tuning. Although AutoML techniques have been successfully applied in various domains, such as image classification and natural language processing, their application to time series classification, particularly for HAR tasks, remains relatively underexplored.

In the field of time series classification, AutoML methods have demonstrated promise in automating the design and optimization of models. Frameworks like Auto-sklearn [48] and TPOT [118] have been developed to select and tune models for time series data, focusing predominantly on traditional machine learning approaches, such as decision trees and ensemble methods. More recently, deep learning-based AutoML frameworks, such as Auto-Keras [72], have begun incorporating neural architectures for time series classification, leveraging their capability to capture temporal dependencies. These frameworks typically explore predefined neural architectures, such as RNNs, CNNs, and their variants, to discover optimal configurations for time series data.

However, despite advancements in applying AutoML to general time series tasks, the specific domain of HAR has seen limited exploration. HAR presents distinct challenges, including its multi-modal nature, sensor noise, and the

need for real-time inference on resource-constrained devices. Current AutoML frameworks are not explicitly designed to address these challenges, as they typically target static datasets with fewer constraints on computational resources. Additionally, custom neural architectures tailored to HAR tasks—particularly those optimized for capturing both spatial and temporal dependencies across multiple sensor modalities—are rarely included in the search space of existing AutoML frameworks.

Given the limited research on AutoML for HAR tasks, there is a clear opportunity to automate the model design process specifically for HAR, addressing its unique challenges and constraints.

5.2 AutoML Framework : ECLSTM

The AutoML framework consists of two fundamental components: the search space and the optimization algorithm. Defining the search space is the first critical step in constructing an AutoML framework. As previous studies [72] have demonstrated, the choice of search space has a substantial impact on the performance of the resulting models. Given the specific challenges of HAR, we propose that the search space for HAR models should incorporate two key aspects: first, it must be designed to capture the unique characteristics of HAR tasks; second, to streamline the search process and reduce complexity, the operators should be stackable, ensuring a consistent structure across layers.

With these considerations, we propose an AutoML framework specifically tailored for HAR tasks, utilizing a novel operator we introduce, called Embedded Convolutional LSTM (ECLSTM) [184].

5.2.1 Methodology

In the following sections, we will first introduce the innovative operators that constitute the search space in Section 5.2.1.1, followed by a detailed discussion of the search space definition in Section 5.2.1.2. Finally, the optimization algorithm will be presented in Section 5.2.1.5.

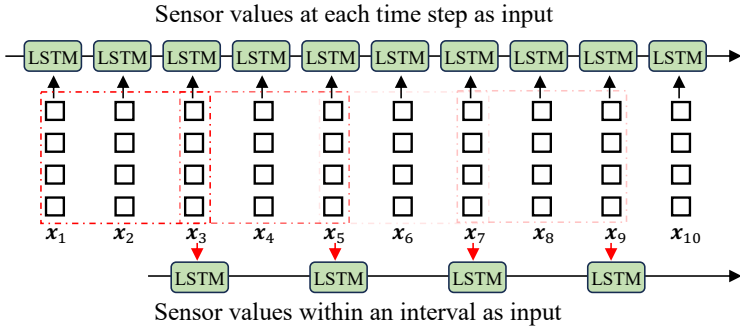


Figure 5.1: Sensor values at each time step or within an interval are sequentially fed into the LSTM.

5.2.1.1 Embedded Convolutional LSTM. In Section 2.3, we reviewed current SOTA HAR models and observed that most designs follow a common strategy: CNNs are typically used to extract local information, followed by LSTM modules to capture temporal dependencies. When LSTMs are applied without CNNs, they often fail to capture local context effectively, leading to sub-optimal results. To address this issue, several approaches have been proposed to enhance the ability of traditional LSTMs to capture local context. Specifically, rather than feeding only the value at each time step, values within an interval are provided to the model. As shown in Figure 5.1, the input to the LSTM is no longer individual values at each time step but an interval, represented by the red dashed bounding box.

However, since traditional LSTMs are designed to accept only one-dimensional inputs, this two-dimensional interval must first be projected into a 1D vector, as illustrated in Figure 5.2. As discussed earlier, HAR time series data have two critical dimensions: one representing temporal dependencies and the other representing sensor channel relationships. In Figure 5.2, sensor values at the same time step are aligned in a column, indicated by the same color, while values from the same sensor channel are arranged in a row, denoted by the same shape, preserving both spatial and temporal relationships. However, after flattening, one of these critical dimensions is inevitably compromised.

Moreover, converting a 2D structure into a 1D vector significantly increases

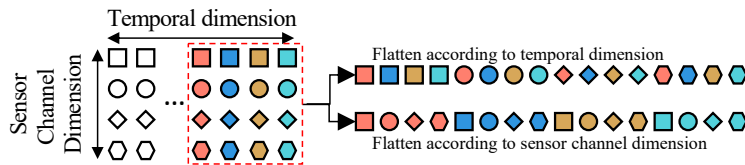


Figure 5.2: Multiple features (represented as shapes) at various sampling times (represented as colors) within a single interval must be flattened into a 1D vector for LSTM input, disrupting the natural sequence. Alternatively, a convolutional kernel can be applied to aggregate local information while preserving the original structure.

the input vector size, leading to a larger number of learnable parameters and heightened computational complexity.

Inspired by these findings, and with the aim of developing a universal operator for constructing the search space, we propose an extension of the standard LSTM: the Embedded Convolutional LSTM (ECLSTM). This model incorporates a set of 1D convolutions within the LSTM structure.

This approach enables the input to remain as a 2-dimensional tensor, preserving the information within the interval along both dimensions. Moreover, the embedded 1D convolutions allow for the extraction of local context as information flows along the temporal dimension, effectively overcoming the limitations of traditional LSTMs in capturing local context. Additionally, this design increases the flexibility of the search space, as ECLSTM can operate effectively even when used in isolation. We hypothesize that the ECLSTM architecture is more powerful than standard LSTMs for handling multivariate time series tasks.

Specifically, to integrate CNNs into the LSTM, we replace the fully connected layers in the standard LSTM with convolutional operations. The resulting equations for the ECLSTM are thus formulated as follows:

$$\begin{aligned}
i_t &= \sigma(W_{xi} * x_t + W_{hi} * h_{t-1} + b_i), \\
f_t &= \sigma(W_{xf} * x_t + W_{hf} * h_{t-1} + b_f), \\
o_t &= \sigma(W_{xo} * x_t + W_{ho} * h_{t-1} + b_o), \\
\tilde{C}_t &= \tanh(W_{xc} * x_t + W_{hc} * h_{t-1} + b_c), \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t, \\
h_t &= o_t \odot \tanh(C_t),
\end{aligned}$$

where $*$ denotes the convolutional operation and \odot the element-wise product. σ is the sigmoid activation function, \tanh is the hyperbolic tangent function, i_t , f_t , o_t , C_t , and h_t are the input gate, forget gate, output gate, cell state, and hidden state at time step t , respectively, and W_{xi} , W_{hi} , W_{xf} , W_{hf} , W_{xo} , W_{ho} , W_{xc} , and W_{hc} are the convolutional weight matrices.

There are three key advantages to using the convolution operator within LSTM. First, the convolution parameters are determined by the kernel size and the number of filters, rather than the length of the input interval, ensuring that the model's complexity remains constant even as the interval size increases. Second, the hidden state (H) and memory (C) are represented as 2D tensors, inherently preserving the temporal relationships within the data. Third, by inputting data as intervals, the model reduces the number of iterative steps required along the temporal dimension, thus enhancing its inference time.

It is well-known that stacking convolutional layers allows for a hierarchical decomposition of raw data and the combination of lower-level features. Therefore, to extract more complex features from the local context, the convolutions in Equation 5.2.1.1 can be stacked as convolutional cells in a chain structure. When three convolutional layers are stacked within the cell, we refer to the ECLSTM as a 3-depth-ECLSTM. For example, considering the input gate in ECLSTM, the formula for the input gate can be expressed as:

$$i_t = \sigma(W_{xi}^{(3)} * \sigma(W_{xi}^{(2)} * \sigma(W_{xi}^{(1)} * x_t) + W_{hi} * h_{t-1} + b_i)$$

where $W_{xi}^{(1)}$, $W_{xi}^{(2)}$ and $W_{xi}^{(3)}$ represent the weights of the three stacked convolutional layers. The other gates follow the same structure but do not share these weights.

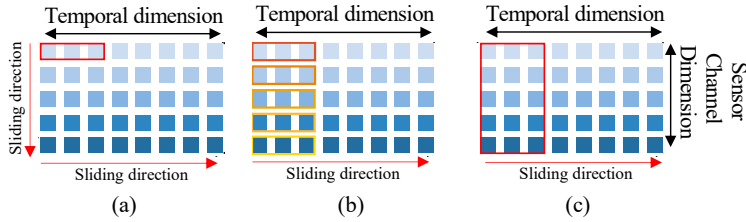


Figure 5.3: (a) In hybrid fusion convolution, the kernel height remains 1, but the kernel slides in two directions: along both the temporal axis and the sensor channel axis. Thanks to weight sharing, this approach significantly reduces the number of parameters. (b) In late fusion convolution, the kernel height is set to 1, with each sensor channel assigned its own convolution kernel, which also slides along the temporal axis. (c) In early fusion convolution, the kernel height is fixed to match the number of sensor channels, and the convolution kernel slides along the temporal axis.

Moreover, results from various HAR models, such as [31], suggest that different fusion strategies can significantly influence performance. Inspired by these findings, the convolutional cell in our ECLSTM operator incorporates three distinct fusion strategies, as illustrated in Figure 5.3. The first is early fusion convolution, which functions similarly to conventional 2D convolution, where features are jointly extracted from all sensory data. The second is late fusion convolution, where features are extracted separately from each sensor using individual convolutions, with each sensor channel having its own convolutional kernel, ensuring no weight sharing. The third is hybrid fusion convolution, where features are also extracted separately from each sensor, but the weights are shared across sensors.

By integrating CNN and LSTM, we combine the strengths of CNN in extracting local features and LSTM in capturing temporal dependencies, while also enabling the use of various fusion strategies. The next challenge is to define the search space, determine the optimal depth of the convolutional layers, and select the appropriate types of convolutions to be used within the ECLSTM architecture.

5.2.1.2 Automatic Prediction Framework. Considering the complexity of hyper-parameter selection, even for domain experts, we introduce an AutoML framework based on the proposed ECLSTM operator. The search space of this framework is designed following the guidelines proposed by TinyHAR. This framework consists of three stages: local context extraction stage, temporal information extraction stage, and prediction stage. The temporal information extraction stage is composed of a series of ECLSTM operators, which not only effectively capture temporal dependencies but also incorporate different fusion strategies from the convolutional layers, facilitating the learning of interactions and fusion across sensor channels. The structure and search space of each stage will be introduced individually.

Local Context Extraction Stage. When the sampling frequency is too high or the sliding window size too large, it becomes impractical to directly use raw data as input to an ECLSTM layer. This is because ECLSTM, as an RNN-based model, processes information iteratively and cannot perform parallel processing, resulting in slow inference times. Therefore, it is essential to reduce the temporal dimension. Compared to pooling or downsampling, a more effective method for reducing the temporal dimension is using convolutional layers. Pooling or downsampling can lead to significant information loss, while convolution provides a more controlled reduction. Consequently, we first apply convolution as a preprocessing step to reduce the temporal dimension. By adjusting the kernel width, stride, and dilation rate, we can fine-tune the level of dimensionality reduction while simultaneously extracting local context. In this step, individual convolutions are used, meaning the kernel height is set to 1.

The search space for this part is presented in Figure 5.4 under the "Pre-Processing" section. There are dependencies between the parameters. If the number of convolutional layers (N_{conv}) is set to 0, no preprocessing (temporal reduction) is applied. Only when the number of layers exceeds 0 are other parameters, such as filter size (d), stride size (st), and kernel size (k), considered. Rather than optimizing these parameters individually for each convolutional layer, we enforce a simple rule: all convolutional layers have the same number of filters, stride, dilation rate, and kernel width.

Part	Conditioned on	Parameter Name	Range	Type
Pre-Processing	–	N_{conv} : No. of Layers	[0,5]	<i>int</i>
	N_{conv} : No. of Layers	d : Filter Size	[2,64]	<i>int</i>
		st : Stride Size	[1,2]	<i>int</i>
		k : Kernel size	(1,3,5,7)	<i>int</i>
		1D Max Pooling	(True, False)	<i>cat</i>
Feature Extraction	–	N_{ectstm} : No. of Layers	[1,3]	<i>int</i>
	N_{ectstm} : No. of Layers	Cell Depth	[1,3]	<i>int</i>
		Cell Depth	d : Filter Size	[4,64]
	k : Kernel size		(1,3,5,7)	<i>int</i>
	Conv Type		(Early,Hybrid,Late)	<i>cat</i>
Activity Prediction	–	N_{fc} : No. of Layers	[1,4]	<i>int</i>
	N_{fc} : No. of Layers	d : Filter Size	[4,64]	<i>int</i>

Figure 5.4: Overview of the search space for the ECLSTM framework.

5.2.1.3 Temporal Feature Extraction Stage. This stage is central to the model, as it is responsible for extracting both temporal and local features. The ECLSTM operator is used in this stage, with multiple ECLSTMs stacked to form the architecture. The "Feature Extraction" section in Figure 5.4 outlines the search space for this stage, which includes parameters such as the number of ECLSTM layers, the depth of the embedded convolutional layers within each ECLSTM, the type of convolution, the number of filters, and the kernel width. These hyperparameters are optimized on a per-layer basis, allowing each ECLSTM to have a unique structure, thereby enhancing the diversity of the network.

Similarly, there are dependencies among these hyperparameters. First, the number of ECLSTM layers must be defined. To accommodate the need for optimized time budgets, we limit the maximum number of stacked ECLSTM layers to four. Once the number of layers is set, the depth of the convolution for each ECLSTM is determined, followed by the specification of the convolution type, the number of filters, and the kernel width for each convolutional layer.

5.2.1.4 Prediction Stage. The prediction stage consists of stacked fully connected layers, with the input being the features extracted from the previous

stage. The final layer produces the model’s output. The "Activity Prediction" section in Figure 5.4 outlines the search space for this stage, where each fully connected layer requires the definition of the number of nodes, the activation function, and the dropout rate.

5.2.1.5 Hyper-parameter Optimization. To optimize hyperparameters, random search is a commonly used method, but it is often inefficient in highly conditional configuration spaces. To address this, our proposed framework employs the BOHB optimizer [46], which combines Bayesian Optimization (BO) with Hyperband (HB). Hyperband efficiently identifies the best configurations through repeated calls to Successive Halving (SH), allocating more computational resources to promising configurations while terminating poorly performing ones early, thereby optimizing the time budget. Meanwhile, Bayesian Optimization uses a kernel density estimator to model the objective function, identifying high-performance regions within the configuration space.

