

Multi-Task Layer Sharing Optimisations for Low-Power tinyML Devices

MICHAEL J. GIBBS

N1054738

A thesis submitted in partial fulfilment of the
requirements of Nottingham Trent University for
the degree of Doctor of Philosophy

March, 2025

In loving memory of my mother, Amanda Jane Gibbs.

Abstract

Deploying [Deep Learning \(DL\)](#) models on low-power [Microcontroller Unit \(MCU\)](#) devices presents significant challenges due to limited computational resources and storage constraints. This thesis explores novel methods for optimising [DL](#) models for edge-based [tiny Machine Learning \(tinyML\)](#) applications, with a focus on [Human Activity Recognition \(HAR\)](#) and stress detection. A multi-*model* system was implemented on an Arduino Nano 33 [BLE Sense](#), where a [HAR](#) model provided contextual information to improve stress detection. The [HAR](#) model achieved 98% accuracy in distinguishing between exercise and no exercise, while the stress detection model reached 88% accuracy. Quantisation techniques were employed to reduce memory requirements, allowing both models to fit within 1.3MB of storage.

To further optimise deployment, this thesis introduces the [tiny Inception Module \(tIM\)](#), a [DL](#) layer-sharing approach designed for [tinyML](#) systems. The [tIM](#) enables pre-existing network layers to be repurposed across models, reducing redundancy while maintaining accuracy. A case study using a [tIM](#) demonstrated a 28.3% storage saving compared to the initial multi-*model* system and a 47.8% reduction when compared to non-layer-sharing approaches, with only a 4% accuracy sacrifice to the stress model.

Additionally, [Echo State Networks \(ESNs\)](#) were investigated as an alternative to traditional [DL](#) approaches due to their low training cost. However, [ESNs](#) have historically been unsuitable for applications due to their high computational complexity. This thesis presents a novel [Multi-Task Learning \(MTL\)](#)-based [ESN](#) approach that shares a single reservoir across multiple datasets, including chaotic systems,

time-series, and image classification tasks. The shared [ESN](#) reservoir achieved competitive performance with state-of-the-art literature while significantly reducing [Floating Point Operations \(FLOPs\)](#) and storage requirements. In a system classifying ten different tasks, the [MTL](#)-based [ESN](#) implementation required fewer [FLOPs](#) than a single convolutional layer used for [MNIST](#) classification, while achieving superior efficiency compared to MobileNetV2 and MCU-Net.

Overall, this PhD research demonstrates that both [tIM](#) and shared [ESN](#) reservoirs offer practical solutions for deploying multiple [DL](#) tasks on low-power resource-constrained devices. The proposed methods enable more efficient model deployment, reducing storage and computation costs while maintaining high inference performance, making them promising candidates for future and [tinyML](#) applications.

Acknowledgements

I would like to thank my supervisor, Prof. Eiman Kanjo. I am extremely grateful for giving a young undergraduate the chance to step up to a PhD. I imagine it has not always been easy since to begin with I was somewhat clueless as to what I should be doing. But without that chance, I would not have been able to grow into the researcher I am today. She has provided me with financial and professional support and opportunities. Her expertise and connections to industry have also meant that I have had the opportunity to meet some amazing people within the tinyML research community, which I am thankful for.

I would also like to thank the rest of my supervisory team for their advice and input into the PhD, particularly Dr. Pedro Machado. Dr. Machado made a habit of asking difficult thought-provoking questions, which I cannot say I always enjoyed, but they have been invaluable in completing this PhD. His support has been constant and has not gone unappreciated.

This PhD journey has been challenging, testing my patience, resolve and drive but I could not have done it without the help of my unofficial supervisor, Dr. Kieran Woodward. He has listened to every idea that I have had, both good and bad and his expertise has given me confidence in my research. Dr. Woodward has led by example and I could not be more thankful for the help he has provided over the years.

A PhD is a long journey which has been made enjoyable by friends and family as well as colleagues. I would first like to thank the rest of the Smart Sensing Lab for making work more enjoyable. You have

all contributed to this PhD through our hundreds of lunchtime conversations.

My best friends, Bethany and Jack, thank you for making me laugh most nights when I come home. Without that outlet, I am unsure how far I would have come in this journey. I especially thank Bethany for her unique perspective on my work. She helped my research communication, as well as provide advice based on her own research background. The same could be said for my girlfriend Amy. She has been more supportive than I could ask.

I owe my achievements to my family. My mother, Amanda, taught me strength and perseverance, while my dad, Anthony, taught me patience and reasoning. Words cannot express how indebted I am to you both. Thank you for shaping me into who I am. Thank you also to my sisters Vicki and Natalie, who have been a pillar of support I can rely on for as long as I can remember.

Finally, I would like to thank extended friends and family who have helped me through these years. Your impact on me does not go unnoticed.

The copyright in this work is held by the author. You may copy up to 5% of this work for private study, or personal, non-commercial research. Any re-use of the information contained within this document should be fully referenced, quoting the author, title, university, degree level and pagination. Queries or requests for any other use, or if a more substantial copy is required, should be directed to the author.

Contents

| | |
|--|------------|
| Abstract | ii |
| Acknowledgments | iv |
| Contents | vii |
| List of Figures | x |
| List of Tables | xiv |
| 1 Introduction | 1 |
| 1.1 Research Gap | 3 |
| 1.2 Research Aim | 4 |
| 1.3 Research Contribution | 5 |
| 1.4 Thesis Outline | 6 |
| 1.5 List of Publications | 7 |
| 2 Literature Review | 9 |
| 2.1 An Introduction to Data | 10 |
| 2.2 Chaotic Systems | 11 |
| 2.3 Artificial Intelligence and Machine Learning | 12 |
| 2.4 Deep Learning | 13 |
| 2.5 TinyML, its Aliases and Variations | 14 |
| 2.6 Constraints of tinyML and Edge Intelligence | 17 |
| 2.6.1 Latency | 18 |
| 2.6.2 Power Consumption | 18 |

| | | |
|----------|--|-----------|
| 2.6.3 | Resource Constraints | 19 |
| 2.7 | Model Reduction Techniques | 20 |
| 2.7.1 | Quantisation | 20 |
| 2.7.2 | Pruning | 21 |
| 2.7.3 | Knowledge Distillation | 22 |
| 2.8 | Context-Aware Solutions for MCUs | 23 |
| 2.9 | Multi-Task Learning and Multi-Domain Learning | 26 |
| 2.10 | Real-World Applications of tinyML | 29 |
| 2.11 | Echo State Networks: A Cognitive Approach | 32 |
| 2.11.1 | ESN Hyperparameters | 34 |
| 2.11.2 | ESNs on a Smaller Scale | 35 |
| 2.12 | Challenges and Opportunities | 36 |
| 2.12.1 | Edge Intelligence and tinyML Deployment | 36 |
| 2.12.2 | Multi-Task Optimisation for Microcontrollers | 41 |
| 2.12.3 | Echo State Network Optimisation | 43 |
| 2.13 | Discussion | 43 |
| 3 | Enabling Multimodal Multi-<i>model</i> Context-Aware tinyML Algorithms for Low-Power Hardware | 46 |
| 3.1 | Introduction | 47 |
| 3.2 | Methodology | 48 |
| 3.2.1 | Datasets | 51 |
| 3.2.2 | Hardware | 53 |
| 3.2.3 | Classification Features | 54 |
| 3.2.4 | Network Architecture | 55 |
| 3.2.5 | Model Quantisation | 56 |
| 3.3 | Results | 57 |
| 3.4 | Discussion | 59 |
| 4 | A Novel Layer Sharing Approach for Multi-Task Learning on tinyML-Enabled Hardware | 63 |
| 4.1 | Introduction | 64 |
| 4.2 | Motivation | 66 |

CONTENTS

| | | |
|----------|--|------------|
| 4.3 | Layer Transformation | 67 |
| 4.4 | Architecture and Setup | 68 |
| 4.5 | Case Study: Stress Detection with Human Activity Recognition | 69 |
| 4.5.1 | Human Activity Recognition Model | 70 |
| 4.5.2 | Stress Model | 72 |
| 4.5.3 | Model Inception | 72 |
| 4.6 | Results Analysis | 74 |
| 5 | Multi-Task Reservoir Sharing Using Echo State Networks | 81 |
| 5.1 | Introduction | 82 |
| 5.2 | Complexity Analysis | 84 |
| 5.3 | Methodology | 88 |
| 5.4 | Results Analysis | 94 |
| 6 | Conclusion and Future Work | 102 |
| 6.1 | Conclusion | 102 |
| 6.2 | Research Contributions | 106 |
| 6.3 | Future Work | 107 |
| | References | 110 |
| | Appendix A. Additional Echo State Network Plots | 150 |

List of Figures

| | | |
|-----|---|----|
| 2.1 | Diagram showing Artificial Intelligence (AI) research and its applications as subsets with tinyML being a subset of both Machine Learning (ML) and DL | 13 |
| 2.2 | Figure depicting three key computing paradigms, the edge, fog and cloud computing paradigms. Processing is completed in the edge layer, hence reducing the burden on the cloud layer. The fog layer can act as an intermediary layer to aid in the processing from a semi-decentralised but likely static location. | 15 |
| 2.3 | Weight pruning of Deep Neural Network (DNN) weights. | 23 |
| 2.4 | Three types of Multi-Domain Learning (MDL) architectures where the gold denotes shared parameters between different input domains. (a) A simple MTL hard parameter sharing architecture where all input domains flow into domain-agnostic layers. (b) An opposing architecture to (a) where each input domain extract domain-specific features to solve a single task. (c) Adapter-based MDL where domain-agnostic fixed parameters act as feature extractors across domains. | 28 |
| 2.5 | A typical ESN with the reservoir input vector \mathbf{W}_{in} feeding into the sparsely connected reservoir \mathbf{W} , which eventually yields an N -dimensional vector output \mathbf{W}_{out} . Typically \mathbf{W}_{out} is a <i>readout</i> layer, such as a ridge regressor. | 33 |
| 2.6 | Figure demonstrating preprocessing blocks can be too much for an MCU device without careful consideration. | 40 |

| | | |
|-----|--|----|
| 3.1 | The proposed system containing the HAR model (top) and stress model (bottom), along with the internal logic. The HAR model constantly infers exercise or not exercise. When exercise is not detected, the stress model is invoked. Data is continuously collected for both models. | 50 |
| 3.2 | Person wearing the Heart Rate (HR) and ElectroDermal Activity (EDA) sensors connected to an Arduino Nano 33 BLE Sense | 54 |
| 3.3 | Graph of performance metrics over varying quantisation policies of the (a) HAR model and (b) the stress model. Latency was unavailable for <i>float32</i> and <i>float16</i> quantisation for the stress model due to its large model size. | 59 |
| 4.1 | Architecture of the transformation adapter layers mapping a three-dimensional input to another three-dimensional output. The transformation block contains only layers compatible with Lite Runtime for Microcontrollers (LiteRTμ) . Each of the layers contains its output shape and the number of parameters said layer incurs (if applicable). | 67 |
| 4.2 | A general overview of the tIM . The layers in blue are model specific, whereas the gold layers are shared. The transformation adapter layers, although blue, are part of the tIM as they are required by the specific model for transformation into the gold layers (or vice versa for the right-hand transformation adapter layers). The output shapes are listed within layers where they are known in a generalised form. | 69 |
| 4.3 | Both model architectures for the HAR model and stress recognition model before any tIM optimisations have been applied. Each node has its layer name, output shape and number of parameters. The total parameters and layers are recorded in the lower left table. | 71 |
| 4.4 | The newly adapted HAR and stress models with a tIM . Both models use the gold layers reducing storage burden. The incurred parameters for each layer are included as well as their output shapes. | 73 |

LIST OF FIGURES

| | | |
|-----|---|-----|
| 4.5 | Graphs of the loss for all the models (top) and the accuracies for all the models (bottom) for each epoch of training (with a callback patience of 10 epochs). | 78 |
| 5.1 | (a) The inspired MTL / MDL structure with several input domains, some single shared layers and several output domains. (b) The traditional ESN structure. (c) The resulting proposed methodology for the MTL -based ESN which assumes M models, each with an input weight matrix $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N \times K_m}$ where m indexes the m th model in M , N represents the number of reservoir units, and K_m denotes the input dimensionality of the m th model. All blue and gold components in (a), (b) and (c) denote individual and shared components, respectively. | 89 |
| 5.2 | Plot of the number of parameters as a function depending on the reservoir size N and the input vector size K . In this plot, the sparsity has not been applied to the total parameters. The red lines denote the values of K used in this chapter, i.e., 1, 3, 60, 600, 784, and 3072. | 91 |
| 5.3 | Visual representation of all pre-existing graph-based initialisation techniques containing 50 nodes; namely, Barabasi-Albert (BA) (left), Newman-Watts-Strogatz (NWS) (centre), and Erdős–Rényi (right) graphs. | 92 |
| 5.4 | Graph of the average accuracy for each reservoir separated by the number of neurons in the reservoir for all classification datasets listed in Table 5.1. All datasets have a single standard deviation of error highlighted to the corresponding dataset. | 94 |
| 5.5 | For each of the regression-based datasets, the average Mean Squared Error (MSE) (blue) as well as the coefficient of determination, or R^2 (gold) on a \log_2 axis denoting the number of neurons in the reservoir. | 96 |
| 1 | Plot showing the accuracy against the log-scaled number of neurons given varying sparsity levels for all classification datasets. . . | 151 |

LIST OF FIGURES

- 2 Plot showing the accuracy against the log-scaled number of neurons given varying initialisation methods for all classification datasets. 152
- 3 Plot showing the log-scaled MSE against the log-scaled number of neurons given varying sparsity levels for all regression-based datasets. 153
- 4 Plot showing the log-scaled Mean Squared Error (MSE) against the log-scaled number of neurons given varying initialisation methods for all regression-based datasets. 154

List of Tables

| | | |
|-----|---|----|
| 2.1 | Table demonstrating some of the limits of double precision (<i>float64</i>), single precision (<i>float32</i>), half precision (<i>float16</i>) and 8 bit integer (<i>int8</i>) precision. Decimal precision denotes the number of decimal places the datatype can accurately place using $\log_2 \text{significand_bits} + 1$ rounded down to an integer. | 21 |
| 2.2 | Table illustrating the available built-in sensors for a range of development boards and highlighting the boards supported by LiteRTμ (formerly TensorFlowLite for Microcontrollers (TFLiteμ)). | 39 |
| 3.1 | Performance metrics for the stress (top) and physical activity (bottom) classification models. For each, the precision, recall, F1 score and accuracy is collected as well as the latency and header file size. | 58 |
| 4.1 | Table presenting the metrics of all the models trained, including variations of the stress model. Stress Model A uses the tIM from the HAR model, whereas Stress Model B is the model with no layer sharing optimisation. | 75 |
| 4.2 | Table comparing the Stress Model A (with tIM) and Stress Model B (without tIM) with the <i>int8</i> quantised stress model from Chapter 3 [1]. However, both of the models in this work are quantised to <i>float16</i> | 76 |
| 4.3 | Table presenting the model sizes between the final stress and HAR models in this article and in Chapter 3 [1] compared to the overall system. | 76 |

| | | |
|-----|--|-----|
| 5.1 | All the datasets used within this chapter including their domain, input shape, and whether it is a classification or regression-based dataset. | 90 |
| 5.2 | A look-up table for the number of parameters and FLOPs (in thousands) for each ESN model varying with the number of neurons and the sparsity level. N denotes the reservoir size and K denotes the one-dimensional input shape. These calculations exclude the FLOPs required to compute the <i>tanh</i> activation function. The corresponding datasets for K are as follows: 1 = Mackey-Glass (MG) ; Non-linear Autoregressive Moving Average (NARMA) , 3 = Lorenz, Rössler ; 60 = Wireless Sensor Data Mining (WISDM) , UCI HAR ; 600 = Stress ; 784 = Modified National Institute of Standards and Technology (MNIST) , MNIST Fashion ; 3072 = CIFAR10 | 93 |
| 5.3 | Table highlighting the significance levels of Multivariate Analysis of Variance (MANOVA) tests for the initialisation strategy, spectral radius and sparsity level. The green indicates a statistically significant value i.e. $P < 0.05$, while red indicates insufficient evidence of significance $P \geq 0.05$. MNIST Fashion and CIFAR10 are not available due to multicollinearity. | 95 |
| 5.4 | For all datasets listed on the left, and their input sizes, K , the total shared parameters when using varying reservoir sizes is demonstrated as well as the total dataset-specific parameters. These dataset-specific parameters are derived from the input weight matrix $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N \times K}$ and the ridge classifier/regressor $\beta \in \mathbb{R}^N$. All values in the table are assumed to be 5% sparsity, i.e., $s = 0.05$. The total dataset-specific parameters to run all 10 classification/regression tasks is included as well as the total including the sparse shared reservoir. | 100 |

Acronyms

AI Artificial Intelligence. [x](#), [1](#), [2](#), [9](#), [12–20](#), [36–38](#), [44](#), [45](#), [47](#), [48](#), [53](#), [82](#), [104](#), [107](#)

AUC Area Under Curve. [74](#)

BA Barabasi-Albert. [xii](#), [34](#), [91](#), [92](#)

BLE Bluetooth Low Energy. [ii](#), [xi](#), [5](#), [6](#), [19](#), [31](#), [38](#), [40](#), [49](#), [53–57](#), [59](#), [62](#), [69](#), [102](#), [103](#), [106](#), [108](#)

BNN Binary Neural Network. [21](#)

BPM Beats Per Minute. [52](#), [54](#), [60](#)

CIFAR10 Canadian Institute for Advanced Research, 10 Classes. [xv](#), [22](#), [23](#), [29](#), [77](#), [90](#), [93](#), [95](#), [99](#)

CNN Convolutional Neural Network. [2](#), [4](#), [6](#), [22](#), [30–32](#), [36](#), [49](#), [55](#), [60](#), [61](#), [63](#), [67](#), [70](#), [86–88](#), [90](#), [98–100](#), [104](#), [105](#), [107](#)

DL Deep Learning. [ii](#), [iii](#), [x](#), [1](#), [2](#), [4–7](#), [9](#), [11](#), [13](#), [14](#), [21–23](#), [30](#), [32](#), [36](#), [41](#), [45](#), [47](#), [54](#), [55](#), [62](#), [64–66](#), [70](#), [79](#), [81](#), [82](#), [102](#), [103](#), [105–107](#)

DNN Deep Neural Network. [x](#), [2–4](#), [6](#), [13](#), [14](#), [23](#), [35](#), [42](#), [43](#), [46](#), [47](#), [55](#), [86–88](#), [90](#), [98–100](#), [104](#), [105](#), [107](#)

EDA ElectroDermal Activity. [xi](#), [24](#), [26](#), [31](#), [32](#), [44](#), [46](#), [49](#), [52](#), [54](#), [60](#), [72](#)

EEG Electroencephalogram. [26](#)

- EI** Edge Intelligence. [7](#), [16](#), [30](#), [36](#), [43](#), [45](#), [81](#)
- ESN** Echo State Network. [ii](#), [iii](#), [x](#), [xii](#), [xv](#), [6](#), [7](#), [9](#), [32–36](#), [43](#), [80–84](#), [87–94](#), [97–102](#), [104](#), [105](#), [107](#), [108](#)
- ESP** Echo State Property. [33](#), [34](#), [96](#)
- FLOP** Floating Point Operation. [iii](#), [xv](#), [6](#), [22](#), [81](#), [83](#), [84](#), [86](#), [87](#), [89](#), [90](#), [93](#), [97](#), [98](#), [100](#), [101](#), [104](#), [105](#), [107](#), [109](#)
- FOMO** Faster Objects More Objects. [29](#)
- FPGA** Field Programmable Gate Array. [21](#)
- FT** Fourier Transform. [3](#), [38](#), [39](#)
- GRU** Gated Recurrent Unit. [70](#)
- HAR** Human Activity Recognition. [ii](#), [xi](#), [xiv](#), [xv](#), [5–7](#), [10](#), [27](#), [30](#), [31](#), [40](#), [45–49](#), [51](#), [54–63](#), [65](#), [69–77](#), [79](#), [90](#), [93](#), [102–104](#), [106–108](#)
- HR** Heart Rate. [xi](#), [31](#), [32](#), [44](#), [47](#), [49](#), [52–54](#), [60](#), [72](#)
- HRV** Heart Rate Variability. [26](#), [32](#), [52–54](#), [60](#), [72](#)
- IMU** Inertial Measurement Unit. [10](#), [31](#), [49](#), [51](#), [54](#), [55](#), [61](#)
- IoT** Internet of Things. [2](#), [16](#), [45](#), [79](#)
- k-NN** k Nearest Neighbour. [32](#)
- LiteRT** Lite Runtime. [37](#), [74](#), [75](#)
- LiteRT μ** Lite Runtime for Microcontrollers. [xi](#), [xiv](#), [37–39](#), [53](#), [55](#), [56](#), [67](#), [68](#), [71](#), [72](#), [105](#)
- LSTM** Long Short-Term Memory. [14](#), [30](#), [32](#), [38](#), [55](#), [60](#), [70](#)
- MAE** Mean Absolute Error. [94](#)

- MANOVA** Multivariate Analysis of Variance. [xv](#), [95](#)
- MCU** Microcontroller Unit. [ii](#), [viii](#), [x](#), [2–6](#), [16](#), [17](#), [19](#), [22](#), [23](#), [27](#), [29](#), [31](#), [36–38](#), [40–46](#), [48](#), [49](#), [51](#), [53–57](#), [59–67](#), [76](#), [79](#), [80](#), [82](#), [100–108](#)
- MDL** Multi-Domain Learning. [x](#), [xii](#), [9](#), [26–28](#), [42](#), [44](#), [66](#), [67](#), [88](#), [89](#)
- MG** Mackey-Glass. [xv](#), [11](#), [12](#), [35](#), [83](#), [89](#), [90](#), [93](#), [99](#)
- ML** Machine Learning. [x](#), [3](#), [9](#), [12](#), [13](#), [16](#), [19](#), [20](#), [24](#), [30–32](#), [36–39](#), [41](#), [44](#), [53](#), [55](#), [56](#), [61](#), [64](#), [65](#), [70](#), [79](#)
- MLP** Multi-Layer Perceptron. [31](#)
- MNIST** Modified National Institute of Standards and Technology. [iii](#), [xv](#), [13](#), [29](#), [33–35](#), [83](#), [90](#), [93](#), [95](#), [97–99](#), [104](#)
- MSE** Mean Squared Error. [xii](#), [xiii](#), [94–96](#), [153](#), [154](#)
- MTL** Multi-Task Learning. [ii](#), [iii](#), [x](#), [xii](#), [6](#), [7](#), [9](#), [26–29](#), [41](#), [42](#), [44](#), [62–64](#), [66](#), [67](#), [81–84](#), [88](#), [89](#), [94](#), [97–101](#), [104](#), [105](#), [107](#), [108](#)
- NARMA** Non-linear Autoregressive Moving Average. [xv](#), [89](#), [90](#), [93](#), [96](#), [99](#)
- NAS** Neural Architecture Search. [21](#), [42](#)
- NWS** Newman-Watts-Strogatz. [xii](#), [34](#), [92](#)
- PAMAP2** Physical Activity Monitoring for Aging People 2. [70](#)
- PPG** Photoplethysmography. [47](#), [53](#), [54](#), [103](#)
- PReLU** Parametric Rectified Linear Unit. [21](#)
- PTQ** Post Training Quantisation. [20](#), [56](#), [75](#)
- QAT** Quantisation Aware Training. [20](#)
- RAM** Random Access Memory. [3](#), [19](#)

ReLU Rectified Linear Unit. [14](#), [21](#), [71](#)

RNN Recurrent Neural Network. [25](#), [30](#), [32](#)

ROC Receiver Operating Characteristic. [74](#)

rs-fMRI resting state-functional Magnetic Resonance Images. [26](#)

SRAM Static Random Access Memory. [3](#), [53](#), [108](#)

TFLite TensorFlow Lite. [74](#), [75](#)

TFLite μ TensorFlowLite for Microcontrollers. [xiv](#), [37](#), [39](#), [53](#)

tIM tiny Inception Module. [ii](#), [iii](#), [xi](#), [xiv](#), [5](#), [7](#), [63–66](#), [68](#), [69](#), [71–80](#), [103](#), [104](#), [106](#), [107](#)

tinyML tiny Machine Learning. [ii](#), [iii](#), [x](#), [2–7](#), [9](#), [13](#), [14](#), [16](#), [17](#), [19](#), [20](#), [22](#), [29](#), [35–38](#), [42–46](#), [48](#), [55](#), [61](#), [63](#), [64](#), [74](#), [79](#), [81](#), [83](#), [103](#), [105](#), [107](#)

VGGNet Visual Geometry Group Network. [22](#)

WISDM Wireless Sensor Data Mining. [xv](#), [51](#), [61](#), [70](#), [83](#), [90](#), [93](#), [95](#)

YOLO You Only Look Once. [29](#)

Chapter 1

Introduction

Due to the COVID-19 pandemic, the world was forced to embrace the cloud computing paradigm. The cloud computing paradigm has aided human-computer interaction by increasing the accessibility of computational resources, user files, and the breadth of applications. However, cloud data centres powering cloud computing systems are expensive both in hardware and in power consumption, with the power consumption of data centres predicted to increase from 200 TWh in 2016 to 2967 TWh in 2030 [2]. Furthermore, in 2023 the amount of electricity consumed by data centres in Ireland was more than that consumed by all urban dwellings combined [3]. Such energy consumption has an environmental and economic impact. For example, Google currently owns and operates data centres in 26 locations with 7 additional locations in development, totalling 33 locations [4], which in 2021 cost Google billions of dollars in investments alone [5]. Governments are also interested in having state owned data centres with the UK government classifying data centres as critical national infrastructure [6]. Reducing reliance on cloud-based systems will lessen the dependence on data centres, thereby lessening their significant environmental and economic impact. To reduce reliance on cloud data centres, a paradigm shift is required. The edge computing paradigm enables the processing of data close to the data source. Thus, the edge computing paradigm can alleviate the burden of processing on cloud data centres. However, edge devices tend to be resource-constrained, limiting the execution of complex [Artificial Intelligence \(AI\)](#) models.

[AI](#) algorithms, particularly gradient-based learning and [Deep Learning \(DL\)](#)

approaches such as [Deep Neural Networks \(DNNs\)](#) and [Convolutional Neural Networks \(CNNs\)](#), are typically large and computationally intensive, requiring substantial resources for both training and inference. As a result, running [AI](#) models traditionally requires powerful cloud-based infrastructure or high-performance computing clusters. This reliance on centralised processing introduced challenges related to latency, bandwidth constraints, and data privacy, particularly in applications that require real-time decision-making or operating in bandwidth-limited environments.

Historically, edge-based hardware lacked computational capabilities to execute complex [AI](#) models directly on-device. Instead, [AI](#) inference at the edge often relied on edge servers or gateways to offload processing from resource-constrained devices, thus reducing the computational burden on embedded systems. This paradigm, known as edge [AI](#) [7], enabled certain [AI](#) functionalities closer to the data source, but still involved communication overhead with intermediate devices.

However, recent advances in [Microcontroller Unit \(MCU\)](#) hardware and efficient [AI](#) model compression techniques have made it possible to execute [AI](#) algorithms directly on low-power, resource-constrained embedded systems. This emerging field, known as [tiny Machine Learning \(tinyML\)](#) [8], represents a significant shift towards on-device [AI](#) processing, where inference can be performed locally with minimal energy consumption. [tinyML](#) enables [AI](#)-driven applications to function with greater privacy and lower latency, as sensitive data no longer need to be transmitted to external servers for processing. These advancements open new possibilities for real-time [AI](#) in wearables, [Internet of Things \(IoT\)](#) devices, and autonomous embedded systems, where energy efficiency and privacy are essential.

Leveraging the advances in edge hardware based on [AI](#), this thesis develops a clear use case to host multiple [DL](#) models on a resource-constrained [MCU](#) device. Subsequently, software optimisations are developed to run several [AI](#) models more effectively than before. The following section highlights the research gap. The rest of this chapter will outline the main research aim and research questions, followed by the research contributions, and thesis outline.

1.1 Research Gap

As hardware capabilities increase, software is able to accommodate more complex algorithms, such as [Fourier Transform \(FT\)](#), and models such as [DNNs](#). By hosting more models, the device can become multi-purpose; for example, data collected from inertial sensors can be processed directly by a smartphone to detect if a user has fallen [\[9\]](#) while also performing biometric recognition [\[10\]](#). Although these models do not run simultaneously, they contribute to the functionality and security of the device, respectively. As smartphones gain more computational resources, the practicality of deploying such models becomes simpler. However, when confronted with smaller wearable devices or resource-constrained low-power (5V) [MCU](#) devices, there is little practical deployment or specific optimisations. A smartphone has gigabytes of [RAM](#) and a processor capable of processing gigahertz. In contrast, [MCU](#) devices have around one megabyte of [SRAM](#) and an [MCU](#) capable of processing megahertz. These differences require different software optimisation strategies, as resources are scarce. Although [tinyML](#) optimisations exist [\[8\]](#), they often fail to consider multiple models on the same device, as it can be challenging enough to deploy a single model to a low-power device.

Software optimisations for resource-constrained low-power [MCU](#) devices can lack replicability due to the specific nature of the target device or dataset. Such devices cannot host an operating system, which can make designing algorithms or optimisations difficult to generalise for many devices. Nonetheless, advancing software strategies in this space has the potential to impact a wide range of domains, including industrial condition monitoring, environmental sensing, precision agriculture, and, notably, smart health. These sectors share a growing reliance on intelligent edge devices that operate independently of the cloud, where latency, bandwidth, and privacy are critical concerns.

Within this landscape, smart health emerges as a particularly compelling and high-impact domain [\[11\]](#). Additional research within [tinyML](#) and increasing model capabilities through software optimisations could mean improved health-care systems. While current systems may exist, most are not intelligent at the point of data collection, which is a key consideration for private health data [\[12\]](#). Furthermore, with the improvement and deployment of several [ML](#) models, it is

possible to gain ubiquitous early prevention technologies through wearable devices. The following section will outline the research aim and three research questions aiming to contribute to the aforementioned research gap.

1.2 Research Aim

Now that the research gap has been identified, a research aim and research questions can be posed. Therefore, this thesis is structured into a single overarching research aim and three related research questions. The research questions are answered by each of the key contribution chapters in this thesis, namely Chapters 3, 4, and 5.

Research Aim

To effectively share information between several Deep Learning models to minimise the number of parameters required to perform multiple tasks independently on low-power Microcontroller Unit devices.

Research Questions

- RQ 1.** How can multiple Deep Learning models be integrated to enhance on-device inference capabilities for multimodal sensor inputs on a wearable Microcontroller Unit device?
- RQ 2.** How can multiple Deep Learning models effectively share information to enable multi-*model* tiny Machine Learning applications?
- RQ 3.** How efficient are sparse shared reservoirs in handling multiple tasks, particularly compared to Deep Neural Networks and Convolutional Neural Networks?

To answer these research questions, multiple models are initially deployed to a single device, before then building on this idea. A new layer sharing architecture is proposed that alleviates some of the drawbacks of deploying multiple models to a single resource-constrained **MCU** device. Finally, a new method based on

reservoir computing to share information is postulated for deployment to these same constrained systems.

1.3 Research Contribution

This thesis can be organised into three key contributions that enhance the field of *tinyML* optimisation for the deployment of multiple models. These contributions can be seen in more detail below.

Contribution 1: Enabling Multimodal Multi-*Model* Context-Aware *tinyML* Algorithms for Low-Power Hardware

Two *DL* models have been constructed for the purpose of deployment onto a single *MCU* device. The *Human Activity Recognition (HAR)* model provides contextual information on whether the stress model should run inference. The novel combination of these two models provides contextual information in real time to improve the accuracy of the second model. The context-aware multimodal multi-*model* approach was subsequently deployed to a resource-constrained Arduino Nano 33 *BLE Sense*¹. Deployment is often missing from literature and has significant practical challenges, such as library and sensor compatibility, which have been addressed.

Contribution 2: A Novel Layer Sharing Architecture for Multi-*Model* *tinyML* Deployment

The novel *tiny Inception Module (tIM)* was introduced for the sharing of network layers between *DL* models. Using a new transformation adapter layer to enable the repurposing of layers for additional tasks, while also allowing models of differing architectures to become entangled. The result yields several models with a subset of shared layers between them. To demonstrate the effectiveness of the *tIM*, a case study on stress recognition and *HAR* was carried out drawing a direct

¹Available online, <https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf>, last accessed 03/03/2025

comparison with the previous chapter. Results showed that, although the HAR model size increased, the whole system had a significant reduction in model size while maintaining accuracies comparable to the literature.

Contribution 3: Evaluating Echo State Network Reservoir Sharing for tinyML Applications

A novel Multi-Task Learning (MTL)-based Echo State Network (ESN) reservoir sharing approach was proposed. Under the MTL-based ESN approach, several tasks reuse the same reservoir but have task-specific input matrices and *readout* layers. The proposed MTL-based ESN was evaluated on 10 datasets covering chaotic systems, time series, and image data. A single hyperparameter configuration was applied across all datasets, varying the number of neurons, spectral radius, sparsity, and initialisation method. Results showed that when a single reservoir is shared, most models still learn appropriate representations of the propagated input, and in several cases, competitive accuracies. Complexity analyses of the MTL-based ESN, a dense layer and a convolutional layer, have been outlined. These analyses were compared to the results of the tested datasets. Results showed that fewer parameters are needed for all 10 datasets compared to popular lightweight DL models for classifying a single task. In addition, fewer Floating Point Operations (FLOPs) are needed for inference in the entire MTL-based ESN system compared to comparative CNN and DNN architectures.

1.4 Thesis Outline

This thesis is organised into the following chapters:

Chapter 2 provides a literature review exploring tinyML, its constraints and applications (particularly using context), as well as model reduction techniques, MTL and ESNs. Chapter 2 then identifies the research challenges and opportunities associated with the aforementioned research topics.

Chapter 3 used a classification model to provide HAR context to a second stress recognition model on the edge using an Arduino Nano 33 BLE Sense, a low-power and low-cost MCU device.

Chapter 4 introduces the **tIM**, a **DL** layer sharing approach specifically designed for **tinyML** systems. The novel transformation adapter layer allows pre-existing network components to be stitched into another model for training. The **tIM** is then evaluated similarly to the previous chapter by using a case study of **HAR** and stress detection.

Chapter 5 proposes a novel **MTL**-based **ESN** reservoir sharing approach for **Edge Intelligence (EI)** and **tinyML** systems. The proposed approach is based on a shared reservoir architecture, which can be used to classify multiple datasets. The approach has been evaluated using a set of benchmarking datasets and real-world data.

Chapter 6 concludes the work presented in this thesis by evaluating the research questions and the main research aim, then subsequently providing potential future research directions.

1.5 List of Publications

During my PhD studies, valuable feedback was gained in the form of peer reviews that led to the publication of some work presented in this thesis. In particular, part of Chapter 2 is based on an *IEEE EDGE* conference paper [13], Chapter 3 is based on a journal paper published in *IEEE Micro* [1], and a journal article based on Chapter 4 is under review. A list of these publications can be seen below.

- Michael Gibbs, Kieran Woodward, and Eiman Kanjo. Combining Multiple tiny Machine Learning Models for Multimodal Context-Aware Stress Recognition on Constrained Microcontrollers. *IEEE Micro*, 44(3):67–75, 5 2024. [1]
- Michael Gibbs and Eiman Kanjo. Realising the Power of Edge Intelligence: Addressing the Challenges in AI and tinyML Applications for Edge Computing. In Proceedings - *IEEE International Conference on Edge Computing*, volume 2023-July, 2023. [13]
- Michael Gibbs, Kieran Woodward, Eiman Kanjo, Pedro Machado and Amir Pourabdollah. tIM: A tiny Inception Module for Layer Sharing on Constrained Devices (under review).