5.2.2 Experiments and Discussions

5.2.2.1 Experiment Setup. In this section, we evaluate the proposed framework on three benchmark datasets: Opportunity (OPPO) [21], Daphnet [11], and PAMAP2 [127]. Considering that the optimization search process of the proposed framework is time-consuming, we used the train-test splits evaluation setup for our experiments, rather than the leave-one-subject-out (LOSO) approach. The specific train-test splits are as follows:

For the OPPO dataset, we used run 2 from subject 1 as the validation set and replicated the most popular recognition challenge by using runs 4 and 5 from subjects 2 and 3 as the test set. The remaining data was used for training. For frame-by-frame analysis, we created sliding windows with a duration of 1 second and 50% overlap. This resulted in approximately 650k training samples (43k frames).

For the PAMAP2 dataset, we used runs 1 and 2 from subject 5 as the validation set and runs 1 and 2 from subject 6 as the test set, with the remaining data used for training. In our analysis, we downsampled the accelerometer data to 33.3Hz to achieve a temporal resolution comparable to the Opportu-

nity dataset. For frame-by-frame analysis, we followed previous work by using non-overlapping sliding windows with a duration of 5.12 seconds and a stepping window of 1 second (78% overlap) [Reiss and Stricker, 2012]. This resulted in a training set of approximately 473k samples (14k frames).

For the Daphnet dataset, we used run 1 from subject 9 as the validation set, runs 1 and 2 from subject 2 as the test set, and the remaining data for training. The accelerometer data was downsampled to 32Hz, and we created sliding windows of 1 second duration with 50% overlap, resulting in approximately 470k training samples (30k frames).

In our framework, the objective function is defined by the model’s performance on the validation dataset, ensuring a more robust evaluation of each configuration by accounting for data variability and reducing the risk of overfitting. All experiments are conducted on a single GPU (RTX 2080 with 8GB RAM), and models are trained using the Adam optimizer [80].

Table 5.1: Comparison of classification performance based on averaged F1-scores on holdout test sequences between the ECLSTM model and other HAR models.

HAR model	OPPO	PAMAP2	Daphnet
Ensembles LSTM [55]	65.9	75.6	76.0
b-LSTM [182]	68.4	83.8	74.1
DeepConvLSTM [119]	67.2	74.8	77.8
DeepConvLSTM-Attn [113]	70.7	87.5	75.6
ECLSTM	72.3	90.9	79.2

5.2.2.2 Comparison to State-of-the-art. The experimental results are presented in Table 5.1. Our framework consistently demonstrates significant recognition improvements over other state-of-the-art (SOTA) models. Notably, we observe a performance gain of 3.4% on the Opportunity dataset compared to the second-best model, B-LSTM. This highlights the strong performance of the models discovered through the search process. Once the search space is defined, the search engine can automatically explore it, significantly reducing manual effort.

However, an uncontrolled increase in model parameters was observed, with our framework using more parameters than the DeepConvLSTM model across nearly all datasets. This issue occurred because the optimizer prioritized model performance during the search process, without factoring in model size or inference time. When lightweight constraints are not included in the search objective, even though the optimizer can discover better-performing models, they may not be suitable for resource-constrained applications.

Furthermore, when attempting to deploy the model on hardware devices, a critical issue emerged: the ECLSTM operator is not supported by standard edge devices. Most hardware platforms come with pre-built inference libraries, and deploying ECLSTM on such devices would require the development of a custom inference library, which significantly increases the development workload.

5.3 Summary

In this chapter, we explored the use of automated machine learning (AutoML) to search for model architectures, successfully optimizing models that outperformed other state-of-the-art HAR models. Notably, this was the first attempt to apply an AutoML framework to optimize HAR models at the time. This work provided several important insights, summarized as follows:

1. **Search efficiency must be improved:** The approach used in this chapter incurred significant overhead due to long optimization times.
2. **Search space design should consider hardware limitations:** In addition to accounting for task-specific characteristics, it is crucial to consider hardware constraints, such as supported operators.
3. **Without deployment constraints, the search process will prioritize performance at the expense of model size:** To optimize lightweight models, hardware limitations must be explicitly incorporated into the search objectives.

The insights gained from this chapter lay the foundation for the work presented in the following chapter.

6 Hardware-aware Model Architecture Optimization

Microcontroller Units (MCUs) are compact, low-power computing systems that are widely used in various devices, including medical equipment, consumer electronics, and wearables. The integration of microprocessors and embedded sensors, such as accelerometers and gyroscopes, enables on-device data analysis, allowing these devices to operate autonomously in privacy-sensitive, real-time environments [179; 22]. However, the limited hardware resources of typical MCUs (e.g., 64 kB SRAM, 64 MHz CPU clock) pose significant challenges for running SOTA HAR models like DeepConvLSTM [119], IF-Conv [178], and Global-Fusion [99] on these devices.

A common approach to overcoming the limited resources of MCUs is to transmit the raw data to a cloud server, where SOTA models can be executed, and then send the results back to the MCU. However, this method is often unsustainable for several reasons. For instance, network communication introduces unpredictable latencies, making it unsuitable for real-time applications or scenarios without reliable network access. Moreover, network communication is energy-intensive, which is particularly problematic for MCUs with limited power resources. Finally, processing data on external servers raises significant privacy concerns.

Another approach is to manually design efficient neural networks tailored for specific use cases. This process is typically carried out by domain experts with specialized knowledge in machine learning. However, it is often prone to errors and can be highly time-consuming [110]. In addition to optimizing model performance, other constraints such as task-specific requirements—like the need for fast inference—or hardware limitations, such as restricted peak memory, further complicate the design process. These multi-constraint design

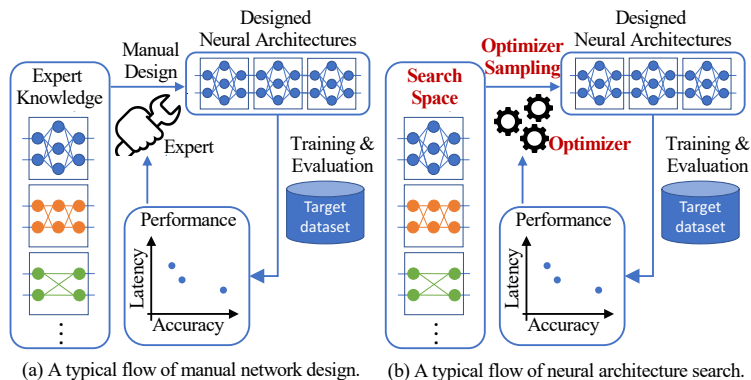


Figure 6.1: Illustration of manual network design and optimization based on neural architecture search.

challenges can be extremely difficult, even for experienced professionals. To address this issue and streamline the design process, neural architecture search (NAS) [189] offers a solution by automatically identifying optimal neural network architectures for specific use cases while simultaneously accounting for hardware constraints [18].

In the following sections, we first review the related work on NAS in Section 6.1. We then introduce our proposed MicroNAS approach in Section 6.2. Following that, we present the integration of NAS and pruning techniques within our SFTNAS framework in Section 6.3, which enables more effective model compression. Finally, we conclude this chapter with a summary and a discussion of potential future research directions in Section 6.4.

6.1 Related Works

6.1.1 Neural Architecture Search

As shown in Figure 6.1, the NAS process mimics the way human experts design models. Experts, based on their specialized knowledge, design a model from a search space of possible architectures. The designed model is then trained and tested, and the results are returned. The experts, using the test results and

model size as feedback, adjust the model architecture and repeat the process until a satisfactory result is achieved. The NAS method replaces the expert with an optimizer. This optimizer iteratively designs new models based on the feedback/reward from the previous designs.

Early NAS systems, as introduced by [189; 190], formulate the search process as a reinforcement learning (RL) problem [140]. While this approach can yield novel and effective architectures, it is highly time-consuming because each iteration of the REINFORCE algorithm [158] requires training a neural network to convergence, and the RL procedure typically necessitates many iterations [45]. To address these inefficiencies, subsequent research has optimized NAS in various ways, including improving training schema [95], adopting alternative optimization algorithms [126; 44; 76], and developing one-shot NAS approaches [162; 122].

One-shot NAS algorithms enhance efficiency by leveraging weight sharing among models within the search space [122], achieved by training a single super-net. This approach eliminates the need to train each model independently from scratch to convergence. Among these advancements, DARTS [97] introduced Differentiable Neural Architecture Search (DNAS), which employs a relaxation strategy to make architecture parameters continuous and differentiable, thereby further improving efficiency. During the architecture search, both the model weights and architecture parameters are jointly optimized using gradient descent. Due to its efficiency, our proposed frameworks, MicroNAS and SFTNAS, utilize DNAS as their foundational approach.

6.1.2 Hardware-aware Neural Architecture Search

Recently, NAS has evolved to become hardware-aware (HW-NAS) [18], where systems optimize not only traditional performance metrics such as accuracy and precision but also hardware-specific metrics like execution latency, peak memory usage, and energy consumption [17; 98]. Optimizing hardware utilization is especially critical when targeting MCUs, which are often constrained by limited resources.

During the optimization process of HW-NAS, it is crucial for the algorithm to accurately estimate hardware-relevant metrics, such as latency and peak

memory consumption, for candidate neural network models with varying architectures. For peak memory consumption, analytical estimation methods can be employed to achieve precise calculations [176]. However, estimating execution latency is more challenging, and various approaches have been proposed to address this issue [17].

One approach, employed during the architecture search process, utilizes a hardware-in-the-loop setup, where the real-time latency of sampled models is measured on the target hardware and returned to the optimizer. Although this method provides accurate hardware-relevant metrics, it significantly extends the search duration [17]. A more efficient alternative is to use the number of floating-point operations (FLOPs) as a proxy for execution latency [159; 152; 93; 176]. While MicroNets [176] and μ NAS [93] suggest that the number of operations in a model is a reliable proxy for execution latency on MCUs, other work [85] argues that this assumption is not always accurate. A middle-ground approach between precise but slow on-device measurements and the fast but less accurate latency estimations using FLOPs is the use of lookup tables [17]. With this method, each candidate operation in the search space is executed once on the target device, and its latency measurement is stored in a lookup table. During the search process, the runtime of any designed model is predicted based on this lookup table. This approach significantly improves the efficiency of latency measurement while providing accurate predictions.

In our proposed MicroNAS approach, we adopt this lookup strategy, whereas in the proposed SFTNAS approach, we use FLOPs as a proxy for latency estimation. The reason for this can be found in the experiments detailed in the MicroNAS experiment section 6.2.2.2.

6.1.3 Neural Architecture Search for HAR

Existing HW-NAS systems have predominantly been developed for image classification tasks [176; 93; 20] and have been relatively under-explored for HAR tasks. As discussed in Chapter 5, a critical element in executing NAS is defining the search space. However, the search spaces designed for image processing fail to account for the unique characteristics of HAR tasks. Only a few studies have investigated HW-NAS frameworks for HAR [82], but they not

only fall short in addressing the specific needs of HAR tasks, but also do not target resource-constrained devices like MCUs. This gap highlights the necessity for a dedicated HW-NAS framework tailored specifically for HAR on MCUs.

In summary, applying HW-NAS to multivariate time series classification, such as HAR tasks, presents two key challenges. The first is accurately estimating the latency and memory usage of models on target hardware. The second is designing an effective search space that not only meets hardware constraints but also accounts for the unique characteristics of HAR tasks, such as multi-modal interactions, the importance of local information, and the need to capture long-term dependencies, among others.

6.2 MicroNAS

We first designed a CNN-based HW-NAS framework specifically for HAR tasks, targeting MCU hardware. We have named this framework MicroNAS [79] to reflect its focus on microcontroller units.

6.2.1 Methodology

The system overview of MicroNAS is illustrated in Figure 6.2. Before designing the search space and training the model, it is essential to define the target hardware constraints, such as setting the peak memory consumption limit (Mem_t) and establishing the maximum allowable execution latency (Lat_t) based on the application requirements.

Once these constraints are defined, the search space can be designed, which will be discussed in Section 6.2.1.1. The second step involves hardware characterization, which refers to predicting hardware-related metrics for each operator within the defined search space on the target hardware, such as latency and peak memory usage. This process will be detailed in Section 6.2.1.3. After characterization, the HW-NAS process can begin, optimizing an optimal architecture from the defined search space that is tailored to the target hardware, as explained in Section 6.2.1.4. Finally, in Section 6.2.3, we will discuss how to deploy the optimized model onto the device.

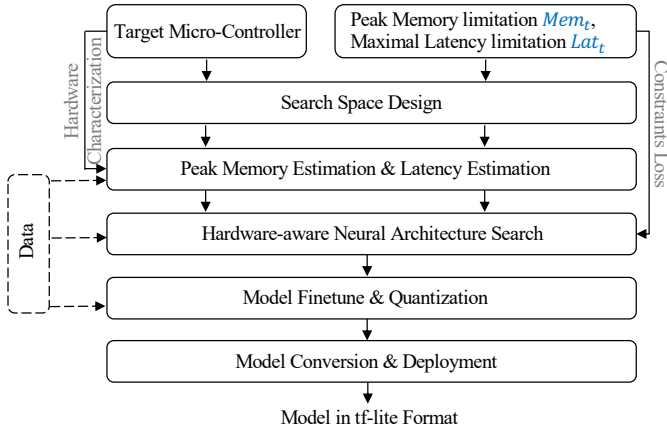


Figure 6.2: Before using the MicroNAS framework, the user needs to specify the dataset to be used, the target micro-controller (MCU_t), and the maximum allowable hardware utilization in terms of execution latency (Lat_t) and peak memory consumption (Mem_t). The output of the system is a corresponding neural network in the TF-Lite format.

6.2.1.1 Search Space. An overview of the search space is depicted in Figure 6.3, which consists of three stages built from a linear stack of architecture-searchable cells, each stage represented by different colors in the figure. The division into three stages is based on guidelines summarized from our previous work, TinyHAR, where each stage extracts different information from the HAR data.

Time-Reduce cells: These cells form the first stage of the architecture. They are designed to extract local temporal context from the incoming time series while simultaneously reducing the temporal dimension. By stacking multiple layers of these cells, the first stage achieves a large temporal receptive field, enabling it to capture global temporal dependencies as well.

Sensor-Fusion cells: These cells form the second stage of the architecture. They facilitate cross-channel interactions and effectively fuse information from multiple sensors.

Prediction cells: These cells form the third stage of the architecture. They

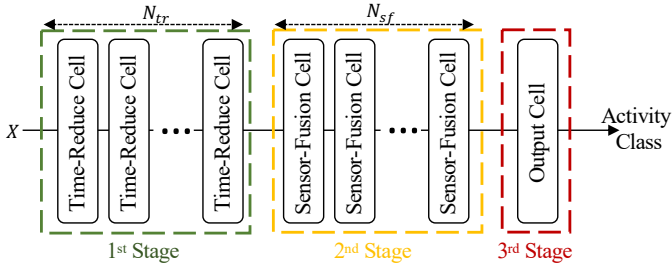


Figure 6.3: High-level overview of the search space for MicroNAS framework.

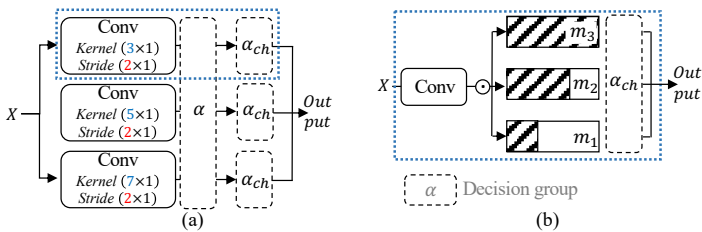


Figure 6.4: (a) Search space of Time-Reduce cell. It contains two decision groups. α to choose a convolution operator and α_{ch} to search for the number of filters for each operator. (b) Dynamic convolution with searchable filter masks.

are responsible for the final prediction, integrating all the features extracted in the previous stages.

In addition to accounting for the specific characteristics of HAR tasks, the cell design also considers hardware constraints. The entire search space is designed to be purely CNN-based, as CNNs are preferred for their high parallelism and fast computational speed on hardware. After each layer in the architecture, batch normalization [67] and ReLU activation [2] are applied (though not shown in Figure 6.3).

The following is a detailed introduction to the structure of the three cell types.

Time-Reduce Cell & Stage. Placing the Time-Reduce cell in the first stage serves two key purposes: extracting local temporal context within each sensor channel and significantly reducing the temporal dimension. The Time-Reduce cell consists of a series of parallel convolutional operators, each using a stride of 2, as shown in Figure 6.4. These convolutional operators vary in their kernel sizes, which are set to 3×1 , 5×1 , and 7×1 . Larger kernels can capture a broader local context, but they also come with increased computational complexity. In this framework, the stride for each convolution is set to 2. This is because when the stride is 2, the temporal dimension of the output is approximately halved (due to the absence of padding) compared to the input.

To effectively capture global temporal information through convolution, multiple layers of Time-Reduce cells are stacked. The number of Time-Reduce cells, denoted as N_{tr} , is calculated as follows:

$$N_{tr} = \left\lceil \log_2 \left(\frac{L}{L_{ml}} \right) \right\rceil$$

where L is the input window size, and L_{ml} is the minimum temporal length of the output from the first stage. In designing the Time-Reduce cells, we deliberately exclude kernel sizes of 1 or identity operators, as they may result in information loss when the stride is set to 2.

For selecting the convolutional operators for each cell (layer), we adopt the DARTS methodology [97]. Each candidate convolutional operator is associated with an architecture parameter α_i^j , $j = 1, 2, 3$, where the vector α_i denotes a decision group for the i -th layer. Each element α_i^j indicates the importance of the corresponding j -th convolutional operator, which is reflected in the forward process by the sampled probability. During forward inference, a convolutional operator is sampled according to α using the Gumbel-Max trick approach [56] as follows:

$$c_i = \text{Onehot_Encoding} \left(\underset{j}{\operatorname{argmax}} \left(g_i^j + \log \left(\alpha_i^j \right) \right) \right) \quad (6.1)$$

where g^i are independent samples drawn from a standard Gumbel distribution ($g^i \sim \text{Gumbel}(0, 1)$). c_i is a one-hot encoded vector representation, with a probability proportional to the value of α_i . The forward process is as follows:

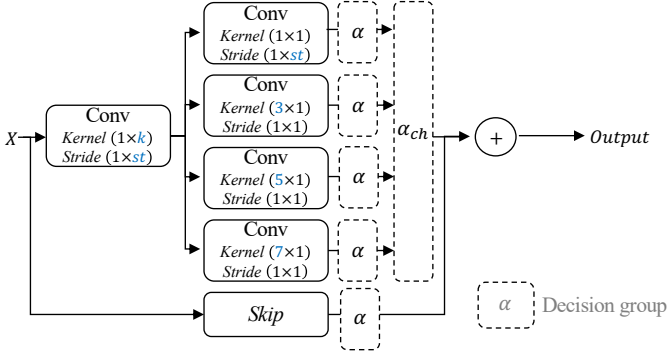


Figure 6.5: Search space of Sensor-Fusion cell. It contains six decision groups. Each decision group is denoted as α in dashed box. Five of them are for the five pathways and one for the filter masks.

$$X_{i+1} = \sum_{j=1}^3 c_i^j op_i^j (X_i) \quad (6.2)$$

At any given instance, only one operator is applied. During gradient back-propagation, we compute the gradient of the *argmax* in equation 6.1 through a differentiable approximation as follows:

$$G_Softmax(\alpha_i^j; \alpha) = \frac{\exp\left(\left(\log(\alpha_i^j) + g_i^j\right) / \theta\right)}{\sum_{k=1}^3 \exp\left(\left(\log(\alpha_i^k) + g_i^k\right) / \theta\right)} \quad (6.3)$$

Here, θ is the temperature parameter controlling the approximation's fidelity to discrete one-hot vectors. Consequently, this allows the model to be trained with discrete operations, using equations 6.1 and 6.2 for the forward pass and the differentiable equation 6.3 for gradient back-propagation.

Sensor-Fusion Cell & Stage. A common challenge in time series data processing is managing the interactions between different sensor modalities. To address this, we designed the Sensor-Fusion cell for the second stage, as illustrated in Figure 6.5. Building upon the Time-Reduce Cell structure, we introduce a fixed convolution operator before the three candidate operators. Unlike

individual convolutions, this pre-convolution operator functions along the sensor channel dimension with a kernel size of $(1, k)$, facilitating cross-channel interaction. This design is inspired by the previously proposed MLP-HAR approach, where cross-sensor information exchange occurs first, followed by intra-sensor information processing. By stacking multiple Sensor-Fusion cells, information can efficiently propagate across both dimensions.

As shown in the Figure 6.5, beyond the three convolutional pathways used in the Time-Reduce cell, the Sensor-Fusion cell incorporates two additional paths: the topmost convolution operator with a kernel size of 1×1 , which provides the option to bypass cross-sensor information extraction, and the bottom identity connection, which allows the entire layer to be skipped.

Similar to the Time-Reduce cell, each pathway in the Sensor-Fusion cell is associated with an α parameter indicating its importance. However, unlike in the Time-Reduce cell, these pathways do not compete with one another. For example, in the Time-Reduce cell, only one convolutional operator can be selected, but in the Sensor-Fusion cell, this limitation does not apply. In other words, each α represents a distinct decision group that determines whether its corresponding pathway is activated. Multiple pathways can be selected simultaneously, allowing for greater diversity within the cell, as convolutions with different kernel sizes can be employed concurrently to capture information across various temporal scales. While activating more pathways increases diversity, it also adds computational complexity. The optimizer is responsible for balancing model complexity and performance, making the trade-off between the two.

Output cell & Stage. The output cell has a fixed architecture with no learnable parameters. It consists of a convolutional layer where the number of filters is set to match the number of target classes. Class probabilities (y_{cls}) are then obtained through a Global Average Pooling (GAP) layer, followed by a Softmax activation function.

6.2.1.2 Dynamic Convolutions. One challenge that remains is determining the appropriate number of filters d for each layer. In previous works [1; 119; 99; 167], this value is typically predefined, with most studies setting $d = 32$ or

$d = 64$. However, the question arises: does every layer require the same number of filters, and do we need as many filters as predefined? To address this, we incorporate dynamic convolution into our design [152]. A dynamic convolution allows the number of filters to be adjusted through a search process.

In a dynamic convolution, the input is first processed through a convolutional layer with the maximum number of allowed filters, denoted as d_{max} . The output is then multiplied by a binary mask along the filter dimension. An example is illustrated in Figure 6.4 (b). Different masks represent different configurations of filter usage, allowing for the use of all filters (m3), two-thirds of the filters (m2), or one-third of the filters (m1). Similar to operator selection, each mask is assigned an architecture decision group weight, $\alpha_{ch} \in \mathcal{R}^3$, which determines the likelihood of each mask being selected. In practice, the mask granularity is finer than just three options in the example. We define a group size g_r , which is a divisor of d_{max} , and the number of mask options is equal to d divided by g_r .

In the Time-Reduce cell, each dynamic convolution has its own set of architecture weights for mask selection. However, in the Sensor-Fusion cell, since all pathways must output tensors of the same shape, the decision group α_{ch} for selecting the number of filters is shared across all pathways within the Sensor-Fusion cell. In the Figure 6.4 and Figure 6.5, dashed boxes represent each decision group. If multiple dashed boxes are shown, they are independent decision groups. If the dashed box spans multiple branches, it indicates that the branches still belong to the same decision group.

6.2.1.3 Latency & Peak Memory Estimation. For MicroNAS to identify architectures that comply with user-defined constraints on execution latency and peak memory consumption, it is crucial to accurately estimate the execution latency and peak memory consumption of designed neural networks within the search space. Below, we describe the estimation algorithms used for these two hardware-related metrics.

Execution Latency. To improve upon FLOPs-based proxy metrics for execution latency estimation, we employed a lookup-table-based approach. Specifically, we profiled the latency of each operator by executing it on the target

MCU device. The lookup table was constructed by varying parameters such as kernel size, stride, number of filters, input shape, and output shape for each candidate operator. In our profiling system, each operator is considered alongside its associated activation function, as the TensorFlow Lite Micro framework [36] fuses operations with their subsequent activation functions. Execution latency is measured using the internal CPU cycle counter of ARM Cortex processors, which can be easily converted to milliseconds. Using this lookup table, we compute the execution latency for any designed models by summing the latency of all operators in the model.

Peak Memory Consumption. When executing a neural network model in the TFLM framework [37], memory is allocated from a user-provided byte array. In TFLM, the byte array is divided into two sections. The first section, referred to as the *Head*, contains non-persistent tensor buffers, and its minimum size corresponds to the peak memory usage, which is critical in the search process. The second section, called the *Tail*, stores persistent allocations required by TFLM. According to the documentation [145], the Tail section involves numerous dynamic allocations, making it difficult to predict the total memory required for execution. As a result, our system focuses solely on optimizing peak memory usage by considering the intermediate tensors during execution, excluding the Tail section from this optimization. To estimate peak memory consumption, we follow the analytical methods proposed in the literature [17; 176]. During neural network execution, both input and output tensors must be stored in memory, and certain operations may require additional memory for computations. For example, the CMSIS-NN kernel library requires extra memory for certain operations such as convolutions [84]. The memory requirement for a given operation is calculated as:

$$mem(op) = mem(input) + mem(output) + extra_mem(op). \quad (6.4)$$

where $mem(x)$ calculates the memory required to store a tensor, taking into account its data format. For instance, *int_8* tensors require only a quarter of the memory compared to *float_32* tensors. The total memory consumption of an

operation is the sum of the memory required for its input and output tensors, along with any additional memory required for computation [84]. In line with the analytical methods discussed in previous studies [17; 176], peak memory consumption is defined as the maximum memory utilized by any operation within the model. This approach allows us to estimate the peak memory usage of a neural network architecture with high precision.

6.2.1.4 Optimization. The goal of the search is to identify an optimal architecture, denoted as architecture parameter α^* , from the search space that maximizes classification performance while adhering to user-defined hardware constraints on execution latency Lat_t and peak memory consumption Mem_t . The entire optimization process can be mathematically formulated as follows:

The objective of the search is to identify the optimal architecture, denoted as the architecture parameter α^* , from the search space. This architecture is expected to maximize classification performance while satisfying user-defined hardware constraints on execution latency Lat_t and peak memory consumption Mem_t . The entire optimization process can be mathematically formulated as follows:

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) \quad (6.5)$$

$$s.t. w^*(\alpha) = \underset{w}{\operatorname{argmin}} \mathcal{L}_{train}(w, \alpha) \quad (6.6)$$

Where \mathcal{L}_{train} and \mathcal{L}_{val} represent the training and validation losses, respectively. This forms a typical bi-level optimization problem [97], where the architecture parameters α and α_{ch} are treated as the upper-level variables and the model weights w as the lower-level variables. The optimization of these two parameters follows the iterative optimization procedure used in DARTS, as outlined in Algorithm 2.

In equation 6.5, the loss function consists of the cross-entropy loss $\mathcal{L}_{crossentropy}$ and hardware-aware constraints, formulated as follows:

$$\mathcal{L} = \mathcal{L}_{crossentropy} + \lambda_{lat} \mathcal{L}_{latency} + \lambda_{mem} \mathcal{L}_{memory} \quad (6.7)$$

Algorithm 2 Optimization Algorithm

Variables: α - Architecture parameter (including α_{ch} , the dynamic filter parameter) w - Model weights ξ_w - Learning rate for updating w ξ_α - Learning rate for updating α and α_{ch} $epoch_{search}$ - Number of epochs**for** $i = 1$ to $epoch_{search}$ **do**

Operator sampling and filter mask sampling

for each mini-batch of data **do**Update distribution α : $\alpha = \alpha - \xi_\alpha \nabla_\alpha \mathcal{L}_{val}(w^*, \alpha)$ Update model weights w : $w = w - \xi_w \nabla_w \mathcal{L}_{train}(w, \alpha)$

Operator sampling and filter mask sampling

end for**end for**

The hardware-aware constraints include the model latency constraint loss $\mathcal{L}_{latency}$ and the peak memory usage constraint loss \mathcal{L}_{memory} . The parameters λ_{lat} and λ_{mem} are trade-off factors that balance the impact of execution latency and memory usage on the overall loss function. These factors enable the optimization process to adjust the focus between performance and hardware efficiency, based on the specific requirements of the application.

The latency loss $\mathcal{L}_{latency}$ is formulated as:

$$\mathcal{L}_{latency} = \eta (Lat(\alpha, MCU_t)) \cdot \log \left(\frac{Lat(\alpha, MCU_t)}{Lat_t} \right) \quad (6.8)$$

where $Lat(\alpha, MCU_t)$ denotes the predicted latency using the lookup table approach. Similarly, the peak memory loss \mathcal{L}_{memory} is formulated as:

$$\mathcal{L}_{memory} = \eta (Mem(\alpha, MCU_t)) \cdot \log \left(\frac{Mem(\alpha, MCU_t)}{Mem_t} \right) \quad (6.9)$$

where $Mem(\alpha, MCU_t)$ denotes the estimated peak memory usage. Both formulations ensure that the loss function increases when the current search space configuration α leads to higher execution latency or peak memory consumption than the allowed limits. Importantly, the function η ensures that the hardware-

related loss terms are set to zero when the current configuration of the search space remains within the predefined targets. The definition of the η function is as follows:

$$\eta(Lat(\alpha, MCU_t)) = \begin{cases} \gamma_{lat}, & \text{if } Lat(\alpha, MCU_t) \geq Lat_t \\ 0, & \text{otherwise} \end{cases} \quad (6.10)$$

$$\eta(Mem(\alpha, MCU_t)) = \begin{cases} \gamma_{mem}, & \text{if } Mem(\alpha, MCU_t) \geq Mem_t \\ 0, & \text{otherwise} \end{cases}$$

where γ_{lat} and γ_{mem} are penalty parameters. For example, if the predicted latency $Lat(\alpha, MCU_t)$ exceeds the limit Lat_t , the latency loss will no longer be zero. The same applies to the memory loss. This ensures that the optimization process focuses on configurations that meet the hardware constraints while still aiming to maximize the model’s performance. To ensure that the user-defined limits on execution latency and peak memory consumption are strictly enforced, the penalty parameters γ_{lat} and γ_{mem} must be set sufficiently high. These parameters strongly discourage configurations that exceed the specified hardware constraints, effectively guiding the search process toward architectures that meet both performance and hardware efficiency requirements.