Chapter 2

Literature Review

Chapter Overview

The previous chapter introduced the overall scope and objectives of this thesis. This chapter provides the necessary background and literature review to contextualise the research. It begins with fundamental concepts such as data, chaotic systems, [Artificial Intelligence \(AI\)](#), [Machine Learning \(ML\)](#), and [Deep Learning \(DL\)](#), then introduces [tiny Machine Learning \(tinyML\)](#). Next, the chapter examines key challenges associated with [tinyML](#), such as computational and memory constraints, and discusses various model reduction techniques designed to mitigate these limitations. It then explores advanced topics, including context-aware solutions, [Multi-Task Learning \(MTL\)](#), and [Multi-Domain Learning \(MDL\)](#), highlighting their relevance to resource-efficient [AI](#). This chapter also discusses real-world applications of [tinyML](#) before introducing [Echo State Networks \(ESNs\)](#), which play a central role in this thesis. Finally, challenges and opportunities are discussed, identifying open research areas for advancements to improve [AI](#) deployment on resource-constrained devices.

2.1 An Introduction to Data

Data is key to analysing and understanding trends. Although data span a vast cross section of research, it is important to understand the basis of what is to come within this thesis. Data can come from several modalities, one of which is time-series. Time-series data can help make informed decisions in a timely manner and provide support to those who need it. Hence, time-series data is sought after because of its direct applicability, particularly in smart health and engineering. However, data can be invariant in time, such as spatial data. Spatial data represents visual information captured by sensors and can be considered time-invariant when the data points remain the same regardless of when they are captured. As a result, the characteristics of the data do not change over time, unless external factors alter them. However, spatial-temporal data processing requires preserving both spatial and temporal components, which can make it more complex and resource-intensive to collect and process than spatial and temporal data. The challenges of handling such data are further compounded by issues of data quality. Poor-quality data, if used for training models, can lead to the phenomenon colloquially referred to as “garbage in garbage out” [14, 15], where flawed inputs inevitably produce flawed outputs. To mitigate this, reducing human influence during data collection can help minimise errors and improve overall data reliability. Although, this does not completely eliminate the error. In addition, some modalities cannot be collected without the use of sensors. These sensors can also vary in performance because higher quality sensors tend to be more resistant to noise. For example, [Human Activity Recognition \(HAR\)](#) data should frequently record movement data collected from an [Inertial Measurement Unit \(IMU\)](#) sensor, at a frequency rate higher than that of human ability, that is, several samples per second. Recognising different human activities, it is possible to determine specific recommendations for habit modification over the long term, or for immediate interventions, there could be the recommendation to stand after prolonged inactivity is detected. Alternatively, detecting a fall could have multiple implications, which could potentially provide life-saving intervention faster than cloud-based solutions. Particularly as falls disproportionately affect the elderly, with approximately 28–35% of people 65 years and older falling each year,

and this number increases as people age [16]. Falls account for approximately 1.6 to 3.0 hospital admissions for people over 60 years of age per year per 10,000 people. These falls can be fatal, with 36.8 deaths per 100,000 falls in the United States [16].

In addition to falls, stress is an emotion that can have a negative impact on people’s lives. Although short-term stress (several minutes or hours) can enhance immune responses by preparing the body for challenges, long-term stress (several hours per day for weeks or months) is generally harmful, as it suppresses immune function, can cause chronic inflammation, and increases susceptibility to illnesses [17]. Therefore, knowing when and how long you have been stressed could provide insight into treatment or prompt a change of routine.

DL is a powerful tool for leveraging the potential of time-series and image data. With its ability to model complex patterns and relationships, DL can enhance the accuracy of time-series analyses and image classifications [18]. Using large datasets, DL models are able to learn intricate features that traditional methods might overlook. In subsequent sections, the specifics of DL are explored, demonstrating its transformative potential to improve predictions, decision-making, and ultimately the quality of insights derived from diverse data sources.

2.2 Chaotic Systems

Data can simulate complex systems of equations in real-world applications. Chaotic systems, in particular, are among the most difficult to predict. This section will introduce chaotic systems and the importance of being able to predict the next step in a chaotic sequence. Chaotic systems themselves are highly sensitive to initial conditions, where small differences in the starting point can lead to vastly different outcomes. They are deterministic in nature, but appear random and unpredictable. By predicting the next step in a chaotic sequence, one can gain insight into how they may behave in advance. Weather forecasting is a well-known example of a chaotic system with vast global implications. Although the deployment of weather forecasting is unlikely on constrained systems, lighter-weight chaotic systems such as the Mackey-Glass (MG) equation could be more

easily deployed.

The MG equation is a chaotic system that can model biological processes such as blood production and is as follows

$$\frac{dx(t)}{dt} = \beta \frac{x(t-\tau)}{1+x(t-\tau)^n} - \gamma x(t), \quad (2.1)$$

where $x(t)$ denotes the state variable at time t . β and γ denote the production and decay rate, respectively, τ is the time delay and n is a parameter to control nonlinearity within the denominator [19, 20]. In the literature, it is common to use the constants $\beta = 0.2$, $\gamma = 0.1$, $\tau = 23$, and $n = 10$ as outlined in [21]. These parameters are often selected to model an attractor, a set of conditions that converge towards a stable *attractive* state.

Alternative attractors, such as the Lorenz and Rössler attractors, represent dynamical systems that are key benchmarks for evaluating the capacity of models to capture and predict chaotic behaviours. These systems, characterised by their non-linear dynamics, provide a foundation for comparative analysis of advanced modelling techniques [22, 23]. A detailed application of chaotic systems and their role in modelling dynamical behaviour is presented later in Section 2.11.

2.3 Artificial Intelligence and Machine Learning

While chaotic systems and attractors offer insights into the modelling of dynamical behaviours, advances in AI and ML have opened up novel ways of analysing and leveraging such systems, particularly in areas like smart health [24, 25] and recommendation systems [26]. Although AI broadly refers to systems capable of tasks that require human intelligence, ML is focused on algorithms that learn patterns from data [27]. This thesis assumes that the distinction between these terms is understood and leverages these technologies to address challenges in modelling complex dynamical systems.

AI encapsulates the entire field of intelligent systems, where ML is an application of AI. ML focuses on statistical methods to simulate intelligence based on experience [27]. ML methods include statistical models such as decision trees, k-nearest neighbours, and support vector machines. Thus, the term AI will be

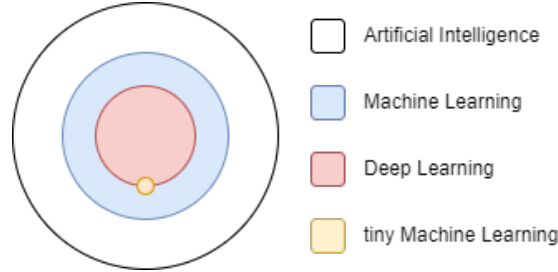


Figure 2.1: Diagram showing AI research and its applications as subsets with tinyML being a subset of both ML and DL.

used more generally than ML but ML may also be used to cover a wide range of more specific models, including DL models.

2.4 Deep Learning

DL algorithms often consist of several layers of neurons, performing a series of functions before filtering to the next layers and repeating [28]. By the last layer, a decision is made, and/or an output is processed. Some of the first instances of modern DL algorithms were used to classify the Modified National Institute of Standards and Technology (MNIST) dataset, a set of handwritten digits [29, 30], where it outperformed the previously superior support vector machines [31]. Now, DL is a vast area of AI with an expanding range of applications in areas such as autonomous vehicles [32], speech recognition [33], and more.

A subsection of DL includes Deep Neural Networks (DNNs), which contain several layers that feedforward to the next, with the final layer producing an output. Most neural network architectures are derived from or at least similar to the DNN, therefore, it is important to understand the underlying principles. Such layers can be mathematically modelled by denoting each layer as some function f , which the neural network is attempting to approximate, such that $y = f(\mathbf{x}; \boldsymbol{\theta})$. Here, y denotes the true value, \mathbf{x} denotes the input value, and $\boldsymbol{\theta}$ denotes the optimal parameters for such a function approximation. In particular, each layer attempts to mould the neural network to fit a particular pattern in the data. Consequently, any new data input into the DNN should output the correct output based on the generalisation of the DNN. For an n layered DNN,

each function is recursive such that

$$f(\mathbf{x}) = (f_n \circ f_{n-1} \circ \dots \circ f_2 \circ f_1)(\mathbf{x}), \quad (2.2)$$

where \circ indicates the function composition operator, i.e. $(f \circ g)(x) = f(g(x))$. Suppose that the output of $f_i = \mathbf{z}_i$ such that the i th layer \mathbf{z}_i , takes the following form

$$\mathbf{z}_i = g(\mathbf{W}_i^\top \mathbf{z}_{i-1} + \mathbf{b}_i), \quad (2.3)$$

where function g denotes an activation function, \mathbf{W}_i and \mathbf{b}_i denote the matrix of trainable weights and its biases for the i th layer. Furthermore, the zeroth layer of the DNN is the input vector, i.e. $\mathbf{z}_0 = \mathbf{x}$ and the output of the final layer is the predicted output label, i.e. $\mathbf{z}_n = \hat{\mathbf{y}}$, an approximation of the true label \mathbf{y} [34].

The activation function g can take various forms, with common examples including Rectified Linear Unit (ReLU), leaky ReLU, sigmoid, softmax, and tanh. The optimal choice of activation function depends on the specific problem and the role of the layer within the network. For example, softmax is often used in the final dense layer for classification tasks, while the Long Short-Term Memory (LSTM) module typically employs tanh and sigmoid activation functions [35].

These mathematical operations, along with the increasing size of $\boldsymbol{\theta}$, make DL a computationally intensive task. Despite this, DL methods are among the most widely deployed to the edge of networks [35–37]. DL methods can adapt well to reliably classify objects or tasks while remaining flexible when adjusting model size and parameters with minimal accuracy loss. Furthermore, DL encompasses a wide variety of network architectures, making it well suited for a variety of edge applications.

2.5 TinyML, its Aliases and Variations

Recently, tinyML has gained significant attention in the research community. In this era of cloud innovation, key technologies are found in edge AI and tinyML. But how did the field gain its name? What research led to tinyML becoming an emerging field? This section answers these questions, while also providing the foundational knowledge for the rest of this thesis.

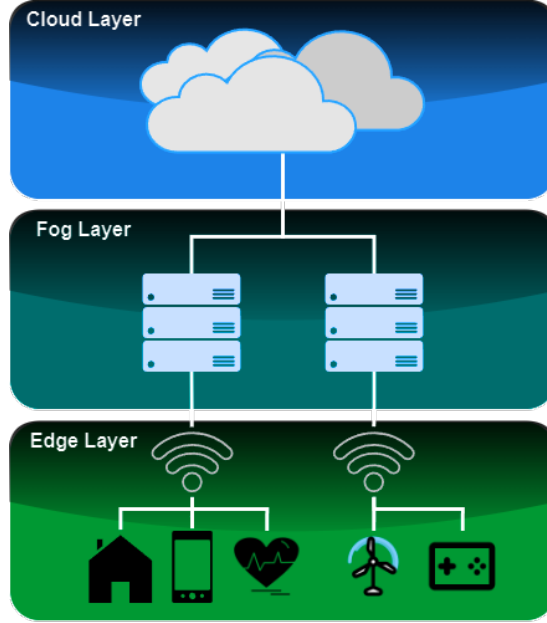


Figure 2.2: Figure depicting three key computing paradigms, the edge, fog and cloud computing paradigms. Processing is completed in the edge layer, hence reducing the burden on the cloud layer. The fog layer can act as an intermediary layer to aid in the processing from a semi-decentralised but likely static location.

The edge computing paradigm consists of devices processing part (or all) of a task close to the end user, therefore reducing the reliance on cloud computing (see Figure 2.2) [11]. Ultimately, the edge computing paradigm aims to decentralise processing, leading to a more environmentally sustainable ecosystem [38]. Although, processing data at the edge of the network has a few drawbacks, namely hardware processing power, power consumption and processing latency among others [13, 39, 40]. Research has therefore focused on migrating cloud-based programs onto edge servers through approaches such as fog computing [41, 42], near-edge computing [43, 44], or directly onto edge devices. Edge devices are those close to the data sources and can range from mobile phones [45], sensor systems [46] or small computers such as a Raspberry Pi [47, 48] or Nvidia Jetson Nano [49].

Where edge computing aims to process tasks close to the data sources, researchers have found it beneficial to bring AI-related tasks to the edge [50]. Such research has therefore coined the name edge AI [51, 52]. Sometimes, edge AI can be categorised into *Edge for AI* and *AI for Edge* for hardware and software so-

lutions, respectively [52–54]. Primarily *Edge for AI* provides hardware solutions whereas *AI for Edge* focuses on on-board optimisations and making full use of the edge’s decentralised capabilities [55]. The greatest value is gained in continuous data collection applications, as the amount of processing and probability of data intercept are reduced [56]. Edge AI, much like edge computing and AI, has several limitations. These are commonly reported within the literature and are the following:

- Latency
- Power Consumption
- Resource Constraints

The above are derived from similar challenges between AI and edge computing. Section 2.6 will go into detail on these challenges.

While edge AI is a common term for the joint subject area of AI and edge computing, other terms such as Edge Intelligence (EI) [40, 48, 57–61] and tinyML [62–66] are popular among the literature. However, each of these terms encompasses different aspects of AI and edge computing. EI is the combination of hardware and software solutions to deploy AI to the edge. EI can be considered synonymous to edge AI due to its focus on deployment. However, several names have been given to this phenomenon. [67] surveys literature on EI and cites papers entitled with edge AI such as [68]. Furthermore, the literature has also cited this area as the *Intelligent Edge* [69] with a particular focus on facilitating edge device capabilities. Whereas tinyML uses ML techniques to deploy onto “tiny” devices such as Microcontroller Units (MCUs). Enabling such devices with complex AI algorithms will mean faster decision-making without relying on other systems to provide the heavy computations. The form factor of tinyML devices makes them desirable as they can be deployed to a range of environments. While other names exist, they tend to deviate from the low-power MCU deployment targets. For example, [70] use Artificial Intelligence of Things (AIoT), which focuses on the application of AI to the Internet of Things (IoT). It is important to establish the distinction between all the terms discussed, as the literature can be misleading if

all these terms are combined as synonymous. Articles focusing on edge AI will not necessarily focus on the same hardware as a paper on tinyML.

As the capabilities of tinyML hardware increase, software solutions must follow leading to a wide range of applications. With appropriate software optimisation, perhaps several applications could be deployed to a single MCU device, making such a device multi-purpose. This thesis will provide insight into potential software optimisations for application of multiple models onto constrained MCU devices.

2.6 Constraints of tinyML and Edge Intelligence

Smaller devices deployed on the edge yield significant benefits compared to the use of cloud-based solutions. Edge devices have increased privacy as data is processed on the device close to the data source. This is particularly relevant for MCU devices, which typically lack the capacity to store sensitive information beyond what is required for processing. Furthermore, due to edge systems' form factor and low reliance on cloud-based services, scaling up systems can become simpler. For example, new edge devices can be added to the network without requiring extensive upgrades to centralised infrastructure. This section focuses on the commonly mentioned limitations within edge computing research, namely;

- Latency: The time delay (in milliseconds) between receiving input and delivering the corresponding output, including computation and communication overhead.
- Power Consumption: The rate of energy usage required to perform tasks such as data collection, inference, and decision-making, typically measured in watts (W).
- Resource Constraints: The limitations of the host hardware, including processing power, memory, storage, and energy capacity.

2.6.1 Latency

Edge computing is capable of offering lower latency in most cases, as the computation is performed close to the data source [39]. Therefore, data do not travel long distances, which negates the latency accrued through travel time. However, there is the caveat of limited computational resources, limited storage, and network limitations at the edge, but these can counteract the lack of travel time and increased privacy that offloading data imposes.

In contrast, suppose that for a system running AI inference, the system latency is divided into two, inference latency and network latency. Therefore,

$$\textit{System Latency} = \textit{Inference Latency} + \textit{Network Latency}. \quad (2.4)$$

For cloud computing, where inference is performed on computationally resource-rich cloud servers, the inference latency may be very low; however, the network latency may be relatively high by comparison because of high data volumes. Compared to edge computing, where inference is on-device, the inference latency may be much slower than that of cloud computing (seconds instead of milliseconds), but the network latency will be nullified as the data does not leave the device. Therefore, the deployment to the edge could cause an unwanted increase in computational latency while only negating a minimal initial network latency, especially in well-connected areas.

Minimising latency is often a priority, but especially when there is a high risk of harm, such as in the case of self-driving vehicles (internet of vehicles) [71]. Performing tasks consistently in real time often requires high memory usage; although, memory itself is limited at the edge [40]. This implies that achieving a system that can process data in real time, i.e. low latency, as well as having a low memory footprint, can be very difficult.

2.6.2 Power Consumption

By moving the data processing close to the data source, access to the power grid is unlikely [38]. Several ubiquitous edge technologies require battery power of varying sizes and voltages, from mobile phones to wearables to self-driving cars.

These devices thrive by conserving battery power wherever possible and operating within their means. It is common for mobile phone/wearable manufacturers to advertise and boast prolonged battery life, but this requires careful planning and meticulous implementation. As such, battery life is very competitive in this industry because of the competitive nature of the optimisation problem involved. An edge device must maintain the balance between operating time and operating cost (in power). With more computation and several peripherals embedded in the system, the device will consequently consume more power. This is especially true when discussing the intelligent edge due to the complexity of AI/ML models. Therefore, when designing an edge AI system, the design must heavily consider the hardware involved [72].

Power consumption is relative; that is, an ML model will consume power per inference cycle, and the device will have an overall operating voltage. A high voltage during inference will require careful consideration of the interference frequency, as maintaining sufficient battery life is essential for task execution over the intended duration.

2.6.3 Resource Constraints

Edge devices consist of small devices, such as MCU devices, to larger devices, such as the Raspberry Pi 4 and Nvidia Jetson/Orin Nano. However, both are resource-constrained in very different ways. The Raspberry Pi 4B hosts a Broadcom BCM2711, Quad core Cortex-A72 which uses the ARM v8 architecture with a 64-bit System-on-Chip clocking 1.5GHz [73]. However, most boards with micro-controllers struggle to clock anywhere close to 1GHz, and host RAM typically in the megabytes or kilobytes. For example, the low-power and resource-constrained Arduino Nicla Sense ME and Arduino Nano 33 BLE Sense have 64KB and 256KB of memory, with 512KB and 1MB of flash, respectively, both clocking at 64MHz. For this reason, the keyword “tiny” is becoming increasingly popular in research to distinguish these two groups of devices, hence tinyML [64, 74–76].

Edge devices and especially tiny edge devices, such as MCU devices, suffer from very limited resources. Models that would normally have millions of trainable parameters would need to be compressed down to the tens or hundreds of

thousands of parameters. But even with model-optimisation techniques, may not always be sufficient for deployment on extremely limited hardware.

2.7 Model Reduction Techniques

To maximise the performance of ML models for the purposes of constrained devices, such models will require software optimisations. For edge AI and tinyML applications, common areas of focus include quantisation and pruning, among other methods. Ultimately, all of these are based on the success of the *lottery ticket hypothesis*, which states that for any network, there exists a sub-network (the “*winning ticket*”) that can reach comparable accuracy in a similar number of iterations [77]. Therefore, effective optimisation techniques must be formulated to find the “*winning ticket*”. Hence, this section will provide an introduction into quantisation, pruning, and knowledge distillation in detail, as these are among the most popular software optimisation techniques.

2.7.1 Quantisation

Quantisation is a key component in model optimisation. It allows data to occupy less space, whilst incurring a reduction in accuracy as a caveat. Several methods of quantisation exist with Post Training Quantisation (PTQ) [78] and Quantisation Aware Training (QAT) [79] among the most popular. These methods are performed after training and during training, respectively, and both yield different benefits. PTQ has a direct impact on a model at the foundational level, since the values are truncated during conversion. Whereas QAT applies the quantisation step during training, which typically means the burden on model performance is less. It is important to note that as the level of precision changes, so do the minimum and maximum possible values as well as floating point accuracy (see Table 2.1). [80] worked on an adaptation of PTQ for use on vision transformers using a bias correction tool to correct inaccuracies gained in the quantisation process. [81] proposed a new process, *AdaRound* for intelligently rounding during the PTQ process to reduce error. [82] developed a quantisation technique that is capable of being hardware-aware, leveraging reinforcement learning to find the

Table 2.1: Table demonstrating some of the limits of double precision (*float64*), single precision (*float32*), half precision (*float16*) and 8 bit integer (*int8*) precision. Decimal precision denotes the number of decimal places the datatype can accurately place using $\log 2^{\text{significand_bits}+1}$ rounded down to an integer.

| | Exponent | Significand Bits | Maximum Value | Decimal Precision |
|----------------|----------|------------------|---|-------------------|
| <i>float64</i> | 11 | 52 | $2^{1023} \times (1 + (1 - 2^{-52})) \approx 10^{308}$ | 15 |
| <i>float32</i> | 8 | 23 | $2^{127} \times (1 + (1 - 2^{-23})) \approx 3.4 \times 10^{38}$ | 7 |
| <i>float16</i> | 5 | 10 | $2^{15} \times (1 + (1 - 2^{-10})) = 65504$ | 3 |
| <i>int8</i> | N/A | N/A | 128 | 0 |

best quantisation policy. [83] extended this by producing an algorithm capable of computing optimal quantisation policies based on memory and computational characteristics. They also use reinforcement learning similar to [Neural Architecture Search \(NAS\)](#).

The impact on quantising to a datatype of less bits detracts model performance the smaller the datatype chosen. Therefore, choosing the smallest quantisation policy may not necessarily be the best action. For this reason, the naive use of binary quantisation, otherwise known as [Binary Neural Networks \(BNNs\)](#), will lead to imperfect results. When training [BNNs](#), backpropagation is not effective because the derivative of discrete binary weights is zero, so most known information would be lost [84]. Not only this, but also the methods of training [BNNs](#) vary in an attempt to reduce the accuracy loss caused by binarisation. However, authors such as [85] and [86] achieve 87.9% and 75.6% Top-5 accuracy, respectively, on the ImageNet dataset. [86] draws upon several considerations, such as an altered regularisation (instead of L2 regularisation), using a small learning rate and using [Parametric Rectified Linear Unit \(PReLU\)](#) [87] (instead of [ReLU](#) or [LeakyReLU](#)). Few [Field Programmable Gate Array \(FPGA\)](#)-based [BNNs](#) exist demonstrating the potential of [BNNs](#) in the future [88, 89].

2.7.2 Pruning

Pruning involves trimming elements of a [DL](#) model such that inference latency and model size are reduced. The idea is based on the optimal brain damage

problem [90, 91] (which is now known as weight pruning), but has since been adapted. Weight pruning looks at trimming the weights of a DL model by setting them to zero based on the least important weights. Determining the least important weights is a heuristic that can be altered by the developer as it is likely application-specific. Weight pruning is the earliest and simplest form of pruning; however, new methods have been constructed to aid more intricate pruning policies. More sophisticated methods look at the structure of the network itself rather than making naive decisions on which areas to prune. This is known as structural pruning. [92] focus on pruning filters within Convolutional Neural Networks (CNNs) and gained a 30% reduction in Floating Point Operations (FLOPs) when using VGGNet on the CIFAR10 dataset. They accomplished this by gradually pruning filters that have minimal impact on the output using the ℓ_1 -norm ($\|x\|_1 = \sum_{i=1}^n |x_i|$). However, [93] found that the ℓ_2 -norm ($\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$) achieved comparable accuracy but with a 41% reduction in FLOPs. Although pruning does not always pertain to CNNs, as weights can be pruned using more advanced techniques for deeper models [94–96]. In addition, deeper methods can be applied to shallower models. [97] developed a method for MCU devices, which achieved a 93% parameter reduction compared to a current tinyML benchmark suite [74, 98].

2.7.3 Knowledge Distillation

Beyond quantisation and pruning, other methods such as knowledge distillation, proposed by [99], have shown positive results. Knowledge distillation is the process of using a larger *teacher* model to aid training a *student* model using soft labels. Soft labels provide knowledge-rich representations to the *student* model, which is typically a smaller architecture designed to capture the key features of the *teacher* model [100]. In some cases, the *student* model can even outperform the *teacher*. In [101], they achieve marginally higher accuracy on the smaller *student* using one-shot and few-shot learning. [102] introduced *similarity-preserving knowledge distillation*, which guides the student network to learn patterns of similarity and dissimilarity from the teacher network’s activations using similarity matrices. Instead of mimicking specific values, the student focuses on preserving

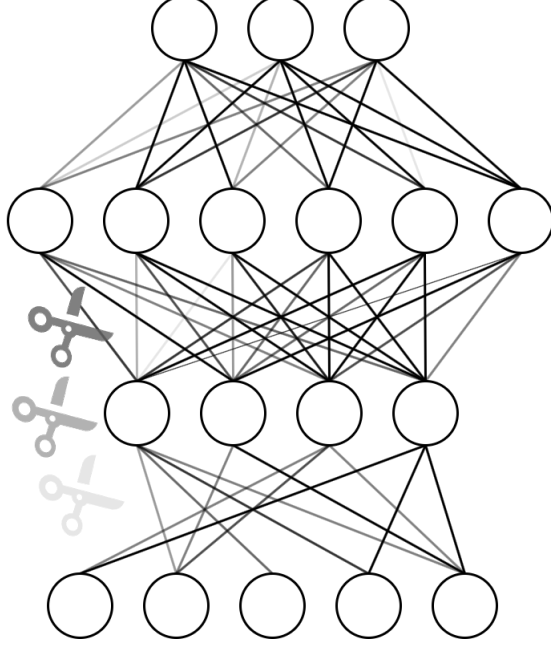


Figure 2.3: Weight pruning of DNN weights.

pairwise relationships, allowing for significant model size reduction with minimal accuracy loss. The authors demonstrate a $5\times$ compression rate with only a 0.3% accuracy reduction when distilling a WideResNet-16-8 network (11M parameters) into a WideResNet-40-2 network (2.2M parameters) on the CIFAR10 dataset. [103] questioned whether larger models imply a better *teacher* model. They found that as the *teacher* becomes sufficiently large, the *student* struggles to fit. Therefore, they propose *early-stopping knowledge distillation*, as smaller *student* models may not have the capacity to learn effectively.

2.8 Context-Aware Solutions for MCUs

Context and contextual recognition are important components of daily life. They can help users understand intent, emotion, and aid one's own personal inference of a situation. Attention mechanisms are similar in concept to context, which has bred a new wave of DL in transformer networks [104, 105]. However, these models carry a heavy processing burden and are currently not suitable for deployment to MCU devices.

Some attempts at contextual-based solutions have been explored in the literature, for example [106] explores the feasibility of creating a contextual algorithm to switch between models depending on the season in aircrafts. This would require several models, which would be interchanged depending on the context given. Furthermore, Gaballah et al. [107] classify stress in hospital workers using contextual cues from speech and physiological features. Participant stress data was collected from a hospital over a 10-week period in which participants self-labelled stress on a five-point scale. They found that the addition of contextual information increased the accuracy and the F1 score by a maximum of 11% and 13.5%, respectively. Rashid et al. [108] propose *SELF-CARE*, an approach applied to stress detection that uses selective fusion from accelerometer modalities while also introducing a late fusion process using a Kalman filter. Both of these fusion techniques accommodate their model, which is composed of several random forest classifiers. They use four input streams including 3-axis accelerometer, blood volume pulse, [ElectroDermal Activity \(EDA\)](#), and skin temperature. Their research demonstrates that incorporating contextual information can significantly improve the accuracy of classification models in a variety of applications.

It can be seen that much of the prior work pertains to stress recognition, which could be due to the obvious tie context holds to stress recognition. A review of contextual methods for stress detection has been conducted in [109] that provides a comprehensive look at how contextual information can improve stress detection performance using [ML](#) models. In all of the different studies, the addition of contextual features consistently improved the accuracy of the model compared to the baseline models, highlighting the importance of contextual data. However, it can be difficult to obtain sound contextual information without the use of another intelligent agent.

When inferring based on a single modality, the accuracy may be lower than that of multiple modalities. For example, suppose that a classifier is tasked with identifying mood based on audio information alone. Suppose that the user is talking very fast and at an elevated noise level. From this description alone, the mood could imply anger/frustration or excitement, two very different emotions. To reduce these misconceptions, an additional modality, such as movement information, could be introduced. Alternatively, another model could be constructed

to transcribe audio from audio information as an additional modality. Although no new information is provided, there is an additional model that will infer new information, which could influence the final outcome. However, fusing features for the optimal output is challenging.

Information fusion is challenging due to the heterogeneous nature of sensor environments [110, 111]. Variations in environmental conditions can lead to unreliable inference, often forcing reliance on a single mode of input. In [112], two older sensor fusion techniques are measured against attention-based fusion. Attention-based sensor fusion takes a set of n feature vectors $\mathbf{F} = [\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_n]$ and apply an attention value \mathbf{a} to each feature vector according to its importance. They train another neural network to generate an estimate for \mathbf{a} , which they denote $\hat{\mathbf{a}}$, such that, when applied to \mathbf{F} , a new attention-based set of feature vectors, $\hat{\mathbf{F}}$, is obtained.

$$\hat{\mathbf{F}} = \mathbf{F} \odot \hat{\mathbf{a}}, \quad \hat{\mathbf{F}} \in R^{n \times m}, \quad (2.5)$$

where \odot denotes the element-wise multiplication operator, n , the number of feature vectors and m , the size of the individual feature vectors. Typically, there are three types of attention mechanisms, soft/global attention [113, 114], local/hard attention [115], and self-attention [104, 111, 116]. Soft/Global attention takes the entire system or set of states into account when calculating the attention, hence the name global. Local/Hard attention looks at the current state the system is in when estimating the attention parameter. Self-attention uses a collection of points from a single sequence to calculate the representation of the entire sequence [104, 116]. All of these attention mechanisms require a large number of mathematical computations. However, some attention mechanisms seem to often involve an array of [Recurrent Neural Networks \(RNNs\)](#), which on the edge would be difficult to implement due to limited computational resources. However, it may be possible to use two small networks, one for the attention estimation and the other for inference on the given task. But, this still is a substantial challenge, which may justify the fact that no such attempts exist within the literature at this time.

Assigning attention to sensors will highlight which sensors need more focus

when processing their inputs. To ensure that multimodal sensor fusion is the correct approach to the problem, the input modalities should be correlated with all sensor inputs. For example, affect can be more accurately identified using both EDA and Heart Rate Variability (HRV) sensors, as both the EDA and HRV sensors are correlated with affect [117].

There are several research articles pertaining to multimodal sensor input, including some of the applications. For example, in [118], they fuse Electroencephalogram (EEG) and resting state-functional Magnetic Resonance Images (rs-fMRI) to localise epileptogenicity at the edge. However, the data is offloaded to an edge server to handle preprocessing, which is necessary due to the amount of resources required to prepare and fuse the information, such as using FWHM = 4 and a fourth-order Butterworth bandpass filter; both are computationally expensive in the context of processing on the resource-constrained edge device. Often preprocessing is left out of the edge computation, leading to data offloading, which goes against one of the main benefits of processing at the edge, i.e. privacy. However, if it were possible to reduce the model size without a statistically significant impact on the performance metrics, it may be possible to employ more preprocessing techniques. In general, incorporating contextual information into a model could require a separate contextual model, but on resource-constrained devices, hosting multiple models is a significant challenge.

2.9 Multi-Task Learning and Multi-Domain Learning

Multi-Task Learning (MTL) and Multi-Domain Learning (MDL) aim to develop a model or a set of models that can effectively map multiple input domains to multiple corresponding output tasks. MTL specifically targets the multiple output tasks and MDL the input domains. MTL architectures are split into two types, hard parameter sharing [119, 120], and soft parameter sharing [121, 122]. Hard parameter sharing involves sharing the hidden layers between all tasks while keeping several task-specific output layers. In contrast, soft parameter sharing adds a constraint to encourage similarities among related parameters, allowing

for more flexibility for the tasks.

Previously, Zhang et al. [123] proposed training a single neural network that shares some layers between all tasks and has some task-specific output layers. Therefore, they create a large model that encompasses all the different tasks in one network architecture. Importantly, all tasks are executed simultaneously, which somewhat limits the practical application. Although, MTL has also been used for simple and complex HAR using wearable sensors [124], multiple language translation [125], and multi-camera person re-identification [126]. As the model is given more tasks, the network expands significantly, making it difficult to train and use, particularly when considering devices with limited resources such as MCU devices.

More recently, MTL has become more focused on optimisation and training. For instance, the total loss function \mathcal{L} has been altered using combinations of several individual loss functions $\{\mathcal{L}_1, \mathcal{L}_2, \dots, \mathcal{L}_n\}$ from n tasks. By combining the loss function, tasks can learn similar representations together while training. These loss functions could even have an attention scalar value attached to each individual loss function. Therefore, individual tasks will have a weighted priority applied to the loss function [127, 128]. However, some tasks are too different and do not generalise well when paired. To overcome significantly different tasks not learning well together, clustering methods have been used to group tasks [129, 130] such that smaller groups of related tasks are trained together. [129] developed a *Task Affinity Grouping* framework to measure related tasks. Tasks with high affinity are trained together (with a combined loss), and tasks with low affinity are trained separately.

Research in MDL overlaps with both MTL and domain adaptation [131]. Domain adaptation alters data from one domain to another. Although, domain adaptation comes before the training process to resolve the dataset imbalance. MDL looks to learn several domain representations. For instance, filtering spam works similarly for most users, but, from subject to body, each user will receive a myriad of incoming emails. Therefore, instead of several models per user, a single MDL model can be adapted to the needs of users [132].

MDL architectures vary; models can come in two main forms, multi-headed and adapter-based (see Figure 2.4) [133]. [134] extended work from [135] for video

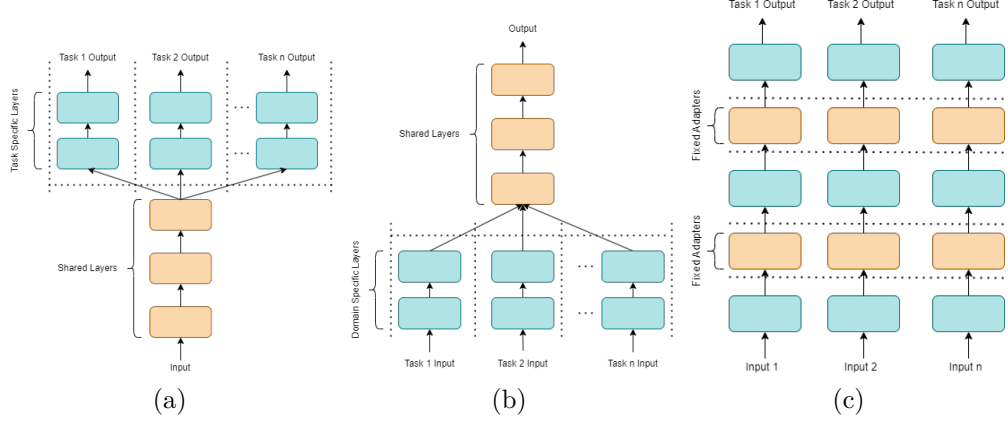


Figure 2.4: Three types of MDL architectures where the gold denotes shared parameters between different input domains. (a) A simple MTL hard parameter sharing architecture where all input domains flow into domain-agnostic layers. (b) An opposing architecture to (a) where each input domain extract domain-specific features to solve a single task. (c) Adapter-based MDL where domain-agnostic fixed parameters act as feature extractors across domains.

motion magnification. They used MDL to train lightweight models using both frequency and spatial domains (at approximately 0.11M and 0.05M parameters respectively). MDL has also been used for health monitoring systems [136], security algorithms [137] and recommendation algorithms [138]. Although MTL is typically applied to computer vision-based tasks [139], it has also been applied to healthcare imaging [140, 141]. For example, [142] use MTL to detect anatomical landmarks within medical imaging data. They analyse three-dimensional brain magnetic resonance and prostate computed tomography data and find that their hard parameter sharing approach meant a lower margin of error compared to other state-of-the-art algorithms.

In [143], they use MTL to simultaneously learn several representations to classify surface defects. They implement three tasks; classification, segmentation, and detection, all of which aid in identifying the aforementioned defects. Using MTL, their model demonstrated improved noise resistance, reduced inference latency, and increased accuracy compared to individual tasks. MTL has been used similarly to extract several features from the same input modality. Therefore, each task is not completely different in what it is classifying, but is instead a

separate task relating to the same input space [144].