6.2.2 Experiments and Discussions

6.2.2.1 Experiment Setup. Before presenting the results, this section outlines the training setup of the proposed MicroNAS framework. For the convolutions in the Time-Reduce cells, the maximum number of filters d_{max} was set to 16, with a group size g_r of 4. In the Sensor-Fusion cells, d_{max} was set to 64, and g_r was set to 8. For the search algorithm, the exploration rate ϵ was set to 0.995, the latency penalty coefficient γ_{lat} to 2, and the memory penalty coefficient γ_{mem} to 4. The super-network was trained for 60 epochs with a batch size of 32. To optimize the weights w , we used the Adam optimizer, while the architectural weights α were optimized using stochastic gradient descent, with the learning rate for the architecture parameters set to 0.36. The training was

conducted on an Intel Core i7-12700K processor.

After the search process of MicroNAS, the optimized model is extracted from the search space according to the architecture parameter α and retrained from scratch using quantization-aware training to maximize classification performance. Finally, the trained, *int_8* quantized neural network is converted to the TensorFlow Lite format, enabling deployment on the target micro-controller, *MCU_t*.

6.2.2.2 Latency Prediction Experiment. Accurately predicting the latency of operators and networks is crucial for performing HW-NAS. Therefore, we first evaluate the accuracy of latency estimation for different neural network architectures by assessing the correlation between estimated latencies and actual execution latencies on hardware. In this experiment, two latency prediction approaches were considered: one using the proposed lookup-table approach and the other employing the FLOPs-based proxy metric. The FLOPs-based proxy metric was implemented by training a linear regression model to estimate latency based on FLOPs.

This experiment was conducted on the MCU NUCLEO-L552ZE-Q [137], equipped with an STM32L552 (single core, ARM Cortex-M33, 80 MHz CPU clock, 512 kB flash, 256 kB SRAM). For this evaluation, 200 random architecture samples were generated from the defined search space. To measure the ground truth inference time, the sampled networks were first quantized to *int_8* and then deployed on the target hardware. The actual execution latency was measured using the internal CPU-cycle counter on the ARM Cortex processors.

The experimental results are shown in Figure 6.6. As can be clearly seen from the figure, the proposed lookup table approach achieves an R^2 -score of 99.97 with a mean absolute error (MAE) of 1.59 ms. In comparison, the FLOPs-based latency estimation achieves an R^2 -score of 96.78 with an MAE of 15.57 ms. These results indicate that while the FLOPs-based estimation method serves as a reasonable proxy with high correlation, the lookup table approach demonstrates a significantly stronger correlation with actual execution latency, providing more accurate predictions.

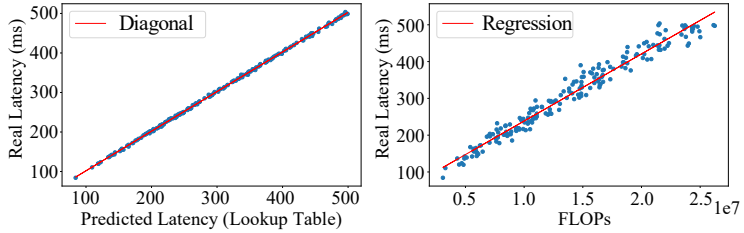


Figure 6.6: Execution latency of whole architectures from our search space. Left: Our lookup-table latency approach. MAE: 1.59 ms, R^2 : 99.97%. Right: Flops based estimate: MAE: 15.57 ms, R^2 : 96.78%.

Additionally, we validated the performance of the lookup table-based prediction method on another device, the Arduino Nicla Sense ME [8], which is equipped with an NRF52832 (single core, ARM Cortex-M4, 64 MHz CPU clock, 512 kB flash, 64 kB SRAM). To clearly illustrate the difference between the predicted and actual latency on the hardware, we sampled eight models of varying sizes from the search space. The results of this experiment are shown in Figure 6.7. For the tested architectures, the latencies predicted using the lookup table closely matched the actual execution latency on the target microcontroller. For the eight models under investigation, the mean difference in execution latency was 1.02 ms, with a standard deviation of 0.63 ms.

In summary, the lookup table-based latency prediction method demonstrated excellent accuracy compared to the FLOPs-based method across two different devices. Our experiment also validated that, when using traditional convolutions, FLOPs can serve as a reasonable proxy for latency estimation.

6.2.2.3 Latency vs. Performance. The primary advantage of the proposed MicroNAS framework lies in its ability to autonomously identify architectures that adhere to the latency constraints of a given task. In this study, we explore the capability of MicroNAS to discover suitable models under varying latency constraints. To isolate the effect of latency, we specifically disabled the loss function related to peak memory consumption. For the UCI-HAR dataset, we set the latency range to 0–250 ms, while for the SkodaR dataset, the latency range was set to 0–500 ms. Under each latency constraint, experiments were

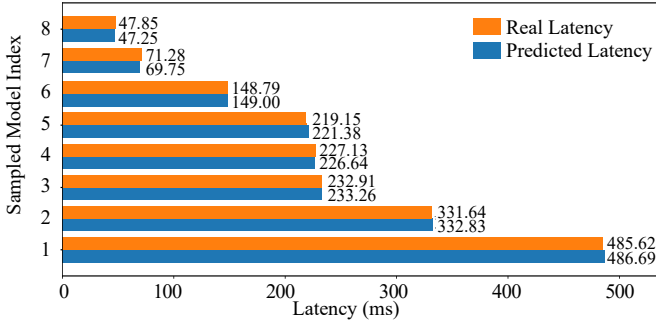


Figure 6.7: Correlation between predicted execution latency and actual execution latency for several architectures from the search space.

conducted five times, and the mean performance metrics along with their variances are illustrated in Figure Figure 6.8 and Figure Figure 6.9.

As anticipated, the classification performance improves as the latency constraints are relaxed, as shown in Figure Figure 6.8 for the UCI-HAR dataset [128] and Figure Figure 6.9 for the SkodaR dataset [170]. With increasing latency allowances, the performance of the discovered models demonstrates an upward trend. Stricter latency constraints significantly degrade performance. This degradation is particularly pronounced in regions with very tight latency constraints. Nonetheless, MicroNAS efficiently optimizes models to meet a wide range of latency constraints. Remarkably, even with an extremely tight constraint of 20 ms, the models identified by MicroNAS still perform well. This is noteworthy as it is typically very challenging for domain experts to optimize models under such extreme latency conditions.

6.2.2.4 Peak Memory vs. Performance. Another advantage of the proposed MicroNAS is its ability to automatically search for the optimal model based on hardware memory constraints. In this experiment, we demonstrate MicroNAS’s capability to discover architectures under different peak memory limitations. To isolate the effect of memory, we disabled the loss function associated with execution latency. As in the previous experiments, we set the range of peak memory consumption between 6 KB and 30 KB. Under each

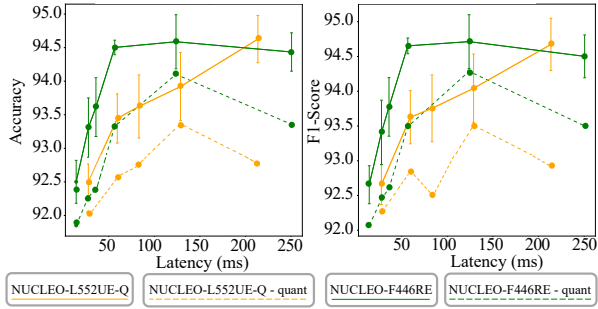


Figure 6.8: Classification performance and latency trade-offs on the UCI-HAR dataset. Left: Accuracy, Right: F1-Score (Macro).

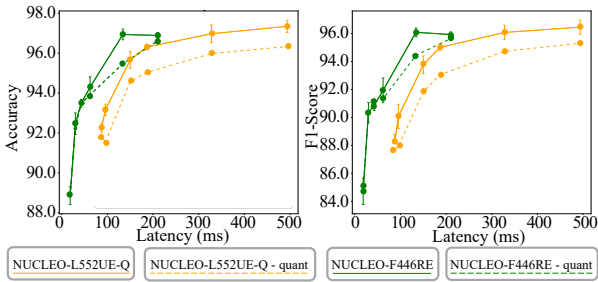


Figure 6.9: Classification performance and latency trade-offs on the SkodaR dataset. Left: Accuracy, Right: F1-Score (Macro).

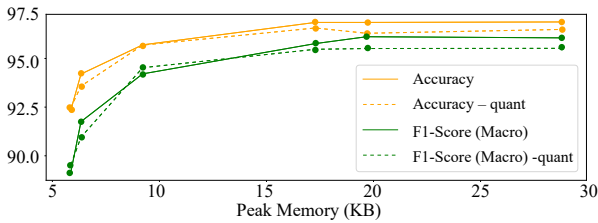


Figure 6.10: Trade-off between peak memory consumption and Accuracy / F1-Score (Macro). Comparison on the SkodaR dataset.

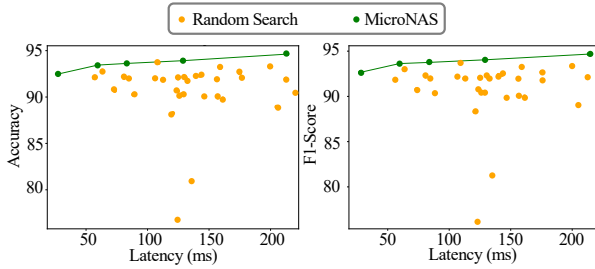


Figure 6.11: MicroNAS compared against random search on the UCI-HAR dataset with the Nucleo-L552ZE-Q.

memory constraint, we conducted the experiments five times and reported the mean performance metrics along with their variances in Figure 6.10. Since the TensorFlow Lite for Microcontrollers (TFLM) framework [37] utilizes the same amount of memory across different microcontrollers, this experiment is independent of specific hardware platforms. Therefore, we conducted this experiment solely on the Nucleo-L552ZE-Q [138] using the SkodaR dataset.

As illustrated in Figure 6.10, and in line with our expectations, performance improves as more memory becomes available. A closer examination of the left portion of the figure, where the peak memory constraints are most stringent, reveals a decline in model performance. However, even with a memory limit as low as 6 KB, MicroNAS remains capable of efficiently and automatically optimizing models that exhibit strong performance.

6.2.2.5 Comparison to Random Search. In this section, we compare MicroNAS against a random search baseline. Random search simulates the behavior of an inexperienced model designer who randomly attempts different architectures. For this comparison, 30 architectures were randomly sampled from the search space and trained for 64 epochs using the same hyper-parameters as in the other experiments. The total training time for these 30 architectures is approximately equivalent to the time required for a full architecture search by MicroNAS.

The results are presented in Figure 6.12 for the SkodaR dataset and Fig-

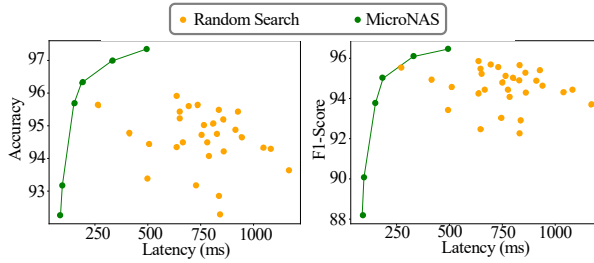


Figure 6.12: MicroNAS compared against random search on the SkodaR dataset with the Nucleo-L552ZE-Q.

ure 6.11 for the UCI-HAR dataset. In both cases, the proposed MicroNAS consistently outperforms the random search baseline across all configurations. Specifically, while the random search generates models of varying sizes, leading to a wide range of latencies, the MicroNAS framework consistently achieves superior performance. This results in a Pareto front that lies consistently above the outcomes of the random search. These findings clearly demonstrate the superior trade-offs between latency constraints and classification performance achieved by the proposed MicroNAS across various latency limitations.

6.2.3 Discussion

This work demonstrates the feasibility of using NAS for designing and deploying neural network models on MCUs. By considering user-defined constraints on execution latency and peak memory consumption for the target MCU, our approach enables the rapid identification of architectures suitable for real-time systems. This significantly reduces the complexity faced by model designers when accounting for multiple hardware limitations. Through extensive experiments, we validate the effectiveness of our HW-NAS method. However, compared to SOTA HAR models, the architectures discovered by our approach, while meeting task, hardware, and user requirements, exhibit a slight decrease in performance. This highlights the untapped potential of HW-NAS for HAR tasks, warranting further research.

6.3 NAS Meets Pruning

To further explore the potential of HW-NAS for model reduction, we conducted an in-depth analysis of the complexity and size of current SOTA HAR models. Our analysis revealed that their large size and computational demands are primarily driven by the extensive number of sensor channels and the use of individual convolution layers, as discussed in Section 6.3.1. Based on these insights, we developed a novel NAS-plus-pruning framework [186], which is detailed in Section 6.3.2. This framework systematically filters sensor channels, prunes filters at each layer, reduces the temporal dimension, and optimizes the model architecture in an end-to-end process. In Section 6.3.3, we will demonstrate its performance on the NUCLEO-L552ZE-Q, equipped with a STM32L552 (Single core, ARM Cortex-M33, 80 MHz CPU clock, 512 kB flash, 256 kB SRAM). Finally, in Section 6.3.4, we provide a summary of this work.

6.3.1 HAR Model Complexity and Size

The challenge in predicting activities from multidimensional sensor data lies in accurately capturing local context, multi-modal interactions, and global temporal information [102; 185]. To capture these different dimensions of information, the feature extraction process in most SOTA HAR models can be divided into four stages: local context extraction, sensor information interaction, sensor information fusion, and temporal information extraction [185], as illustrated in Figure 6.13.

Following these design principles, as shown in Figure 6.13, the architecture of most SOTA HAR models—such as Attend-and-Discriminate [1], If-ConvTransformer [178], DeepSense [167], AttenSense [102], TinyHar [185], GlobalFusion [99], and ALAE-TAE [3]—can be categorized into these stages. The primary differences among these models lie in their approaches to sensor fusion and temporal information extraction. However, nearly all these networks begin with individual convolution blocks in the first stage for local context extraction, which are then passed on to downstream modules.

We selected four SOTA HAR models and evaluated their model complexity

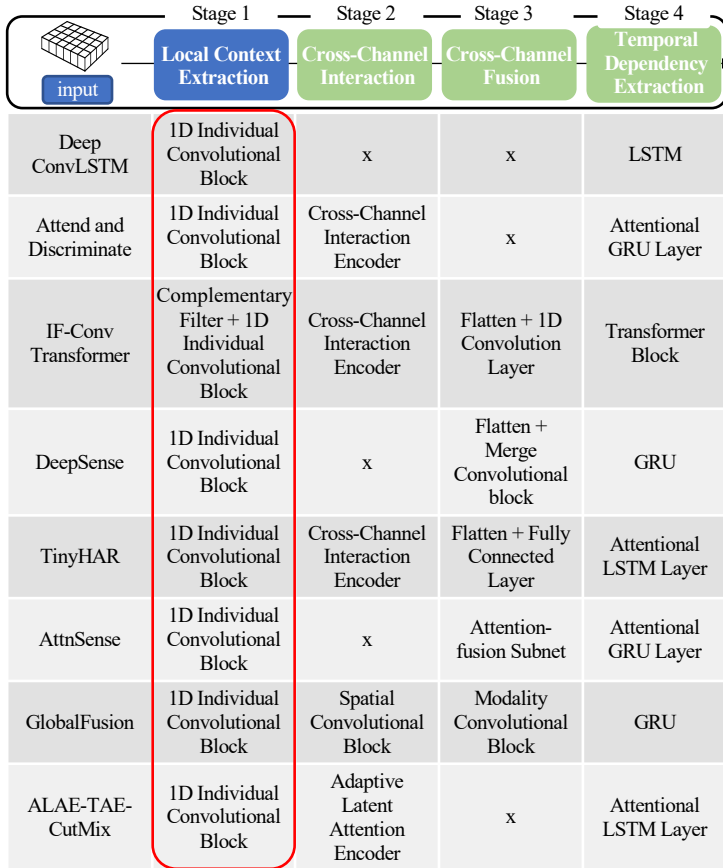


Figure 6.13: Overview of the architecture design of HAR models.

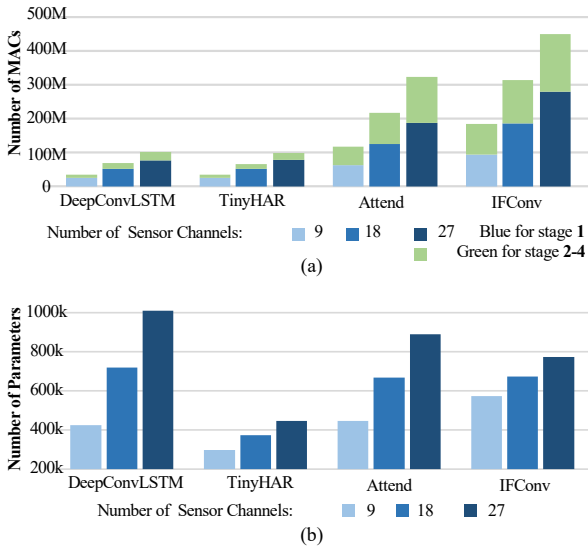


Figure 6.14: Impact of the number of sensor channels on model complexity (a) and on number of trainable parameters (b).

at each stage. To quantify model complexity, we used Multiply-Accumulate Operations (MACs) as the evaluation metric. To highlight the complexity of the initial stage, we divided the four stages into two parts: the first part includes only the first stage, while the second part encompasses the subsequent stages. This evaluation was carried out using the UCI-HAR dataset, which utilizes a single Inertial Measurement Unit (IMU) corresponding to 9 sensor channels for data collection. Furthermore, we increased the number of sensor channels from 9 to 18 and 27 to simulate scenarios involving the use of additional IMUs. This modification enabled us to investigate how the number of sensor channels impacts both model size and complexity.

As shown in Figure 6.14 (a), the majority of the MACs across all analyzed models are concentrated in the first stage. In the figure, the computational complexity of the first stage is represented by blue bars, while that of the subsequent stages is shown in green. It is evident that the blue bars dominate. This is primarily because the temporal dimension of the HAR data is signifi-

cantly larger than the other dimensions, and the first stage operates directly on these original dimensions, leading to high computational intensity. This finding further supports our previous work, which demonstrated that an effective strategy to reduce model complexity is to shorten the temporal dimension in this convolutional stage.

Interestingly, we observed that increasing the number of IMUs results in a substantial rise in both the model's parameter count and overall complexity. As shown in Figure 6.14 (a) and (b), when the number of sensor channels increases from 9 to 18, the model's size and computational cost nearly double. This increase can be attributed to two key factors. First, the individual convolutions applied to the expanded sensor channels significantly contribute to the additional computational load. Second, models such as TinyHAR and If-Conv employ a flatten operation followed by a fully connected layer for feature fusion, while other models like DeepConvLSTM and Attend directly pass vectorized features to subsequent temporal extraction stages, which also include fully connected layers. It is well-established that the number of parameters and the complexity of a fully connected layer are proportional to its input and output dimensions. Thus, an increase in sensor channels drastically enlarges the input dimension, leading to a significant rise in computational demands for these layers. Given the rapid advancement of sensor technologies and the increasing ease of deploying wearable devices, the number of sensor channels in future applications is expected to grow. This trend poses a major challenge for processing data from multiple IMU sources, as it results in oversized models with heightened computational requirements.

In conclusion, these investigations have uncovered two critical findings: first, the initial stage of the model accounts for a substantial portion of the computational resources; second, the model's size and computational complexity increase significantly with the number of sensor channels. To develop more lightweight models, the most effective strategies involve optimizing the first stage by minimizing the temporal dimension, carefully selecting efficient sensor channels when their number is large, and reducing the number of filters in each convolutional layer.

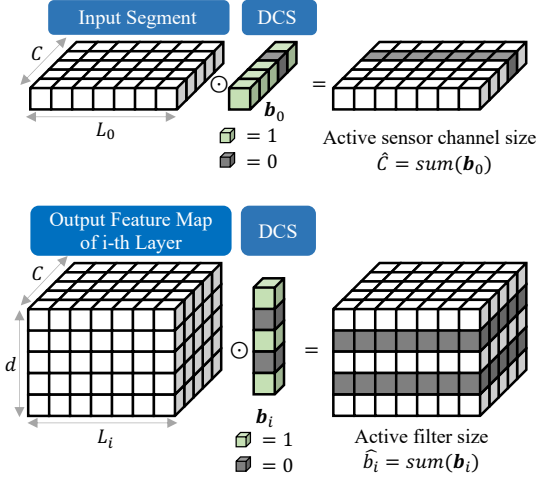


Figure 6.15: DCS layer for the input layer and convolutional layer.

6.3.2 Methodology

Based on the previous findings, we have designed an NAS-plus-pruning framework that can be applied to any HAR model featuring an individual convolution as the first stage.

6.3.2.1 Sensor and Convolutional Channel Pruning. To effectively reduce both the number of sensor channels and convolutional filters, a pruning strategy is incorporated into the framework. Specifically, a Differentiable Channel-wise Scaling (DCS) layer [92] is introduced at the beginning of the model and at the end of each convolutional layer, as illustrated in Figure 6.15. The DCS layer is parameterized by a binarized vector \mathbf{b}^* . The mathematical formulation of this process is given by:

$$\mathbf{X}_i = \mathbf{b}_i^* \odot \mathbf{X}_i \quad i = 0 \quad (6.11)$$

$$\mathbf{X}_{i+1} = \mathbf{b}_i^* \odot (\mathbf{X}_{i-1} \otimes \mathbf{w}_i) \quad i = 1, 2, \dots, L \quad (6.12)$$

In this equation, the symbol \otimes denotes the convolution operation, while \odot represents element-wise multiplication. Each value in the binarized parameter vector \mathbf{b}^* serves as an indicator of the importance of the corresponding sensor or filter channel. When $i = 0$, referring to the input layer, no convolutional operation occurs. The initial DCS layer, parameterized by $\mathbf{b}_0^* \in \mathbb{R}^{1 \times C_0 \times 1}$, is responsible for selecting important sensor channels and pruning non-informative sensor channels from the original input data. In contrast, the subsequent DCS layers focus on selecting important filters and pruning redundant filters in the preceding convolutional layers, where $\mathbf{b}_i^* \in \mathbb{R}^{1 \times 1 \times d}$ for $i = 1, 2, \dots, N_{conv}$. A value of 1 in \mathbf{b}^* signifies importance, ensuring the retention of the corresponding feature map region for subsequent layers. Conversely, a value of 0 in \mathbf{b}^* indicates irrelevance, leading to the nullification of the associated feature map section. Hence, the tasks of sensor channel (filter) selection and pruning are framed as a binary optimization problem, governed by the parameter \mathbf{b}_i^* .

To address the issue of gradient back-propagation associated with this binarized parameter, the Straight-Through Estimator (STE) technique [16] was employed. The forward and backward propagation processes are defined as follows:

$$Forward : \mathbf{b}_i^* = \begin{cases} 0 & \mathbf{v}_i \leq thres \\ 1 & \mathbf{v}_i > thres \end{cases} \quad (6.13)$$

$$Backward : \frac{\partial \mathcal{L}}{\partial \mathbf{v}_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{b}_i^*} \quad (6.14)$$

In this context, the binarized parameter \mathbf{b}_i^* is derived from a trainable continuous parameter \mathbf{v}_i . The threshold value *thres* is a hyper-parameter, set to 0.5 in this work. Through the integration of DCS layers and the STE method, both the model parameters \mathbf{w}_i and the binarized informative parameters \mathbf{b}_i^* , parameterized by \mathbf{v}_i , can be trained simultaneously.

6.3.2.2 Search Space. As mentioned in the previous section, in addition to pruning sensor and filter channels, it is crucial to effectively condense the temporal dimension. To achieve this, we have devised a search space, as illustrated

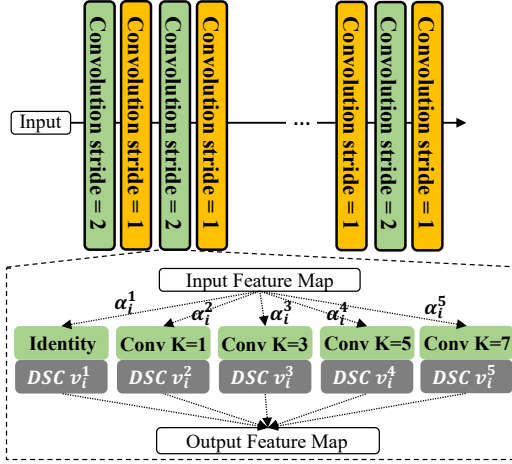


Figure 6.16: Search space of each layer for SFTNAS framework.

in Figure 6.16. This space is structured by alternating convolutional layers with strides of 2 and strides of 1. For each searchable layer, the available operations include five candidate operators: convolutions with kernel sizes of 1, 3, 5, and 7, along with an identity operator. To select operators for each layer, we adopt the DARTS methodology [97], combined with the Gumbel-Max trick, as discussed in section 6.2.1.1. Each candidate operator is associated with an architecture parameter α_i^j , $j = 1, 2, 3, 4, 5$.

Since this work focuses on optimizing only the first stage of the HAR model, the search space is defined to include more layers and a broader variety of operators compared to our previous work, MicroNAS. The number of convolutional layers with a stride of 2 is determined by the length of the input L . To prevent excessive reduction of the temporal dimension, a specific strategy is employed: the number of layers with a convolution stride of 2 is defined as $\lfloor \log_2 \left(\frac{L}{4} \right) \rfloor$. The denominator 4 ensures that, even when all convolutional layers with a stride of 2 are applied, the resulting size in the temporal dimension will be at least 4. This setup maximizes the inclusion of convolutional layers within the search space. Compared to our previous work, MicroNAS, the number of convolutional layers has increased and can be flexibly adjusted ac-

ording to the specific task requirements during the optimization process. For example, when a layer is designated as an identity layer, it reduces the total number of layers in the first stage. However, in our previous work, MicroNAS, the number of convolutional layers was fixed.

6.3.2.3 Optimization. following our previous work as described in section 6.2.1.4, the entire optimization process can be mathematically defined as follows,

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), v^*(\alpha), \alpha) \quad (6.15)$$

$$s.t. w^*(\alpha), v^*(\alpha) = \underset{w, v}{\operatorname{argmin}} \mathcal{L}_{train}(w, v, \alpha) \quad (6.16)$$

Where \mathcal{L}_{train} and \mathcal{L}_{val} denote the training and validation loss, respectively. The key difference in our approach is that, in addition to optimizing the model weights w , we also incorporate the trainable parameters v from the DCS layer. This results in a typical bi-level optimization problem [97], with α as the upper-level variable and the model weights w and learnable scaling factors v as the lower-level variables. The optimization of these two sets of parameters follows the iterative optimization procedure used in DARTS. In equation 6.15, the loss function comprises the cross-entropy loss $\mathcal{L}_{crossentropy}$ and hardware-aware constraints, defined as follows:

$$\mathcal{L} = \mathcal{L}_{crossentropy} + \lambda_{lat} \mathcal{L}_{latency} + \lambda_{mem} \mathcal{L}_{memory} \quad (6.17)$$

The hardware-aware constraints include model latency constraint and peak memory usage limitation. λ_{lat} and λ_{mem} are trade-off parameters that control the influence of efficiency and memory usage on the overall loss. During the total loss minimization process, a policy is learned that decides whether to remove or retain a layer, as well as modify its number of filters and sensor channels.

Compared to our previous work, MicroNAS, a significant difference in this study is that we use model computational complexity as a proxy for execution latency. The model latency loss $\mathcal{L}_{latency}$ is analytically estimated through the

number of FLOPs (Floating Point Operations) as follows:

$$\mathcal{L}_{latency} = \sum_{i=1}^{N_{conv}} \sum_j^5 \mathbf{c}_i^j \left| \hat{\mathbf{b}}_{i-1}^* \cdot \hat{\mathbf{b}}_i^{*j} \cdot \hat{C}_0 \frac{L_{i-1}}{st_i} \cdot k_i^j \cdot 1 \right|^2 \quad (6.18)$$

For each layer i , the computational complexity of the selected operator op_i^j is included in the L2 norm regularization. The vector \mathbf{c}_i^j is a one-hot encoded vector indicating which operator is sampled. The term $\hat{\mathbf{b}}_i^{*j} = \sum (\mathbf{b}_i^{*j})$ denotes the number of active filters for the selected operator, representing the output dimension, while $\hat{\mathbf{b}}_{i-1}^*$ represents the input dimension. The stride of the current layer is denoted by st_i , and k_i^j represents the current kernel size. The size of the active sensor channels, $\hat{C}_0 = \sum (\hat{\mathbf{b}}_0^*)$, is derived from the first DCS layer for the input segment. It is evident that the number of active sensor channels directly impacts the computational complexity of all subsequent layers. Selecting and utilizing only the important sensor channels can significantly reduce the model's computational complexity. L_{i-1} represents the length of the input in the temporal dimension. Although FLOPs are not a perfect proxy for latency, they exhibit a strong correlation, as demonstrated in the evaluation section 6.2.2.2. Using FLOPs directly as a proxy for latency allows us to bypass the cumbersome process of profiling all operators on hardware.