MTL is limited in the number of tasks because the more tasks added, the more likely the total loss will increase disproportionately. The increased loss could be due to the shared information having to be more general between more tasks, therefore reducing task specificity. Thus, grouping algorithms can be used to sort several multi-tasking models into specified groups [145]. Overall, **MTL** has the ability to reduce the number of parameters in an intelligent system while maintaining performance. As a result of lower parameters, the memory footprint could be reduced, resulting in reduced power consumption [146], which is often very limited in **MCU** devices.

2.10 Real-World Applications of tinyML

Classification tasks independent of time take into account data at a single time point, such as any image classification. As the image is taken at a particular moment and classified, it is invariant to temporal dependency. For **tinyML** applications, image classification is popular but limited. Image classification tends to be intensive, since convolutional layers require significant processing capabilities, especially when grouped together. However, some methods have been optimised specifically for use on **MCU** devices, such as Edge Impulse’s *Faster Objects More Objects (FOMO)* [147] and *TinyissimoYOLO* [148]. **FOMO** has enabled applications similar to MobileNet and **You Only Look Once (YOLO)** while still compact enough to effectively run inference on a microcontroller at 30 (on an Arduino Nicla Vision). Inference is possible on an **MCU** device as centroids are used to discretise the image into smaller squares. Whereas *TinyissimoYOLO* is inspired by **YOLOv1**, an earlier iteration of the model. [149] compared the two of these models with the introduction of tiling, which repeatedly infers smaller subimages to achieve a 91% F1 score from the 28% prior. With these models, more complex tasks can be brought to the edge of the network. However, for newer methods that either improve or challenge those prior, reliable datasets are required for validation. For image classification datasets, **MNIST** and **CIFAR10** are among the most common. **CIFAR10** requires a more complex model to capture the vast differences between its ten classification tasks compared to **MNIST**. Some image-

based tasks can depend on time, such as person detection algorithms [150]. Tasks which rely on time, i.e. time-series data, are desirable at the edge of the network, particularly when handling sensitive data. Such is typically the case in smart healthcare applications.

As discussed in Section 2.1, falls attribute a significant risk to those over 65 years of age [16]. Providing patients with a small device to carry to detect falls will mean a faster response from emergency services and, consequently, fewer deaths as a result. Such a device would need to host an ML model that can perform inference quickly with high accuracy and recall. [151] makes use of edge-based technologies to develop a system (named *Whoops*) capable of calling emergency services if the user is unresponsive after a detected fall. However, *Whoops* does not use the computational resources on-device. [152] utilise the edge devices' capabilities but still offloads some processing to an edge server, while [35] migrates all the data collection and processing to on-device using an RNN.

Healthcare resources are often limited and therefore require careful allocation. Some EI devices will be able to detect illnesses or attacks in patients before they occur. For example, devices can be given to patients prone to strokes to provide timely notification of an attack's onset, thus mitigating damage to the patient. [42] produced an algorithm capable of detecting falls in stroke patients, as strokes have a high risk of morbidity. Although, the implementation used in [42] used the fog computing paradigm. In addition, EI may help prioritise certain cases over others. In general, the adoption of EI technologies could lead to fewer hospital visits [153] and greater public physical health.

Beyond smart health, extensive research has shown the proficiency of sensor-based configurations in classifying HAR. Wearable sensors attached to users are commonly used for data collection; these sensors include accelerometers, gyroscopes, and magnetometers and are often embedded within smartphones and smartwatches.

Sensor-based HAR is a time-series classification problem. DL enables high-level feature representations to be automatically learned from raw data. Both CNNs and LSTM networks have shown promise for HAR [154]. CNNs have been widely used to learn spectral patterns of sensor signals, whereas LSTMs have been used to capture temporal dependencies.

HAR is a common application area when developing ML on low-power devices, as a CNN-based feature learning approach has been developed for HAR [155]. In their approach, they used IMU data from 20 healthy subjects to classify walking, walking upstairs, walking downstairs, sedentary, and sleeping, achieving 96.4% accuracy. Considerable computational speed-up was achieved using the proposed approach compared to and Multi-Layer Perceptron (MLP). The model was deployed onto a smartphone for use in the real-world. A one-dimensional CNN-based method for HAR has also been developed [156]. Activity data was collected using a smartphone IMU, including walking, running, and standing still. Acceleration data were converted to vector magnitude data and used as input to the one-dimensional CNN achieving an accuracy of 92.71%.

Approaches have been developed to improve classification on constrained MCU devices. Optimisation techniques, such as pruning and quantisation, have enabled the size of the model to be reduced by 10 times without severely impacting model accuracy [157]. However, [158] have shown the possibility of classifying human activities on the edge using a low-power MCU device using quantisation. A one-dimensional CNN was used to develop a classification model to infer three activities (running, falling, and normal state). The model was deployed on an Arduino Nano 33 BLE Sense and achieved 97% accuracy with quantisation reducing the model size by 53%. Similarly, a CNN has been used to classify five specific movements in bed (agitation, idle, in bed, out of bed, and bed movement) for the elderly [76]. The model was deployed to an nRF52 development board achieving 88.96% accuracy. However, the data used in this study were collected from only one person, creating bias.

While edge computing includes powerful devices such as smartphones, the use of low-power MCU devices shows greater potential due to the ability to add external sensors such as for physiological monitoring. Furthermore, although HAR research has made extensive use of smartphones for on-device processing, it is not feasible to use a smartphone for the processing of external sensors, such as physiological sensors required for stress detection.

Non-invasive physiological sensors such as EDA and Heart Rate (HR) present a significant opportunity to assess stress in the real-world due to their direct correlation with the sympathetic nervous system. EDA and HR have been shown

to detect poor mental wellbeing, such as stress, using a one-dimensional CNN with 92.3% accuracy, outperforming comparative models such as LSTM [117]. Similarly, EDA and HRV were used in a wearable device to measure stress during driving [159]. The wearable device took measurements over a 5-minute period to detect stress levels with an accuracy of 97.4% and found that HRV and EDA are highly valuable. EDA and HR signals have also been used to infer stressed and relaxed states using k Nearest Neighbour (k -NN) and Fisher discriminant analysis, achieving 95% accuracy stating that 5–10 second intervals are suitable for real-time stress detection [160].

While traditional ML methods like k -NNs and DL achieve accurate stress detection, RNNs can model complex temporal dependencies in physiological signals. ESNs, a lightweight yet powerful subclass of RNNs, provide an efficient way to capture these temporal dynamics.

2.11 Echo State Networks: A Cognitive Approach

The Echo State Network (ESN), introduced by [161], is a subset of reservoir computing, whereby a random reservoir of neurons governs the dynamics of a system. ESNs are a form of RNN which propagate an input state into a higher dimensional state to output into a *readout* layer. The state update equation for an ESN [161, 162] is as follows

$$\mathbf{x}(t+1) = \tanh(\mathbf{W}_{\text{in}}\mathbf{u}(t+1) + \mathbf{W}\mathbf{x}(t)), \quad (2.6)$$

where $\mathbf{x}(t) \in \mathbb{R}^N$ is the N -dimensional reservoir state, $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N \times K}$ denotes the $N \times K$ input matrix, $\mathbf{W} \in \mathbb{R}^{N \times N}$ denotes $N \times N$ reservoir and $\mathbf{u}(t) \in \mathbb{R}^K$ denotes the K -dimensional input state. With the reservoir states obtained from equation 2.6, they can be fed into a regression or classification model to produce an output. Typically, the output, or *readout* layer takes the form of a ridge regressor.

In reservoir computing and ESNs, the reservoir itself, once initialised, does not require training. Only the final *readout* layer needs training, making ESNs desirable in scenarios where training is to be minimised. Therefore, the initial-

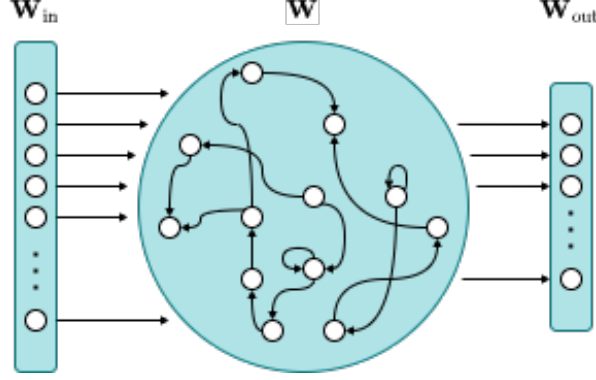


Figure 2.5: A typical ESN with the reservoir input vector \mathbf{W}_{in} feeding into the sparsely connected reservoir \mathbf{W} , which eventually yields an N -dimensional vector output \mathbf{W}_{out} . Typically \mathbf{W}_{out} is a *readout* layer, such as a ridge regressor.

isation process should be carefully considered, especially to maintain the [Echo State Property \(ESP\)](#). The [ESP](#) is a composite condition that ensures that the internal states of an [ESN](#), or reservoir activations, are stable, responsive, and primarily influenced by recent inputs rather than initial conditions or states from a distant past. The [ESP](#) includes: a fading memory effect, which ensures that the influence of past inputs gradually diminishes; stability in dynamics, typically controlled through the spectral radius; and a trade-off between non-linear transformation capabilities and memory retention. Reservoirs with higher non-linear transformations will respond dynamically to recent changes in inputs, but may struggle to retain long-term dependencies due to their sensitivity. Conversely, with low non-linear transformations, the reservoir weights will be less sensitive to changes, improving its ability to recall long-term information. These characteristics allow [ESNs](#) to effectively encode temporal patterns, providing the foundation for robust performance on time-dependent tasks [\[161, 163\]](#). A reservoir without the [ESP](#) will not generalise well on tasks, leading to poor preservation of input dynamics.

An [ESN](#) is commonly used to predict time-series data through regression tasks, although that does not mean that it cannot perform well for classification tasks. In [\[164\]](#), they used several reservoirs of 16K neurons to achieve 0.92% error rate on [MNIST](#) (total 528K parameters, which are mostly reservoir neurons).

Others, such as [165], have used ESNs for anomaly detection. They proposed an ESN auto encoder which achieved 91.2% accuracy with a model of only 89.2K parameters. They make use of 0.9 spectral radius and 90% sparsity (i.e., $s = 0.1$) and a single fully connected layer containing fewer neurons than the input as a *readout* layer. However, more commonly, ESNs are used to predict behaviour in dynamical/chaotic systems [166–169].

2.11.1 ESN Hyperparameters

There are several hyperparameters that can affect an ESN, which will be used throughout this thesis. The following are the parameters that have a high impact on base ESN performance.

- The way in which the reservoir is initialised can have a large impact on reservoir performance. In [170], they used three initialisation methods (Random Regular, Newman-Watts-Strogatz (NWS) and Barabasi-Albert (BA)) in their experiment to assess the effectiveness of decentralised incremental federated learning. Although reservoir initialisation is not a focal point of their work, [171] incorporate a NWS graph to generate a reservoir and test speech emotion recognition tasks. They find that this graph style enhances the range of the ESP by monitoring the spectral radius.
- ESN reservoirs have a spectral radius (ρ), which is a value that aims to keep the reservoir at the edge of chaos [172]. It is applied to the reservoir during initialisation to rescale the current spectral radius to the target such that

$$\mathbf{W} = \rho \cdot \frac{\mathbf{W}_{\text{init}}}{P(\mathbf{W}_{\text{init}})}, \quad (2.7)$$

where $P(\cdot)$ will calculate the spectral radius of the input matrix. Typically, the spectral radius is set between 0.8 and 1 [173, 174].

- The size of the reservoir has a great impact on the performance of the entire system. In [175], they classify the MNIST handwritten digits achieving a 0.93% error rate using a reservoir of 4,000 neurons. They also experimented with a few other reservoirs using 1,200 and 1,000 neurons which achieved

an error rate of 1.68% and 1.25% respectively. In particular, the increase in neurons increased performance. However, their work does not focus on the implications of a large reservoir, particularly for a simple benchmarking task such as the [MNIST](#) dataset.

- Reservoir sparsity, s , describes the percentage of non-zero connections within the reservoir. A sparse reservoir ($s \ll 1$) often performs better than a dense reservoir [161]. Most applications use a sparsity of 1% [167], 5% [176], or 10% [177].

In general, all components of the [ESN](#) contribute to the effectiveness of the system. For certain applications, some parameters may be adjusted accordingly. For example, for an application that focuses on accuracy, the number of neurons would be higher, and the spectral radius and initialisation method can be adjusted as per the application. However, for an application in a constrained system, the number of neurons in the reservoir should be reduced, and the sparsity should be as low as possible.

2.11.2 ESNs on a Smaller Scale

[ESNs](#) are desirable for devices with low computational power, especially when training is involved due to the low training cost of an [ESN](#). However, their use on constrained devices is limited within the literature. So far, [ESNs](#) for [tinyML](#) have only been investigated in the context of on-device training/learning [178–181]. The model optimisation methods described in Section 2.7 cannot be naively applied to [ESNs](#) as the model structure is different to a typical [DNN](#) model. However, [182] developed *intESN*, a quantisation policy to reduce the bit precision of a reservoir using a Hamming distance clipping algorithm. Their experiment showed that integer quantisation could be achieved but with a significant reduction in accuracy, particularly in multivariate tasks. Nevertheless, when additional neurons were added to compensate for the space saved, *intESN* outperformed other traditional [ESNs](#) for the same tasks. [183] performs time-series prediction on the [MG](#) and Lorenz systems using time-delay reservoirs, a subset of reservoir computing. They outline that reservoir computing could be more suited to reduce

the number of matrix multiplications compared to CNNs. However, the same could be appropriate for applications to constrained devices.

2.12 Challenges and Opportunities

Within this section, challenges and opportunities from the reviewed literature will be explored but limited to the following areas;

- Edge Intelligence and tinyML deployment,
- Multi-Task Optimisation for tinyML,
- Echo State Networks (ESNs).

2.12.1 Edge Intelligence and tinyML Deployment

EI applications are few compared to cloud computing, which is due to the constrained nature of the devices on which AI algorithms are deployed [7, 184]. Such devices often do not have access to mains power. Therefore, there is an additional consideration (beyond the AI algorithm itself) of power consumption when developing EI systems. Furthermore, AI algorithms and DL algorithms in particular are known to be resource-intensive and, as a consequence, power-intensive. Therefore, the lack of power and computational resources yields a delicate optimisation problem in which the aim is to maximise model performance, while minimising the resources required to run inference timely. Typically, larger ML models gain higher accuracy metrics, but also require more resources to compute inference. Therefore, research on model optimisation has been conducted through pruning [185] and quantisation [82, 83]. Many research projects target Raspberry Pis [48, 186, 187], Nvidia Jetson/Orin Nanos [49, 188], or mobile phones [45], which are all edge devices. However, the comparison between the aforementioned devices and MCU devices is unfair due to the magnitude of the difference in almost every technical specification, from clock speed to physical footprint. These challenges that research does not necessarily highlight may contribute to why EI is not as widely used as it could be.

One such challenge is the choice of programming language for [tinyML](#) development. Python is known to be a high-level interpreted language, implying a slower runtime compared to lower-level languages, such as C / C# / C++. Arduino is an open-source electronic platform that simplifies the development of software for specific hardware [189]. As Arduino uses C++, it inherits some of the benefits of lower-level runtime and therefore will be a better choice of language for resource-constrained devices.

However, when developing on edge systems and notably [MCU](#) devices, the actual choice of programming languages become very limited. More specifically, a developer would likely use a C compiled language, such as C/C++ or MicroPython. Although, some boards do not have support for MicroPython and are therefore limited to a single language (often C/C++). This makes development for [MCU](#) devices more challenging as developers are required to be proficient in both Python and C++.

Python is known for the overwhelming online support and [AI](#) libraries such as Google’s TensorFlow [190] and Meta AI’s PyTorch [191]. When using C/C++ on a development board, a significantly lower level of abstraction is required, especially with Arduino hardware, as these boards can struggle to use MicroPython. Arduino boards use a variation of C++ including the Arduino header file (Arduino.h) to aid in development and are simple to understand development at a beginner level. However, libraries such as TensorFlow are no longer available for [MCU](#) devices as they are too large. To overcome this, Google developed a condensed library named [Lite Runtime for Microcontrollers \(LiteRT \$\mu\$ \)](#), previously known as [TensorFlowLite for Microcontrollers \(TFLite \$\mu\$ \)](#) [192]. This compact version of TensorFlow means that models can be developed using Python and [Lite Runtime \(LiteRT\)](#) but inference can subsequently be executed using [LiteRT \$\mu\$](#) . However, the ease of use of TensorFlow in Python versus [LiteRT \$\mu\$](#) in C/C++ is very different. Much more preamble and understanding of the [MCU](#) device is required for C/C++ development over the few lines of code that it would take using Python. Furthermore, [LiteRT \$\mu\$](#) supports only a limited number of [MCU](#) devices. Within Table 2.2, a selection of current [MCU](#) devices for [AI/ML](#) applications were chosen. Of these boards, only 3 have official support for [LiteRT \$\mu\$](#) and are marked with a [†]. In addition, [LiteRT \$\mu\$](#) supports a limited suite of neural network

layers. Notably, only a two-dimensional convolutional layer is supported but its one-dimensional counterpart is not. Furthermore, `LiteRT μ` only added support for `LSTM` modules in 2022 [193], limiting the deployment of certain `AI/ML` models on some development boards. As a result, hardware compatibility remains a key consideration when selecting a platform for a project. Beyond performance and hardware specifications, community support plays a crucial role, as widely adopted boards benefit from extensive resources, troubleshooting solutions, and additional libraries. These resources account for a positive ecosystem, as over time older problems will be well documented and these libraries will be available to the new users. However, an online community is not easy to build and can be considered a feature itself. Therefore, to gain this feature, companies need to build a strong foundation of examples and documentation to allow building a user base. Maintaining this community engagement can have a significant impact on the adoption and effectiveness of `tinyML`.

However, in reality, support for new boards is often limited. Subsequently, limited support makes development more difficult for new users and, in some cases, not viable at all. For example, suppose that a developer is using a newer `MCU` device such as the Arduino Nicla Vision to collect and process heart rate data. The Arduino Nicla Vision’s intended programming language is Arduino C++ and therefore has several libraries available. However, the most common library to convert an analogue signal to heart rate features (`PulseSensorPlayground` [194]) is not compatible with the `MCU` device (as of 05/2023). This example could force the developer to regress and use a significantly older board with less computational power.

In contrast, a board with an overwhelming amount of community support is the Arduino Nano 33 `BLE Sense`. Because of this, even though the board is behind in hardware specification, it still maintains modern day use in research [195]. The range of support means that preprocessing can be performed before inference can be run. However, preprocessing can be costly, especially when running computationally expensive feature engineering algorithms.

Feature engineering uses mathematical methods to extract key features from data. For example, a common way to extract more features from a signal is to use the `Fourier Transform (FT)`, which transforms temporal data into the

2. Literature Review

Table 2.2: Table illustrating the available built-in sensors for a range of development boards and highlighting the boards supported by [LiteRT \$\mu\$](#) (formerly [TFLite \$\mu\$](#)).

| | | <i>Development Board</i> | | | | | | | | | | |
|---------|----------------------|--|--|--|------------------------------------|-----------------------------------|-----------------------------|---|-------------------------------|-----------------------------|----------------------------|--|
| | | <i>Himax WE-I Plus EVB[†]</i> | <i>Sony Spresense Main Board[†]</i> | <i>Syntiant TinyML Development Board</i> | <i>Sparkfun MicroMod (Artemis)</i> | <i>Sparkfun MicroMod (SAMD51)</i> | <i>Raspberry Pi Pico4ML</i> | <i>Arduino Nano 33 BLE Sense [†]</i> | <i>Arduino Nicla Sense ME</i> | <i>Arduino Nicla Vision</i> | <i>Arduino Nicla Voice</i> | |
| Sensors | Camera | ✓ | ✗* | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | |
| | Accelerometer | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| | Microphone | ✓ | ✗* | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| | Bluetooth | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| | Light/Proximity | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | |
| | Pressure | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | |
| | Temperature/Humidity | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | |
| | Gas | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | |
| | Time of Flight (ToF) | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | |

* Extendable Sony Spresense hardware available

[†] Supported boards for LiteRT for Microcontrollers (as of 02/2025)

frequency domain. The [FT](#), however, does include complex calculus, which can be solved numerically when discretised. Edge devices are resource-constrained and, therefore, do not have the capacity to perform the [FT](#), especially for further feature engineering before then passing this data into an [ML](#) model. Variations of the [FT](#) have since been devised, sacrificing a level of precision but achieving low latency. The Fast [Fourier Transform](#) and discrete cosine transform rely on lookup tables for evaluating trigonometric functions. Without careful feature

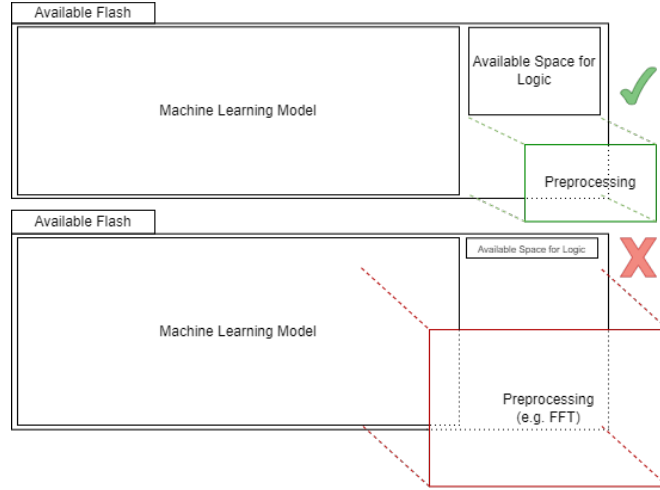


Figure 2.6: Figure demonstrating preprocessing blocks can be too much for an MCU device without careful consideration.

engineering, a developer risks exceeding the memory and processing constraints of an MCU, making it difficult to fit both the model and its preprocessing onto the device. In such cases, upgrading to a more powerful board may be necessary, but this introduces another challenge—sensor compatibility.

Depending on the problem being solved, the sensors required for reliable results will vary greatly. For example, [196] monitored mental wellbeing using air pollution data, which required a significant number of sensors compared to HAR, which can be accomplished with an accelerometer [197]. Lots of new tiny edge hardware come with sensors built in. A common MCU device used for development is the Arduino Nano 33 BLE Sense. It contains a wide range of sensors, all packaged in a small form factor. A full list of sensors can be found in Table 2.2.

By choosing a board with built-in sensors, compatibility is guaranteed and footprint is reduced, as there is no need for the sensor to be added externally. However, most built-in sensors are generalised for a range of applications. Meaning that for specialised applications such as air quality monitoring, it is unlikely that a board has built-in particulate matter sensors [198]. However, sensors can be added externally. In fact, one of the highest specification boards in Table 2.2 is the Sony Spresense main board, which does not have built-in sensors. Therefore, due to its higher specification, boards such as the Sony Spresense main board may

be used to include higher-quality sensors or even as an edge gateway connecting to other boards providing data collection.

However, in some applications, it makes more sense to use a board with built-in sensors. For example, when processing accelerometer data, a built-in sensor helps minimise footprint and mitigate compatibility risks, as many development boards already integrate an accelerometer. Beyond hardware selection, data collection is another key consideration. Before training a supervised ML classification model, labelled data is required. Gradient-based learning, such as DL, relies on this data to evaluate generalisation and adjust the model to better fit the population distribution. While this is the goal, obtaining cleaned and labelled datasets remains a challenge. Benchmarking datasets provide developers with standardised data for testing models and algorithms, aiding in performance evaluation [65, 98, 199].

Furthermore, data labelling remains a laborious task that a human must oversee in some capacity. There are methods such as semi-supervised learning which aim to label data using a subset of previously labelled data, but these methods work best with big data applications. Research on labelling data on an MCU device is very limited with only [200] attempting it. Although this does not completely automate the process, it has the ability to simplify data collection on a target device.

2.12.2 Multi-Task Optimisation for Microcontrollers

MTL exploits pre-existing elements of a model to share components between several tasks. However, since these tasks run simultaneously, adding more tasks will quickly lead to a system that is too large for constrained systems. Perhaps, by restricting which tasks run at a particular time, constrained devices such as MCU devices could become more functional, as tasks share a common storage burden. This idea has been partly explored in the form of gated MTL. Gated MTL dynamically adjusts the contribution of each task to the overall learning process [201]. Nonetheless, there has been little exploration into how MTL can be optimised for resource-constrained devices such as MCU devices. Although, [202] shows that MTL and their *AdaShare* approach reduces the number of parameters

by at least 50% across 5 tasks.

MDL can include fixed adapters that are shared between all domains. Adapters are typically lightweight sets of layers within the model, but can take various forms such as residual adapters [203, 204]. Similarly to ResNet, residual adapters make use of residual/skip connections while also reducing dimensionality and maintaining the initial input data. The application of adapters has expanded with [205] employing **NAS** to identify the most suitable adapter within a search space comprising three adapter modules, in addition to determining the optimal position within a trunk model. Also, adapters can be used to attach sections of pre-trained models that are domain-agnostic, but often this does require the incoming input shapes to be identical. Perhaps, the idea of adapters could be applied to **tinyML** models to optimise smaller scale models to save parameters.

Context and multi-tasking can compliment each other particularly at the edge of the network, close to the data source. However, there are several limitations that until recently have halted the development of both together. Context itself is difficult to quantify and discretise for the purpose of model training. Context should be gathered in real-time before the contextual information becomes outdated. Thus, a device close to the data source could be valuable. Recent hardware advancements in **MCU** devices have allowed developers to use more complex software solutions. However, as aforementioned in Section 2.12.1, new hardware has its own limitations. With more capable devices and better software solutions, such as the introduction of , low-power devices can support heavier tasks, or more tasks such as **MTL**.

Crawshaw et al. [206] construct a survey and find that there is a general lack of theoretical understanding of **MTL** with **DNNs**. However, overall, the idea of models sharing information between them seems promising as more tasks migrate to the edge of the network. Due to the limited computation on **MCU** devices, **MTL** / **MDL** could improve the functionality of low-power devices while preserving reasonable task accuracy. More specifically, hard parameter sharing demonstrates opportunity as model size can be greatly reduced across tasks.

2.12.3 Echo State Network Optimisation

ESNs are able to effectively contribute to tasks if the correct reservoir conditions are met. However, in some cases, the reservoir can grow large and still struggle to learn complex representations. Similarly to a DNN, a deep ESN can be constructed by stacking several reservoirs [207]. The stacking of several reservoirs shows positive accuracy and low loss, but to the detriment of significant additional processing cost [208, 209]. Deploying ESNs to the edge of the network is challenging, but with further optimisation of inference, as partially addressed by [182], they may become suitable for widespread adoption. In particular, as more tasks aim to migrate to the edge of the network, perhaps the sharing of reservoirs will be more efficient than larger neural network layers, especially when processing intensive layers such as convolutional layers. In addition, the reduced training cost could pave the way for on-device learning.

While ESNs can enable on-device learning, they can also alleviate processing for multi-tasking in the same way. The reservoirs themselves are static beyond initialisation and do not rely on any input data during the initialisation phase. Perhaps, under the correct conditions, several tasks could use the same or similar reservoirs. This concept has not been explored and poses an interesting investigation, which is explored more in Chapter 5.

2.13 Discussion

tinyML as a research area is new and is growing rapidly. Most research surrounding tinyML has been published post 2019/2020 [8, 210]. tinyML can be applied to most systems and appears to be an important next step following the edge computing paradigm, although tinyML has some strict limitations as outlined in 2.6. After reviewing tinyML related literature, research gaps have been exposed, particularly optimisation methods that have not yet been migrated to the edge of the network. Optimisation methods such as quantisation and pruning are standard in EI and tinyML applications and will be considered in all experiments within this thesis.

As described in Section 2.12.1, the evaluation of algorithms for MCU devices

are often not deployed on-device. However, all of these challenges come before needing the [AI/ML](#) model. As much as [AI/ML](#) is the primary challenge due to model size and limitations on development, these other unexpected challenges mean that such development becomes much more taxing than one may anticipate. Therefore, the evaluation of new algorithms on-device is a weakness of previous work. Thus, deployment should be a high priority when answering [RQ 1](#).

Contextual conditions are complex to understand and implement. Transformers make use of attention and self-attention which is close to contextualising data. Deploying these large-scale systems to the edge of the network is not feasible. But this does not mean that context itself cannot have value for low-power devices. The literature relating to context often refers to information fusion or attention mechanisms.

Reviewing [MTL](#) and [MDL](#) outlined clear potential for parameter reduction in larger models spanning multiple input domains. Low-power devices could benefit from multiple models interleaved together through hard parameter sharing. This gap provides the basis for [RQ 2](#). The literature focuses on reducing the size of larger-scale models for energy efficiency, which also applies to [tinyML](#) research.

The ability to use non-invasive sensors to measure [HR](#) and [EDA](#) allows small devices to accurately determine stress levels in real-time and should be further utilised to detect stress. In doing so on-device, it may be possible to react to sudden stress increases and recommend specific actions to mitigate the compounding of additional stress. Although, physiological signals do not account for the context in which devices are used, as context can play a significant role in perceived stress levels, meaning additional sensors are required. In particular, physical activity can significantly alter human physiology, including [HR](#) and [EDA](#) [[211](#)], showing consideration is needed to not infer stress when exercising to avoid false positives. The ability to classify sensor data using small [MCU](#) devices offers many opportunities for real-world inference due to their small footprint. However, edge computing still faces many challenges such as the constrained nature of the devices and the high memory requirement of [AI](#) models. It is a demanding proposition to overcome many of these challenges; however, advances in software techniques such as quantisation and the use of context aim to improve the classification of models on limited hardware. Overall, while there has been much

research on effective DL models for stress classification, they have rarely been deployed on MCU devices, and there has been no consideration of how human activity context could improve the performance of stress modelling.

In general, the fusion of edge computing and AI has the potential to unlock a wide range of application areas, especially IoT. Although EI and tinyML are becoming a particularly coveted area of research, the deployment of AI to MCU devices poses several challenges that have been discussed in Section 2.12.1. The next chapter will focus on deployment of multiple models onto a single MCU device. In addition, Chapter 3 will provide a real-world case study using HAR to contextualise stress detection.

Chapter 3

Enabling Multimodal Multi-*model* Context-Aware tinyML Algorithms for Low-Power Hardware

Chapter Overview

Deep Neural Networks (DNNs) have shown their effectiveness in accurately classifying stress, but most existing solutions rely on the cloud or large obtrusive devices for inference. The emergence of tinyML provides an opportunity to bridge this gap and enable ubiquitous intelligent systems. In this chapter, a context-aware stress detection approach is proposed that uses a Microcontroller Unit (MCU) device to continuously infer physical activity to mitigate motion artifacts when inferring stress from heart rate and ElectroDermal Activity (EDA). Two DNNs are deployed onto a single resource-constrained MCU device for real-world stress recognition, with the resultant stress and Human Activity Recognition (HAR) models achieving 88% and 98% accuracy, respectively. The proposed context-aware approach improves the accuracy and privacy of stress detection systems while eliminating the need to store or transmit sensitive health data.

3.1 Introduction

In recent years, stress has become a major health concern affecting individuals across different age groups and professions. Stress can lead to a wide range of physical and mental health issues, such as anxiety, depression, and cardiovascular disease [212]. As a result, there is growing interest in developing effective stress management systems that can detect and mitigate stress in real-world environments, with many adopting artificially intelligent methods.

Advances in DL are resulting in ever expanding capabilities and applications ranging from voice assistants to autonomous driving. However, the successful deployment of a DNN relies upon two stages: training and inference. Much of the literature focuses on the development and training of DNNs but not the real-world inference, which is often due to the fact that DNNs are difficult to employ, as AI-based solutions commonly require a large amount of computational power. Thus, these systems often use cloud-based solutions or systems that are impractical for deployment.

In Chapter 2, DNNs have shown their effectiveness in accurately classifying stress, but most existing solutions rely on the cloud for inference. Edge computing has the potential to aid real-world stress recognition, as bridging these capabilities to embedded devices can reduce the amount of data stored and transmitted, hence increasing the privacy of sensitive health data. Photoplethysmography (PPG) sensors enable the non-invasive monitoring of Heart Rate (HR). Automatic analysis of PPG has rendered it valuable in clinical and non-clinical settings. However, tracking HR with PPG is difficult due to motion artifacts, which are major causes of signal degradation, as they obscure the location of the heart rate peak in the spectra [213]. Therefore, context recognition in the form of detecting physical activity is essential to improve the performance of stress detection. By integrating exercise recognition, stress detection systems can become more reliable by mitigating any motion artifacts from the signal.

HAR is the task of inferring actions carried out by a person and is a popular research topic with the potential to improve healthcare [214, 215], which can provide context to allow for more accurate stress detection. However, to implement such systems on a large scale, efficient and low-power intelligent algorithms that

3. Multimodal Multi-*model* tinyML Deployment

can run on low-cost, resource-constrained MCU devices should be developed. The context can play an important role in the accuracy of systems [216]. Context itself is ambiguous and requires clarification in each application in which it is used. However, context has previously been used to aid in inferring road traffic conditions such as congestion using context such as school breaks and weather [217]. Although the concept of applying context to the edge of the network is relatively unexplored.

AI on the edge research is still in its infancy and the trade-off between classification accuracy and embedded device constraints must be evaluated. Most edge computing research frequently uses larger devices, such as Raspberry Pis or smartphones [45, 48]; however, many real-world applications, such as wearable devices for stress recognition, require smaller electronics as they are to be worn on-body. These tiny MCU devices offer an ideal opportunity for real-world, on-device inference. This chapter presents the deployment of a novel multimodal, multi-*model* approach for context-aware stress detection, where stress inference occurs only if no physical activity is detected by a separate classification model. All processing is performed on-device using a low-power, low-cost, and resource-constrained MCU. The proposed tinyML approach can be applied to both models and will demonstrate the potential of using multiple models while also leveraging several input modalities. The contributions of this chapter are twofold.

- The novel combination of two models to provide real-time contextual HAR information to improve the accuracy of stress inference.
- The deployment of the context-aware multi-*model* approach on a low-power, resource-constrained MCU device for real-world on-device inference.

The remainder of this chapter is organised as follows; Section 3.2 explores the methodology, Section 3.3 outlines the results, and Section 3.4 provides a discussion based on the results.

3.2 Methodology

This work outlines a tinyML multimodal multi-*model* approach combining a HAR classifier with stress detection. The multi-*model* context-aware system consists

3. Multimodal Multi-*model* tinyML Deployment

of the following key components:

- **Inertial Measurement Unit (IMU)** sensors - The built-in **IMU** of the Arduino Nano 33 **BLE** Sense is used to collect three-axis accelerometer data for **HAR**.
- **HR** and **EDA** sensors - These external sensors connect to the **MCU** device to provide physiological signals for stress detection.
- **HAR** model - A **Convolutional Neural Network (CNN)** classifies each incoming accelerometer sample as either resting or active.
- Stress detection model - A separate **CNN** acts on physiological data to classify stress, but only once resting state is inferred.
- **MCU** device - The Arduino Nano 33 **BLE** Sense provides computing power, sensor connectivity and executes both models.