The loss for peak memory consumption is defined as follows:

$$\mathcal{L}_{memory} = \begin{cases} \log \left(\frac{Mem(\alpha, \mathbf{v}, MCU_t)}{Mem_t} \right), & \text{if } Mem(\alpha, \mathbf{v}, MCU_t) > Mem_t \\ 0, & \text{otherwise} \end{cases} \quad (6.19)$$

A penalty is applied when peak memory usage exceeds the defined maximum memory limit Mem_t .

6.3.3 Experiments and Discussions

6.3.3.1 Experiment Setup. We use three HAR benchmark datasets to validate the proposed framework and provide empirical evidence of its generalizability: **Skoda** [170], the UCI-HAR dataset [128], and the MotionSense dataset [107]. To demonstrate the applicability of the proposed framework

across various models, we consider three HAR models in our experiments: DeepConvLSTM (DCL) [119], TinyHAR [185], and Attend-and-Discriminate (Attend) [1]. All three models utilize an individual convolution block in the initial stage. Therefore, we replace only the initial stage with our proposed searchable framework, while the structure of the subsequent stages remains consistent with the original models.

Pruning is a commonly used method to make models more lightweight. In addition to our proposed method, we include two SOTA pruning methods, AutoSlim [169] and PaS [92], as baselines. Since we did not modify these two algorithms, they only prune the number of filters in each convolutional layer. However, our algorithm not only prunes the Sensor channels and Filters of each layer but also reduces the length of the Temporal dimension. Thus, we have named our model as SFTNAS.

To optimize the model weights w and informative parameters \mathbf{v} , we use the same setup as described in Section 2.4. For the optimization of the architecture parameters α , we employ an additional Adam optimizer with a learning rate of $\xi_\alpha = 5 \times 10^{-3}$, momentum (0.5, 0.999), and a weight decay of 10^{-3} . The parameters α are initialized to 10^{-3} . The temperature θ is initially set to $\theta_0 = 5.0$, and its decay is governed by the equation $\theta = \theta_0 \times \exp(-0.05 \times epoch)$.

6.3.3.2 Comparison to State-of-the-art. The three tables, Table 6.1, Table 6.3, and Table 6.2, provide performance results for the algorithms applied to different HAR models across three datasets: Motion Sense, UCI-HAR, and SKODA. The algorithms with the best performance are highlighted in bold. From a general point of view, SFTNAS consistently stands out as the most effective algorithm in reducing model size across all datasets and models. It achieves significant reductions in the number of parameters compared to the original models. In addition, SFTNAS also proves to be highly efficient in terms of model complexity reduction, achieving the lowest MMACs across all combinations of datasets and models, indicating its ability to significantly improve inference time. While AutoSlim and PaS also contribute to reducing model size and complexity, they are not as effective as SFTNAS.

Regarding classification performance, SFTNAS enables all three HAR models to have the smallest model size and the least complexity on the Motion

Table 6.1: Performance comparison between the SFTNAS framework and other pruning methods on the MotionSense dataset.

Model + Methods	Parameters	MMACs	$F1_m$
DCL	522.57 k	138.73	92.23
DCL + SFTNAS	81.92 k	2.32	89.88
DCL + AutoSlim	131.24 k	35.14	88.67
DCL + PaS	95.33 k	25.33	88.21
Attend	519.11 k	155.62	94.42
Attend + SFTNAS	56.45 k	3.74	91.37
Attend + AutoSlim	81.76 k	24.92	90.98
Attend + PaS	73.15 k	19.02	90.46
TinyHAR	322.76 k	47.17	92.12
TinyHAR + SFTNAS	41.28 k	1.79	90.17
TinyHAR + AutoSlim	50.03 k	7.69	89.21
TinyHAR + PaS	54.27 k	6.15	90.20

Sense and UCI-HAR datasets, achieving optimal results. On the SKODA dataset, however, SFTNAS shows slightly lower results than the other two pruning algorithms. Nevertheless, it is important to emphasize that the reductions in both model size and complexity on the SKODA dataset are significant compared to the other two algorithms. This is because the proposed SFTNAS not only reduces the number of filters, but also prunes the temporal dimension and the sensor channels. One limitation is that SFTNAS only optimizes the initial stage, while the subsequent model structures remain fixed.

It is important to note that, compared to the original models, the models compressed by SFTNAS are reduced in size by approximately 7-20 times, while the decrease in performance across the three datasets is no more than 3%. Compared to MicroNAS, the performance degradation is significantly mitigated.

6.3.3.3 Performance on Hardware. We also performed an evaluation on the NUCLEO-L552ZE-Q, equipped with a STM32L552 (Single core, ARM Cortex-M33, 80 MHz CPU clock, 512 kB flash, 256 kB SRAM), using the UCI-HAR dataset. Given the extreme limitations of this hardware, the three

Table 6.2: Performance comparison between the SFTNAS framework and other pruning methods on the Skoda(r) dataset.

Model + Methods	Parameters	MMACs	$F1_m$
DCL	1.11 M	534.44	91.12
DCL + SFTNAS	61.66 k	3.54	90.01
DCL + AutoSlim	173.93 k	83.72	88.45
DCL + PaS	187.93 k	70.42	89.33
Attend	962.0 k	592.41	92.39
Attend + SFTNAS	63.87 k	5.03	91.04
Attend + AutoSlim	139.17 k	90.44	91.23
Attend + PaS	150.82 k	87.65	91.59
TinyHAR	470.73 k	183.2	90.48
TinyHAR + SFTNAS	38.01 k	4.27	87.48
TinyHAR + AutoSlim	141.73 k	57.53	87.01
TinyHAR + PaS	104.38 k	43.01	88.31

models, as well as the variants pruned by AutoSlim and PaS, could not be deployed on it.

To deploy the model on this hardware, we applied the proposed SFTNAS to TinyHAR. In addition to considering the model complexity during training, we also included peak memory in the training loss, with L_{Mem} set to 96 kB. After the model was trained, this optimized architecture was further fine-tuned using quantization-aware training, and later the resulting model was deployed to the MCU using int_8 quantization. Adjusting the size of β_{mem} , we could obtain models with different computational complexities. We experimented with various values of β_{mem} and plotted the trade-off between the inference time and the performance of the resulting models in Figure 6.17 (a). This experiment demonstrates SFTNAS’s ability to find architectures under different model complexity constraints. When higher model complexities are allowed, classification performance increases.

We further analyze the distribution of the number of filters per layer in the optimize models and plot the distribution in Figure 6.17 (b). As can be observed from the figure, there is an increasing trend in the number of filters

Table 6.3: Performance comparison between the SFTNAS framework and other pruning methods on the UCI-HAR dataset.

Model + Methods	Parameters	MMACs	$F1_m$
DCL	424.26 k	105.95	92.51
DCL + SFTNAS	33.74 k	1.86	92.49
DCL + AutoSlim	46.21 k	11.72	91.83
DCL + PaS	37.89 k	9.64	92.04
Attend	445.38 k	120.55	94.07
Attend + SFTNAS	41.11 k	1.37	93.29
Attend + AutoSlim	48.62 k	13.59	92.88
Attend + PaS	39.89 k	11.2	92.71
TinyHAR	298.18 k	36.27	94.58
TinyHAR + SFTNAS	13.96 k	0.767	94.36
TinyHAR + AutoSlim	39.99 k	6.21	93.58
TinyHAR + PaS	21.34 k	3.56	94.19

as the number of layers increases. This trend can serve as a heuristic for the manual design of HAR models.

6.3.4 Discussion

In this work, we have optimized the design of individual blocks in the HAR model by using NAS in combination with a pruning technique. This approach is designed to minimize the size of various dimensions, namely the sensor channel dimension, time dimension, and filter dimension, without significantly sacrificing the model’s performance. It has been experimentally demonstrated that the proposed method can significantly reduce the size and complexity of the model while maintaining its performance. In many cases, the complexity is only about 2% of the original. This observation indirectly shows that reducing the complexity of the model in the first stage can also benefit subsequent computations. Finally, we demonstrate the usability of the proposed method by applying it on a microcontroller.

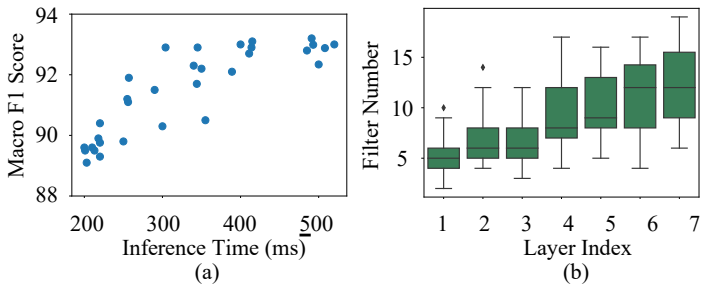


Figure 6.17: (a) Trade-off between inference time and performance from SFTNAS framework. (b) Distribution of optimized filter numbers from SFTNAS framework.

6.4 Summary

In this chapter, we explored the application of HA-NAS to the optimization of lightweight HAR models. HW-NAS provides an automated framework for discovering models that not only achieve high performance but also adhere to hardware constraints, greatly reducing the complexity of manual model design. In the context of SFTNAS, we identified that the majority of the complexity in HAR models stems from the first convolutional stage, which is significantly impacted by the sensor channels. By optimizing this initial stage, we were able to substantially reduce the model size while preserving comparable performance.

However, we also highlighted a limitation of this framework: it focuses solely on optimizing the first stage of HAR models, while the subsequent stages remain unaltered, adhering to the original model design. In contrast, MicroNAS provides a more flexible solution by enabling global optimization across the entire model. Future work could extend the search space to include the later stages of HAR models, allowing for more comprehensive optimization and model compression.

7 Conclusion

Wearable HAR systems are increasingly used in real-time monitoring applications. Many of these applications depend on continuous sensor data collection and real-time processing on edge devices such as smartwatches and micro-controllers. However, the limited computational power, memory, and battery life of these devices present significant challenges for deploying complex deep learning models.

In recent years, numerous HAR models have been developed with progressively better classification accuracy, yet these gains often come at the expense of increased model complexity and size. Relatively few studies have focused on lightweight HAR models, especially in the deep learning domain. This dissertation addresses this gap by introducing a series of methods that not only maintain high classification accuracy but also minimize resource consumption, enabling wearable HAR systems to operate efficiently on resource-constrained edge devices such as micro-controllers.

Contribution 1: Data Preparation for Model Size Reduction. The first major contribution of this work focuses on data preparation as a means to reduce model size. This approach is model-agnostic and can be applied to any HAR model, making it particularly useful for non-expert users working in resource-constrained environments. A common approach to reducing model size is to manually decrease the number of layers or filters in a pre-existing model. However, this often reduces the model's learning capacity, resulting in diminished performance, even if the model becomes deployable on edge devices.

To address this issue, we proposed techniques such as **automated data augmentation** and a **learnable wavelet layer**, which can be easily applied to manually downsized models, creating an end-to-end learning framework. Experimental results show that these approaches effectively mitigate performance

loss, allowing for smaller models that maintain or even improve upon the original performance.

For example, the task-specific data augmentation optimization framework **AutoAugHAR** enriched the training dataset, providing the model with more diverse and informative inputs, thus enabling smaller models to learn more robust patterns. Similarly, the **learnable wavelet layer** improved the model's ability to capture frequency-domain information, which is crucial for activity classification, enhancing both model efficiency and compactness.

Contribution 2: Design Principles for Lightweight Models. The second major contribution is the establishment of design principles specifically tailored for lightweight models. These principles focus on two key aspects: (1) identifying the essential information required for accurate HAR predictions, and (2) understanding the unique characteristics of HAR data and the reasons why HAR models are often large and computationally intensive.

Building on these insights, we developed models such as **TinyHAR**, **Cross-Atten**, and **MLP-HAR**. Despite their reduced size, these models demonstrate robust information extraction capabilities by explicitly embedding necessary information flows into the model architecture. For example, **TinyHAR** incorporates a hierarchical information flow, progressing from coarse-grained to global features and facilitating cooperation across different modalities, significantly enhancing its ability to capture key activity patterns. The **Cross-Atten** model further extends TinyHAR by effectively leveraging multiple representations to enhance performance. Notably, the **MLP-HAR** model is specifically optimized for edge devices, utilizing an architecture composed entirely of fully connected layers. Despite its simplicity, MLP-HAR achieves SOTA performance in terms of classification accuracy, while offering fast inference times and efficient memory usage. These design principles not only serve as a blueprint for developing lightweight models but also provide valuable insights for structuring the search space in the AutoML frameworks introduced later.

Contribution 3: AutoML Framework for Model Optimization. The third major contribution addresses the challenge of optimizing models for various application scenarios and hardware environments. To simplify the model design

process, we developed two AutoML frameworks (**MicroNAS** and **SFTNAS**) that automate model architecture optimization. Before initiating the optimization process, users define the application scenario, target hardware, memory limitations, and inference time requirements. The AutoML framework then balances these objectives and outputs a model that is both deployable on the specified hardware and capable of high performance.

This automation is essential in scenarios where manually optimizing models is impractical due to the complexity of balancing classification performance, model size, and computational efficiency. For instance, when deploying models on micro-controllers, manually tuning models to meet strict memory and inference time requirements would be highly challenging. Our frameworks ensure that these constraints are incorporated into the optimization process, facilitating efficient deployment without compromising performance.

Moreover, our **STFNAS** framework integrates **pruning techniques** within the model search process, further reducing model complexity by automatically selecting the most important sensor channels. This not only simplifies the model but also reduces data collection and transmission burdens, which are critical factors in real-world applications where data acquisition can be costly and energy-intensive.

Conclusion. In summary, this dissertation explores multiple strategies for developing lightweight HAR models, addressing challenges from various perspectives, including data preparation, model design, and automated optimization. Each approach has been validated through extensive experimentation, demonstrating its effectiveness in reducing model size while maintaining high performance.

The work presented in this dissertation represents a significant advancement in the field of wearable HAR systems on edge devices. It provides key insights and sets the foundation for further research into developing highly efficient, lightweight models for real-world deployment. The future of wearable HAR holds great promise, with more diverse applications and enhanced user experiences as lightweight models become increasingly feasible for deployment in various real-world scenarios.