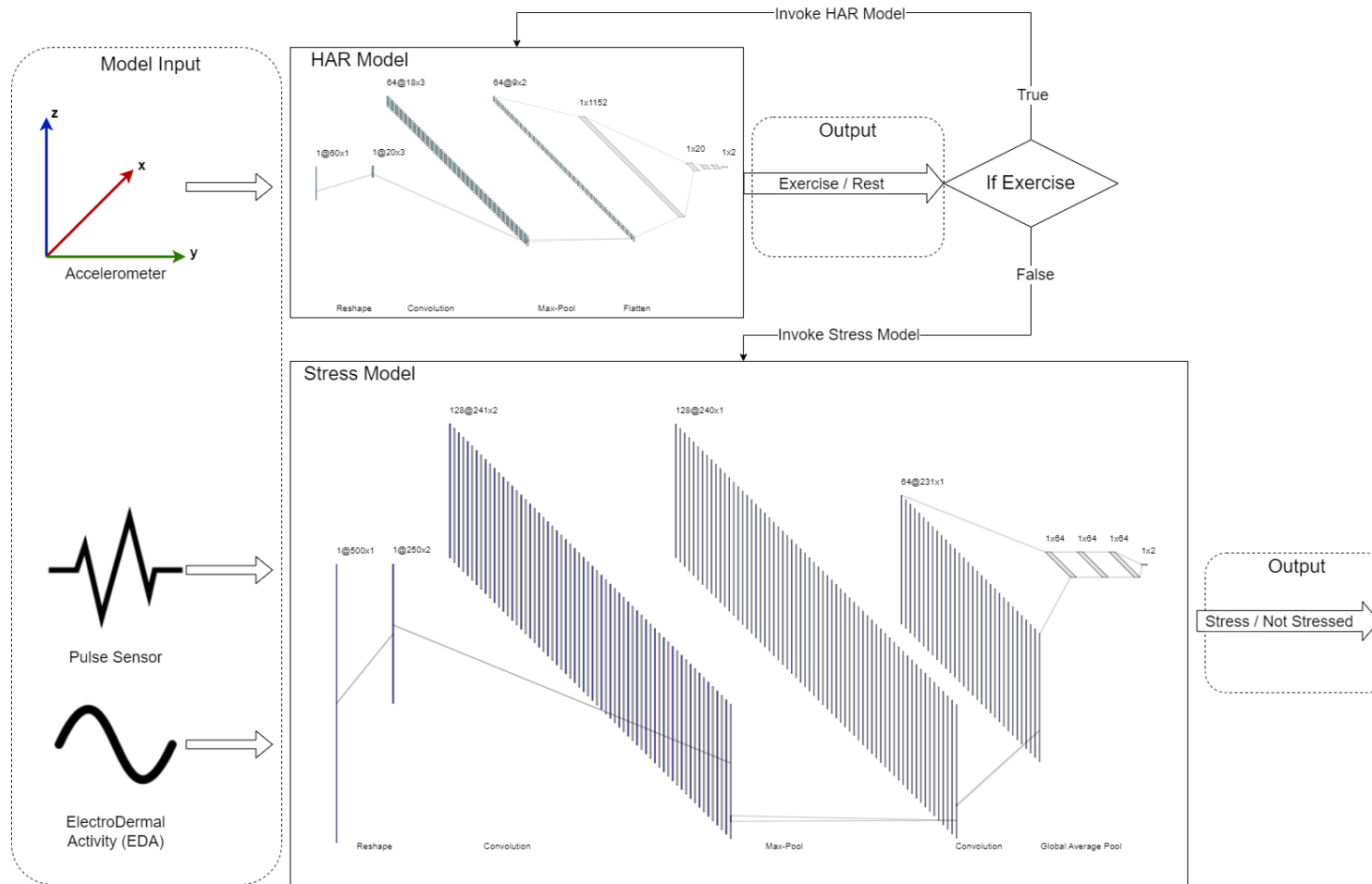


Figure 3.1: The proposed system containing the HAR model (top) and stress model (bottom), along with the internal logic. The HAR model constantly infers exercise or not exercise. When exercise is not detected, the stress model is invoked. Data is continuously collected for both models.

3. Multimodal Multi-*model* tinyML Deployment

The [IMU](#) continuously collects motion data from the accelerometer input which is streamed to the [HAR](#) model for low-latency classification as resting or active. Simultaneously, physiological sensors acquire data. Only when the [HAR](#) model predicts the resting state is the stress model triggered to analyse the physiological data and output a stress prediction to reduce motion artifacts and increase accuracy. Both models run independently on the [MCU](#) device.

Evaluation is performed by simultaneously collecting all sensor data streams but withholding the physiological data from the stress model until inactive periods are detected, mirroring real-world operation. The same datasets are used for both training and evaluation.

3.2.1 Datasets

In order to complete the multi-*model* context-aware approach, a dataset was required for both the [HAR](#) and stress recognition to train the classifiers.

In this chapter, the [Wireless Sensor Data Mining \(WISDM\)](#) dataset is used [218]. The dataset contains three-axis accelerometer data sampled at 20Hz for walking, jogging, climbing upstairs, climbing downstairs, sitting, and standing. Data were collected from 29 participants using an Android phone in their front trouser leg pocket.

Preprocessing the data was required as the data was recorded using an Android phone, and the timestamp was recorded using the phone's up-time. Some rows of data had a timestamp, but no attached data. These rows were removed during the cleaning process.

Due to the requirement of only needing to infer when the user is performing physical activity or resting, the classes were condensed into exercise and rest. Walking, jogging, climbing upstairs, and climbing downstairs were combined into the exercise class, while standing and sitting were combined to form the rest class.

For the stress dataset, a lab-based stressor experiment was conducted in which participants' stress response was stimulated using the Montreal stress test [219]. This experiment induced stress in 20 healthy participants aged 18–50 between June and September 2019 as approved by Nottingham Trent University Human Ethics Board, application number 600. Participants wore hand-held non-invasive

3. Multimodal Multi-*model* tinyML Deployment

sensors on their fingers. The sensors recorded [HR Beats Per Minute \(BPM\)](#), raw [HR](#) amplitude, [Heart Rate Variability \(HRV\)](#), and [EDA](#), each sampled at 30Hz to collect physiological data while experiencing relaxed and stressed states of mental wellbeing.

To allow the effects of stress and mental arithmetic to be investigated separately, the experiment had two test conditions: a relaxed control condition and a stressed experimental condition. All participants completed both conditions. Each participant was briefed before completing a 3-minute rest period in which they looked at a static computer screen where no tasks were displayed. This ensured participants were relaxed before starting the control condition. They then completed the control condition for 3 minutes whereby a series of mental arithmetic questions were displayed that the participants answered followed by another 3-minute rest period. The participants then completed the second stressed experimental condition in which the difficulty of the questions from the control condition increased. The task time limit was also adjusted to be 10% less than the average time taken to answer the questions during the control, taking it just beyond the individual’s mental capacity. The time pressure along with the progress bar showing their progress compared with an artificially inflated average were designed to induce stress during this condition. A similar number of samples were collected from both relaxed and stressed data, helping to reduce bias in the classification model. The final experiment dataset resulted in a total of 417,251 sensor data samples when relaxed and 475, 232 data samples when stressed. The dataset consists of physiological sensor data, including [HR](#) (mean: $\mu = 79.4$ [BPM](#), standard deviation: $\sigma = 11.6$ [BPM](#)), [HRV](#) ($\mu = 773.8$ ms, $\sigma = 152.6$ ms), and [EDA](#) ($\mu = 320.4$ k Ω , $\sigma = 158$ k Ω). This stress dataset is used in this chapter and throughout the thesis.

3.2.2 Hardware

The Arduino Nano 33 BLE Sense¹ is a small form factor, low-power, and cost-effective MCU device. It is based on the Arm Cortex-M4 32-bit processor², which is an energy-efficient MCU device that is well-suited for a wide range of applications. Although the Cortex-M4 processor is capable of performing advanced calculations and processing, it has limited processing power. The limited processing power means that the AI capabilities of the board are limited by the complexity and size of the ML models that can be run on the MCU device.

In addition to its processing power limitations, the Arduino Nano 33 BLE Sense also has limited memory with only 256KB of SRAM, which can make it difficult to store larger samples and complex models. The inability to store complex models means that methods such as model compression and quantisation are required to reduce the size of the models and make them more suitable for deployment. Therefore, Lite Runtime for Microcontrollers (LiteRT μ) (formerly TensorFlow Lite for Microcontrollers (TFLite μ)) has been used to develop small models capable of fitting on the MCU device. While there are other MCU devices that are more powerful, the Arduino Nano 33 BLE Sense is an extremely low-cost MCU device enabling much wider deployment and also has extensive support for LiteRT μ . Furthermore, the Arduino Nano 33 BLE Sense has many built-in sensors as shown in Table 2.2.

A non-invasive sensor-based approach presents the most significant opportunity to assess stress, as sensors can be easily connected to MCU devices and used inconspicuously in the real-world.

HR sensors are commonly used within wearable computing systems, as they can be embedded in a wide range of devices due to their small footprint and provide insights into the autonomous nervous system. Therefore, the same PPG sensor used in the Montreal stressor data collection experiment was connected to the MCU device to measure HR.

Impaired parasympathic activity including stress often results in reduced HRV

¹Available online, <https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf>, last accessed 03/03/2025

²Available online, <https://developer.arm.com/Processors/Cortex-M4>, last accessed 03/03/2025

3. Multimodal Multi-*model* tinyML Deployment

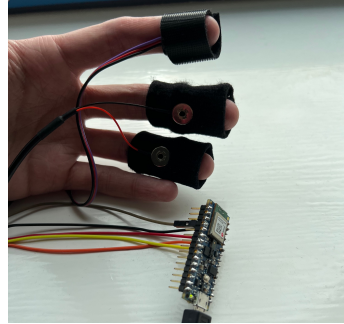


Figure 3.2: Person wearing the HR and EDA sensors connected to an Arduino Nano 33 BLE Sense.

- variation in time between heartbeats - which can be measured using non-invasive, low-cost PPG sensors. Similarly, the same EDA sensor from the data collection experiment has been connected to the MCU device. EDA is often used to train affective models to classify stress as it is directly related to the sympathetic nervous system, which controls rapid involuntary responses to dangerous or stressful situations [220].

Finally, motion data measures movement through accelerometers, gyroscopes, and magnetometers which can be used to measure physical activity. The Arduino Nano 33 BLE Sense has an LSM9DS1¹ IMU which includes a three-dimensional digital linear acceleration sensor. This built-in accelerometer has been used for real-world HAR inference [221].

3.2.3 Classification Features

Feature extraction is a crucial process when developing classification models. However, advances in DL have resulted in the ability to classify raw sensor data. However, the memory limitations of the Arduino 33 BLE Sense limit the ability to extract complex features.

Due to the constrained nature of the MCU device, the only feature used from the PPG sensor was the raw HR amplitude along with the raw EDA values. Additional features such as BPM and HRV were examined, but when the model

¹Available online, <https://www.st.com/resource/en/datasheet/lsm9ds1.pdf>, last accessed 03/03/2025

3. Multimodal Multi-*model* tinyML Deployment

was trained using these features, it was too large to fit on the MCU device in addition to the HAR model due to the limited 1MB of flash memory. Similarly, for the HAR model, the raw three-axis accelerometer values were used from the Arduino Nano 33 BLE Sense on-board IMU sensor.

3.2.4 Network Architecture

DNNs can provide higher accuracy compared to some classic ML approaches, and their sample-by-sample streaming capabilities better match real-time edge processing. Previous research shows the benefits of using CNNs for both HAR and stress detection [45]. While Long Short-Term Memory (LSTM) networks are frequently used to classify time-series data, and even HAR [154, 222], previous work shows that CNNs can outperform LSTMs for time-series classification, such as stress recognition [117]. Furthermore, testing showed that the resultant model size remains smaller when using one-dimensional CNN, making them ideal for tinyML applications.

A one-dimensional CNN was selected for the HAR model due to its high performance and smaller model sizes compared to an LSTM. A CNN is composed of convolutional layers that employ a filter to slide over the one-dimensional time series data. Although, due to LiteRT μ compatibility issues, two-dimensional convolutional layers have been used with a one-dimensional filter and stride, thus simulating a one-dimensional convolutional layer. The CNN architecture consists of a single convolutional layer, a dropout layer with a rate of 0.2 to prevent overfitting followed by three dense layers, and a softmax output layer. The high accuracy and automated feature learning of CNNs makes it better suited for generalised mobility detection, despite its higher complexity compared to most other LiteRT μ compatible DL layers.

Similarly, a one-dimensional CNN was selected for the stress detection model. The original input data is partitioned into fixed length segments. This data is segmented over an overlapping sliding window with an experimentally chosen window size of 256 samples and a step size of 24, after testing various window sizes ranging from 16 to 256. The network architecture consists of two one-dimensional convolutional layers, followed by max-pooling operations. Batch normalisation

3. Multimodal Multi-*model* tinyML Deployment

layers are incorporated, as well as a dropout layer with a rate of 0.2 to avoid overfitting, prior to the softmax activation function. As in the [HAR](#) model, a two-dimensional convolutional layer has been practically implemented but simulating a one-dimensional behaviour to ensure compatibility with the deployment library [LiteRT \$\mu\$](#) .

To reduce model size, the stress model was tested using only one convolutional layer. However, using only a single convolutional layer resulted in a significant accuracy sacrifice of approximately 27%. Thus, two convolutional layers must be in the network architecture to achieve significant accuracy.

3.2.5 Model Quantisation

The Arduino Nano 33 [BLE](#) Sense is a powerful and versatile [MCU](#) device that is capable of performing a variety of [ML](#) tasks on-device. However, due to its limited memory, it is often necessary to use techniques such as model quantisation to reduce the size of [ML](#) models and make them more suitable for deployment.

Two commonly used policies for [Post Training Quantisation \(PTQ\)](#) are *int8* and *float16* quantisation, which are both methods to reduce the precision of floating-point numbers used in [ML](#) models. In a standard [ML](#) model, floating-point numbers are typically represented using 32 bits, which can make the model large and memory intensive. Using *float16* quantisation, the number of bits used to represent each floating-point number is reduced to 16 bits. For the *int8* method, the number of bits is reduced from 32 to 8, thus a maximum model reduction by a factor of 4. However, using *int8* quantisation means representing the model parameters as full 8-bit integers, truncating them at the decimal place. These techniques significantly reduced the size of both models, making them more suitable for deployment on devices with limited memory, such as the Arduino Nano 33 [BLE](#) Sense.

It was necessary to ensure that both models could fit onto the target board, namely the Arduino Nano 33 [BLE](#) Sense. To do so, [PTQ](#) of varying policy (*int8*, *float16* and *float32* / unquantised) were tested. By introducing a smaller space for the data to occupy, the information incurs a rounding error from the original value. Typically, as a consequence of rounding error, the accuracy of a model is

reduced. An acceptable accuracy sacrifice varies depending on the application.

3.3 Results

An iterative process was adopted during development, which meant setting some performance targets. The results should be more significant than a guess, as this demonstrates that the model performs beyond randomness. Therefore, the accuracy should be more significant than one standard deviation in a normal distribution. As both models are binary classifiers, this puts the target accuracy at 68%.

The stress model was initially tested using *float16* quantisation, but this meant using 92% of the available storage. Therefore, to ensure that both models could fit on the MCU device, they each required quantisation, with the HAR model requiring *float16* quantisation and the stress model *int8* quantisation. Therefore, the accuracy of the HAR model was only reduced by 0.34% while the latency was reduced from 102 to 70 ms. Similarly, *int8* quantisation had no impact on the accuracy of the stress model, but decreased latency and reduced memory usage. Therefore, a minor accuracy sacrifice was deemed appropriate to demonstrate two models on a single MCU device while still performing at a high level of significance. Upon the deployment of the models, the stress and HAR models achieved an accuracy of 88% and 98%, respectively, both using hold-out validation with a 20% test split. A further breakdown of the performance metrics for each model can be found in Table 3.1. The resulting models were able to detect both physical activity and stress with high precision. This enabled the final context-aware approach to be deployed on the Arduino Nano 33 BLE Sense, where physical activity is continuously inferred. Thence, once no physical activity is detected, stress is inferred.

Robust evaluation of the multi-*model* context-aware system requires validating both the individual model accuracies and the overall workflow. Hold-out validation allows one to report performance on unseen data in an unbiased manner. However, evaluating the models separately does not fully validate the real-world operation of the system. Thus, end-to-end validation with live sensor streams was also performed. The HAR model classifies the incoming motion data and

3. Multimodal Multi-*model* tinyML Deployment

Table 3.1: Performance metrics for the stress (top) and physical activity (bottom) classification models. For each, the precision, recall, F1 score and accuracy is collected as well as the latency and header file size.

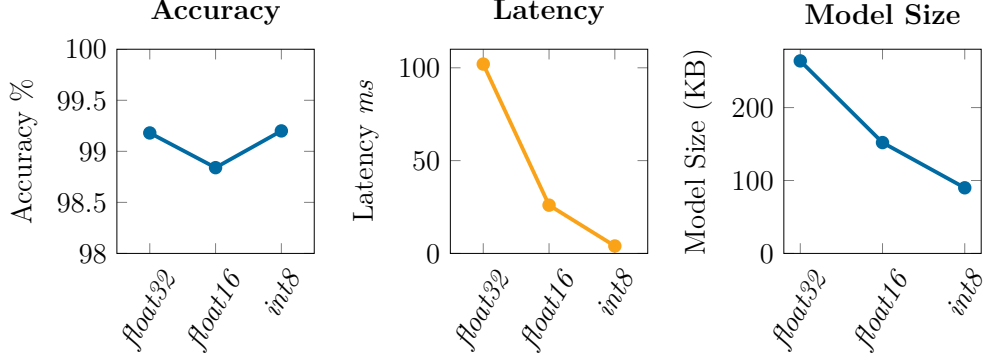
| Stress Model | | | |
|--------------|----------------------|--------|-------|
| Class | Precision | Recall | F1 |
| Non-Stress | 0.86 | 0.95 | 0.90 |
| Stress | 0.94 | 0.83 | 0.88 |
| Latency | Accuracy | | 0.88 |
| 3642ms | Model Size (.h file) | | 1.1MB |

| HAR Model | | | |
|-----------|----------------------|--------|-------|
| Class | Precision | Recall | F1 |
| Rest | 0.97 | 0.92 | 0.94 |
| Exercise | 0.99 | 1.00 | 0.99 |
| Latency | Accuracy | | 0.98 |
| 26ms | Model Size (.h file) | | 152KB |

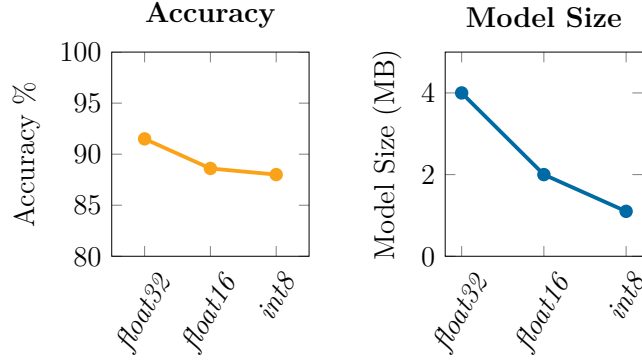
only triggers stress classification during periods of inactivity. The end-to-end approach ensures that models are evaluated with live data capturing noise and artifacts. The classifiers were run concurrently as designed rather than separately to validate the activity context modelling and switching logic.

The end-to-end validation of the multi-*model* system with live sensor data streams confirmed the expected performance based on the hold-out testing of individual models. The [HAR](#) model operated with high accuracy on motion data, properly classifying periods of activity versus inactivity. During these classified inactive states, the stress detection model was selectively triggered as intended and output predictions within the expected accuracy range reported during hold-out testing. The overall workflow of selective stress classification based on activity context was successfully validated. No problems were encountered with concurrency, data flows, or model switching logic. The latency remained low and suitable for real-time operation. This end-to-end validation on live data from multiple sensors confirmed that the integrated system performs as designed based on the individual model metrics.

3. Multimodal Multi-*model* tinyML Deployment



(a) HAR Model metrics



(b) Stress Recognition Model metrics

Figure 3.3: Graph of performance metrics over varying quantisation policies of the (a) [HAR](#) model and (b) the stress model. Latency was unavailable for *float32* and *float16* quantisation for the stress model due to its large model size.

3.4 Discussion

In this chapter, the feasibility of running a context-aware [HAR](#) model that triggers a stress detection model, all independently performed on an Arduino Nano 33 [BLE](#) Sense, has been examined. The results confirm the potential to run two classification models on a resource-constrained, low-cost [MCU](#) device. Each of the models achieved high accuracy (0.98 and 0.88 for activity recognition and stress detection respectively), demonstrating the potential for real-world inference. This demonstrates similar results for [HAR](#) [76, 223], however, the results demonstrate improved performance for rest compared to [76] that achieved 67.3% for the respective idle class, demonstrating the benefits of binary classification.

3. Multimodal Multi-*model* tinyML Deployment

Furthermore, many studies do not report on key metrics such as latency [158] or model size [76] which are key considerations when deploying on MCU devices. Most of the research at the edge for HAR relies on smartphones where the model sizes can be larger, such as 16MB for an activity recognition CNN [155], which is not comparable to the deployment on resource-constrained MCU devices. Context plays an important role in stress detection, as the activity in which an individual is engaged can significantly impact physiology. In particular, physical activity can drastically alter both HR and EDA, which are common biomarkers for stress detection. Therefore, this work combining a HAR classification model with stress detection on an MCU device provides a novel approach for context-aware stress detection that can help improve the reliability of real-world stress inference.

However, limitations were encountered; in particular, the limited memory of the Arduino Nano 33 resulted in the use of simpler models. Initially, a CNN with four convolutional layers was used for stress detection that achieved 95.6% accuracy, a 7.6% increase over the two layer CNN; however, this used 97% of the MCU device’s available dynamic memory when using *int8* quantisation. Therefore, the number of convolutional layers was reduced to two layers, reducing the model size by 67%. To further reduce the size of the stress model, the model was trained using only one convolutional layer; however, using one convolutional layer reduced the accuracy to 61.8%, a 26.7% reduction in accuracy. The model was tested using *float16* quantisation with two convolutional layers but this used 92% of the MCU device’s available storage. Therefore, two convolutional layers were used to train the model and *int8* quantisation was used to ensure a suitable model size. The use of LSTM networks was explored during model experimentation, but the models performed worse and were larger than comparative CNNs. Although unusual, the poor performance of LSTM networks compared to CNNs exists in related literature [224]. Due to the size of LSTM models and poor performance, an LSTM-based model was not considered within the results. Additionally, only raw HR and EDA data were used to classify stress. Additional features, such as BPM and HRV, made the model too large to deploy to the MCU device with the additional HAR model, demonstrating the trade-offs required for real-world deployment. Wearable sensors are assumed to offer clean,

3. Multimodal Multi-*model* tinyML Deployment

unaffected signals and consistent placement among participants. For modelling, it is assumed that the CNN architectures generalise well to new data based on the validation results. The model architectures are constrained to relatively simple one-dimensional CNNs given the hardware limitations. Only a single context factor of physical activity was used to trigger stress classification rather than exploring multiple contextual variables.

Figure 3.3 illustrates that the latency and model size of the HAR model are reduced using *float16* and *int8* quantisation, while the accuracy remains uniform. This demonstrates the benefits of quantisation when deploying onto low-power hardware. However, the HAR model was limited to a minimum of *float16* quantisation due to the inconsistent nature of the WISDM dataset [218]. The expected LSM9DS1 IMU input data would range between $[0, 1]$, however, the training data were rarely between these values. Most training samples had a high range and interquartile range, implying a need for complex scaling for reliable integer results. Therefore, if *int8* quantisation was used, this could incur a high preprocessing cost in the final implementation.

These limitations demonstrate the challenging nature of running two classification models on a single MCU device. While previous work has demonstrated the deployment of individual ML models to MCU devices, to the author’s knowledge, this is the first approach to successfully deploy multiple ML models on a single MCU device, structured in a context-aware inference pipeline. Specifically, one lightweight model (HAR model) provides real-time context and dynamically informs the operation of a second, more complex model (stress model).

Therefore, this chapter introduces a novel paradigm in the tinyML domain: on-device model orchestration, where multiple specialised models operate in coordination within tight memory and compute constraints. The technical challenges were considerable, including maintaining low latency, ensuring memory-efficient operation across models, and implementing selective execution based on real-time context, all within the constrained runtime environment of a MCU device. To achieve this, optimisations such as quantisation, were employed.

This approach demonstrates the process required to deploy multiple ML models to many real-world applications and could be applied beyond stress detection. It highlights the need for modular, context-driven on-device intelligence, which

3. Multimodal Multi-*model* tinyML Deployment

can remain functional despite its hardware target. In the future, new [MCU](#) devices with increased memory may simplify deployment with increasingly complex models.

Both models achieved accuracy comparable to previous work for both the stress and [HAR](#) models. Thus, this chapter outlines a key contribution in using multiple [DL](#) models with one informing the other, both of which are deployed to a low-cost constrained Arduino Nano 33 [BLE](#) Sense. However, what if more models were added? An [MCU](#) device lacks the storage requirements to store such a system. The following chapter provides an answer to this question by developing a layer-sharing approach inspired by [Multi-Task Learning \(MTL\)](#).

Chapter 4

A Novel Layer Sharing Approach for Multi-Task Learning on tinyML-Enabled Hardware

Chapter Overview

In the previous chapter, multiple [Convolutional Neural Networks \(CNNs\)](#) were deployed to a single resource-constrained [Microcontroller Unit \(MCU\)](#) device. [Human Activity Recognition \(HAR\)](#) data was used to contextualise stress data exemplifying an increasing need to deploy complex models on devices with limited resources. However, as shown in Chapter 3, multiple complex neural networks often exceed the limited memory and compute capabilities of [MCU](#) devices. Thus, [Multi-Task Learning \(MTL\)](#), an approach that combines models to share parameters between several tasks, has inspired a new layer-sharing approach. This chapter presents the [tiny Inception Module \(tIM\)](#), a novel layer-sharing approach that combines the idea of [MTL](#) and [tinyML](#) software optimisations. Therefore, the [tIM](#) allows layers to be shared between tasks of different architectures. This chapter introduces the [tIM](#) and presents a case study that demonstrates the efficiency of the [tIM](#) for application to stress detection and [HAR](#). Furthermore, this chapter demonstrates a single [tIM](#) for the sharing of network layers between a [HAR](#) model and a stress detection model of differing architectures. The stor-

age savings of both models is 28.3% compared to Chapter 3 [1] and 47.8% when deploying both models without layer-sharing optimisation. The **tIM** has shown an impressive ability to reduce redundancy while only acquiring a 4% accuracy sacrifice on the incepted stress detection model.

4.1 Introduction

In recent years, the interest of processing **ML** algorithms on-device has become a focal point in industry as well as research [210, 225]. With on-device processing boasting privacy and latency gains, it is clear that enabling **ML** on-device is currently a desired field of research. Software optimisations have increased what limited hardware is capable of, helping their range of possible applications grow. Some optimisations, such as quantisation [79, 81] and pruning [92, 93], are popular among software model optimisations. However, other methods, such as parameter sharing, can help reduce the storage of some larger models.

Optimisation methods aid in the deployment of **ML** algorithms; however, some devices, such as health-related wearables, may seek to solve several tasks. Perhaps to recognise a patient who is a fall risk as to when they have fallen [226], or even identify if and when they experience prolonged periods of stress [227]. The idea of classifying several tasks simultaneously while sharing parameters among tasks is known as **MTL** [228–230]. Although **MTL** is often not considered for on-device deployment because tasks run concurrently, increasing the computational load of the device. Nevertheless, sharing information between tasks ensures a reduction in redundancy within a system as the storage burden is reduced. The reduced storage burden is particularly valuable when considering **tinyML**; however, sharing the maximum information between tasks typically requires similar model architectures. Most **DL** models have their own architecture and input shapes, thus maintaining their specificity. Although, some tasks may share some similar feature extraction/abstraction methods.

Thus, combining the idea of **MTL** and **tinyML** software optimisations, this chapter introduces a new layer sharing approach, the **tIM**. The **tIM** unlocks multi-tasking on tiny, low-power (5V) **MCU** devices with significantly reduced storage overhead. The **tIM** makes use of similar tasks and small transformative layers to

have smaller neural networks work together to classify a larger, yet unique new task. More specifically, **tIM** will make use of pre-trained layers to entangle multiple neural networks to reuse layers. In this chapter, the use of a single **tIM** is demonstrated for the application to stress detection combined with **HAR**, mimicking a real-world application. In particular, the contributions in this chapter are twofold.

- Introducing the novel **tiny Inception Module (tIM)** for the sharing of neural network layers between **DL** models of different architectures. A new transformation adapter layer is used to enable the repurposing of layers for additional tasks.
- A case study demonstrating the efficiency of the **tiny Inception Module (tIM)** on stress recognition and **HAR** is presented.

The proposed method has the potential to significantly improve the development of machine learning algorithms on embedded hardware, particularly on low-power **MCU** devices. By reducing the storage occupancy of **DL** models through the deconstruction of similar models into shared and specialised components, multiple complex networks can be deployed on **MCU** devices without a significant increase in resources usually required per model. The shared layers reduce redundancy, while the specialised layers retain accuracy on each task. Reduced redundancy enables sophisticated multi-*model* intelligence on resource-constrained **MCU** devices such as those presented in Chapter 3 [1], resulting in more accurate and efficient **ML** algorithms. Increased accuracy and efficiency are essential for most applications, particularly in fields such as healthcare [231] and transportation [232].

The remainder of this chapter is organised as follows, Section 4.2 outlines the motivation behind the **tIM** in detail, Section 4.3 introduces the proprietary transformer adapter and Section 4.4 highlights the overall architecture of a **tIM**. Section 4.5 onwards demonstrates the use of a single **tIM** within a real-world application, with Section 4.6 presenting and discussing the results.

4.2 Motivation

Much like in human behaviour, during classification tasks people do not consider everything they have ever seen. There is a process of eliminating as much of the search space as possible to maximise the chance of correct classification while minimising computational effort. However, in the context of [DL](#), such a task would be very computationally intensive.

Drawing inspiration from the human approach, the proposed methodology breaks down the problem into specialised models that share parameters where beneficial, instead of training a single large model that covers all possible tasks. This concept overlaps with those introduced in Chapter 2 in [MTL](#) and [Multi-Domain Learning \(MDL\)](#). In [MDL](#), lower-level visual features, such as edges, are useful in many vision-based domains. Thus, by sharing these parameters, redundant computation and storage is avoided. Then, deeper layers can diverge to capture task-specific representations.

The [MTL](#) approach considers the computational burden and attempts to make use of as many pre-existing weights as possible, like in hard parameter sharing [228]. Therefore, the memory footprint and storage footprint is reduced compared to a single large model containing all the classes of the smaller models. As outlined in Section 2.9, hard parameter sharing is a method of layer sharing that shares large sets of parameters, mostly in layers. Although, [MTL](#) / [MDL](#) have not been optimised for microcontroller deployment. [MCU](#) devices typically cannot handle processing multiple computational tasks simultaneously, especially as the number of tasks increases. Therefore, the [tIM](#) makes use of pre-existing layers to extract features. Each shared layer will be pre-trained and frozen while the newly built model is training. The key benefit of using a [tIM](#) is that by storing large chunks of a pre-existing model within another could mean a significant reduction in parameters in memory leading to a decreased storage burden, while achieving comparable accuracy. With an increase in depth and feature extraction, a model could yield competitive performance metrics while containing key information for another task.

Hard parameter sharing and [MDL](#) techniques can share lots of information between tasks or domains. Although this typically implies that there is a larger

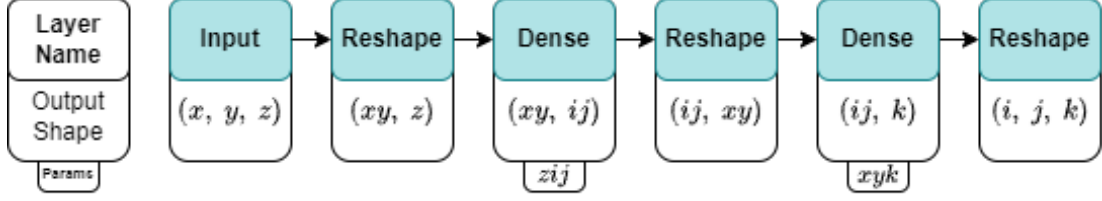


Figure 4.1: Architecture of the transformation adapter layers mapping a three-dimensional input to another three-dimensional output. The transformation block contains only layers compatible with [Lite Runtime for Microcontrollers \(LiteRT \$\mu\$ \)](#). Each of the layers contains its output shape and the number of parameters said layer incurs (if applicable).

pre-trained model acting as a backbone model. Adapters can then help differentiate from the shared components to the domain/task specific representations. A drawback to this is that, on [MCU](#) devices, one may seek to classify data across different modalities, meaning that input dimensions are unlikely to be similar. Therefore, in this chapter, a different take on [MTL](#) using a transformation adapter layer is used to embed a model within another for multimodal applications.

4.3 Layer Transformation

When a model is incepted into another, it is likely that the input/output dimensions of the layers will no longer align. Therefore, some transformation is needed to allow these dimensions to expand or shrink without losing significant information. Drawing upon [MDL](#) adapter modules [203, 205], the transformation adapter is proposed. The transformation adapter is a small sequential architecture, of which can be seen in Figure 4.1.

The architecture in Figure 4.1 incurs further parameters to the model due to the addition of two dense layers. As [CNNs](#) are common in image classification, and sensor data collected over a window encompass three-dimensions, Figure 4.1 is modelled for a three-dimensional input to another three-dimensional output. For said three-dimensional input, (x, y, z) , and three-dimensional output, (i, j, k) , the number of parameters incurred by the transformation layer, θ , is as follows

$$\theta = zij + xyk. \quad (4.1)$$

The above implies that any higher input or output dimensions will cause the layer to have a significant number of parameters. The transformation layer facilitates attaching a new model within an existing model, meaning at most for each **tIM** within another model two transformation layers will be required. The transformation layer could be made up of two-dimensional convolutional layers in place of the dense layers, but this adds a small number of parameters. Only layers compatible with **LiteRT μ** are used in constructing the transformation adapter layer, hence there is no use of a permute layer to transpose the dimensions of the layer. The usage of transformation layers should therefore be consistent with dense layers. They incur a parameter cost and should not immediately cascade from a large layer into a low number of neurons. Transformation layers themselves do incur a depth cost as this layer alone consists of six layers. However, most of these layers are low-cost, with the exception of dense layers depending on the incoming and outgoing dimensions. Nonetheless, the transformation layer architecture ensures an effective and efficient transformation between the current dimension and the target dimension.

4.4 Architecture and Setup

The **tIM** requires careful construction and entanglement of the target models. Suppose that you have m models, each with n_m layers, where n_m denotes the number of layers in model m (which implies varying architectures). First, a suitable candidate for sharing must be chosen, and then trained in isolation. Once trained and fine-tuned, other models can be constructed and the layers shared. Identifying where layers should share can be task-specific. For maximum storage savings, the layers with the highest parameters would be shared / replaced. However, recall that the transformation layer incurs a cost. If not careful, much like a poorly placed dense layer, you will not only incur a heavy storage burden, but the model may not learn well due to the majority of parameters being in a single layer. An in-depth real-world example of choosing layers to share is explained in Section 4.5.3.

Although unlikely, if there are layers that share the same output shape, then the initial transformation adapter layer is not required before the incepted layers.

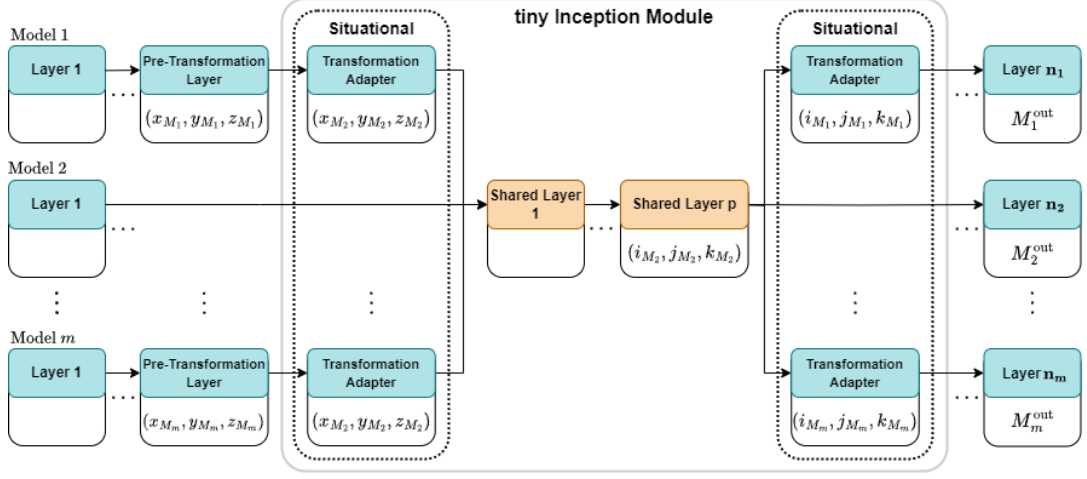


Figure 4.2: A general overview of the **tIM**. The layers in blue are model specific, whereas the gold layers are shared. The transformation adapter layers, although blue, are part of the **tIM** as they are required by the specific model for transformation into the gold layers (or vice versa for the right-hand transformation adapter layers). The output shapes are listed within layers where they are known in a generalised form.

Also, the second transformation adapter layer is only required if the returning shape is not one-dimensional. For example, if some feature extraction layers are shared and return a three-dimensional output, a transformation adapter layer will be required to allow the layer dimensions to match.

Each other model can be built up using the pre-trained layers to be shared, enclosed by the transformation adapter layers (if required), followed by their final layers. Upon following this methodology, an architecture similar to that in Figure 4.2 will be achieved for m models.

4.5 Case Study: Stress Detection with Human Activity Recognition

In this section, the effectiveness of a **tIM** within a real-world application is presented in a stress detection and **HAR** application. These datasets have been used to achieve an accuracy of 88% and 98%, respectively, on an Arduino Nano 33 **BLE** Sense in Chapter 3 [1], which will be the basis for comparison. The stress

detection dataset is the same as used in Chapter 3. Thus, the stress dataset is two-class, whereas the HAR dataset contains six classes from the publicly available Wireless Sensor Data Mining (WISDM) HAR dataset [218]. These six classes are as follows; *Walking, Jogging, Upstairs, Downstairs, Sitting, and Standing*.

Stress is a complex emotion to classify due to its individuality [117]. Therefore, stress recognition tasks are important to detect and notify the user when they may be experiencing stress. Several studies have been conducted with a range of ML classifiers. [233] use an Long Short-Term Memory (LSTM) module to classify stress using physiological data, mobile phone usage data and mobility pattern data, achieving 81.4% accuracy. A CNN was used by [234] with image encoding to enable stress classification achieving 98.5%. [159] collected data and developed a system to classify three levels of stress while driving in Boston. They determine that heart rate variability and skin conductivity are the two most correlated features when classifying stress, achieving 97.4% accuracy.

The WISDM dataset [218] has been evidenced to achieve good performance among several research articles. [235] achieved 98.2% using two fine-tuned CNNs, a and a Wigner-Ville transform. However, these transformations are computationally expensive without considering the DL component of the system. [236] achieved 96.20%, 97.21%, and 95.27% on the UCI-HAR, WISDM, and PAMAP2 HAR datasets respectively. They used a CNN and Gated Recurrent Unit (GRU) in an attempt to compensate for less preprocessing.

Although, since stress recognition is a more complex task compared to HAR, this model is used as the basis of which to incept the HAR model. The following two subsections provide information on the setup of the model and the hyperparameters of both the HAR model and the stress model. These subsections will also detail how to incept a portion of a model into another, highlighting key considerations and justification for parameter choices.

4.5.1 Human Activity Recognition Model

Choosing a window size for a model will affect the performance of the model, as well as the shape of the input itself. If a window is too small, the model may not pick out key features. The literature typically supports a window size be-

4. tiny Inception Module

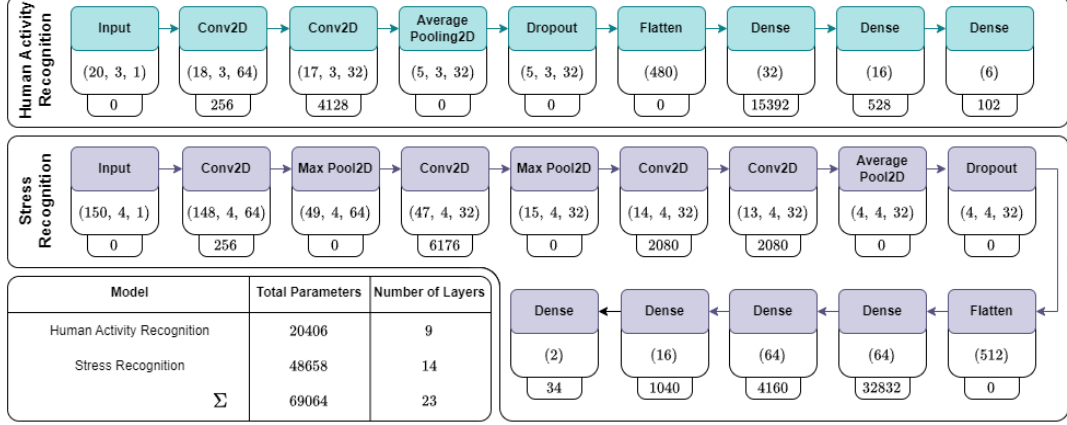


Figure 4.3: Both model architectures for the HAR model and stress recognition model before any `tim` optimisations have been applied. Each node has its layer name, output shape and number of parameters. The total parameters and layers are recorded in the lower left table.

tween 2 and 3 seconds [237] for HAR. However, in an effort to reduce the number of parameters, a window size of 1 second (20 samples) was used. In addition, raw accelerometer data was used to save preprocessing cost and reduce the input shape of the model. An overlapping window of size 50% (that is, 10 samples) was used [238], resulting in a large number of train/test sequences. These sequences are of size (20, 3, 1), which is therefore the input shape of the HAR model. The HAR model consists of two two-dimensional convolutional layers with Rectified Linear Unit (ReLU) activation, the first with a 3×1 kernel and 64 filters, and the second with a 2×1 kernel and 32 filters. These are followed by two-dimensional average pooling with a 3×1 pool, a 25% neuron dropout, and a flatten layer. Next, the network includes two dense layers consisting of 32, 16 neurons with ReLU activations. Finally, the network concludes with an output dense layer with 6 neurons with softmax activation. A visual representation can be seen in Figure 4.3. The architecture only includes layers and operations compatible with `LiteRT μ` [239]. It includes two two-dimensional convolutional layers, enough to gain effective accuracy, while still conscious of limited processing capabilities. Furthermore, to ensure compatibility with `LiteRT μ` , both convolution layers contain one-dimensional convolution kernels; therefore, in principle, they

act as one-dimensional convolution layers.

4.5.2 Stress Model

For the stress model, a window size of 150 samples was used, which is equivalent to 5 seconds with a sampling rate of 30Hz. [160] determines that 5–10 second intervals are suitable for real-time stress detection, thus supporting the chosen window size, while maximising the number of training samples. Four features are processed, namely; the raw [Heart Rate \(HR\)](#) analogue signal, [HR](#), [Heart Rate Variability \(HRV\)](#), and [ElectroDermal Activity \(EDA\)](#) analogue signal.

As expected, the stress model architecture is more complex than the [HAR](#) model architecture, as the nature of the classification task is more complex. Therefore, the initial architecture used three two-dimensional convolutional layers and two-dimensional max pooling pairs for appropriate feature extraction. Next, a dropout layer to avoid overfitting, a flatten layer, and four dense fully connected layers. Similarly to the [HAR](#) model, all the layer types used within the stress model have been chosen because of their compatibility with [LiteRT \$\mu\$](#) . However, the construction of the stress model is more complex due to the tiny inception module. Figure 4.3 depicts the aforementioned stress architecture.

4.5.3 Model Inception

Model Inception will almost certainly require a single transformation adapter layer as it is unlikely that the layers between the incepted layer and the pre-existing layer are compatible. Between our two architectures, there are no commonalities between the output shapes, and as such at least one transformation layer needs to be added before the incepted layers. Ideally, the number of additional parameters that the transformation adapter layer incurs should be minimised for two points, the input shape (x, y, z) and the output (i, j, k) . Therefore, the more similar the shapes are to each other, the better. If not, there would be an explosive number of parameters for the transformation block, which would outweigh the storage saving that a [tIM](#) would yield.

To identify where to insert a model within the model, first the input shapes of the base [HAR](#) and stress models are studied. These can be seen in Figure 4.3. It

4. tiny Inception Module

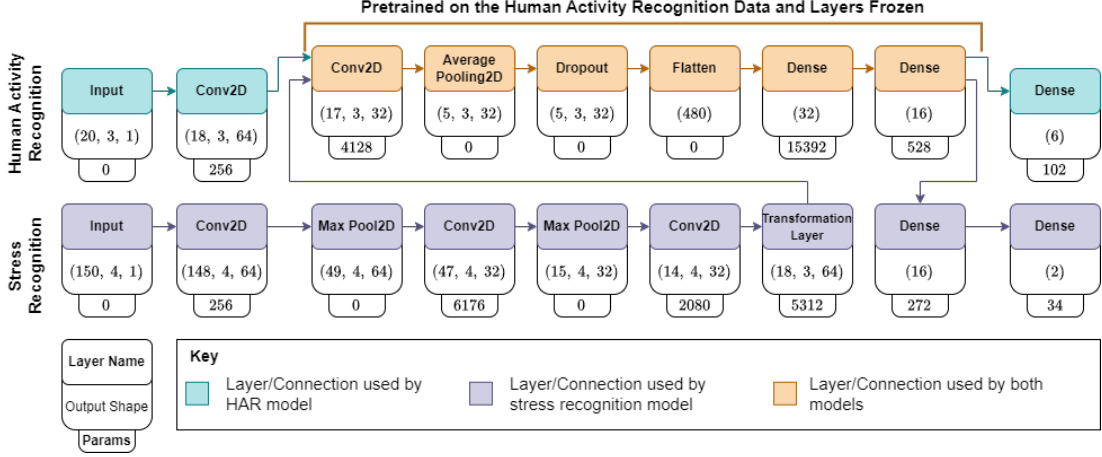


Figure 4.4: The newly adapted HAR and stress models with a tIM. Both models use the gold layers reducing storage burden. The incurred parameters for each layer are included as well as their output shapes.

can be seen that in the first half of the model architectures, both models approach a similar smaller output shape. The stress model approaches (14, 4, 32) in its third two-dimensional convolutional layer, which is similar to the input shape of the HAR model. It can also be seen that the majority of both model parameters come from a single two-dimensional convolutional layer at the beginning and a dense layer after the flatten layers. Therefore, if these layers can be effectively shared through a tIM, a significant number of parameters can be saved.

Subsequently, the tIM was added to the model after the third two-dimensional convolutional layer. The tIM will link together the HAR model in place of the pre-existing layers in the centre of the stress model. Upon adding the transformation layer and inserting the untrainable (frozen) HAR layers, a new architecture is obtained, which can be seen in Figure 4.4. Some layers have therefore been lost but instead in place of pre-trained layers. These earlier layers could yield some effective feature extraction properties on sensor data that can be used without the need for an entirely new set of values.

Now that the model architecture and the shared layers have been decided, the models can be constructed and trained in order according to Algorithm 1.

Algorithm 1 Training Order for the HAR and Stress models with a tIM

Require: Pre-defined layer indices to share from α to β

Ensure: Data preloaded into separate test/train splits

$\lambda \leftarrow$ Compile and train HAR model

$\mu \leftarrow$ Construct layers before tIM for the stress model

if $\mu^{\text{out}}.\text{shape} \neq \lambda_{\alpha}^{\text{in}}.\text{shape}$ **then**

$\triangleright (\cdot, \cdot) =$ transformation adapter (input_shape, output_shape)

$\gamma \leftarrow \text{get_transformation_adapter}(\mu^{\text{out}}.\text{shape}, \lambda_{\alpha}^{\text{in}}.\text{shape})$

$\mu.\text{append}(\gamma)$

end if

for layer in $\lambda_{\alpha:\beta}$ **do**

layer.trainable \leftarrow False

$\mu.\text{append}(\text{layer})$

end for

$\mu.\text{append}(\text{Remaining layers})$

Compile and train model μ

4.6 Results Analysis

For testing purposes, a version of the HAR model has been trained; however, there are two variations of the stress model exist; these are labelled as follows.

A The proposed architecture as depicted in Figure 4.4, labelled ‘*after*’.

B The initial architecture as in Figure 4.3 which has been labelled ‘*before*’.

For each of these stress models and the HAR model, the accuracy, precision, recall, and F1 score were recorded, as these are standardised metrics for model evaluation. The Area Under Curve (AUC) and Receiver Operating Characteristic (ROC) were also recorded for both stress models, as these are 2-class/binary classifiers. Additionally, the model sizes for the header files (.h) and TensorFlow Lite (TFLite)¹ files (.tflite) have been recorded. These file extensions are required for deployment to C++ and Python systems, respectively, which are among the two most common for embedded systems and tinyML applications. Model sizes combined with model accuracy metrics aptly demonstrate storage savings while ensuring good model performance. All results are from the test

¹Name recently changed from TFLite to Lite Runtime (LiteRT)

Table 4.1: Table presenting the metrics of all the models trained, including variations of the stress model. Stress Model A uses the [tIM](#) from the [HAR](#) model, whereas Stress Model B is the model with no layer sharing optimisation.

| | Accuracy | Precision | Recall | F1 | AUC-ROC |
|-------------------|----------|-----------|--------|--------|---------|
| HAR | 0.9434 | 0.9430 | 0.9434 | 0.9430 | N/A |
| Stress A (after) | 0.8364 | 0.8384 | 0.8364 | 0.8355 | 0.9118 |
| Stress B (before) | 0.8776 | 0.8782 | 0.8776 | 0.8777 | 0.9485 |

set using a [TFLite](#)¹ interpreter with all models undergoing *float16* [Post Training Quantisation](#) (PTQ).

For the six-class classification, the [HAR](#) model achieved an accuracy of 94.34%. It was set to train for 300 epochs with an early stopping callback with a patience of 10 epochs. The callback was triggered stopping the training of the [HAR](#) model at 206 epochs. For Stress Model A (including the [tIM](#)), the stress model achieved 83.64% accuracy. It includes a single [tIM](#) that consists of six shared layers and one transformation adapter layer, which can be seen in [Figure 4.4](#). It was set to train for 500 epochs as, during experimentation, the model seemed resilient to overfitting. However, the callback was triggered at epoch 492. The initial Stress Model B prior to the [tIM](#) achieved 87.10% accuracy on the test set. It was set to train for 250 epochs and started to show signs of overfitting and triggered the early stopping callback at epoch 220.

Overall, there has been a 4.12% loss in accuracy between Stress Model A and Stress Model B (see [Table 4.1](#)). However, the header file size between these two models has been reduced by 27.3% (or 330KB). Stress Model B is comparable to [\[1\]](#) achieving similar accuracy and model size (see [Table 4.2](#)). Compared to [Chapter 3 \[1\]](#), the [HAR](#) model has seen an increase in model size of 362KB (+130.3%), while Stress Model A (with the [tIM](#)) has decreased by 224KB (−20.3%). Additionally, the overall system, that is, both models combined in a single system, has been reduced by 402KB (−28.3%). [Table 4.3](#) demonstrates this comparison among other evaluation metrics.

The stress recognition model without [tIM](#) (Stress Model B) took on an architecture similar to the model presented in [Chapter 3 \[1\]](#). Therefore, key model

¹Name recently changed from [TFLite](#) to [LiteRT](#)

Table 4.2: Table comparing the Stress Model A (with [tIM](#)) and Stress Model B (without [tIM](#)) with the *int8* quantised stress model from Chapter 3 [1]. However, both of the models in this work are quantised to *float16*.

| | | Model Size | | | Accuracy | |
|-----|----------|------------|-------|------------------|----------|------------------|
| | Model | .tflite | .h | $\approx \Delta$ | Base | $\approx \Delta$ |
| [1] | Stress | — | 1.1MB | — | 0.88 | — |
| tIM | Stress A | 143KB | 876KB | -224KB | 0.8364 | -0.0436 |
| | Stress B | 196KB | 1.2MB | +106KB | 0.8776 | -0.0024 |

metrics remained similar (see Table 4.2). However, between using a [tIM](#), Stress Model A is significantly smaller while only sacrificing approximately 4% accuracy.

The [tIM](#) has shown to be effective at sharing relevant information between models to preserve precious storage space, particularly for deployment onto [MCU](#) devices. The reduction in parameters means more tasks can be deployed onto the same board or even allow for more complex structures between models. Previous works have classified stress data with accuracies ranging from 75% to 85% [240–242]. Despite taking an accuracy sacrifice (to 83.64% from 87.76%), our lightweight stress detection classification is comparable to that of these articles. Previous work in [HAR](#) has consistently achieved accuracy between 96.0%–98.2% [223, 235, 243, 244]. Our lightweight shareable implementation is comparable, reaching marginally below the 96.0%–98.2% accuracy threshold at 94.34%. Although the accuracy of the [HAR](#) model is slightly less than the literature, the [HAR](#) task is usually the focal point of the system, often employing complex feature extraction methods that cost precious computational resources. Furthermore, the slightly lower accuracy could be attributed to the smaller window size

Table 4.3: Table presenting the model sizes between the final stress and [HAR](#) models in this article and in Chapter 3 [1] compared to the overall system.

| | | Model Size | | | Accuracy | | Classes | |
|---------|-------|------------|-------|------------------|----------|------|----------|----------|
| | | .tflite | .h | $\approx \Delta$ | | | | |
| Model | tIM | [1] | tIM | [1] | tIM | [1] | tIM | [1] |
| Stress | 143KB | 876KB | 1.1MB | -224KB | 0.8364 | 0.88 | 2 | 2 |
| HAR | 84KB | 514KB | 152KB | +362KB | 0.9434 | 0.98 | 6 | 2 |
| Overall | 146KB | 898KB | 1.3MB | -402KB | — | — | 8 | 4 |

to reduce the input shape into the HAR model.

The overall system has been reduced by 28.3% from Chapter 3 [1]. However, importantly in Chapter 3 an *int8* quantisation policy was used for the stress model, whereas *float16* was used in this chapter. Reducing the storage burden by 28.3% at a higher level of bit precision demonstrates the feasibility of the tIM. Reducing the quantisation policy to *int8* could further reduce the model size by half, but at a potential detriment to accuracy. Furthermore, the storage burden compared to without parameter/layer sharing techniques (i.e. without the tIM) is reduced by 47.8%, further demonstrating the effectiveness of the tIM. Also, between the work presented in Chapter 3 [1], the HAR model has increased in complexity and size by approximately 362 KB, which is more than double. However, the number of classes has increased from 2 to 6, thus it is now a more complex task, requiring a more complex model. Incorporating the tIM module provided the opportunity to introduce additional complexity to the model.

Related work in model reduction is complex depending on the problem being solved. [102] use knowledge distillation to demonstrate a $5\times$ compression rate with only a 0.3% accuracy reduction. They used a large teacher model, WideResNet-16-8 network (11M parameters), and a smaller WideResNet-40-2 network (2.2M parameters) student model on the CIFAR10 dataset. Although the CIFAR10 dataset could be argued as a more complex problem, their final model had 2.2m parameters, versus our 35k ($\approx 62\times$ smaller). Furthermore, the system presented in this chapter solves two tasks that encompass a total of 8 classes.

As expected, Stress Model A performs marginally worse than Stress Model B, as some of the specificity in the model is lost due to the sharing process. Where one could argue that the accuracy loss of 4% is too great, there is 330KB of leverage to potentially increase the number of parameters / complexity without meeting the size of the initial Stress Model B.

When comparing the training loss and accuracy in Figure 4.5, with the overall test set metrics in Table 4.1, it can be seen that Stress Model B has begun to overfit. However, Stress Model A, regardless of training 492 epochs (versus 206), achieved similar accuracy between training and testing. Due to the fixed training parameters, it seems that when using a tIM, the model containing the tIM will be more resistant to overfitting than without. However, this idea seems appropriate

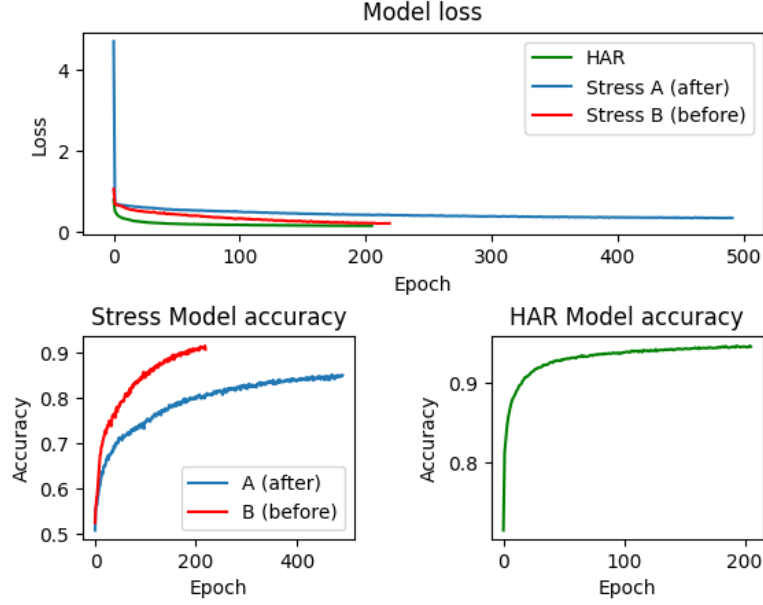


Figure 4.5: Graphs of the loss for all the models (top) and the accuracies for all the models (bottom) for each epoch of training (with a callback patience of 10 epochs).

as the model contains fixed layers, as the **tIM** layers will not adapt, and thus fewer parameters will be adjusted during backpropagation, meaning that the model would need to see more data to overfit. Conversely, this does imply that the model will learn slower, which is consistent with the number of epochs that both models trained (before the callback was triggered). In addition, if the incepted layers contain alternating gradients to the target model, the system could suffer what is known as “*catastrophic forgetting*” [245, 246]. This phenomena becomes more likely as the task space increases.

Although the proposed approach has been effective in reducing model size while maintaining accuracy, there are limitations. The **tiny Inception Module** requires knowledge of which model to incept. In the case study presented, it was known that stress detection is a more complex task and therefore would likely require a more complex model. Therefore, the stress model became a good candidate for the **tIM**. A similar rationale should be used in future implementations. However, where there is uncertainty, one could perform a bespoke architecture

search using the construction and training process outlined in Algorithm 1.

The **tIM** has been shown to be successful due to its adaptability. Layer sharing is not always feasible as without the use of transformation adapter layers, layer sharing will be extremely rare, as layer dimensions will likely not match. Typically, this implies that for simple layer sharing, the architectures should be similar, although similar architectures limit the models' ability to become task-specific. Therefore, with the use of a **tIM**, models can be trained to share important information while remaining task-specific. In fact, for even larger systems (beyond the targeted **tinyML**), it could be possible to use several **tIMs** to entangle similar layers within a system. Therefore, **tIM** is an effective architectural modification, specifically when targeting storage saving when running multiple **DL** tasks.

As processing is migrated to the edge of the network, the number of tasks to be solved will increase. With methods like the **tiny Inception Module**, this migration process could facilitate low-power devices, such as wearables, to execute multiple tasks. Overall, as **ML** expands into more embedded and **Internet of Things (IoT)** devices, sustainable computing practices become increasingly important. Large neural networks have provided impressive capabilities, but also require substantial computing resources [247]. This work shows how careful model architecture and sharing of parameters can enable sophisticated intelligence whilst adhering to the tight constraints of **MCU** devices.

In summary, this chapter demonstrates that the introduction of the **tIM** enables effective model compression and parameter sharing, allowing multiple tasks to run on resource-constrained devices without significantly compromising accuracy. While Stress Model A, which incorporates the **tIM**, showed a 4.12% drop in accuracy compared to Stress Model B, it achieved a substantial 27.3% reduction in model size, translating to a combined system saving of 402KB (28.3%) compared to Chapter 3 [1]. In particular, these reductions were achieved using a *float16* quantisation policy, with further compression possible using *int8*. The **tIM** not only helped mitigate overfitting by reducing the number of trainable parameters but also supported greater model adaptability using transformation layers, which allow non-matching architectures to share layers. Despite a slight accuracy reduction in both the **HAR** and stress models compared to the state-of-

the-art, the overall system strikes an effective balance between performance and deployability.

This chapter has introduced the [tIM](#) while also compounding the necessity for multiple models on a single resource-constrained [MCU](#) device. As more tasks are added, a more generalised layer sharing approach is required. Thus, in the following chapter, reservoir computing will be explored as an alternative to layer sharing. More specifically, while focusing on complexity analyses, [Chapter 5](#) shares [Echo State Network \(ESN\)](#) reservoirs between unrelated tasks for deployment to [MCU](#) devices.

Chapter 5

Multi-Task Reservoir Sharing Using Echo State Networks

Chapter Overview

In the previous chapter, a novel layer sharing approach for DL models was introduced. However DL models can be cumbersome especially as the number of tasks grows. In this chapter, a novel Multi-Task Learning (MTL)-based Echo State Network (ESN) reservoir sharing approach is proposed for Edge Intelligence (EI) and tinyML systems. The proposed approach is based on a shared reservoir architecture, which can be used to classify multiple datasets. The approach has been evaluated using a set of benchmarking datasets and real-world data. The model was found to learn better than randomness on most datasets, and in the majority of cases, showed performance competitive with the state-of-the-art literature. In addition, the performance of the proposed system is compared to alternative methods using complexity analyses and Floating Point Operation (FLOP) comparisons, demonstrating the viability of reservoir sharing, particularly for resource-constrained devices.

5.1 Introduction

As AI technologies improve, their expansion to the edge of the network promises enhanced resource efficiency and real-time processing, allowing tasks to run directly on-device. In some cases, such as mobile phones, multiple AI tasks are able to run on-device but are limited by the lack of processing power, and power consumption. As previously outlined in this thesis, smaller devices such as Microcontroller Unit (MCU) devices are even more limited in processing power by orders of magnitude but could still offer benefits. Developing the way edge devices can store and process multiple tasks will improve the breadth and complexity of current and new tasks developed. With more tasks at the edge of the network, industrial applications such as anomaly detection [248] and fault diagnosis [249] could improve manufacturing industries while healthcare applications such as atrial fibrillation detection [250] could improve patient care. The idea of multiple tasks learning together, thus sharing parameters, is known as MTL [230]. Typically, MTL models run their tasks simultaneously because the input domain is similar but is attempting to solve a different task. Although in doing so, DL models can be cumbersome, especially for resource-constrained devices such as MCU devices. Layers such as convolutional layers carry a significant processing burden, while dense fully connected layers have a more significant storage burden when compared to other neural network layers. In addition, training these DL models requires more computational power than inference.

ESNs, introduced by [161], have proven to be effective tools for predicting chaotic systems. ESNs typically do not require as many resources as DL models in the training/initialisation process, which is promising for edge AI applications, especially as data processing is migrated closer to the data source. Although ESNs are often computationally efficient due to their untrained reservoirs, the high dimensionality of the reservoir, particularly in large-scale implementations, can lead to increased computational and memory demands [251]. However, with sparse matrices, the number of operations can be greatly reduced. With the random nature of ESNs, perhaps some tasks will generalise well with similar reservoir dynamics, thus sharing the reservoir would benefit storage overhead. Several separate reservoirs have been used for separate tasks for MTL [252].

[253] use multiple reservoirs for image classification using the [Modified National Institute of Standards and Technology \(MNIST\)](#) and [MNIST Fashion](#) datasets. They find that this multi-reservoir approach yields better results than deep [ESNs](#) and regular [ESNs](#). However, they use 20×1000 neuron reservoirs for each task. If the reservoir is shared among tasks, the reservoir would not need to move in and out of memory, reducing overhead.

Where Chapter 4 reused several neural network layers to perform inference on a single task sequentially, this chapter aims to extend the layer sharing to share a reservoir. Thus, [ESNs](#) with varying hyperparameters and initialisation strategies are applied to benchmarking datasets, such as the [Mackey-Glass \(MG\)](#) equation and Lorenz attractors, demonstrating the viability of sharing reservoirs between tasks. Beyond these benchmarking datasets, more complex classification datasets have been tested, such as [Wireless Sensor Data Mining \(WISDM\)](#) and [MNIST](#) to gain an understanding of the versatility of a single shared reservoir. By exploring these more complex classification datasets, the future of [ESNs](#) in edge intelligent and [tinyML](#) systems is postulated. Comparing neural network architectures can be difficult, particularly when focusing on architectures suitable for [tinyML](#). Therefore, the evaluation of these datasets is compared with a single dense layer and a single convolutional layer to gain insight into the scalability of the [MTL-based ESN](#) approach. Thus, the key contribution of this chapter is the proposal and implementation of a novel [MTL-based ESN](#) reservoir sharing approach. The [MTL-based ESN](#) is subsequently validated through the following methods.

- The proposed [MTL-based ESN](#) is evaluated on 10 datasets including chaotic systems, time-series, and image data for ranging hyperparameters.
- Complexity analyses of the proposed system and the results of the tested datasets have been carried out and compared to a dense and a convolutional layer.

The rest of this chapter is organised as follows, Section 5.2 highlights key complexity analyses and [FLOPs](#) equations to use within the discussion, Section 5.3 outlines the novel [MTL](#) inspired reservoir sharing technique and how it will be evaluated, and Section 5.4 presents and discusses the results.

5.2 Complexity Analysis

To understand the impact of ESNs on a system, the number of FLOPs can be calculated. This section will also evaluate the FLOPs, time and spatial complexities of a single dense layer and a single two-dimensional convolutional layer. These insights will be used to evaluate and compare the performance of the proposed MTL-based ESN approach. However, firstly, the state update of an ESN is as follows

$$\mathbf{x}(t+1) = \tanh(\mathbf{W}_{\text{in}}\mathbf{u}(t+1) + \mathbf{W}\mathbf{x}(t)), \quad (5.1)$$

where the reservoir of weights $\mathbf{W} \in \mathbb{R}^{N \times N}$, the input weight matrix $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N \times K}$, the input signal $\mathbf{u} \in \mathbb{R}^K$ and the current state at time t , $\mathbf{x}(t) \in \mathbb{R}^N$. Therefore, the number of FLOPs required for a single time step, t is

$$\text{Total FLOPs} = \underbrace{2N^2 - N}_{\mathbf{W}\mathbf{x}(t)} + \underbrace{2NK - N}_{\mathbf{W}_{\text{in}}\mathbf{u}(t+1)} + \underbrace{N}_{\text{Bias}} + \underbrace{zN}_{\tanh}. \quad (5.2)$$

Here, \tanh is assumed to be constant in time, costing z FLOPs [254]. However, the solution assumes that 100% of \mathbf{W} is non-zero. Thus, sparsity scalars $s_{\text{in}} \ll 1$ and $s_{\text{res}} \ll 1$ are introduced to the input weight matrix and reservoir weight matrix respectively yielding an overall

$$\text{Total FLOPs} = 2s_{\text{res}}N^2 + 2s_{\text{in}}NK + (z-1)N. \quad (5.3)$$

Incorporating the total time T for each state update yields the following average case time complexity

$$\mathcal{O}(s_{\text{res}}TN^2 + s_{\text{in}}TNK). \quad (5.4)$$

The theoretical average case storage complexity is

$$\mathcal{O}(s_{\text{res}}N^2 + s_{\text{in}}NK), \quad (5.5)$$

where $s_{\text{res}}N^2$ is the storage of the reservoir weight matrix and $s_{\text{in}}NK$ is the input weight matrix.

Suppose that each task in P tasks have *readout* layers R_p , where p indicates the p th *readout* layer. Each *readout* layer, R_p , takes $T \times N$ input and maps it to

its required output. Therefore, the storage complexity of the whole system takes into account all the *readout* layers but only a single reservoir matrix. Thus, the storage complexity of a multi-task reservoir sharing architecture is as follows

$$\mathcal{O} \left(s_{\text{res}} N^2 + s_{\text{in}} N K P + \sum_{p=1}^P R_p \right). \quad (5.6)$$

However, the time complexity for running a single task remains the same. Although, as reservoirs are shared between tasks, the reservoir weight matrix itself will not need to be loaded to and from memory, reducing the memory overhead. Without the shared reservoir, the storage complexity of P tasks is as follows

$$\mathcal{O} \left(s_{\text{res}} N^2 P + s_{\text{in}} N K P + \sum_{p=1}^P R_p \right), \quad (5.7)$$

assuming that all reservoirs are the same, implying a saving of $\approx (P - 1) s_{\text{res}} N^2$ parameters. Therefore, as the number of tasks increases, the storage savings will increase, although the dominating term is likely to be N^2 . In this chapter, the sparsity of both the input weight matrix and the reservoir weight matrix are equal and will be denoted s , i.e.,

$$s_{\text{in}} = s_{\text{res}} = s. \quad (5.8)$$

An example *readout* layer is the ridge regressor. The ridge regressor can be expressed mathematically by the following

$$L(\beta) = \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2, \quad (5.9)$$

where $y \in \mathbb{R}^m$ denotes the target vector of m samples, $X \in \mathbb{R}^{m \times n}$ denotes the input feature matrix, $\beta \in \mathbb{R}^n$ denotes the learned parameters and λ a regularisation parameter. Solving Equation 5.9 the following solution for the parameters β is found

$$\beta = (X^T X + \lambda I)^{-1} X^T y. \quad (5.10)$$

Here, I denotes the identity matrix. Equation 5.10 appears to yield high algorithmic time complexity due to a matrix multiplication as well as matrix inversion.

However, these expensive calculations are found during the training phase. The inference cost of a ridge regressor and ridge classifier for a single sample is linear in time, $\mathcal{O}(n)$, since only the following is computed:

$$y = X\beta. \quad (5.11)$$

Thus a ridge regression *readout* layer is ideal for testing purposes.

For comparable methods such as [Deep Neural Networks \(DNNs\)](#), the attainment of the time and space complexities are complex. Although, for a single dense layer, the storage complexity is

$$\text{Dense Parameters} = MK + M + K, \quad (5.12)$$

$$\mathcal{O}(\text{Dense Parameters}) = \mathcal{O}(MK), \quad (5.13)$$

where M denotes the number of neurons in the dense layer. The [FLOPs](#) required for a dense fully connected layer is as follows

$$\text{Total FLOPs} = \underbrace{2MK - M}_{\mathbf{W}\mathbf{x}} + \underbrace{M}_{\text{Bias}} + \underbrace{zM}_{\text{Activation}}, \quad (5.14)$$

$$= 2MK + zM. \quad (5.15)$$

The time complexity of a dense fully connected layer is as follows

$$\mathcal{O}(MK), \quad (5.16)$$

since there is a single multiplication in the $\mathbf{W}\mathbf{x}$ component. It is common for models to be composed of several dense fully connected layers with varying input sizes, and therefore it is difficult to generalise. Moreover, models such as [Convolutional Neural Networks \(CNNs\)](#) typically include layers of different types such as flatten, dropout, and two-dimensional convolutional layers to name a few. Flatten and dropout layers impose negligible complexities, but the same is not true for two-dimensional convolutional layers.

Two-dimensional convolutional layers typically take a three-dimensional layer as an input ($C_{\text{in}} \times H_{\text{in}} \times W_{\text{in}}$) which denote the input channels, input height,

and input width, respectively. The convolution kernel, K , consists of C_{in} input channels, a kernel height, K_h and a kernel width, K_w . The shape of the output of a convolutional layer is of size $(C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}})$, where C_{out} denotes the output kernels and $H_{\text{out}} \times W_{\text{out}}$ denote the height and width of the output, respectively. The width and height of the output are as follows

$$H_{\text{out}} = \frac{H_{\text{in}} - K_h + 2P}{S} + 1, \quad (5.17)$$

$$W_{\text{out}} = \frac{W_{\text{in}} - K_w + 2P}{S} + 1, \quad (5.18)$$

where P and S are the padding and stride configurations for the layer.

Therefore, to work out the **FLOPs** for a convolutional layer, the kernel must be convolved over $H_{\text{in}} \times W_{\text{in}}$, thus making $C_{\text{in}} \times K_h \times K_w$ multiplications and $C_{\text{in}} \times K_h \times K_w - 1$ additions across $H_{\text{out}} \times W_{\text{out}}$ positions and C_{out} kernels. Hence, the total **FLOPs** for a single convolutional layer is as follows

$$\text{Total FLOPs} = (2C_{\text{in}}K_hK_w - 1)H_{\text{out}}W_{\text{out}}C_{\text{out}}z, \quad (5.19)$$

where z denotes the number of **FLOPs** required to compute the activation function. Therefore the time complexity is

$$\mathcal{O}(C_{\text{in}}K_hK_wH_{\text{out}}W_{\text{out}}C_{\text{out}}). \quad (5.20)$$

Overall, equation 5.20 implies that an **ESN** has a higher magnitude of time complexity; however, this does not always mean that they are less efficient. An **ESN** highly depends on the reservoir size, which is likely the dominating term. Whereas a **DNN** or **CNN** will contain several layers of both convolutional and dense fully connected layers. Although the input dimensions and output dimensions are more likely to be the dominating terms in a **DNN** or **CNN** model.