Bibliography

- [1] A. Abedin, M. Ehsanpour, Q. Shi, H. Rezatofghi, and D. C. Ranasinghe. Attend and discriminate: Beyond the state-of-the-art for human activity recognition using wearable sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(1):1–22, 2021.
- [2] A. Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.
- [3] N. Ahmad and H.-f. Leung. Alae-tae-cutmix+: Beyond the state-of-the-art for human activity recognition using wearable sensors. In *2023 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 222–231. IEEE, 2023.
- [4] L. Alawneh, T. Alsarhan, M. Al-Zinati, M. Al-Ayyoub, Y. Jararweh, and H. Lu. Enhancing Human Activity Recognition Using Deep Learning and Time Series Augmented Data. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–16, 2021.
- [5] G. Anandalingam and T. L. Friesz. Hierarchical optimization: An introduction. *Annals of Operations Research*, 34:1–11, 1992.
- [6] D. Anguita, A. Ghio, L. Oneto, X. Parra, J. L. Reyes-Ortiz, et al. A Public Domain Dataset for Human Activity Recognition Using Smartphones. In *Esann*, volume 3, page 3, 2013.
- [7] Ankita, S. Rani, H. Babbar, S. Coleman, A. Singh, and H. M. Aljahdali. An efficient and lightweight deep learning model for human activity recognition using smartphones. *Sensors*, 21(11):3845, 2021.

- [8] Arduino. Arduino nicla sense me. <https://www.bosch-sensortec.com/software-tools/tools/arduino-nicla-sense-me>, 2022. (Accessed: 19.05.2022).
- [9] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [10] M. Bachlin, D. Roggen, G. Troster, M. Plotnik, N. Inbar, I. Meidan, T. Herman, M. Brozgol, E. Shaviv, N. Giladi, et al. Potentials of Enhanced Context Awareness in Wearable Assistants for Parkinson’s Disease Patients with the Freezing of Gait Syndrome. In *2009 International Symposium on Wearable Computers*, pages 123–130. IEEE, 2009.
- [11] M. Bachlin, D. Roggen, G. Troster, M. Plotnik, N. Inbar, I. Meidan, T. Herman, M. Brozgol, E. Shaviv, N. Giladi, et al. Potentials of Enhanced Context Awareness in Wearable Assistants for Parkinson’s Disease Patients with the Freezing of Gait Syndrome. In *2009 International Symposium on Wearable Computers*, pages 123–130. IEEE, 2009.
- [12] A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh. The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*, 2018.
- [13] O. Banos, R. Garcia, J. A. Holgado-Terriza, M. Damas, H. Pomares, I. Rojas, A. Saez, and C. Villalonga. mhealthdroid: a novel framework for agile development of mobile health applications. In *Ambient Assisted Living and Daily Activities: 6th International Work-Conference, IWAAL 2014, Belfast, UK, December 2-5, 2014. Proceedings 6*, pages 91–98. Springer, 2014.
- [14] B. Barshan and M. C. Yükses. Recognizing Daily and Sports Activities in Two Open Source Machine Learning Environments Using Body-worn Sensor Units. *The Computer Journal*, 57(11):1649–1667, 2014.
- [15] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

- [16] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.
- [17] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang. Hardware-aware neural architecture search: Survey and taxonomy. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4322–4329. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/592. URL <https://doi.org/10.24963/ijcai.2021/592>. Survey Track.
- [18] H. Benmeziane, K. E. Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang. A comprehensive survey on hardware-aware neural architecture search. *arXiv preprint arXiv:2101.09336*, 2021.
- [19] M. Bock, A. Hölzemann, M. Moeller, and K. Van Laerhoven. Improving Deep Learning for HAR with Shallow LSTMs. In *2021 International Symposium on Wearable Computers*, pages 7–12, 2021.
- [20] H. Cai, L. Zhu, and S. Han. Proxylessnas: Direct neural architecture search on target task and hardware, 2019.
- [21] R. Chavarriaga, H. Sagha, A. Calatroni, S. T. Digumarti, G. Tröster, J. d. R. Millán, and D. Roggen. The Opportunity Challenge: A Benchmark Database for On-body Sensor-based Activity Recognition. *Pattern Recognition Letters*, 34(15):2033–2042, 2013.
- [22] J. Chen and X. Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019. doi: 10.1109/JPROC.2019.2921977.
- [23] K. Chen, L. Yao, D. Zhang, B. Guo, and Z. Yu. Multi-agent attentional activity recognition. *arXiv preprint arXiv:1905.08948*, 2019.
- [24] K. Chen, D. Zhang, L. Yao, B. Guo, Z. Yu, and Y. Liu. Deep learning for sensor-based human activity recognition: Overview, challenges, and opportunities. *ACM Computing Surveys (CSUR)*, 54(4):1–40, 2021.

- [25] Y. Chen, K. Zhong, J. Zhang, Q. Sun, and X. Zhao. Lstm networks for mobile human activity recognition. In *2016 International conference on artificial intelligence: technologies and applications*, pages 50–53. Atlantis Press, 2016.
- [26] H. Cheng, M. Zhang, and J. Q. Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [27] S. Chung, J. Lim, K. J. Noh, G. Kim, and H. Jeong. Sensor Data Acquisition and Multimodal Sensor Fusion for Human Activity Recognition Using Deep Learning. *Sensors*, 19(7):1716, 2019.
- [28] Y. L. Coelho, F. d. A. S. dos Santos, A. Frizzera-Neto, and T. F. Bastos-Filho. A lightweight framework for human activity recognition on wearable devices. *IEEE Sensors Journal*, 21(21):24471–24481, 2021.
- [29] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.
- [30] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. Randaugment: Practical Automated Data Augmentation with A Reduced Search Space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.
- [31] Z. Cui, W. Chen, and Y. Chen. Multi-scale convolutional neural networks for time series classification. 2016. doi: 10.48550/ARXIV.1603.06995. URL <https://arxiv.org/abs/1603.06995>.
- [32] F. Daghero, A. Burrello, C. Xie, M. Castellano, L. Gandolfi, A. Calimera, E. Macii, M. Poncino, and D. J. Pagliari. Human activity recognition on microcontrollers with quantized and adaptive deep neural networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 21(4):1–28, 2022.

- [33] L. M. Dang, K. Min, H. Wang, M. J. Piran, C. H. Lee, and H. Moon. Sensor-based and vision-based human activity recognition: A comprehensive survey. *Pattern Recognition*, 108:107561, 2020.
- [34] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.
- [35] J. G. Daugman. Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-dimensional Visual Cortical Filters. *JOSA A*, 2(7):1160–1169, 1985.
- [36] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, et al. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems*, 3:800–811, 2021.
- [37] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, et al. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems*, 3:800–811, 2021.
- [38] A. Dempster, D. F. Schmidt, and G. I. Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 248–257, 2021.
- [39] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.
- [40] M. Dong, J. Han, Y. He, and X. Jing. Har-net: Fusing deep representation and hand-crafted features for human activity recognition. In *Signal and Information Processing, Networking and Computers: Proceedings of the 5th International Conference on Signal and Information Processing, Networking and Computers (ICSINC)*, pages 32–40. Springer, 2019.

- [41] X. Dong and Y. Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1761–1770, 2019.
- [42] D. L. Donoho and M. Elad. Optimally Sparse Representation in General (Nonorthogonal) Dictionaries via ℓ_1 Minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.
- [43] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [44] T. Elsken, J. H. Metzen, and F. Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*, 2018.
- [45] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.
- [46] S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*, 2018.
- [47] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy. A survey of data augmentation approaches for nlp. *arXiv preprint arXiv:2105.03075*, 2021.
- [48] M. Feurer, A. Klein, K. Eggenberger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28, 2015.
- [49] E. Fons, P. Dawson, X.-j. Zeng, J. Keane, and A. Iosifidis. Adaptive weighting scheme for automatic time-series data augmentation. *arXiv preprint arXiv:2102.08310*, 2021.
- [50] S. Gedam and S. Paul. A review on mental stress detection using wearable sensors and machine learning techniques. *IEEE Access*, 9:84045–84066, 2021.

- [51] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022.
- [52] M. Goubeaud, P. Joußen, N. Gmyrek, F. Ghorban, L. Schelkes, and A. Kummert. Using Variational Autoencoder to Augment Sparse Time Series Datasets. In *2021 7th International Conference on Optimization and Applications (ICOA)*, pages 1–6. IEEE, 2021.
- [53] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.
- [54] F. Gu, M.-H. Chung, M. Chignell, S. Valaee, B. Zhou, and X. Liu. A survey on deep learning for human activity recognition. *ACM Computing Surveys (CSUR)*, 54(8):1–34, 2021.
- [55] Y. Guan and T. Plötz. Ensembles of deep lstm learners for activity recognition using wearables. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, 1(2):1–28, 2017.
- [56] E. J. Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1948.
- [57] S. Ha and S. Choi. Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors. In *2016 international joint conference on neural networks (IJCNN)*, pages 381–388. IEEE, 2016.
- [58] S. Ha, J.-M. Yun, and S. Choi. Multi-modal convolutional neural networks for activity recognition. In *2015 IEEE International conference on systems, man, and cybernetics*, pages 3017–3022. IEEE, 2015.
- [59] R. Hataya, J. Zdenek, K. Yoshizoe, and H. Nakayama. Faster augmentation: Learning augmentation strategies using backpropagation.

- In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXV 16*, pages 1–16. Springer, 2020.
- [60] S. Holm. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.
- [61] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [62] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.
- [63] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
- [64] S. Huang, P.-Y. Chen, and J. McCann. Diffar: adaptive conditional diffusion model for temporal-augmented human activity recognition. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 3812–3820, 2023.
- [65] D. Hughes and N. Correll. Distributed convolutional neural networks for human activity recognition in wearable robotics. In *Distributed Autonomous Robotic Systems: The 13th International Symposium*, pages 619–631. Springer, 2018.
- [66] Z. Hussain, M. Sheng, and W. E. Zhang. Different approaches for human activity recognition: A survey. *arXiv preprint arXiv:1906.05074*, 2019.
- [67] S. Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

- [68] H. Ismail Fawaz, B. Lucas, G. Forestier, C. Pelletier, D. F. Schmidt, J. Weber, G. I. Webb, L. Idoumghar, P.-A. Muller, and F. Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, 2020.
- [69] B. K. Iwana and S. Uchida. An Empirical Survey of Data Augmentation for Time Series Classification with Neural Networks. *Plos one*, 16(7): e0254841, 2021.
- [70] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- [71] C. Y. Jeong, H. C. Shin, and M. Kim. Sensor-Data Augmentation for Human Activity Recognition with Time-Warping and Data Masking. *Multimedia Tools and Applications*, 80:20991–21009, 2021.
- [72] H. Jin, Q. Song, and X. Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956, 2019.
- [73] A. Jordao, A. C. Nazare Jr, J. Sena, and W. R. Schwartz. Human activity recognition based on wearable sensor data: A standardization of the state-of-the-art. *arXiv preprint arXiv:1806.05226*, 2018.
- [74] G. Kaiser and L. H. Hudgins. *A Friendly Guide to Wavelets*, volume 300. Springer, 1994.
- [75] G. Kalouris, E. I. Zacharaki, and V. Megalooikonomou. Improving CNN-Based Activity Recognition by Data Augmentation and Transfer Learning. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 1387–1394. IEEE, 2019.
- [76] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31, 2018.

- [77] D. M. Karantonis, M. R. Narayanan, M. Mathie, N. H. Lovell, and B. G. Celler. Implementation of A Real-Time Human Movement Classifier Using A Triaxial Accelerometer for Ambulatory Monitoring. *IEEE transactions on information technology in biomedicine*, 10(1):156–167, 2006.
- [78] U. Khandelwal, H. He, P. Qi, and D. Jurafsky. Sharp nearby, fuzzy far away: How neural language models use context. *arXiv preprint arXiv:1805.04623*, 2018.
- [79] T. King, Y. Zhou, T. Röddiger, and M. Beigl. Micronas: Memory and latency constrained hardware-aware neural architecture search for time series classification on microcontrollers. *arXiv preprint arXiv:2310.18384*, 2023.
- [80] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [81] N. Kitaev, Ł. Kaiser, and A. Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.
- [82] S. Kobayashi, T. Hasegawa, T. Miyoshi, and M. Koshino. Marnasnets: Towards cnn model architectures specific to sensor-based human activity recognition. *IEEE Sensors Journal*, 2023.
- [83] J. R. Kwapisz, G. M. Weiss, and S. A. Moore. Activity Recognition Using Cell Phone Accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.
- [84] L. Lai, N. Suda, and V. Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus, 2018. URL <https://arxiv.org/abs/1801.06601>.
- [85] L. Lai, N. Suda, and V. Chandra. Not all ops are created equal!, 2018. URL <https://arxiv.org/abs/1801.04326>.
- [86] A. Le Guennec, S. Malinowski, and R. Tavenard. Data Augmentation for Time Series Classification Using Convolutional Neural Networks. In

ECML/PKDD workshop on advanced analytics and learning on temporal data, 2016.

- [87] B. LEE. Emg-eeg dataset for upper-limb gesture classification, 2023. URL <https://dx.doi.org/10.21227/5ztn-4k41>.
- [88] S.-M. Lee, S. M. Yoon, and H. Cho. Human activity recognition from accelerometer data using convolutional neural network. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 131–134. IEEE, 2017.
- [89] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, et al. Yolov6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976*, 2022.
- [90] X. Li, J. Luo, and R. Younes. ActivityGAN: Generative Adversarial Networks for Data Augmentation in Sensor-Based Human Activity Recognition. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*, pages 249–254, 2020.
- [91] Y. Li, G. Hu, Y. Wang, T. Hospedales, N. M. Robertson, and Y. Yang. Dada: Differentiable automatic data augmentation. *arXiv preprint arXiv:2003.03780*, 2020.
- [92] Y. Li, P. Zhao, G. Yuan, X. Lin, Y. Wang, and X. Chen. Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization. *arXiv preprint arXiv:2206.01198*, 2022.
- [93] E. Liberis, L. Dudziak, and N. D. Lane. μ nas: Constrained neural architecture search for microcontrollers. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, EuroMLSys '21, page 70–79, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450382984. doi: 10.1145/3437984.3458836. URL <https://doi.org/10.1145/3437984.3458836>.