The storage complexity of a two-dimensional convolutional layers rely on the convolution kernels and the input and output channels C_{in} and C_{out} . The convolution kernels convolve over the input shape, which is most of the time complexity cost; however, the input shape is not stored directly. Thus, the number of pa-

rameters can be expressed as follows

$$\text{Conv Parameters} = \underbrace{K_h K_w C_{\text{in}} C_{\text{out}}}_{\text{Filters}} + \underbrace{C_{\text{out}}}_{\text{Bias}}. \quad (5.21)$$

Therefore the storage complexity is obtained as follows

$$\mathcal{O}(K_h K_w C_{\text{in}} C_{\text{out}}). \quad (5.22)$$

Understanding the time and space complexities yield a basis for comparison. [DNNs](#) and [CNNs](#) have varying architectures, making comparison to an entirely different structure challenging. However, by comparing a single layer case for the dense and convolutional layer, one could gain an understanding of the scale of efficiency presented by the proposed method.

5.3 Methodology

Previously, [ESNs](#) have been used to solve problems, be they regression-based tasks [\[166\]](#) or classification tasks [\[164\]](#). However, as far as the author is aware, no literature appears to share a reservoir between several tasks. That is, the input weight matrix remains as in traditional [ESNs](#), but the reservoir itself is initialised once and reused across many tasks. The idea of reusing a newly initialised reservoir draws upon the structure of [MTL](#) / [Multi-Domain Learning \(MDL\)](#) by hosting several input domains to their own independent output domains while reusing a portion of the model. The shared reservoir [MTL](#)-based [ESN](#) approach can be seen in [Figure 5.1](#). The input and output domains are trained using an identical pre-initialised reservoir. Although the [MTL](#)-based [ESN](#) method does not share features or a loss function such as in [MTL](#), the architecture is heavily influenced from [MTL](#) and [MDL](#) building from [Chapter 4](#). Therefore, each model will have a unique input weight matrix \mathbf{W}_{in} , and a specific *readout* layer. As with [MTL](#), the [MTL](#)-based [ESN](#) approach is expected to reduce the storage burden across multiple tasks due to the shared nature of the reservoir.

Due to the heterogeneous structure of a [DNN](#) versus an [ESN](#), a direct comparison analytically is not feasible. To draw a conclusion on the effectiveness of

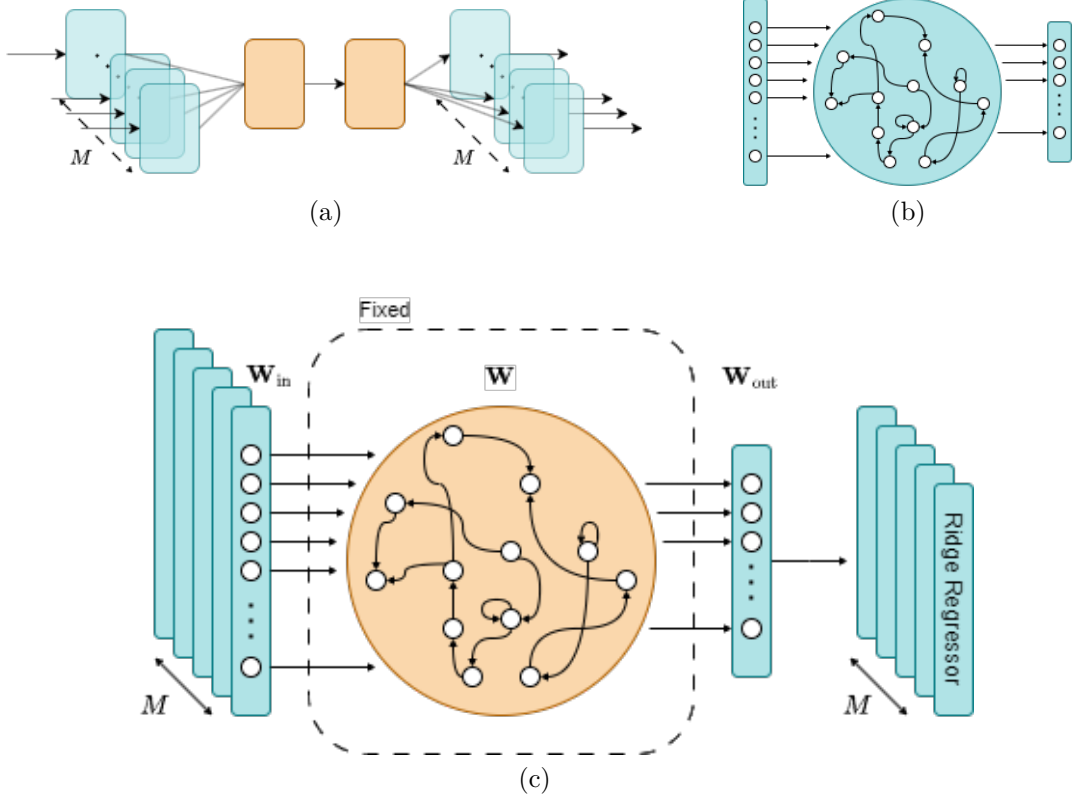


Figure 5.1: (a) The inspired **MTL** / **MDL** structure with several input domains, some single shared layers and several output domains. (b) The traditional **ESN** structure. (c) The resulting proposed methodology for the **MTL**-based **ESN** which assumes M models, each with an input weight matrix $\mathbf{W}_{in} \in \mathbb{R}^{N \times K_m}$ where m indexes the m th model in M , N represents the number of reservoir units, and K_m denotes the input dimensionality of the m th model. All blue and gold components in (a), (b) and (c) denote individual and shared components, respectively.

reservoir sharing, several benchmark datasets will test the effectiveness of reservoir sharing in terms of both model accuracy/loss and runtime efficiency through the number of **FLOPs**. [255] review **ESNs** and discover that the most common datasets used to evaluate **ESNs** are the **Non-linear Autoregressive Moving Average (NARMA)** [208], **MG** equation [256] and **Lorenz attractor** [169] datasets. Therefore, in this chapter, these datasets will be used to evaluate the effectiveness of an **ESN** for reservoir sharing. In addition, additional datasets will be tested to gain a greater cross-section of the research landscape as well as limi-

5. Reservoir Sharing Using ESNs

Table 5.1: All the datasets used within this chapter including their domain, input shape, and whether it is a classification or regression-based dataset.

| Dataset | Domain | Input Shape | Type |
|---------------------|----------------|-------------|----------------|
| NARMA [258] | Chaotic System | 1,1 | Regression |
| Mackey-Glass [20] | Chaotic System | 1,1 | Regression |
| Lorenz [259] | Chaotic System | 3,1 | Regression |
| Rössler [260] | Chaotic System | 3,1 | Regression |
| WISDM [218] | Time-Series | 20,3,1 | Classification |
| UCI HAR [261] | Time-Series | 20,3,1 | Classification |
| Stress | Time-Series | 150,4,1 | Classification |
| MNIST [262] | Image | 28,28,1 | Classification |
| MNIST Fashion [263] | Image | 28,28,1 | Classification |
| CIFAR10 [264] | Image | 32,32,3 | Classification |

tations of reservoir sharing. These are listed in Table 5.1. The image datasets, MNIST, MNIST Fashion and CIFAR10 typically perform well with DNNs and CNNs [257]. However, by evaluating these image datasets, it may be feasible to replace resource-intensive neural network models with an appropriately sized reservoir (depending on the hardware target device).

The datasets listed in Table 5.1 will be used to evaluate the number of FLOPs for varying hyperparameters across shared reservoirs. The function for the number of parameters Θ , is identical to the storage complexity of an ESN, i.e.,

$$\Theta = s(N^2 + NK). \quad (5.23)$$

That is, the number of parameters increases polynomially as N increases, meaning that the reservoir size will likely have the largest impact on the result. A graphical representation of the number of parameters as a function can be seen in Figure 5.2. Thus, the reservoir size parameter has been chosen to vary the most in its input to gain an understanding of behaviour of its output. Other hyperparameters such as the spectral radius and sparsity will be set between [0.8–1] [173, 174] and [0.01–0.1] [167, 177], respectively, based upon the literature.

The *readout* layer in the ESN architecture will be a ridge regressor or a ridge classifier depending on the problem type of the dataset (as shown in Table 5.1). Thus, the constant, low-cost *readout* layer for all datasets will simplify the perfor-

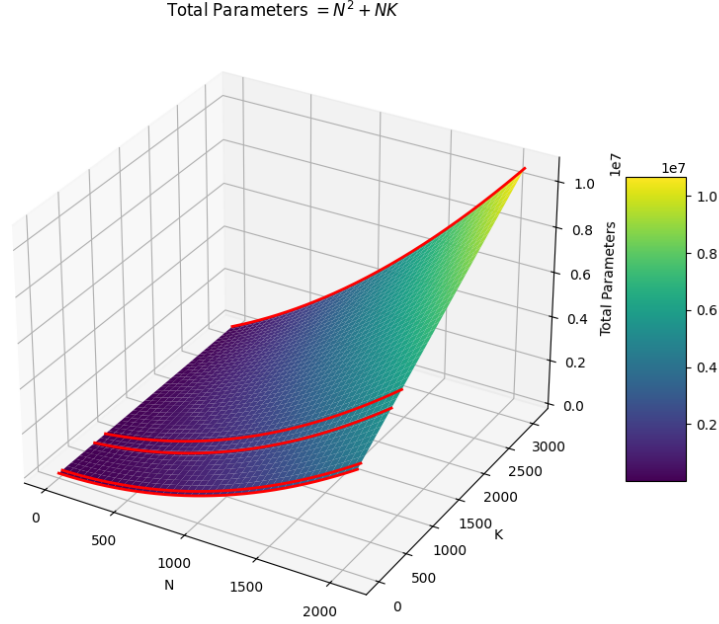


Figure 5.2: Plot of the number of parameters as a function depending on the reservoir size N and the input vector size K . In this plot, the sparsity has not been applied to the total parameters. The red lines denote the values of K used in this chapter, i.e., 1, 3, 60, 600, 784, and 3072.

mance evaluation comparison. The number of parameters for all ESNs depends on the number of neurons, the sparsity, and the input vector size. Therefore, Table 5.2 shows the number of parameters for all dataset configurations outlined in Table 5.1.

The reservoir hyperparameters are important, as they all determine how the reservoir will be initialised. However, the initialisation of the random weights and how they may be connected is an open problem in ESN literature [265]. In this chapter, four different initialisations will be tested. The first is a random initialisation from a uniform distribution. There is no consideration of other connections or their weighting in the random initialisation. The second method uses a Barabasi-Albert (BA) graph for initialisation [266]. The BA graph priori-

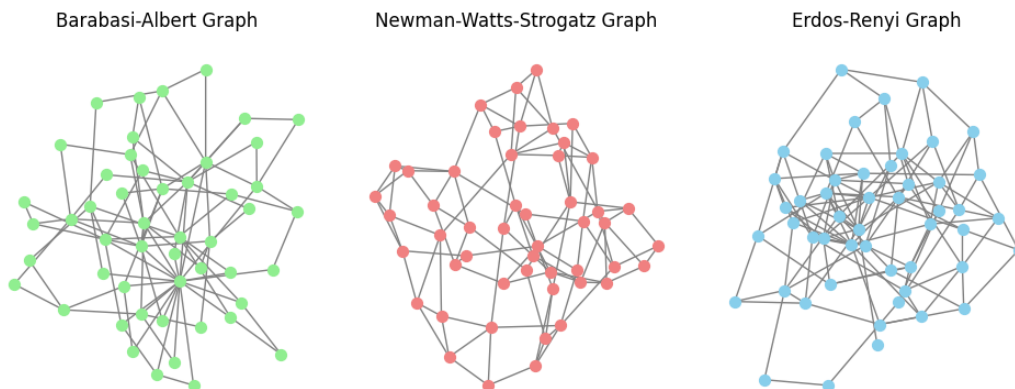


Figure 5.3: Visual representation of all pre-existing graph-based initialisation techniques containing 50 nodes; namely, [Barabasi-Albert \(BA\)](#) (left), [Newman-Watts-Strogatz \(NWS\)](#) (centre), and Erdős–Rényi (right) graphs.

tises well-connected nodes using what is known as preferential attachment [267], which implies a reduced probability of stranded nodes. The third initialisation method is the *small world* graph also known as a [Newman-Watts-Strogatz \(NWS\)](#) graph [268] and the final initialisation method is the Erdős–Rényi graph [269]. These initialisation methods have been used in previous [ESN](#) research [171, 270]. Using the four initialisation methods, it may be possible to see significantly improved performance for one or more of the datasets. Ultimately, this could inform further optimisations without affecting the overall complexity of inference, since all matrices contain the same sparsity.

5. Reservoir Sharing Using ESNs

Table 5.2: A look-up table for the number of parameters and **FLOPs** (in thousands) for each **ESN** model varying with the number of neurons and the sparsity level. N denotes the reservoir size and K denotes the one-dimensional input shape. These calculations exclude the **FLOPs** required to compute the *tanh* activation function. The corresponding datasets for K are as follows: 1 = **MG**; **NARMA**, 3 = Lorenz, Rössler; 60 = **WISDM**, UCI **HAR**; 600 = Stress; 784 = **MNIST**, **MNIST** Fashion; 3072 = **CIFAR10**.

| | | Parameters ($\times 10^3$) FLOPs ($\times 10^3$) | | | | | | | |
|------|------|---|---------|----------|-------|-------|--------|--------|--------|
| | | Total | | Sparsity | | | | | |
| | | | | 1% | | 5% | | 10% | |
| K | N | Param | FLOP | Param | FLOP | Param | FLOP | Param | FLOP |
| 1 | 256 | 65.7 | 131.3 | 0.6 | 1.0 | 3.2 | 6.3 | 6580 | 12.9 |
| | 512 | 262.6 | 524.8 | 2.6 | 4.7 | 13.1 | 25.7 | 26.2 | 52.0 |
| | 1024 | 1049.6 | 2098.1 | 10.4 | 19.9 | 52.4 | 103.9 | 104.9 | 208.8 |
| | 2048 | 4196.3 | 8390.6 | 41.9 | 81.8 | 209.8 | 417.5 | 419.6 | 837.2 |
| 3 | 256 | 66.3 | 132.3 | 0.6 | 1.0 | 3.3 | 6.3 | 6.6 | 13.0 |
| | 512 | 263.6 | 526.8 | 2.6 | 4.7 | 13.1 | 25.8 | 26.3 | 52.2 |
| | 1024 | 1051.6 | 2102.2 | 10.5 | 20.0 | 52.5 | 104.1 | 105.1 | 209.3 |
| | 2048 | 4200.4 | 8398.8 | 42.0 | 81.9 | 210.0 | 417.9 | 420.0 | 838.0 |
| 60 | 256 | 80.8 | 161.5 | 0.8 | 1.3 | 4.0 | 7.8 | 8.0 | 15.9 |
| | 512 | 292.8 | 585.2 | 2.9 | 5.3 | 14.6 | 28.7 | 29.2 | 58.0 |
| | 1024 | 1110.0 | 2219.0 | 11.1 | 21.1 | 55.5 | 109.9 | 111.0 | 220.9 |
| | 2048 | 4317.1 | 8632.3 | 43.1 | 84.2 | 215.8 | 429.6 | 431.7 | 861.3 |
| 600 | 256 | 219.1 | 438.0 | 2.1 | 4.1 | 10.9 | 21.6 | 21.9 | 43.5 |
| | 512 | 569.3 | 1138.1 | 5.6 | 10.8 | 28.4 | 56.4 | 56.9 | 113.3 |
| | 1024 | 1662.9 | 3324.9 | 16.6 | 32.2 | 83.1 | 165.2 | 166.2 | 331.5 |
| | 2048 | 5423.1 | 10844.1 | 54.2 | 106.4 | 271.1 | 540.2 | 542.3 | 1082.5 |
| 784 | 256 | 266.2 | 532.2 | 2.6 | 5.0 | 13.3 | 26.3 | 26.6 | 52.9 |
| | 512 | 663.5 | 1326.5 | 6.6 | 12.7 | 33.1 | 65.8 | 66.3 | 132.1 |
| | 1024 | 1851.3 | 3701.7 | 18.5 | 36.0 | 92.5 | 184.1 | 185.1 | 369.2 |
| | 2048 | 5799.9 | 11597.8 | 58.0 | 113.9 | 289.9 | 577.9 | 579.9 | 1157.9 |
| 3072 | 256 | 851.9 | 1703.6 | 8.5 | 16.7 | 42.5 | 84.9 | 85.1 | 170.1 |
| | 512 | 1835.0 | 3669.5 | 18.3 | 36.1 | 91.7 | 182.9 | 183.5 | 366.4 |
| | 1024 | 4194.3 | 8387.5 | 41.9 | 82.8 | 209.7 | 418.4 | 419.4 | 837.8 |
| | 2048 | 10485.7 | 20969.4 | 104.8 | 207.6 | 524.2 | 1046.5 | 1048.5 | 2095.1 |

5.4 Results Analysis

For the ridge classifier results, accuracy, precision, recall, and F1 score were selected to evaluate performance. For the ridge regression datasets, the metrics [Mean Squared Error \(MSE\)](#), [Mean Absolute Error \(MAE\)](#), and R^2 have been collected. To evaluate the effectiveness of the [MTL-based ESN](#) the regression-based datasets are initially evaluated separately to the classification-based tasks. Evaluating the aforementioned performance metrics ensuring the individual models are learning better than randomness will imply that the [MTL-based ESN](#) implementation is feasible. Subsequently, the storage overhead can be evaluated.

Figure 5.4 shows the average accuracy for the classification-based tasks across all hyperparameters, including the spectral radius, the sparsity, and the initialisation method. The plot also includes a single standard deviation error, indicating

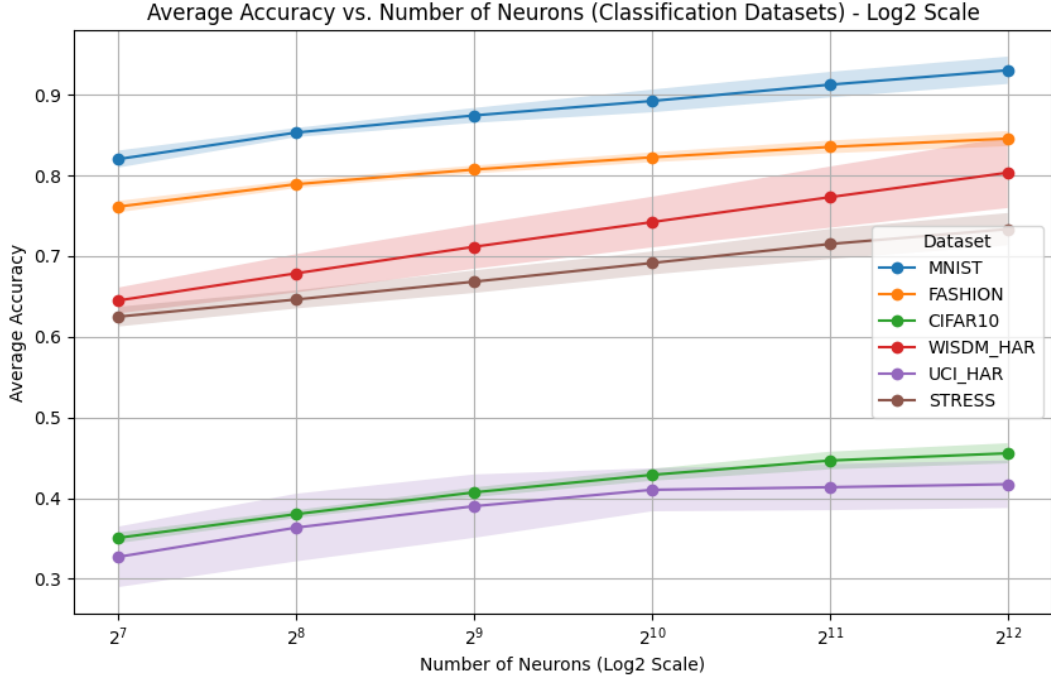




Figure 5.4: Graph of the average accuracy for each reservoir separated by the number of neurons in the reservoir for all classification datasets listed in Table 5.1. All datasets have a single standard deviation of error highlighted to the corresponding dataset.

5. Reservoir Sharing Using ESNs

Table 5.3: Table highlighting the significance levels of [MANOVA](#) tests for the initialisation strategy, spectral radius and sparsity level. The green indicates a statistically significant value i.e. $P < 0.05$, while red indicates insufficient evidence of significance $P \geq 0.05$. [MNIST](#) Fashion and [CIFAR10](#) are not available due to multicollinearity.

| Dataset | Initialisation | Spectral Radius | Sparsity |
|---|----------------|--|----------|
| NARMA | 0.9165 | 0.0000 | 0.0000 |
| Mackey-Glass | 0.8304 | 0.0194 | 0.0000 |
| Lorenz | 0.0198 | 0.2871 | 0.0000 |
| Rössler | 0.9340 | 0.0001 | 0.0000 |
| WISDM HAR | 0.9951 | 0.9333 | 0.0000 |
| UCI HAR | 0.0859 | 0.0000 | 0.0000 |
| Stress | 0.9944 | 0.9833 | 0.0000 |
| MNIST | 0.2553 | 0.3548 | 0.0000 |
| Not Significant | | Significant | |
|  | |  | |

variation in accuracies across the datasets. The highest standard deviation is $\pm 4.3\%$ for the [WISDM](#) dataset, perhaps implying that the other hyperparameters had a significant impact on the accuracy. This interpretation is supported by [Multivariate Analysis of Variance \(MANOVA\)](#) tests, which found significance for all hyperparameters as expected ($P < 0.05$). However, some datasets did not present enough evidence to suggest that some of the hyperparameters were significant. The specific significant hyperparameters and their significance values can be seen in Table 5.3.

It can be seen that the sparsity is significant across all the datasets, which is unsurprising, since more information is lost at higher sparsity levels. This could be particularly impactful for the input weight matrix \mathbf{W}_{in} as some important input channels could be blocked. It is possible that further optimisation of these hyperparameters can yield better results. For some datasets, the spectral radius was insignificant, which could be due to the ideal spectral radius being within the tested range, i.e. 0.8–0.9.

For the regression-based datasets, the [MSE](#) and the coefficient of determination (R^2) follow positive trends as the number of neurons increases (see Figure 5.5). That is, the loss tends to zero and the coefficient of determination tends

5. Reservoir Sharing Using ESNs

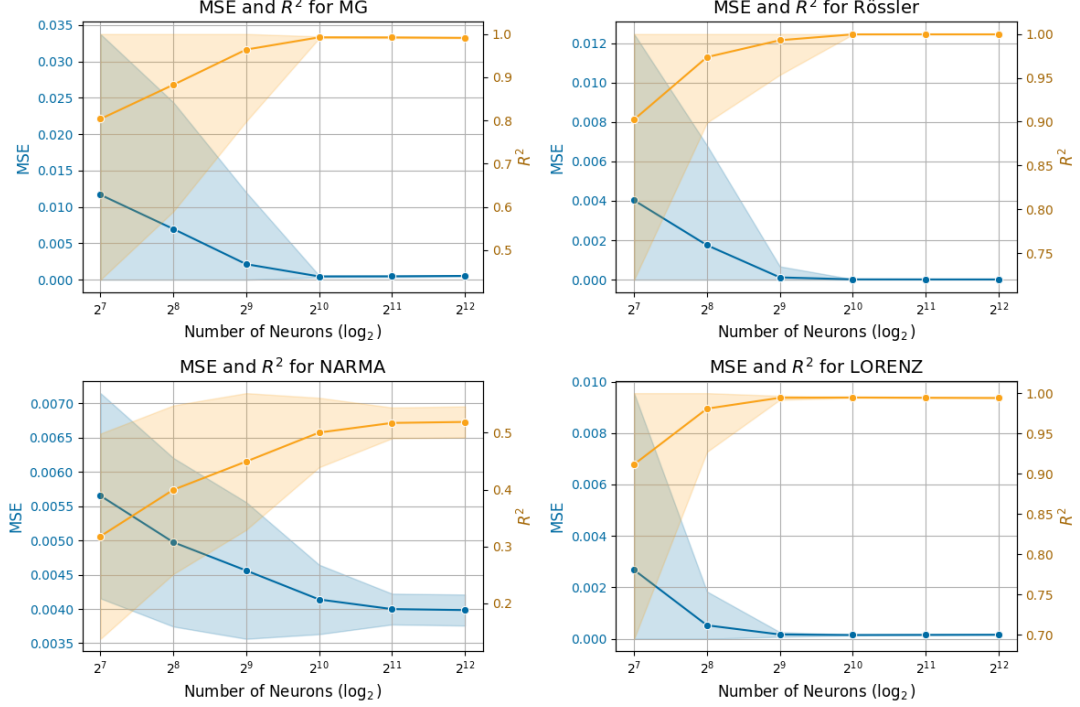


Figure 5.5: For each of the regression-based datasets, the average MSE (blue) as well as the coefficient of determination, or R^2 (gold) on a \log_2 axis denoting the number of neurons in the reservoir.

to one as the number of neurons increases with the exception of the NARMA dataset. The fact that the NARMA dataset does not tend toward an R^2 of one may result from the model hyperparameters not being well-matched to the incoming data. Figure 5.5 also illustrates a single standard deviation of error. The standard deviation error converges as the number of neurons increases, but at different rates. In the literature, these chaotic system datasets do not require a large number of neurons to achieve good results [270, 271]. In [270], a reservoir of 50 neurons produced an MSE in the order $1e-4$ and $1e-3$ for the Rössler and Lorenz systems respectively. These results are consistent with the smallest reservoir of 128 neurons. However, the high standard deviation is likely due to the reservoir not being able to effectively propagate the input signal, implying a breakdown in the Echo State Property (ESP).

The sparsity is significant for all datasets, while the initialisation method is

insignificant for all datasets when evaluated over the key performance metrics mentioned above. However, the relationship between these variables and overall accuracy or loss can be seen through additional plots in Appendix 6.3. In particular, the 1% sparsity caused a lower performance, particularly for smaller reservoirs. Although this is likely due to an over-reliance on fewer weight values. For example, for a 1% sparsity for a 128 neuron reservoir implies 164 non-zero values, whereas for a 1024 neuron reservoir, there is over 10,000 non-zero values.

All of these datasets can have improved accuracies or reduced losses from optimising their hyperparameters and/or providing alternative *readout* layers. However, for consistency, the implementation the ridge regressor/classifier was used. Choosing this *readout* layer type could be at the detriment to the performance of some of the datasets. Regardless, all models from all datasets have demonstrated learning capacity beyond randomness and in some cases competitive results (MNIST, Fashion MNIST, and Stress datasets). MNIST have been optimised to reach classification accuracies of 98/99% [272], although for ESN-based implementations [164, 253], such is only possible via large reservoir computing architectures and vast preprocessing. Although [175] achieve 90.52% accuracy without optimisations for a reservoir of 1200 neurons, which is comparable to the 89.22% average accuracy achieved for a reservoir of 1024 neurons. Such is similar for MNIST Fashion, which has achieved 70%-80% for 1024+ neurons. However, [253] demonstrate accuracies of around 88% but for a reservoir of 20,000 neurons, much larger than in this chapter. In chapter 3, if the stress dataset achieves an accuracy of over 68%, this implies a single standard deviation of confidence that the model has performed better than randomness. This was true for some 512 neuron reservoirs but became consistent at reservoirs of size 1024 and above.

The MTL-based ESN implementation has been shown to provide results beyond randomness. Now it is possible to evaluate the storage savings and FLOP savings using the information from Section 5.2. All dense and convolutional layers contain an activation function which is commonly either the sigmoid or tanh function. Both activation functions are similar in complexity; therefore, the parameter that accounts for the activation function, that is, z will be discounted. Thus, FLOPs for each of the ESN reservoir configurations can be seen in Ta-

ble 5.2. In the Results section (Section 5.4), the performances of most datasets began to, or had already, plateaued with a reservoir of 1024 neurons with a sparsity of 5%. According to Table 5.2, the number of FLOPs a 1024 neuron reservoir with 5% sparsity ranges between approximately 104k to 420k. These results can be less than that of a single convolutional layer, let alone an entire CNN. For example, suppose that a single convolutional layer is constructed that has an input shape of (28, 28, 1) for the classification of MNIST data. Suppose that a 3×3 convolution kernel is used along with a stride of 1, *same* padding and 32 filters, most of which are common for image classification in particular [273–275]. Thus the number of FLOPs for this single convolutional layer would be as follows

$$\begin{aligned} \text{FLOPs} &= H_{\text{out}} \cdot W_{\text{out}} \cdot C_{\text{out}} \cdot (2C_{\text{in}} \cdot K_h \cdot K_w - 1), \\ &= \underbrace{(28 \times 28)}_{H_{\text{out}} \times W_{\text{out}}} \times \underbrace{(32)}_{C_{\text{out}}} \times \underbrace{(2 \times 1 \times 3 \times 3 - 1)}_{2C_{\text{in}} \cdot K_h \cdot K_w - 1}, \\ &= 426496. \end{aligned}$$

Furthermore, as convolutional layers are stacked, it is likely that the FLOPs will increase exponentially as C_{in} will equal the number of filters in the previous convolutional layer.

A dense layer will not be as computationally expensive with the same approach. However, by using the same example input as above for a dense layer with 32 neurons, the FLOPs cost is as follows

$$\begin{aligned} \text{Total FLOPs} &= 2MK, \\ &= 2 \times \underbrace{(32)}_M \times \underbrace{(28 \times 28 \times 1)}_K, \\ &= 50176. \end{aligned}$$

Compared to the 184k FLOPs used by the MNIST ESN implementation, the 50k FLOPs required by the dense layer represents a reduction of approximately 72.74%. However, a DNN will likely use several dense layers that could contain more neurons than the 32 in the previous example. Thus, the FLOPs of the MTL-based ESN could be considered more efficient because of its sparse representation.

However, [DNNs](#) and [CNNs](#) can have additional optimisations such as pruning [\[92\]](#) to increase the sparsity of the weight matrices and filters.

Although the number of parameters used for more complex datasets, such as [CIFAR10](#), may be more efficient, less complex datasets, such as [MG](#) and [MNIST](#), could be using a larger reservoir than required. For example, [\[271\]](#) only considers reservoirs between 50 and 200 neurons for the [NARMA](#) dataset. Furthermore, according to [Figure 5.5](#), most of the regression-based datasets plateau at 512 neurons and their standard deviation is significantly reduced compared to 128 and 256 neuron reservoirs. This trend could imply that these datasets do not gain a significant benefit as the reservoir neurons increase. Therefore, for these datasets, perhaps the use of the shared reservoir approach could be detrimental to these individual tasks. However, several tasks that share the same reservoir will likely save storage between multiple tasks.

To evaluate the storage burden of the [MTL-based ESN](#), the number of parameters will be used as a comparative metric. In doing so, the precision of these parameters will become obsolete. That is, any additional precision optimisations, such as integer quantisation [\[182\]](#), can be applied to the parameter numbers to obtain a representation of the total storage size. In addition, the comparisons made assume that sparse matrix representations are used to store the [MTL-based ESN](#) implementation. Taking the same configuration of the [MTL-based ESN](#) as above, i.e., 1024 neurons at a sparsity of 5%, the total number of parameters is therefore approximately 338k. These 338k parameters include all 10 input weight matrices and ridge regressors/classifiers, as well as the shared reservoir weight matrix shown in depth in [Table 5.4](#). Using the same convolutional layer setup as above, the number of parameters required for storage is as follows

$$\text{Conv Parameters} = \underbrace{3 \times 3}_{\text{Kernel}} \times \underbrace{1}_{C_{\text{in}}} \times \underbrace{32}_{C_{\text{out}}} = 228.$$

For a dense layer with 32 neurons as used previously, the following is obtained

$$\text{Dense Parameters} = \underbrace{32}_M \times \underbrace{28 \times 28 \times 1}_K + \underbrace{32}_M + \underbrace{28 \times 28 \times 1}_K = 25904.$$

5. Reservoir Sharing Using ESNs

Table 5.4: For all datasets listed on the left, and their input sizes, K , the total shared parameters when using varying reservoir sizes is demonstrated as well as the total dataset-specific parameters. These dataset-specific parameters are derived from the input weight matrix $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N \times K}$ and the ridge classifier/regressor $\beta \in \mathbb{R}^N$. All values in the table are assumed to be 5% sparsity, i.e., $s = 0.05$. The total dataset-specific parameters to run all 10 classification/regression tasks is included as well as the total including the sparse shared reservoir.

| | | Reservoir Size (N) | | | | | |
|--------------------------|------|------------------------|-------|--------|--------|--------|---------|
| | | 128 | 256 | 512 | 1024 | 2048 | 4096 |
| Shared Params (sN^2) | | 820 | 3277 | 13108 | 52429 | 209716 | 838861 |
| Dataset | K | $sNK + N$ | | | | | |
| NARMA, MG | 1 | 135 | 269 | 538 | 1076 | 2151 | 4301 |
| Lorenz, Rössler | 3 | 148 | 295 | 589 | 1178 | 2356 | 4711 |
| WISDM, UCI_HAR | 60 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 |
| Stress | 600 | 3968 | 7936 | 15872 | 31744 | 63488 | 126976 |
| MNIST, MNIST Fashion | 784 | 5146 | 10292 | 20583 | 41165 | 82330 | 164660 |
| CIFAR10 | 3072 | 19789 | 39578 | 79156 | 158311 | 316621 | 633242 |
| Total | | 35639 | 71274 | 142544 | 285085 | 570167 | 1140330 |
| Total Inc. sN^2 | | 36459 | 74551 | 155652 | 337514 | 779883 | 1979191 |

Although for both the convolutional layer and dense layer the number of parameters are much less than that of the entire MTL-based ESN implementation, these individual layers make up a smaller portion of a larger model. The same is true for the FLOPs. Due to the heterogeneous architectures of CNNs and DNNs, direct comparison is not feasible. However, taking these layer examples for classification of a single dataset as a base for comparison exemplifies that the shared reservoir architecture to classify multiple datasets is an efficient use of storage space.

MobileNetv2 [273], a low-cost pre-trained network, has 3.4m parameters and takes approximately 600m FLOPs. However, this model configuration expects a 224×224 input. Models with MCUs in mind, such as MCU-Net, have managed to achieve results comparable to MobileNetv2 but with 1.2m parameters and 168m

FLOPs [276]. Both of these models are much larger than the MTL-based ESN presented in this chapter. Furthermore, both MobileNetv2 and MCU-Net classify a single task, while the MTL-based ESN can classify 10 different classification problems. These results, as well as the low training cost of an ESN, exemplify the potential for ESNs on MCU devices. Although a single ESN can contribute to a larger system, by reusing the reservoir, it is possible to significantly reduce the storage burden while maintaining a low number of FLOPs.

This chapter builds on Chapter 4 using a shared reservoir instead of neural network layer sharing approaches. In addition, the same datasets from Chapters 3 and 4 have been used to build a comparison basis for varying optimisation techniques. From Chapter 4, the sharing of information between tasks requires knowledge of which information is best shared, which can be a time-consuming process. With shared reservoir optimisation, others may be able to more easily share information between tasks by using a shared reservoir approach, which could see more multi-*model* applications on edge-based systems. The next chapter will conclude the work presented in this PhD thesis, highlight its implications, and provide potential future works based on the presented work.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The previous chapter focused on a shared reservoir approach to [Echo State Networks \(ESNs\)](#), which was tested on 10 datasets that represent a cross section of current [ESN](#) research. This chapter will draw a conclusion to this thesis beginning with how the research questions and research aim proposed in Chapter 1 of this thesis have been met.

Research Question 1

How can multiple Deep Learning models be integrated to enhance on-device inference capabilities for multimodal sensor inputs on a wearable Microcontroller Unit device?

In Chapter 3, a classification model was used to provide context to a second model on the edge using an Arduino Nano 33 [BLE Sense](#), a low-power and low-cost [Microcontroller Unit \(MCU\)](#) device. [Human Activity Recognition \(HAR\)](#) has been assessed to help reduce motion artifacts in real-world stress detection. Previously, it has been challenging to deploy a single [DL](#) model to an [MCU](#) device; however, in Chapter 3 two models, [HAR](#) and stress detection, have been deployed. The methodology presents how the [HAR](#) model contextualise the other, while outlining how the [HAR](#) and stress detection system is composed. The [HAR](#) model achieved 98% accuracy when detecting exercise or no exercise, while the

stress detection model achieved 88% accuracy. The [HAR](#) and stress detection models were quantised to *float16* and *int8*, respectively, reducing the complexity of the model for a minimal accuracy sacrifice. The size of both header files for each of the models was approximately 1.3MB. To gain two practical [DL](#) models to occupy 1.3MB on a low-power [MCU](#) device meant fewer architectures were able to be deployed. Additionally, as discussed in Chapter 2, the availability of libraries for sensors on alternate boards are few. The Arduino Nano 33 [BLE](#) Sense was chosen due to its compatibility with a [Photoplethysmography \(PPG\)](#) sensor library required for stress detection. Although, regardless of these limitations, the [HAR](#) model was able to aptly provide context to the stress detection model. The deployment of multiple models on a single resource-constrained low-power [MCU](#) device demonstrates the feasibility of a multi-*model* system to improve the functionality of the device.

Research Question 2

How can multiple Deep Learning models effectively share information to enable multi-*model* tiny Machine Learning applications?

To answer [RQ 2](#), Chapter 4 introduces the [tiny Inception Module \(tIM\)](#), a [DL](#) layer sharing approach specifically designed for [tinyML](#) systems. The novel transformation adapter layer allows pre-existing network components to be stitched into another model for training. The [tIM](#) has proven to significantly reduce the storage burden of a system by reusing pre-existing layers between multiple models. Tasks of differing complexities can be considered even if their architectures vary by making use of the presented transformation adapter layer. As shown early in Chapter 4, the [tIM](#) is a generalised approach that can be applied to any dataset. A case study on real-world data is presented for [tIM](#) using stress detection and [HAR](#). The [HAR](#) model achieved 94.34% classifying 6 classes and the stress recognition model achieved 83.64% classifying between stressed and not stressed. The [tIM](#) demonstrated significant storage savings of 28.3% for the entire system compared to Chapter 3 [1] with the entire system quantised to *float16* precision. Furthermore, compared to no layer sharing approaches for the same models, the [tIM](#) achieved a storage saving of 47.8%. Ultimately, the [tIM](#)

has shown an impressive ability to reduce redundancy while only acquiring a 4% accuracy sacrifice on the inception stress detection model. Therefore, by entangling models using the [tIM](#), much like the [HAR](#) model, multiple models can be shared at a higher complexity than before. Thus, higher specification models can occupy the same device, improving the functionality of the device.

Research Question 3

How efficient are sparse shared reservoirs in handling multiple tasks, particularly compared to Deep Neural Networks and Convolutional Neural Networks?

In Chapter 5, [ESN](#) reservoirs have been studied as candidates to replace layer sharing approaches. Due to an [ESNs](#) low training cost, they become a particularly desirable target for [MCU](#) devices. However, due to the high complexity reservoir, solving tasks using [ESNs](#) is not considered for edge [AI](#) applications. Chapter 5 explores potential [ESN](#) optimisations to make them more suited to edge-based applications. Thus, a novel [Multi-Task Learning \(MTL\)](#)-based approach groups chaotic systems, time-series and image datasets with a common reservoir. Using chaotic systems popular in [ESN](#) literature, as well as real-world benchmarking datasets, yields a detailed cross-section of the shared reservoir approach. In addition, unrelated tasks can outline the limits of reservoir sharing. Overall, most tasks generalise and learn effectively beyond randomness, with the majority of datasets competing with state-of-the-art literature. By sharing a reservoir between all tested datasets, the system contains approximately 338k parameters and between 104k–419k [Floating Point Operations \(FLOPs\)](#) for a 1024 neuron reservoir with 5% sparsity, which has been compared to a single dense, and two-dimensional convolutional layer. In sharing the reservoir between ten datasets, the [FLOP](#) cost was smaller than that of a single two-dimensional convolutional layer classifying [Modified National Institute of Standards and Technology \(MNIST\)](#) handwritten digits with 32 filters. While the number of parameters is more for the [MTL-based ESN](#) approach, [Deep Neural Networks \(DNNs\)](#) and [Convolutional Neural Networks \(CNNs\)](#) are comprised of several layers compounding the number of parameters depending on the depth of the network.

Compared to MobileNetV2 and MCU-Net, both of which classify a single task, the MTL-based ESN implementation, which can classify ten different tasks cost a fraction of the parameters and FLOPs. Thus, sharing a reservoir between tasks can offer significant benefits when multiple tasks are deployed to a single device, particularly compared to DNNs and CNNs. Although most of the savings are in storage, there is also a significant saving in FLOPs during inference that can reduce power consumption.

Research Aim

To effectively share information between several Deep Learning models to minimise the number of parameters required to perform multiple tasks independently on low-power Microcontroller Unit devices.

Having answered all the research questions, it is possible to evaluate the overall research aim. This thesis contributes significantly to the research aim by first deploying multiple models to a low-power MCU device, secondly implementing a layer sharing approach to complete the same tasks but with reduced model size and increased model complexity, and finally implementing and evaluating a novel ESN reservoir sharing approach. In sharing multiple models on-device, the device becomes multi-purpose and perhaps decreases dependence on additional hardware. In addition, multiple tasks could yield more information based on the same input stream. However, all chapters have only used layers compatible with Lite Runtime for Microcontrollers (LiteRT μ), which limits the possible DL model architectures tested. Furthermore, because no new data was collected within this thesis, all datasets used were publicly available, with the exception of the stress dataset. Therefore, it is unclear how some of the tasks developed will generalise in real-world environments, particularly with practical sensors.

Additionally, the work presented in this thesis has implications for society. A researcher or developer could use any of the optimisation methods presented within their own work. Chapters 3 and 4 could contribute to new ideas of health monitoring, or inspire alternate multi-*model* context-aware applications. In particular, Chapter 3 is readily available and deployable, pushing the current bounds of tiny Machine Learning (tinyML). Although Chapter 4 pushes these boundaries

further but requires some additional hardware and library adjustments to become fully functional, preliminary results show that significant storage savings can be made while maintaining significant accuracy. The technologies shown in both Chapters 3 and 4 could help medical professionals monitor patients, particularly when monitoring stress in daily life. Alternatively, with multiple tasks on-device, industrial pipelines can be maintained by detecting a range of faults based on different modalities. The work presented in this thesis has the potential to increase the functionality of MCU devices, leading to a lower reliance on cloud data storage and processing. Furthermore, since all data (including sensitive data) is only stored for processing and decision-making, such systems will improve data privacy.

6.2 Research Contributions

There have been several novel contributions presented within this PhD thesis. These are as follows:

- Two DL models have been constructed for the purpose of deployment onto a single MCU device. The Human Activity Recognition provides contextual information on whether the stress model should run inference. The novel combination of these two models provides real-time contextual HAR information to improve the accuracy of stress inference.
- The context-aware multimodal multi-*model* approach was subsequently deployed to a resource-constrained Arduino Nano 33 BLE Sense. Deployment is often missing from literature and has significant practical challenges, such as library and sensor compatibility, which have been addressed.
- The novel tiny Inception Module was introduced for the sharing of network layers between DL models. Using a new transformation adapter layer to enable the repurposing of layers for additional tasks, while also allowing models of differing architectures to become entangled. The result yields several models with a subset of shared layers between them.

- To demonstrate the effectiveness of the [tiny Inception Module](#), a case study on stress recognition and [Human Activity Recognition](#) was carried out drawing a direct comparison with the previous chapter. Results showed that, although the [HAR](#) model size increased, the whole system had a significant reduction in model size while maintaining accuracies comparable to the literature.
- A novel [Multi-Task Learning](#)-based [Echo State Network](#) reservoir sharing approach is proposed. Under the [MTL](#)-based [ESN](#) approach, several tasks reused the same reservoir but have task-specific input matrices and readout layers.
- The proposed [Multi-Task Learning](#)-based [Echo State Network](#) is evaluated on 10 datasets that cover chaotic systems, time series, and image data. A single hyperparameter configuration is applied across all datasets, varying the number of neurons, spectral radius, sparsity, and initialisation method. The results showed that when a single reservoir is shared, most models still learn appropriate representations of the propagated input and, in several cases, competitive accuracies.
- Complexity analyses of the [Multi-Task Learning](#)-based [Echo State Network](#), a dense layer and a convolutional layer have been outlined. These analyses were compared with the results of the tested datasets. The results showed that fewer parameters are needed for all 10 datasets compared to popular lightweight [DL](#) models to classify a single task. In addition, fewer [FLOPs](#) are needed for inference in the entire [MTL](#)-based [ESN](#) system compared to comparative [CNN](#) and [DNN](#) architectures.

6.3 Future Work

This thesis presents optimisation methods to execute inference on multiple [DL](#) models on resource-constrained [MCU](#) devices. However, these methods can be extended pushing the boundaries of edge [AI](#) and [tinyML](#) further. This section will outline potential future research directions.

In Chapter 3, one model is used to provide contextual information that affects the other. Transformer networks are an increasingly popular research topic [277] which makes use of attention-based information (which could be considered a form of context). Future work could explore deployment of a transformer network to an MCU device or even a larger edge device. The deployment of transformer networks could allow for the inclusion of more complex tasks and the migrating of natural language processing to the edge of the network, which could significantly reduce the reliance on data centres.

In Chapters 3 and 4, models are tested on two datasets, HAR and stress datasets. These datasets proved sufficient to demonstrate the viability and use case for multiple models on a single MCU device for their respective methods. However, to truly understand the limits of these methods, a wider range of modalities should be tested.

Furthermore, within this thesis, the Arduino Nano 33 BLE Sense was the only board used within experiments due to library and sensor restrictions. The Arduino Nano 33 BLE Sense¹ has a 64MHz clock speed, 256KB SRAM and 1MB flash. However, newer boards such as the Arduino Portenta H7² or the Arduino Nicla³ range have around double the technical specification in all areas. By upgrading the hardware and testing these methods, perhaps more functional tasks or modalities can be developed. Subsequently, the MTL-based ESN could be used on larger models as a way to reduce redundancy; thus, choosing a hardware target of higher specification than an MCU device, such as a smartphone, could yield significant storage and memory benefits regardless.

In Chapter 5, all tasks were made to follow the same convention. That is, they all had the same reservoir size and hyperparameters as well as the same readout layer. However, future work could group tasks based on similar reservoir configurations, potentially yielding increased accuracy and/or reduced loss per group. Additionally, to extend the MTL-based ESN implementation further,

¹Available online, <https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf>, last accessed 03/03/2025

²Available online, <https://docs.arduino.cc/resources/datasheets/ABX00042-ABX00045-ABX00046-datasheet.pdf>, last accessed 03/03/2025

³Available online, <https://docs.arduino.cc/resources/datasheets/ABX00050-datasheet.pdf>, last accessed 03/03/2025

6. Conclusion and Future Work

perhaps less complex datasets, such as chaotic systems, do not need such a large reservoir. Instead, a sub-reservoir could exist within the main reservoir to further reduce the total **FLOP** cost. In addition, all datasets used a ridge regressor, which may not produce adequate accuracy for the dataset. Future work could tune these parameters and apply the system to a more practical application.

References

- [1] Michael Gibbs, Kieran Woodward, and Eiman Kanjo. Combining Multiple Tiny Machine Learning Models for Multimodal Context-Aware Stress Recognition on Constrained Microcontrollers. *IEEE Micro*, 44(3):67–75, 5 2024. doi:10.1109/MM.2023.3329218. xiv, 7, 64, 65, 69, 75, 76, 77, 79, 103
- [2] Martijn Koot and Fons Wijnhoven. Usage impact on data center electricity needs: A system dynamic forecasting model. *Applied Energy*, 291:116798, 6 2021. doi:10.1016/J.APENERGY.2021.116798. 1
- [3] Data Centres Metered Electricity Consumption 2023 - Central Statistics Office. URL: <https://www.cso.ie/en/releasesandpublications/ep/p-dcmec/datacentresmeteredelectricityconsumption2023/>. 1
- [4] Discover our data center locations. URL: <https://www.google.co.uk/about/datacenters/locations/>. 1
- [5] Sundar Pichai. Our plans to invest \$9.5 billion in the U.S. in 2022. URL: <https://blog.google/inside-google/company-announcements/investing-america-2022/>. 1
- [6] Data centres to be given massive boost and protections from cyber criminals and IT blackouts - GOV.UK. URL: <https://www.gov.uk/government/news/data-centres-to-be-given-massive-boost-and-protections-from-cyber-criminal>. 1

REFERENCES

- [7] Rongkang Dong, Yuyi Mao, and Jun Zhang. Resource-Constrained Edge AI with Early Exit Prediction. *Journal of Communications and Information Networks*, 7(2):122–134, 9 2022. doi:10.23919/JCIN.2022.9815196. 2, 36
- [8] Partha Pratim Ray. A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1595–1623, 4 2022. doi:10.1016/J.JKSUCI.2021.11.019. 2, 3, 43
- [9] Jiangpeng Dai, Xiaole Bai, Zhimin Yang, Zhaohui Shen, and Dong Xuan. Mobile phone-based pervasive fall detection. *Personal and Ubiquitous Computing*, 14(7):633–643, 10 2010. URL: <https://link.springer.com/article/10.1007/s00779-010-0292-x>, doi:10.1007/S00779-010-0292-X/FIGURES/14. 3
- [10] Shah Faisal Darwaish, Esmiralda Moradian, Tirdad Rahmani, and Martin Knauer. Biometric Identification on Android Smartphones. *Procedia Computer Science*, 35(C):832–841, 1 2014. doi:10.1016/J.PROCS.2014.08.250. 3
- [11] Alaa Awad Abdellatif, Amr Mohamed, Carla Fabiana Chiasserini, Mounira Tlili, and Aiman Erbad. Edge computing for smart health: Context-aware approaches, opportunities, and challenges. *IEEE Network*, 33(3):196–203, 5 2019. doi:10.1109/MNET.2019.1800083. 3, 15
- [12] Xian Gao, Peixiong He, Yi Zhou, and Xiao Qin. Artificial Intelligence Applications in Smart Healthcare: A Survey. *Future Internet 2024*, Vol. 16, Page 308, 16(9):308, 8 2024. URL: <https://www.mdpi.com/1999-5903/16/9/308/htm><https://www.mdpi.com/1999-5903/16/9/308>, doi:10.3390/FI16090308. 3
- [13] Michael Gibbs and Eiman Kanjo. Realising the Power of Edge Intelligence: Addressing the Challenges in AI and tinyML Applications for Edge Computing. In *Proceedings - IEEE International Conference on Edge Computing*, volume 2023-July, 2023. doi:10.1109/EDGE60047.2023.00056. 7, 15
- [14] L. Todd Rose and Kurt W. Fischer. Garbage In, Garbage Out: Having Useful Data Is Everything. *Measurement: Interdisciplinary*

REFERENCES

- Research & Perspective*, 9(4):222–226, 10 2011. URL: <https://www.tandfonline.com/doi/abs/10.1080/15366367.2011.632338>, doi: 10.1080/15366367.2011.632338. 10
- [15] Monique F. Kilkenny and Kerin M. Robinson. Data quality: “Garbage in – garbage out”. <https://doi.org/10.1177/1833358318774357>, 47(3):103–105, 5 2018. URL: <https://journals.sagepub.com/doi/full/10.1177/1833358318774357>, doi:10.1177/1833358318774357. 10
- [16] WHO Global Report on Falls Prevention in Older Age. 2008. 11, 30
- [17] Firdaus S. Dhabhar. Effects of stress on immune function: the good, the bad, and the beautiful. *Immunologic Research* 2014 58:2, 58(2):193–210, 5 2014. URL: <https://link.springer.com/article/10.1007/s12026-014-8517-0>, doi:10.1007/S12026-014-8517-0. 11
- [18] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 7 2019. URL: <https://link.springer.com/article/10.1007/s10618-019-00619-1>, doi:10.1007/S10618-019-00619-1/FIGURES/16. 11
- [19] Michael C. Mackey and Leon Glass. Oscillation and Chaos in Physiological Control Systems. *Science*, 197(4300):287–289, 1977. URL: <https://www.science.org/doi/10.1126/science.267326>, doi: 10.1126/SCIENCE.267326. 12
- [20] Leon Glass and Michael Mackey. Mackey-Glass equation. *Scholarpedia*, 5(3):6908, 2010. doi:10.4249/scholarpedia.6908. 12, 90
- [21] Peter Grassberger and Itamar Procaccia. Measuring the strangeness of strange attractors. *Physica D: Nonlinear Phenomena*, 9(1-2):189–208, 10 1983. doi:10.1016/0167-2789(83)90298-1. 12
- [22] Jacob Morra and Mark Daley. Using Connectome Features to Constrain Echo State Networks. *Proceedings of the International Joint Conference*

REFERENCES

- on *Neural Networks*, 2023-June, 2023. doi:[10.1109/IJCNN54540.2023.10191832](https://doi.org/10.1109/IJCNN54540.2023.10191832). 12
- [23] Naima Chouikhi, Boudour Ammar, Nizar Rokbani, and Adel M. Alimi. PSO-based analysis of Echo State Network parameters for time series forecasting. *Applied Soft Computing*, 55:211–225, 6 2017. doi:[10.1016/J.ASOC.2017.01.049](https://doi.org/10.1016/J.ASOC.2017.01.049). 12
- [24] Arjun Panesar. *Machine learning and AI for healthcare*. Springer, 2019. 12
- [25] A.V.L.N. Sujith, Guna Sekhar Sajja, V. Mahalakshmi, Shibili Nuhmani, and B. Prasanalakshmi. Systematic review of smart health monitoring using deep learning and Artificial intelligence. *Neuroscience Informatics*, 2(3):100028, 9 2022. doi:[10.1016/J.NEURI.2021.100028](https://doi.org/10.1016/J.NEURI.2021.100028). 12
- [26] Ivens Portugal, Paulo Alencar, and Donald Cowan. The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications*, 97:205–227, 5 2018. doi:[10.1016/J.ESWA.2017.12.020](https://doi.org/10.1016/J.ESWA.2017.12.020). 12
- [27] AI vs. Machine Learning: How Do They Differ? | Google Cloud. URL: <https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning?hl=en>. 12
- [28] Francis McNamee, Schahram Dustadar, Peter Kilpatrick, Weisong Shi, Ivor Spence, and Blesson Varghese. A Case For Adaptive Deep Neural Networks in Edge Computing. *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 43–52, 8 2021. URL: <https://ieeexplore.ieee.org/abstract/document/9582182>, doi:[10.1109/CLOUD53861.2021.00017](https://doi.org/10.1109/CLOUD53861.2021.00017). 13
- [29] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, 7 2006. doi:[10.1162/NECO.2006.18.7.1527](https://doi.org/10.1162/NECO.2006.18.7.1527). 13
- [30] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. In Bernhard Schölkopf,

REFERENCES

- John Platt, and Thomas Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. 13
- [31] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. doi:10.1109/TPAMI.2013.50. 13
- [32] Hongbo Gao, Bo Cheng, Jianqiang Wang, Keqiang Li, Jianhui Zhao, and Deyi Li. Object Classification Using CNN-Based Fusion of Vision and LIDAR in Autonomous Vehicle Environment. *IEEE Transactions on Industrial Informatics*, 14(9):4224–4230, 9 2018. doi:10.1109/TII.2018.2822828. 13
- [33] Navdeep Jaitly, Patrick Nguyen, Andrew Senior, and Vincent Vanhoucke. Application Of Pretrained Deep Neural Networks To Large Vocabulary Speech Recognition. 2012. URL: <https://research.google/pubs/pub38130/>. 13
- [34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016. URL: <https://www.deeplearningbook.org/>. 14
- [35] Emanuele Torti, Alessandro Fontanella, Mirto Musci, Nicola Blago, Danilo Pau, Francesco Leporati, and Marco Piastra. Embedded real-time fall detection with deep learning on wearable devices. *Proceedings - 21st Euromicro Conference on Digital System Design, DSD 2018*, pages 405–412, 10 2018. doi:10.1109/DSD.2018.00075. 14, 30
- [36] Brian Coffen and Md Shaad Mahmud. TinyDL: Edge computing and deep learning based real-time hand gesture recognition using wearable sensor. *2020 IEEE International Conference on E-Health Networking, Application and Services, HEALTHCOM 2020*, 3 2021. doi:10.1109/HEALTHCOM49281.2021.9399005. 14
- [37] Chung Wen Hung, Jun Rong Wu, and Ching Hung Lee. Device Light Fingerprints Identification Using MCU-Based Deep Learning Approach. *IEEE Access*, 9:168134–168140, 2021. doi:10.1109/ACCESS.2021.3135448. 14

-
- [38] Maurizio Capra, Riccardo Peloso, Guido Masera, Massimo Ruo Roch, and Maurizio Martina. Edge Computing: A Survey On the Hardware Requirements in the Internet of Things World. *Future Internet 2019, Vol. 11, Page 100*, 11(4):100, 4 2019. URL: <https://www.mdpi.com/1999-5903/11/4/100/htm><https://www.mdpi.com/1999-5903/11/4/100>, doi:10.3390/FI11040100. 15, 18
- [39] Rajesh Gupta, Dakshita Reebadiya, and Sudeep Tanwar. 6G-enabled Edge Intelligence for Ultra -Reliable Low Latency Applications: Vision and Mission. *Computer Standards & Interfaces*, 77:103521, 8 2021. doi:10.1016/J.CSI.2021.103521. 15, 18
- [40] Xingzhou Zhang, Yifan Wang, Sidi Lu, Liangkai Liu, Lanyu Xu, and Weisong Shi. OpenEI: An open framework for edge intelligence. *Proceedings - International Conference on Distributed Computing Systems*, 2019-July:1840–1851, 7 2019. doi:10.1109/ICDCS.2019.00182. 15, 16, 18
- [41] Luis M. Vaquero and Luis Roderio-Merino. Finding your Way in the Fog. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 10 2014. URL: <https://dl.acm.org/doi/10.1145/2677046.2677052>, doi:10.1145/2677046.2677052. 15
- [42] Yu Cao, Songqing Chen, Peng Hou, and Donald Brown. FAST: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation. *Proceedings of the 2015 IEEE International Conference on Networking, Architecture and Storage, NAS 2015*, pages 2–11, 9 2015. doi:10.1109/NAS.2015.7255196. 15, 30
- [43] Ryan A. Cooke and Suhaib A. Fahmy. A model for distributed in-network and near-edge computing with heterogeneous hardware. *Future Generation Computer Systems*, 105:395–409, 4 2020. doi:10.1016/J.FUTURE.2019.11.040. 15
- [44] Megha Sharma, Abhinav Tomar, and Abhishek Hazra. Edge Computing for Industry 5.0: Fundamental, Applications, and Research Challenges. *IEEE*

-
- Internet of Things Journal*, 11(11):19070–19093, 6 2024. doi:10.1109/JIOT.2024.3359297. 15
- [45] Song-Mi Lee, Sang Min Yoon, and Heeryon 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, 2 2017. doi:10.1109/BIGCOMP.2017.7881728. 15, 36, 48, 55
- [46] Md Zia Uddin. A wearable sensor-based activity prediction system to facilitate edge computing in smart healthcare system. *Journal of Parallel and Distributed Computing*, 123:46–53, 1 2019. doi:10.1016/J.JPDC.2018.08.010. 15
- [47] Baotong Chen, Jiafu Wan, Antonio Celesti, Di Li, Haider Abbas, and Qin Zhang. Edge Computing in IoT-Based Manufacturing. *IEEE Communications Magazine*, 56(9):103–109, 2018. doi:10.1109/MCOM.2018.1701231. 15
- [48] Fei Luo, Salabat Khan, Yandao Huang, and Kaishun Wu. Binarized Neural Network for Edge Intelligence of Sensor-Based Human Activity Recognition. *IEEE Transactions on Mobile Computing*, 22(3):1356–1368, 3 2023. doi:10.1109/TMC.2021.3109940. 15, 16, 36, 48
- [49] Haogang Feng, Gaoze Mu, Shida Zhong, Peichang Zhang, and Tao Yuan. Benchmark Analysis of YOLO Performance on Edge Intelligence Devices. *Cryptography 2022*, Vol. 6, Page 16, 6(2):16, 4 2022. URL: <https://www.mdpi.com/2410-387X/6/2/16/html><https://www.mdpi.com/2410-387X/6/2/16>, doi:10.3390/CRYPTOGRAPHY6020016. 15, 36
- [50] Massimo Merenda, Carlo Porcaro, and Demetrio Iero. Edge Machine Learning for AI-Enabled IoT Devices: A Review. *Sensors 2020*, Vol. 20, Page 2533, 20(9):2533, 4 2020. URL: <https://www.mdpi.com/1424-8220/20/9/2533/html><https://www.mdpi.com/1424-8220/20/9/2533>, doi:10.3390/S20092533. 15

REFERENCES

- [51] Rongkang Dong, Yuyi Mao, and Jun Zhang. Resource-Constrained Edge AI with Early Exit Prediction. *Journal of Communications and Information Networks*, 7(2):122–134, 6 2022. doi:10.23919/JCIN.2022.9815196. 15
- [52] L Lovén, T Leppänen, E Peltonen, J Partala The 1st 6G wireless ..., and undefined 2019. EdgeAI: A vision for distributed, edge-native artificial intelligence in future 6G networks. *The 1st 6G Wireless Summit*, pages 1–2, 2019. URL: <http://jultika.oulu.fi/files/nbnfi-fe2019050314180.pdf>. 15, 16
- [53] Wei Yang Bryan Lim, Zehui Xiong, Dusit Niyato, Xianbin Cao, Chunyan Miao, Sumei Sun, and Qiang Yang. Realizing the Metaverse with Edge Intelligence: A Match Made in Heaven. *IEEE Wireless Communications*, pages 1–9, 2022. doi:10.1109/MWC.018.2100716. 16
- [54] Ella Peltonen, Mehdi Bennis, Michele Capobianco, Merouane Debbah, Aaron Ding, Felipe Gil-Castiñeira, Marko Jurmu, Teemu Karvonen, Markus Kelanti, Adrian Kliks, Teemu Leppänen, Lauri Lovén, Tommi Mikkonen, Ashwin Rao, Sumudu Samarakoon, Kari Seppänen, Paweł Sroka, Sasu Tarkoma, and Tingting Yang. 6G White Paper on Edge Intelligence. 9 2020. URL: <https://arxiv.org/abs/2004.14850v1>. 16
- [55] Wei Yang Bryan Lim, Zehui Xiong, Dusit Niyato, Xianbin Cao, Chunyan Miao, Sumei Sun, and Qiang Yang. Realizing the Metaverse with Edge Intelligence: A Match Made in Heaven. *IEEE Wireless Communications*, 2022. doi:10.1109/MWC.018.2100716. 16
- [56] Rajan Gupta, Sanjana Das, and Saibal Kumar Pal. EdgeAI: Concept and Architecture. *EdgeAI for Algorithmic Government*, pages 31–55, 2023. URL: https://link.springer.com/chapter/10.1007/978-981-19-9798-3_3, doi:10.1007/978-981-19-9798-3{_}3. 16
- [57] Md Motaharul Islam, Mohammad Kamrul Hasan, Shayla Islam, Mohammed Balfaqih, Ahmed Ibrahim Alzahrani, Nasser Alalwan, Nurhizam Safie, Zaheed Ahmed Bhuiyan, Rahul Thakkar, and Taher M. Ghazal. Enabling pandemic-resilient healthcare: Narrowband Internet of Things and edge intelligence for real-time monitoring.

- CAAI *Transactions on Intelligence Technology*, 2024. URL: <https://onlinelibrary.wiley.com/doi/full/10.1049/cit2.12314><https://onlinelibrary.wiley.com/doi/abs/10.1049/cit2.12314><https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/cit2.12314>, doi:10.1049/CIT2.12314. 16
- [58] Chen Chen, Guorun Yao, Lei Liu, Qingqi Pei, Houbing Song, and Schahram Dustdar. A Cooperative Vehicle-Infrastructure System for Road Hazards Detection With Edge Intelligence. *IEEE Transactions on Intelligent Transportation Systems*, 24(5):5186–5198, 5 2023. doi:10.1109/TITS.2023.3241251. 16
- [59] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 9 2020. doi:10.1109/JIOT.2020.2984887. 16
- [60] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proceedings of the IEEE*, 2019. doi:10.1109/JPROC.2019.2918951. 16
- [61] Yin Zhang, Chi Jiang, Binglei Yue, Jiafu Wan, and Mohsen Guizani. Information fusion for edge intelligence: A survey. *Information Fusion*, 81:171–186, 9 2022. URL: <https://linkinghub.elsevier.com/retrieve/pii/S1566253521002438>, doi:10.1016/J.INFFUS.2021.11.018. 16
- [62] Yu Yuan Liu, Hong Sheng Zheng, Yu Fang Hu, Chen Fong Hsu, and Tsung Tai Yeh. TinyTS: Memory-Efficient TinyML Model Compiler Framework on Microcontrollers. *Proceedings - International Symposium on High-Performance Computer Architecture*, pages 848–860, 2024. doi:10.1109/HPCA57654.2024.00070. 16
- [63] Kunran Xu, Huawei Zhang, Yishi Li, Yuhao Zhang, Rui Lai, and Yi Liu. An Ultra-Low Power TinyML System for Real-Time Visual Processing at Edge. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 70(7):2640–2644, 7 2023. doi:10.1109/TCSII.2023.3239044. 16

-
- [64] Partha Pratim Ray. A review on TinyML: State-of-the-art and prospects. *Journal of King Saud University - Computer and Information Sciences*, 34(4):1595–1623, 4 2022. doi:[10.1016/J.JKSUCI.2021.11.019](https://doi.org/10.1016/J.JKSUCI.2021.11.019). 16, 19
 - [65] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong. MLPerf Tiny Benchmark. 9 2021. URL: <https://arxiv.org/abs/2106.07597v4>, doi: [10.48550/arxiv.2106.07597](https://doi.org/10.48550/arxiv.2106.07597). 16, 41
 - [66] Yongchang Li, Juncheng Jia, Yan Zuo, and Weipeng Zhu. TinyOOD: Effective out-of-Distribution Detection for TinyML. *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 6 2023. URL: <https://ieeexplore.ieee.org/document/10094746/>, doi:[10.1109/ICASSP49357.2023.10094746](https://doi.org/10.1109/ICASSP49357.2023.10094746). 16
 - [67] Xiaojie Wang, Beibei Wang, Yu Wu, Zhaolong Ning, Song Guo, and Fei Richard Yu. A Survey on Trustworthy Edge Intelligence: From Security and Reliability To Transparency and Sustainability. *IEEE Communications Surveys & Tutorials*, pages 1–1, 2024. URL: <https://ieeexplore.ieee.org/document/10640100/>, doi:[10.1109/COMST.2024.3446585](https://doi.org/10.1109/COMST.2024.3446585). 16
 - [68] Yuanming Shi, Kai Yang, Tao Jiang, Jun Zhang, and Khaled B. Letaief. Communication-Efficient Edge AI: Algorithms and Systems. *IEEE Communications Surveys and Tutorials*, 22(4):2167–2191, 10 2020. doi:[10.1109/COMST.2020.3007787](https://doi.org/10.1109/COMST.2020.3007787). 16
 - [69] Pedro Nunes, Eugénio Rocha, and José Paulo Santos. Using Intelligent Edge Devices for Predictive Maintenance on Injection Molds. *Applied Sciences* 2023, Vol. 13, Page 7131, 13(12):7131, 6 2023. URL: <https://www.mdpi.com/2076-3417/13/12/7131/htmhttps://www.mdpi.com/2076-3417/13/12/7131>, doi:[10.3390/APP13127131](https://doi.org/10.3390/APP13127131). 16

-
- [70] Bowei Dong, Qiongfeng Shi, Yanqin Yang, Feng Wen, Zixuan Zhang, and Chengkuo Lee. Technology evolution from self-powered sensors to AIoT enabled smart homes. *Nano Energy*, 79:105414, 9 2021. doi:10.1016/J.NANOEN.2020.105414. 16
- [71] Zhihan Lv, Dongliang Chen, and Qingjun Wang. Diversified Technologies in Internet of Vehicles under Intelligent Edge Computing. *IEEE Transactions on Intelligent Transportation Systems*, 22(4):2048–2059, 4 2021. doi:10.1109/TITS.2020.3019756. 18
- [72] Warren J. Gross, Brett H. Meyer, and Arash Ardakani. Hardware-Aware Design for Edge Intelligence. *IEEE Open Journal of Circuits and Systems*, 2:113–127, 12 2020. doi:10.1109/OJCAS.2020.3047418. 19
- [73] Raspberry Pi 4 Model B specifications – Raspberry Pi. URL: <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>. 19
- [74] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Peter Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Niu Wenxu, and Xu Xuesong. MLPerf Tiny Benchmark. 6 2021. URL: <https://arxiv.org/abs/2106.07597v4>, doi:10.48550/arxiv.2106.07597. 19, 22
- [75] Brian Coffen and Md Shaad Mahmud. TinyDL: Edge computing and deep learning based real-time hand gesture recognition using wearable sensor. *2020 IEEE International Conference on E-Health Networking, Application and Services, HEALTHCOM 2020*, 3 2021. doi:10.1109/HEALTHCOM49281.2021.9399005. 19
- [76] Kristof Tjonck, Chandrakanth R. Kancharla, Jens Vankeirsbilck, Hans Hallez, Jeroen Boydens, and Bozheng Pang. Real-Time Activity Tracking using TinyML to Support Elderly Care. In *2021 XXX International Scientific Conference Electronics (ET)*, pages 1–6. IEEE, 9 2021. doi:10.1109/ET52713.2021.9579991. 19, 31, 59, 60

-
- [77] Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. 3 2018. 20
- [78] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. Resiliency of Deep Neural Networks under Quantization. 11 2015. 20
- [79] Tian Huang, Tao Luo, and Joey Tianyi Zhou. Adaptive precision training for resource constrained devices. *Proceedings - International Conference on Distributed Computing Systems*, 2020-November:1403–1408, 11 2020. doi:10.1109/ICDCS47774.2020.00185. 20, 64
- [80] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-Training Quantization for Vision Transformer. *Advances in Neural Information Processing Systems*, 34:28092–28103, 12 2021. URL: <https://gitee.com/mindspore/models/tree/master/research/cv/VT->. 20
- [81] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or Down? Adaptive Rounding for Post-Training Quantization. 2020. 20, 64
- [82] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8612–8620, 2019. 20, 36
- [83] Manuele Rusci, Marco Fariselli, Alessandro Capotondi, and Luca Benini. Leveraging Automated Mixed-Low-Precision Quantization for Tiny Edge Microcontrollers. *Communications in Computer and Information Science*, 1325:296–308, 2020. URL: https://link.springer.com/chapter/10.1007/978-3-030-66770-2_22, doi:10.1007/978-3-030-66770-2{_}22/TABLES/2. 21, 36
- [84] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks. In *Advances in Neural Information Processing Systems 29*, number 4, 2016. URL: <https://github.com/itayhubara/BinaryNet>. 21

-
- [85] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards Accurate Binary Convolutional Neural Network. In I Guyon, U Von Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/b1a59b315fc9a3002ce38bbe070ec3f5-Paper.pdf. 21
- [86] Wei Tang, Gang Hua, and Liang Wang. How to Train a Compact Binary Neural Network with High Accuracy? In *AAAI Publications, Thirty-First AAAI Conference on Artificial Intelligence*, 2017. URL: <https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/viewPaper/14619>. 21
- [87] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034. IEEE, 12 2015. doi:10.1109/ICCV.2015.123. 21
- [88] Nael Fasfous, Manoj Rohit Vemparala, Alexander Frickenstein, Lukas Frickenstein, Mohamed Badawy, and Walter Stechele. BinaryCoP: Binary Neural Network-based COVID-19 Face-Mask Wear and Positioning Predictor on Edge Devices. *2021 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2021 - In conjunction with IEEE IPDPS 2021*, pages 108–115, 6 2021. doi:10.1109/IPDPSW52791.2021.00024. 21
- [89] B. S. Ajay and Madhav Rao. Binary neural network based real time emotion detection on an edge computing device to detect passenger anomaly. *Proceedings of the IEEE International Conference on VLSI Design*, 2021-February:175–180, 2 2021. doi:10.1109/VLSID51830.2021.00035. 21
- [90] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal Brain Surgeon. In S Hanson, J Cowan, and C Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL: https://proceedings.neurips.cc/paper_files/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf. 22