- [94] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.
- [95] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018.
- [96] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.
- [97] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.
- [98] P. Liu, B. Wu, H. Ma, and M. Seok. Memnas: Memory-efficient neural architecture search with grow-trim learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.
- [99] S. Liu, S. Yao, J. Li, D. Liu, T. Wang, H. Shao, and T. Abdelzaher. Giobalfusion: A global attentional deep learning framework for multisensor information fusion. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(1):1–27, 2020.
- [100] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Computer Vision—ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.
- [101] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning Efficient Convolutional Networks Through Network Slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

- [102] H. Ma, W. Li, X. Zhang, S. Gao, and S. Lu. Attnsense: Multi-level attention mechanism for multimodal human activity recognition. In *IJCAI*, pages 3109–3115, 2019.
- [103] C. J. Maddison, D. Tarlow, and T. Minka. A* sampling. *Advances in neural information processing systems*, 27, 2014.
- [104] S. Mahmud, M. Tanjid Hasan Tonmoy, K. Kumar Bhaumik, A. Mahbubur Rahman, M. Ashraful Amin, M. Shoyaib, M. Asif Hossain Khan, and A. Ahsan Ali. Human activity recognition from wearable sensor data using self-attention. In *ECAI 2020*, pages 1332–1339. IOS Press, 2020.
- [105] S. Mahmud, M. Tonmoy, K. K. Bhaumik, A. M. Rahman, M. A. Amin, M. Shoyaib, M. A. H. Khan, and A. A. Ali. Human Activity Recognition from Wearable Sensor Data Using Self-attention. *arXiv preprint arXiv:2003.09018*, 2020.
- [106] R. Mahony, T. Hamel, and J.-M. Pfimlin. Complementary filter design on the special orthogonal group $so(3)$. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 1477–1484. IEEE, 2005.
- [107] M. Malekzadeh, R. G. Clegg, A. Cavallaro, and H. Haddadi. Mobile sensor data anonymization. In *Proceedings of the international conference on internet of things design and implementation*, pages 49–58, 2019.
- [108] S. Mallat et al. *A Wavelet Tour of Signal Processing: the Sparse Way. AP Professional, Third Edition, London*, 2009.
- [109] Y. Mei, T. Jiang, X. Ding, Y. Zhong, S. Zhang, and Y. Liu. Wi-wave: Wifi-based Human Activity Recognition Using the Wavelet Integrated CNN. In *2021 IEEE/CIC International Conference on Communications in China (ICCC Workshops)*, pages 100–105, 2021. doi: 10.1109/ICCCWorkshops52231.2021.9538931.

- [110] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter. Towards automatically-tuned neural networks. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *Proceedings of the Workshop on Automatic Machine Learning*, volume 64 of *Proceedings of Machine Learning Research*, pages 58–65, New York, New York, USA, 24 Jun 2016. PMLR. URL https://proceedings.mlr.press/v64/mendoza_towards_2016.html.
- [111] F. Meng, H. Liu, Y. Liang, J. Tu, and M. Liu. Sample fusion network: An end-to-end data augmentation network for skeleton-based human action recognition. *IEEE Transactions on Image Processing*, 28(11): 5281–5295, 2019.
- [112] S. Münzner, P. Schmidt, A. Reiss, M. Hanselmann, R. Stiefelhagen, and R. Dürichen. CNN-Based Sensor Fusion Techniques for Multimodal Human Activity Recognition. In *Proceedings of the 2017 ACM international symposium on wearable computers*, pages 158–165, 2017.
- [113] V. S. Murahari and T. Plötz. On Attention Models for Human Activity Recognition. In *Proceedings of the 2018 ACM international symposium on wearable computers*, pages 100–103, 2018.
- [114] A. Nedorubova, A. Kadyrova, and A. Khlyupin. Human activity recognition using continuous wavelet transform and convolutional neural networks. *arXiv preprint arXiv:2106.12666*, 2021.
- [115] K. Nguyen-Trong, H. N. Vu, N. N. Trung, and C. Pham. Gesture recognition using wearable sensors with bi-long short-term memory convolutional neural networks. *IEEE Sensors Journal*, 21(13):15065–15079, 2021.
- [116] A. Q. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.
- [117] K. Ojiako and K. Farrahi. Mlps are all you need for human activity recognition. *Applied Sciences*, 13(20):11154, 2023.

- [118] R. S. Olson and J. H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. In *Workshop on automatic machine learning*, pages 66–74. PMLR, 2016.
- [119] F. J. Ordóñez and D. Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1), 2016. ISSN 1424-8220. doi: 10.3390/s16010115. URL <https://www.mdpi.com/1424-8220/16/1/115>.
- [120] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. SpecAugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.
- [121] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [122] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.
- [123] J. Piskozub. Letters of polish sign language alphabet, 2023. URL <https://dx.doi.org/10.21227/w90m-m764>.
- [124] A. Poria. Uncertainty principles for the fourier and the short-time fourier transforms. *Journal of Mathematical Physics*, 62(11), 2021.
- [125] V. Radu, C. Tong, S. Bhattacharya, N. D. Lane, C. Mascolo, M. K. Marina, and F. Kawsar. Multimodal deep learning for activity and context recognition. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, 1(4):1–27, 2018.
- [126] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *International conference on machine learning*, pages 2902–2911. PMLR, 2017.

- [127] A. Reiss and D. Stricker. Introducing A New Benchmarked Dataset for Activity Monitoring. In *2012 16th international symposium on wearable computers*, pages 108–109. IEEE, 2012.
- [128] J.-L. Reyes-Ortiz, L. Oneto, A. Samà, X. Parra, and D. Anguita. Transition-aware human activity recognition using smartphones. *Neurocomputing*, 171:754–767, 2016.
- [129] J. Roberts and T. D. Roberts. Use of the butterworth low-pass filter for oceanographic data. *Journal of Geophysical Research: Oceans*, 83 (C11):5510–5514, 1978.
- [130] M. X. B. Rodriguez, A. Gruson, L. Polania, S. Fujieda, F. Prieto, K. Takayama, and T. Hachisuka. Deep Adaptive Wavelet Network. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.
- [131] P. J. Rousseeuw. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.
- [132] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.
- [133] S. R. Shakya, C. Zhang, and Z. Zhou. Comparative study of machine learning and deep learning architecture for human activity recognition using accelerometer data. *Int. J. Mach. Learn. Comput*, 8(6):577–582, 2018.
- [134] S. Shao and V. Sanchez. A study on diffusion modelling for sensor-based human activity recognition. In *2023 11th International Workshop on Biometrics and Forensics (IWBF)*, pages 1–7. IEEE, 2023.
- [135] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

- [136] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen. Smart devices are different: Assessing and mitigating mobile sensing heterogeneities for activity recognition. In *Proceedings of the 13th ACM conference on embedded networked sensor systems*, pages 127–140, 2015.
- [137] STMicroelectronics. Nucleo-1552ze-q - stm32 nucleo-144 development board. <https://www.st.com/en/evaluation-tools/nucleo-1552ze-q.html>, 2023. (Accessed on 05/31/2022).
- [138] STMicroelectronics. Nucleo-1552ze-q - stm32 nucleo-144 development board. <https://www.st.com/en/evaluation-tools/nucleo-1552ze-q.html>, 2023. (Accessed on 05/31/2022).
- [139] D. Strigl, K. Kofler, and S. Podlipnig. Performance and scalability of gpu-based convolutional neural networks. In *2010 18th Euromicro conference on parallel, distributed and network-based processing*, pages 317–324. IEEE, 2010.
- [140] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.
- [141] T. Sztyler and H. Stuckenschmidt. On-body localization of wearable devices: An investigation of position-aware activity recognition. In *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–9. IEEE, 2016.
- [142] M. Tan. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.
- [143] C. I. Tang, I. Perez-Pozuelo, D. Spathis, S. Brage, N. Wareham, and C. Mascolo. Selfhar: Improving human activity recognition through self-training with unlabeled data. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, 5(1):1–30, 2021.

- [144] Y. Tang, Q. Teng, L. Zhang, F. Min, and J. He. Layer-wise training convolutional neural networks with smaller filters for human activity recognition using wearable sensors. *IEEE Sensors Journal*, 21(1):581–592, 2020.
- [145] Tensorflow. `tflite-micro/memory_management.md` at `main` · tensorflow/tflite-micro. <https://github.com/tensorflow/tflite-micro/>, 2022. (Accessed on 06/12/2022).
- [146] Y. Tian, J. Zhang, J. Wang, Y. Geng, and X. Wang. Robust Human Activity Recognition Using Single Accelerometer via Wavelet Energy Spectrum Features and Ensemble Feature Selection. *Systems Science & Control Engineering*, 8(1):83–96, 2020.
- [147] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.
- [148] M. T. H. Tonmoy, S. Mahmud, A. Mahbubur Rahman, M. Ashraf Amin, and A. A. Ali. Hierarchical self attention based autoencoder for open-set human activity recognition. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 351–363. Springer, 2021.
- [149] T. T. Um, F. M. Pfister, D. Pichler, S. Endo, M. Lang, S. Hirche, U. Fietzek, and D. Kulić. Data Augmentation of Wearable Sensor Data for Parkinson’s Disease Monitoring Using Convolutional Neural Networks. In *Proceedings of the 19th ACM international conference on multimodal interaction*, pages 216–220, 2017.
- [150] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
- [151] C. Vonesch, T. Blu, and M. Unser. Generalized Daubechies Wavelet Families. *IEEE Transactions on Signal Processing*, 55(9):4415–4429, 2007.

- [152] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen, P. Vajda, and J. E. Gonzalez. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12962–12971, 2020. doi: 10.1109/CVPR42600.2020.01298.
- [153] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023.
- [154] J. Wang, C. Yang, X. Jiang, and J. Wu. When: A wavelet-dtw hybrid attention network for heterogeneous time series analysis. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2361–2373, 2023.
- [155] S. Wang, H. Wu, X. Shi, T. Hu, H. Luo, L. Ma, J. Y. Zhang, and J. Zhou. Timemixer: Decomposable multiscale mixing for time series forecasting. *arXiv preprint arXiv:2405.14616*, 2024.
- [156] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu. Time Series Data Augmentation for Deep Learning: A Survey. *arXiv preprint arXiv:2002.12478*, 2020.
- [157] F. Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in Statistics: Methodology and Distribution*, pages 196–202. Springer, 1992.
- [158] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.
- [159] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10726–10734, 2019. doi: 10.1109/CVPR.2019.01099.

- [160] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. *arXiv preprint arXiv:2210.02186*, 2022.
- [161] R. Xi, M. Hou, M. Fu, H. Qu, and D. Liu. Deep dilated convolution on multimodality time series for human activity recognition. In *2018 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2018.
- [162] S. Xie, H. Zheng, C. Liu, and L. Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.
- [163] T. Xu and W. Takano. Graph stacked hourglass networks for 3d human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16105–16114, 2021.
- [164] Z.-Q. J. Xu, Y. Zhang, and T. Luo. Overview frequency principle/spectral bias in deep learning. *arXiv preprint arXiv:2201.07395*, 2022.
- [165] S. K. Yadav, K. Tiwari, H. M. Pandey, and S. A. Akbar. A review of multimodal human activity recognition with special emphasis on classification, applications, challenges and future directions. *Knowledge-Based Systems*, 223:106970, 2021.
- [166] J. Yang, M. N. Nguyen, P. P. San, X. Li, and S. Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Ijcai*, volume 15, pages 3995–4001. Buenos Aires, Argentina, 2015.
- [167] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. In *Proceedings of the 26th international conference on world wide web*, pages 351–360, 2017.
- [168] F. Yu, D. Wang, E. Shelhamer, and T. Darrell. Deep layer aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2403–2412, 2018.

- [169] J. Yu and T. Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.
- [170] P. Zappi, C. Lombriser, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Tröster. Activity Recognition from On-body Sensors: Accuracy-power Trade-off by Dynamic Sensor Selection. In *European Conference on Wireless Sensor Networks*, pages 17–33. Springer, 2008.
- [171] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. Understanding and robustifying differentiable architecture search. *arXiv preprint arXiv:1909.09656*, 2019.
- [172] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, and J. Zhang. Convolutional neural networks for human activity recognition using mobile sensors. In *6th international conference on mobile computing, applications and services*, pages 197–205. IEEE, 2014.
- [173] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 2114–2124, 2021.
- [174] D. Zhang and D. Zhang. Wavelet transform. *Fundamentals of image data mining: Analysis, Features, Classification and Retrieval*, pages 35–44, 2019.
- [175] D. Zhang, L. Zhang, Q. Yi, L. Huang, and G. Zhang. Human Activity Recognition Based on Wavelet-CNN Architecture. In *2021 5th Asian Conference on Artificial Intelligence Technology (ACAIT)*, pages 77–85, 2021. doi: 10.1109/ACAIT53529.2021.9731338.
- [176] S. Zhang and X. Zhou. Micronet: Realizing micro neural network via binarizing ghostnet. In *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, pages 1340–1343, 2021. doi: 10.1109/ICSP51882.2021.9408972.

- [177] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.
- [178] Y. Zhang, L. Wang, H. Chen, A. Tian, S. Zhou, and Y. Guo. If-convtransformer: A framework for human activity recognition using imu fusion and convtransformer. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(2):1–26, 2022.
- [179] Z. Zhang, L. Wang, and C. Lee. Recent advances in artificial intelligence sensors. *Advanced Sensor Research*, 2(8):2200072, 2023. doi: <https://doi.org/10.1002/adrs.202200072>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/adrs.202200072>.
- [180] H. Zhao, C. Funk, B. Noack, U. Hanebeck, and M. Beigl. Kalman Filtered Compressive Sensing Using Pseudo-Measurements. In *2021 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pages 1–8. IEEE, 2021.
- [181] H. Zhao, Y. Zhou, T. Riedel, M. Hefenbrock, and M. Beigl. Improving human activity recognition models by learnable sparse wavelet layer. In *Proceedings of the 2022 ACM International Symposium on Wearable Computers*, pages 84–88, 2022.
- [182] Y. Zhao, R. Yang, G. Chevalier, X. Xu, and Z. Zhang. Deep residual bidir-lstm for human activity recognition using wearable sensors. *Mathematical Problems in Engineering*, 2018:1–13, 2018.
- [183] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.
- [184] Y. Zhou, M. Hefenbrock, Y. Huang, T. Riedel, and M. Beigl. Automatic remaining useful life estimation framework with embedded convolutional lstm as the backbone. In *Machine Learning and Knowledge*

Discovery in Databases: Applied Data Science Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part IV, pages 461–477. Springer, 2021.

- [185] Y. Zhou, H. Zhao, Y. Huang, T. Riedel, M. Hefenbrock, and M. Beigl. Tinyhar: A lightweight deep learning model designed for human activity recognition. In *Proceedings of the 2022 ACM International Symposium on Wearable Computers*, pages 89–93, 2022.
- [186] Y. Zhou, T. King, Y. Huang, H. Zhao, T. Riedel, T. Röddiger, and M. Beigl. Enhancing efficiency in har models: Nas meets pruning. In *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 33–38. IEEE, 2024.
- [187] Y. Zhou, T. King, H. Zhao, Y. Huang, T. Riedel, and M. Beigl. Mlp-har: Boosting performance and efficiency of har models on edge devices with purely fully connected layers. In *2024 ACM International Symposium on Wearable Computers (ISWC'24)*, 2024.
- [188] Y. Zhou, H. Zhao, Y. Huang, T. Röddiger, M. Kurnaz, T. Riedel, and M. Beigl. Autoaughar: Automated data augmentation for sensor-based human activity recognition. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 8(2):1–27, 2024.
- [189] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.
- [190] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.
- [191] S. Zuo, V. Fortes, S. Suh, S. Sigg, and P. Lukowicz. Unsupervised diffusion model for sensor-based human activity recognition. In *Adjunct Proceedings of the 2023 ACM International Joint Conference on Pervasive*

*and Ubiquitous Computing & the 2023 ACM International Symposium
on Wearable Computing*, pages 205–205, 2023.