REFERENCES

- [91] Yann LeCun, John Denker, and Sara Solla. Optimal Brain Damage. In D Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf. 22
- [92] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. 8 2016. 22, 64, 99
- [93] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks. 8 2018. 22, 64
- [94] Sunil Vadera and Salem Ameen. Methods for Pruning Deep Neural Networks. *IEEE Access*, 10:63280–63300, 2022. doi:10.1109/ACCESS.2022.3182659. 22
- [95] Yang He and Lingao Xiao. Structured Pruning for Deep Convolutional Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(5):2900–2919, 5 2024. doi:10.1109/TPAMI.2023.3334614. 22
- [96] Seul-Ki Yeom, Philipp Seegerer, Sebastian Lapuschkin, Alexander Binder, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Pruning by explaining: A novel criterion for deep neural network pruning. *Pattern Recognition*, 115:107899, 7 2021. doi:10.1016/j.patcog.2021.107899. 22
- [97] Josen Daniel De Leon and Rowel Atienza. Depth Pruning with Auxiliary Networks for Tinnym. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3963–3967. IEEE, 5 2022. doi:10.1109/ICASSP43922.2022.9746843. 22
- [98] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David Patterson, Danilo Pau, Jae-sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. Benchmarking TinyML Systems: Challenges and Direction. 3 2020. 22, 41

REFERENCES

- [99] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. 3 2015. 22
- [100] Zhiqiang Shen and Eric Xing. A Fast Knowledge Distillation Framework for Visual Recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 13684 LNCS:673–690, 2022. URL: https://link.springer.com/chapter/10.1007/978-3-031-20053-3_39, doi: 10.1007/978-3-031-20053-3{_}39/TABLES/11. 22
- [101] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:3962–3971, 6 2019. doi:10.1109/CVPR.2019.00409. 22
- [102] Frederick Tung and Greg Mori. Similarity-Preserving Knowledge Distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019*, pages 1365–1374, 2019. 22, 77
- [103] Jang Hyun Cho and Bharath Hariharan. On the Efficacy of Knowledge Distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019*, pages 4794–4802, 2019. 23
- [104] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. *Advances in Neural Information Processing Systems*, 30, 2017. 23, 25
- [105] Karim Ahmed, Nitish Shirish Keskar, and Richard Socher. Weighted Transformer Network for Machine Translation. 11 2017. URL: <https://arxiv.org/abs/1711.02132v1>. 23
- [106] Nathalia Nascimento, Paulo Alencar, Carlos Lucena, Donald D Cowan, David R Cheriton, and Donald Cowan. A Context-Aware Machine Learning-based Approach. 2018. URL: <https://www.researchgate.net/publication/328874557>. 24

-
- [107] Amr Gaballah, Abhishek Tiwari, Shrikanth Narayanan, and Tiago H. Falk. Context-Aware Speech Stress Detection in Hospital Workers Using Bi-LSTM Classifiers. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2021-June:8348–8352, 2021. doi:10.1109/ICASSP39728.2021.9414666. 24
- [108] Nafiul Rashid, Trier Mortlock, and Mohammad Abdullah Al Faruque. SELF-CARE: Selective Fusion with Context-Aware Low-Power Edge Computing for Stress Detection. *Proceedings - 18th Annual International Conference on Distributed Computing in Sensor Systems, DCOSS 2022*, pages 49–52, 2022. doi:10.1109/DCOSS54816.2022.00019. 24
- [109] Md Saif Hassan Onim, Elizabeth Rhodus, and Himanshu Thapliyal. A Review of Context-Aware Machine Learning for Stress Detection. *IEEE Consumer Electronics Magazine*, 2023. doi:10.1109/MCE.2023.3278076. 24
- [110] Yingying Jiang, Wei Li, M. Shamim Hossain, Min Chen, Abdulhameed Alelaiwi, and Muneer Al-Hammadi. A snapshot research and implementation of multimodal information fusion for data-driven emotion recognition. *Information Fusion*, 53:209–221, 1 2020. doi:10.1016/J.INFFUS.2019.06.019. 25
- [111] Ze Zhang, Michael Farnsworth, Boyang Song, Divya Tiwari, and Ashutosh Tiwari. Deep Transfer Learning With Self-Attention for Industry Sensor Fusion Tasks. *IEEE Sensors Journal*, 22(15):15235–15247, 8 2022. doi:10.1109/JSEN.2022.3186505. 25
- [112] Wenjin Tao, Haodong Chen, Md Moniruzzaman, Ming C. Leu, Zhaozheng Yi, and Ruwen Qin. Attention-Based Sensor Fusion for Human Activity Recognition Using IMU Signals. 12 2021. URL: <https://arxiv.org/abs/2112.11224v1>, doi:10.48550/arxiv.2112.11224. 25
- [113] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *32nd International Conference on Machine Learning, ICML 2015*, 3:2048–2057,

- 2 2015. URL: <https://arxiv.org/abs/1502.03044v3>, doi:10.48550/arxiv.1502.03044. 25
- [114] Heechan Yang, Ji Ye Kim, Hyongsuk Kim, and Shyam P. Adhikari. Guided Soft Attention Network for Classification of Breast Cancer Histopathology Images. *IEEE Transactions on Medical Imaging*, 39(5):1306–1315, 5 2020. doi:10.1109/TMI.2019.2948026. 25
- [115] Minh Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 8 2015. URL: <https://arxiv.org/abs/1508.04025v5>, doi:10.48550/arxiv.1508.04025. 25
- [116] Di Song, Junxian Shen, Tianchi Ma, and Feiyun Xu. Two-level fusion of multi-sensor information for compressor blade crack detection based on self-attention mechanism. *Structural Health Monitoring*, 8 2022. URL: <https://journals.sagepub.com/doi/full/10.1177/14759217221116599>, doi:10.1177/14759217221116599/ASSET/IMAGES/LARGE/10.1177{_}14759217221116599-FIG11.JPEG. 25
- [117] Kieran Woodward, Eiman Kanjo, David J Brown, and T M McGinnity. Towards Personalised Mental Wellbeing Recognition On-Device using Transfer Learning "in the Wild". In *IEEE International Smart Cities Conference 2021*, 2021. 26, 32, 55, 70
- [118] Mohammad Parsa Hosseini, Tuyen X. Tran, Dario Pompili, Kost Elisevich, and Hamid Soltanian-Zadeh. Deep Learning with Edge Computing for Localization of Epileptogenicity Using Multimodal rs-fMRI and EEG Big Data. *Proceedings - 2017 IEEE International Conference on Autonomic Computing, ICAC 2017*, pages 83–92, 8 2017. doi:10.1109/ICAC.2017.41.26
- [119] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1930–1939,

REFERENCES

- 7 2018. URL: <https://dl.acm.org/doi/10.1145/3219819.3220007>, doi:10.1145/3219819.3220007/SUPPL{_}FILE/MA{_}MODELING{_}RELATIONSHIPS.MP4. 26
- [120] Lijun Zhang, Qizheng Yang, Xiao Liu, and Hui Guan. Rethinking Hard-Parameter Sharing in Multi-Domain Learning. *Proceedings - IEEE International Conference on Multimedia and Expo*, 2022-July, 2022. doi:10.1109/ICME52920.2022.9859706. 26
- [121] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-Stitch Networks for Multi-Task Learning, 2016. 26
- [122] Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. Latent Multi-Task Architecture Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4822–4829, 7 2019. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/4410>, doi:10.1609/AAAI.V33I01.33014822. 26
- [123] Yu Zhang and Qiang Yang. Special Topic: Machine Learning An overview of multi-task learning. *National Science Review*, 5:30–43, 2018. URL: <https://academic.oup.com/nsr/article/5/1/30/4101432>, doi:10.1093/nsr/nwx105. 27
- [124] Liangying Peng, Ling Chen, Zhenan Ye, and Yi Zhang. AROMA: A Deep Multi-Task Learning Based Simple and Complex Human Activity Recognition Method Using Wearable Sensors. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, pages 439–461, 7 2018. URL: <https://dl.acm.org/doi/10.1145/3214277>, doi:10.1145/3214277. 27
- [125] D Dong, H Wu, W He, and D Yu. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 1723–1732, 2015. URL: <https://aclanthology.org/P15-1166.pdf>. 27

-
- [126] Chi Su, Fan Yang, Shiliang Zhang, Qi Tian, Larry Steven Davis, and Wen Gao. Multi-Task Learning with Low Rank Attribute Embedding for Multi-Camera Person Re-Identification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1167–1181, 5 2018. doi:[10.1109/TPAMI.2017.2679002](https://doi.org/10.1109/TPAMI.2017.2679002). 27
- [127] Shikun Liu, Edward Johns, and Andrew J. Davison. End-To-End Multi-Task Learning With Attention, 2019. 27
- [128] Keishi Ishihara, Anssi Kanervisto, Jun Miura, and Ville Hautamaki. Multi-Task Learning With Attention for End-to-End Autonomous Driving, 2021. 27
- [129] Lei Han and Yu Zhang. Learning Multi-Level Task Groups in Multi-Task Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1):2638–2644, 2 2015. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/9581>, doi:[10.1609/AAAI.V29I1.9581](https://doi.org/10.1609/AAAI.V29I1.9581). 27
- [130] Ammar Sherif, Abubakar Abid, Mustafa Elattar, and Mohamed ElHelw. STG-MTL: scalable task grouping for multi-task learning using data maps. *Machine Learning: Science and Technology*, 5(2):025068, 6 2024. URL: <https://iopscience.iop.org/article/10.1088/2632-2153/ad4e04><https://iopscience.iop.org/article/10.1088/2632-2153/ad4e04/meta>, doi:[10.1088/2632-2153/AD4E04](https://doi.org/10.1088/2632-2153/AD4E04). 27
- [131] Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R. Arabnia. A Brief Review of Domain Adaptation. pages 877–894, 2021. URL: https://link.springer.com/chapter/10.1007/978-3-030-71704-9_65, doi:[10.1007/978-3-030-71704-9_{_}65](https://doi.org/10.1007/978-3-030-71704-9_{_}65). 27
- [132] Mark Dredze, Alex Kulesza, and Koby Crammer. Multi-domain learning by confidence-weighted parameter combination. *Machine Learning*, 79(1-2):123–149, 10 2010. URL: <https://link.springer.com/article/10.1007/s10994-009-5148-0>, doi:[10.1007/S10994-009-5148-0/METRICS](https://doi.org/10.1007/S10994-009-5148-0/METRICS). 27
- [133] Kazuki Omi, Jun Kimata, and Toru Tamaki. Model-Agnostic Multi-Domain Learning with Domain-Specific Adapters for Action Recognition.

- IEICE Transactions on Information and Systems*, E105D(12):2119–2126, 12 2022. doi:10.1587/TRANSINF.2022EDP7058. 27
- [134] Jasdeep Singh, Subrahmanyam Murala, and G Sankara Raju Kosuru. Multi Domain Learning for Motion Magnification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13914–13923, 2023. URL: <https://github.com/jasdeep-singh-007/Multi-Domain-Learning-for-Motion-Magnification>. 27
- [135] Tae-Hyun Oh, Ronnachai Jaroensri, Changil Kim, Mohamed Elgharib, Frédo Durand, William T Freeman, and Wojciech Matusik. Learning-based Video Motion Magnification. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 633–648, 2018. 27
- [136] Shiyi Jiang, Yuan Li, Farshad Firouzi, and Krishnendu Chakrabarty. Federated clustered multi-domain learning for health monitoring. *Scientific Reports 2023 14:1*, 14(1):1–12, 1 2024. URL: <https://www.nature.com/articles/s41598-024-51344-9>, doi:10.1038/s41598-024-51344-9. 28
- [137] Keyao Wang, Guosheng Zhang, Haixiao Yue, Ajian Liu, Gang Zhang, Haocheng Feng, Junyu Han, Errui Ding, and Jingdong Wang. Multi-Domain Incremental Learning for Face Presentation Attack Detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(6):5499–5507, 3 2024. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/28359>, doi:10.1609/AAAI.V38I6.28359. 28
- [138] Zihan Zhang, Xiaoming Jin, Lianghao Li, Guiguang Ding, and Qiang Yang. Multi-Domain Active Learning for Recommendation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1):2358–2364, 3 2016. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/10291>, doi:10.1609/AAAI.V30I1.10291. 28
- [139] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. Multi-Task Learning for Dense Prediction Tasks: A Survey. *IEEE Transactions on Pattern Analysis and*

-
- Machine Intelligence*, 44(7):3614–3633, 7 2022. doi:[10.1109/TPAMI.2021.3054719](https://doi.org/10.1109/TPAMI.2021.3054719). 28
- [140] Wenlu Zhang, Rongjian Li, Tao Zeng, Qian Sun, Sudhir Kumar, Jieping Ye, and Shuiwang Ji. Deep model based transfer and multi-task learning for biological image analysis. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015-August:1475–1484, 8 2015. URL: <https://dl.acm.org/doi/10.1145/2783258.2783304>, doi:[10.1145/2783258.2783304](https://doi.org/10.1145/2783258.2783304). 28
- [141] Pim Moeskops, Jelmer M. Wolterink, Bas H.M. van der Velden, Kenneth G.A. Gilhuijs, Tim Leiner, Max A. Viergever, and Ivana Išgum. Deep learning for multi-task medical image segmentation in multiple modalities. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9901 LNCS:478–486, 2016. URL: https://link.springer.com/chapter/10.1007/978-3-319-46723-8_55, doi:[10.1007/978-3-319-46723-8_{55}/FIGURES/3](https://doi.org/10.1007/978-3-319-46723-8_{55}/FIGURES/3). 28
- [142] Jun Zhang, Mingxia Liu, and Dinggang Shen. Detecting Anatomical Landmarks from Limited Medical Imaging Data Using Two-Stage Task-Oriented Deep Neural Networks. *IEEE Transactions on Image Processing*, 26(10):4753–4764, 10 2017. doi:[10.1109/TIP.2017.2721106](https://doi.org/10.1109/TIP.2017.2721106). 28
- [143] Vignesh Sampath, Iñaki Murtua, Juan José Aguilar Martín, Andoni Rivera, Jorge Molina, and Aitor Gutierrez. Attention-Guided Multitask Learning for Surface Defect Identification. *IEEE Transactions on Industrial Informatics*, 19(9):9713–9721, 9 2023. doi:[10.1109/TII.2023.3234030](https://doi.org/10.1109/TII.2023.3234030). 28
- [144] Deblina Bhattacharjee, Tong Zhang, Sabine Süssstrunk, and Mathieu Salzmann. MulT: An End-to-End Multitask Learning Transformer, 2022. 29
- [145] Chris Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently Identifying Task Groupings for Multi-Task Learning. *Advances in Neural Information Processing Systems*, 34:27503–27516, 12 2021. 29

- [146] Aimé Cedric Muhoza, Emmanuel Bergeret, Corinne Brdys, and Francis Gary. Power consumption reduction for IoT devices thanks to Edge-AI: Application to human activity recognition. *Internet of Things*, 24:100930, 12 2023. doi:10.1016/J.IOT.2023.100930. 29
- [147] FOMO: Object detection for constrained devices | Edge Impulse Documentation. URL: <https://docs.edgeimpulse.com/docs/edge-impulse-studio/learning-blocks/object-detection/fomo-object-detection-for-constrained-devices>. 29
- [148] Julian Moosmann, Marco Giordano, Christian Vogt, and Michele Magno. TinyissimoYOLO: A Quantized, Low-Memory Footprint, TinyML Object Detection Network for Low Power Microcontrollers. *AICAS 2023 - IEEE International Conference on Artificial Intelligence Circuits and Systems, Proceeding*, 2023. doi:10.1109/AICAS57966.2023.10168657. 29
- [149] Liam Boyle, Julian Moosmann, Nicolas Baumann, Seonyeong Heo, and Michele Magno. DSORT-MCU: Detecting Small Objects in Real-Time on Microcontroller Units. *IEEE Sensors Journal*, 2024. doi:10.1109/JSEN.2024.3425904. 29
- [150] Jayakanth Kunhoth, Mahdi Alkaeed, Adeel Ehsan, and Junaid Qadir. VisualAid+: Assistive System for Visually Impaired with TinyML Enhanced Object Detection and Scene Narration. *2023 International Symposium on Networks, Computers and Communications, ISNCC 2023*, 2023. doi:10.1109/ISNCC58260.2023.10323988. 30
- [151] Dariusz Mrozek, Anna Koczur, and Bożena Małysiak-Mrozek. Fall detection in older adults with mobile IoT devices and machine learning in the cloud and on the edge. *Information Sciences*, 537:132–147, 10 2020. doi:10.1016/J.INS.2020.05.070. 30
- [152] Daohua Pan, Hongwei Liu, Dongming Qu, and Zhan Zhang. CNN-Based Fall Detection Strategy with Edge Computing Scheduling in Smart Cities. *Electronics 2020, Vol. 9, Page 1780*, 9(11):1780, 10 2020. URL: <https://www.mdpi.com/2079-9292/9/11/>

- [1780/htmhttps://www.mdpi.com/2079-9292/9/11/1780](https://www.mdpi.com/2079-9292/9/11/1780), doi:10.3390/ELECTRONICS9111780. 30
- [153] Vahideh Hayyolalam, Moayad Aloqaily, Oznur Ozkasap, and Mohsen Guizani. Edge Intelligence for Empowering IoT-Based Healthcare Systems. *IEEE Wireless Communications*, 28(3):6–14, 6 2021. doi:10.1109/MWC.001.2000345. 30
- [154] Ronald Mutegeki and Dong Seog Han. A CNN-LSTM Approach to Human Activity Recognition. *2020 International Conference on Artificial Intelligence in Information and Communication, ICAIIC 2020*, pages 362–366, 2 2020. doi:10.1109/ICAIIIC48513.2020.9065078. 30, 55
- [155] Tahmina Zebin, Patricia J Scully, Niels Peek, Alexander J Casson, and Krikor B Ozanyan. Design and Implementation of a Convolutional Neural Network on an Edge Computing Smartphone for Human Activity Recognition. *IEEE Access*, 7:133509–133520, 2019. 31, 60
- [156] Song Mi Lee, Sang Min Yoon, and Heeryon Cho. Human activity recognition from accelerometer data using Convolutional Neural Network. *2017 IEEE International Conference on Big Data and Smart Computing, Big-Comp 2017*, pages 131–134, 3 2017. 31
- [157] Shubham Gupta, Sweta Jain, Bholanath Roy, and Abhishek Deb. A TinyML Approach to Human Activity Recognition. *Journal of Physics: Conference Series*, 2273(1), 2022. 31
- [158] Alessandro Ghibellini, Luciano Bononi, and Marco Di Felice. Intelligence at the IoT Edge: Activity Recognition with Low-Power Microcontrollers and Convolutional Neural Networks. *Proceedings - IEEE Consumer Communications and Networking Conference, CCNC*, pages 707–710, 2022. 31, 60
- [159] Jennifer A. Healey and Rosalind W. Picard. Detecting stress during real-world driving tasks using physiological sensors. *IEEE Transactions on Intelligent Transportation Systems*, 6(2):156–166, 6 2005. doi:10.1109/TITS.2005.848368. 32, 70

REFERENCES

- [160] Alberto De Santos Sierra, Carmen Sanchez Avila, Javier Guerra Casanova, Gonzalo Bailador Del Pozo, and Vicente Jara Vera. Two stress detection schemes based on physiological signals for real-time applications. In *Proceedings - 2010 6th International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIHMSP 2010*, 2010. 32, 72
- [161] Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, (34), 1 2001. URL: <https://www.ai.rug.nl/minds/uploads/EchoStatesTechRep.pdf>. 32, 33, 35, 82
- [162] Herbert Jaeger. Echo state network. *Scholarpedia*, 2(9):2330, 2007. doi: 10.4249/scholarpedia.2330. 32
- [163] Michael Buehner and Peter Young. A tighter bound for the echo state property. *IEEE Transactions on Neural Networks*, 17(3):820–824, 5 2006. doi:10.1109/TNN.2006.872357. 33
- [164] Azarakhsh Jalalvand, Glenn Van Wallendael, and Rik Van De Walle. Real-Time Reservoir Computing Network-Based Systems for Detection Tasks on Visual Contents. *Proceedings - 7th International Conference on Computational Intelligence, Communication Systems and Networks, CICSyN 2015*, pages 146–151, 10 2015. doi:10.1109/CICSYN.2015.35. 33, 88, 97
- [165] Fabrizio De Vita, Giorgio Nocera, Dario Bruneo, and Sajal K. Das. A Novel Echo State Network Autoencoder for Anomaly Detection in Industrial IoT Systems. *IEEE Transactions on Industrial Informatics*, 19(8):8985–8994, 8 2023. doi:10.1109/TII.2022.3224981. 34
- [166] Daniel Estévez-Moya, Ernesto Estévez-Rams, and Hölger Kantz. Echo State Networks for the Prediction of Chaotic Systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 14335 LNCS:119–128, 2024. URL: https://link.springer.com/chapter/10.1007/978-3-031-49552-6_11, doi:10.1007/978-3-031-49552-6_{_}11/FIGURES/6. 34, 88

REFERENCES

- [167] Allen Hart, James Hook, and Jonathan Dawes. Embedding and approximation theorems for echo state networks. *Neural Networks*, 128:234–247, 8 2020. [doi:10.1016/J.NEUNET.2020.05.013](https://doi.org/10.1016/J.NEUNET.2020.05.013). 34, 35, 90
- [168] Dingyuan Li, Fu Liu, Junfei Qiao, and Rong Li. Structure optimization for echo state network based on contribution. *Tsinghua Science and Technology*, 24(1):97–105, 2 2019. [doi:10.26599/TST.2018.9010049](https://doi.org/10.26599/TST.2018.9010049). 34
- [169] Brian Whiteaker and Peter Gerstoft. Reducing echo state network size with controllability matrices. *Chaos*, 32(7), 7 2022. URL: [/aip/cha/article/32/7/073116/2835900/Reducing-echo-state-network-size-with](https://aip/cha/article/32/7/073116/2835900/Reducing-echo-state-network-size-with), [doi:10.1063/5.0071926/2835900](https://doi.org/10.1063/5.0071926/2835900). 34, 89
- [170] Geremia Pompei, Patrizio Dazzi, Valerio De Caro, and Claudio Gallicchio. Decentralized Incremental Federated Learning with Echo State Networks. pages 1–8, 9 2024. [doi:10.1109/IJCNN60899.2024.10650756](https://doi.org/10.1109/IJCNN60899.2024.10650756). 34
- [171] Rebh Soltani, Emna Benmohamed, and Hela Ltifi. Newman-Watts-Strogatz topology in deep echo state networks for speech emotion recognition. *Engineering Applications of Artificial Intelligence*, 133:108293, 7 2024. [doi:10.1016/J.ENGAPPAI.2024.108293](https://doi.org/10.1016/J.ENGAPPAI.2024.108293). 34, 92
- [172] Andrea Ceni and Claudio Gallicchio. Edge of Stability Echo State Network. *IEEE Transactions on Neural Networks and Learning Systems*, 2024. [doi:10.1109/TNNLS.2024.3400045](https://doi.org/10.1109/TNNLS.2024.3400045). 34
- [173] Ganesh K. Venayagamoorthy and Bashyal Shishir. Effects of spectral radius and settling time in the performance of echo state networks. *Neural Networks*, 22(7):861–863, 9 2009. [doi:10.1016/J.NEUNET.2009.03.021](https://doi.org/10.1016/J.NEUNET.2009.03.021). 34, 90
- [174] Ken Caluwaerts, Francis Wyffels, Sander Dieleman, and Benjamin Schrauwen. The spectral radius remains a valid indicator of the Echo state property for large reservoirs. *Proceedings of the International Joint Conference on Neural Networks*, 2013. [doi:10.1109/IJCNN.2013.6706899](https://doi.org/10.1109/IJCNN.2013.6706899). 34, 90

REFERENCES

- [175] Nils Schaetti, Michel Salomon, and Raphael Couturier. Echo State Networks-Based Reservoir Computing for MNIST Handwritten Digits Recognition. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pages 484–491. IEEE, 8 2016. doi:[10.1109/CSE-EUC-DCABES.2016.229](https://doi.org/10.1109/CSE-EUC-DCABES.2016.229). 34, 97
- [176] Menna Allah Soliman, Mostafa A. Mousa, Mahmood A. Saleh, Mahmoud Elsamanty, and Ahmed G. Radwan. Modelling and implementation of soft bio-mimetic turtle using echo state network and soft pneumatic actuators. *Scientific Reports* 2021 11:1, 11(1):1–11, 6 2021. URL: <https://www.nature.com/articles/s41598-021-91136-z>, doi: [10.1038/s41598-021-91136-z](https://doi.org/10.1038/s41598-021-91136-z). 35
- [177] Sanjukta Krishnagopal and Michelle Girvan. *Uncovering Patterns in Complex Data with Reservoir Computing and Network Analytics: A Dynamical Systems Approach*. PhD thesis, United States – Maryland, 2020. URL: <https://ntu.idm.oclc.org/login?url=https://www.proquest.com/dissertations-theses/uncovering-patterns-complex-data-with-reservoir/docview/2435287169/se-2?accountid=14693>. 35, 90
- [178] Fabrizio De Vita, Giorgio Nocera, Dario Bruneo, Valeria Tomaselli, and Mirko Falchetto. On-Device Training of Deep Learning Models on Edge Microcontrollers. In *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 62–69. IEEE, 8 2022. doi:[10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics55523.2022.00018](https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics55523.2022.00018). 35
- [179] Danilo Pau and Prem Kumar Ambrose. Automated Neural and On-Device Learning for Micro Controllers. *MELECON 2022 - IEEE Mediterranean*

- Electrotechnical Conference, Proceedings*, pages 758–763, 2022. doi:10.1109/MELECON53508.2022.9843050. 35
- [180] Danilo Pietro Pau, Prem Kumar Ambrose, and Fabrizio Maria Aymone. A Quantitative Review of Automated Neural Search and On-Device Learning for Tiny Devices. *Chips 2023, Vol. 2, Pages 130-141*, 2(2):130–141, 5 2023. URL: <https://www.mdpi.com/2674-0729/2/2/8/html><https://www.mdpi.com/2674-0729/2/2/8>, doi:10.3390/CHIPS2020008. 35
- [181] Davide Nadalini, Manuele Rusci, Luca Benini, and Francesco Conti. Reduced precision floating-point optimization for Deep Neural Network On-Device Learning on microcontrollers. *Future Generation Computer Systems*, 149:212–226, 12 2023. doi:10.1016/J.FUTURE.2023.07.020. 35
- [182] Denis Kleyko, Edward Paxon Frady, Mansour Kheffache, and Evgeny Osipov. Integer Echo State Networks: Efficient Reservoir Computing for Digital Hardware. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1688–1701, 4 2022. doi:10.1109/TNNLS.2020.3043309. 35, 43, 99
- [183] Bogdan Penkovsky, Xavier Porte, Maxime Jacquot, Laurent Larger, and Daniel Brunner. Coupled Nonlinear Delay Systems as Deep Convolutional Neural Networks. *Physical Review Letters*, 123(5):054101, 8 2019. URL: <https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.123.054101>, doi:10.1103/PHYSREVLETT.123.054101/SUPPINFO.PDF. 35
- [184] Jiawei Shao and Jun Zhang. Communication-Computation Trade-off in Resource-Constrained Edge Inference. *IEEE Communications Magazine*, 58(12):20–26, 9 2020. doi:10.1109/MCOM.001.2000373. 36
- [185] Gao Yang, Gong Hao, Lu Weijia, Wang Qinghua, Su Chen, and Ni Zhang. An Attentive Pruning Method for Edge Computing. *ACM International Conference Proceeding Series*, pages 6–10, 9 2020. doi:10.1145/3383972.3384008. 36
- [186] Arunima Sambhuta Pattanayak, Bhawani Shankar Pattnaik, Siba K Udghata, and Ajit Kumar Panda. Development of Chemical Oxygen on

- Demand (COD) Soft Sensor Using Edge Intelligence. *IEEE Sensors Journal*, 20(24):14892–14902, 9 2020. doi:10.1109/JSEN.2020.3010134. 36
- [187] Shaojun Zhang, Wei Li, Yongwei Wu, Paul Watson, and Albert Y Zomaya. Enabling edge intelligence for activity recognition in smart homes. *Proceedings - 15th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2018*, pages 228–236, 9 2018. doi:10.1109/MASS.2018.00044. 36
- [188] Duc-Liem Dinh, Hong-Nam Nguyen, Huy-Tan Thai, and Kim-Hung Le. Towards AI-Based Traffic Counting System with Edge Computing. *Journal of Advanced Transportation*, 2021:5551976, 2021. doi:10.1155/2021/5551976. 36
- [189] What is Arduino?, 2023. URL: <https://docs.arduino.cc/learn/starting-guide/whats-arduino>. 37
- [190] TensorFlow. URL: <https://www.tensorflow.org/>. 37
- [191] PyTorch. URL: <https://pytorch.org/>. 37
- [192] TensorFlow Lite for Microcontrollers. URL: <https://www.tensorflow.org/lite/microcontrollers>. 37
- [193] GitHub - tensorflow/tflite-micro: Infrastructure to enable deployment of ML models to low-power resource-constrained embedded targets (including microcontrollers and digital signal processors). URL: <https://github.com/tensorflow/tflite-micro>. 38
- [194] Pulse Sensor Playground. URL: <https://pulsesensor.com/pages/pulsesensor-playground-toolbox>. 38
- [195] Pravin R Kshirsgar, Hariprasath Manoharan, B Ebenezer Abishek, A V Bharadwaja, K Kavita, and P Vijayakumar. Dual Power Supply with Real Time GPS Tracking for Vehicular Application Using Machine Learning Algorithm. *Proceedings of the International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics, ICIITCEE 2023*, pages 227–231, 2023. doi:10.1109/IITCEE57236.2023.10091039. 38

-
- [196] Thomas Johnson, Eiman Kanjo, and Kieran Woodward. DigitalExposome: quantifying impact of urban environment on wellbeing using sensor fusion and deep learning. *Computational Urban Science 2023* 3:1, 3(1):1–17, 9 2023. URL: <https://link.springer.com/article/10.1007/s43762-023-00088-9>, doi:10.1007/S43762-023-00088-9. 40
- [197] Rajesh Gupta, Dakshita Reebadiya, and Sudeep Tanwar. 6G-enabled Edge Intelligence for Ultra -Reliable Low Latency Applications: Vision and Mission. *Computer Standards & Interfaces*, 77:103521, 9 2021. doi:10.1016/J.CSI.2021.103521. 40
- [198] Thomas Johnson and Eiman Kanjo. Urban Wellbeing: A Portable Sensing Approach to Unravel the Link Between Environment and Mental Wellbeing. *IEEE Sensors Letters*, 7(3), 9 2023. doi:10.1109/LSENS.2023.3243790. 40
- [199] Colby Banbury, Emil Njor, Andrea Mattia Garavagno, Matthew Stewart, Pete Warden, Manjunath Kudlur, Nat Jeffries, Xenofon Fafoutis, and Vijay Janapa Reddi. Wake Vision: A Tailored Dataset and Benchmark Suite for TinyML Computer Vision Applications. 5 2024. URL: <https://arxiv.org/abs/2405.00892v4>. 41
- [200] Kieran Woodward, Eiman Kanjo, Andreas Oikonomou, and Alan Chamberlain. LabelSens: enabling real-time sensor data labelling at the point of collection using an artificial intelligence-based approach. *Personal and Ubiquitous Computing*, 24(5):709–722, 9 2020. URL: <https://link.springer.com/article/10.1007/s00779-020-01427-x>, doi:10.1007/S00779-020-01427-X/FIGURES/10. 41
- [201] Liqiang Xiao, Honglun Zhang, and Wenqing Chen. Gated Multi-Task Network for Text Classification. *NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 2:726–731, 2018. URL: <https://aclanthology.org/N18-2114>, doi:10.18653/V1/N18-2114. 41

REFERENCES

- [202] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. AdaShare: Learning What To Share For Efficient Deep Multi-Task Learning. *Advances in Neural Information Processing Systems*, 33:8728–8740, 2020. [41](#)
- [203] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. *Advances in Neural Information Processing Systems*, 30, 2017. [42](#), [67](#)
- [204] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient Parametrization of Multi-Domain Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8119–8127, 2018. [42](#)
- [205] Hanbin Zhao, Hao Zeng, Xin Qin, Yongjian Fu, Hui Wang, Bourahla Omar, and Xi Li. What and Where: Learn to Plug Adapters via NAS for Multidomain Learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33(11):6532–6544, 11 2022. [doi:10.1109/TNNLS.2021.3082316](#). [42](#), [67](#)
- [206] Michael Crawshaw. Multi-Task Learning with Deep Neural Networks: A Survey. 9 2020. URL: <https://arxiv.org/abs/2009.09796v1>. [42](#)
- [207] Niccolo Federici, Danilo Pau, Nicola Adami, and Sergio Benini. Tiny Reservoir Computing for Extreme Learning of Motor Control. *Proceedings of the International Joint Conference on Neural Networks*, 2021-July, 7 2021. [doi:10.1109/IJCNN52387.2021.9534304](#). [43](#)
- [208] J Qiao, F Li, H Han, and W Li. Growing Echo-State Network With Multiple Subreservoirs. *IEEE Transactions on Neural Networks and Learning Systems*, 28(2):391–404, 2017. [doi:10.1109/TNNLS.2016.2514275](#). [43](#), [89](#)
- [209] Danil Koryakin, Johannes Lohmann, and Martin V Butz. Balanced echo state networks. *Neural Networks*, 36:35–45, 2012. URL: <https://www.sciencedirect.com/science/article/pii/S0893608012002213>, [doi: https://doi.org/10.1016/j.neunet.2012.08.008](#). [43](#)
- [210] Hui Han and Julien Siebert. TinyML: A Systematic Review and Synthesis of Existing Research. *4th International Conference on Artificial Intelligence*

REFERENCES

- in Information and Communication, ICAIIC 2022 - Proceedings*, pages 269–274, 2022. doi:10.1109/ICAIIIC54071.2022.9722636. 43, 64
- [211] Solveig Vieluf, Tanuj Hasija, Rasmus Jakobsmeier, Peter J Schreier, and Claus Reinsberger. Exercise-induced changes of multimodal interactions within the autonomic nervous network. *Frontiers in Physiology*, 10(MAR):240, 3 2019. 44
- [212] Neil Schneiderman, Gail Ironson, and Scott D Siegel. Stress and Health: Psychological, Behavioral, and Biological Determinants. *Annual Review of Clinical Psychology*, 1:607–628, 11 2004. 47
- [213] Shahid Ismail, Usman Akram, and Imran Siddiqi. Heart rate tracking in photoplethysmography signals affected by motion artifacts: a review. *Eurasip Journal on Advances in Signal Processing*, 2021(1):1–27, 12 2021. URL: <https://link.springer.com/articles/10.1186/s13634-020-00714-2>, doi:10.1186/s13634-020-00714-2, doi:10.1186/s13634-020-00714-2/FIGURES/7. 47
- [214] Muhammad Haseeb Arshad, Muhammad Bilal, and Abdullah Gani. Human Activity Recognition: Review, Taxonomy and Open Challenges. *Sensors* 2022, Vol. 22, Page 6463, 22(17):6463, 8 2022. URL: <https://www.mdpi.com/1424-8220/22/17/6463/html><https://www.mdpi.com/1424-8220/22/17/6463>, doi:10.3390/S22176463. 47
- [215] Ong Chin Ann and Lau Bee Theng. Human activity recognition: A review. *Proceedings - 4th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2014*, pages 389–393, 3 2014. doi:10.1109/ICCSCE.2014.7072750. 47
- [216] Liane Marina Messmer, Christoph Reich, and Djaffar Ould Abdeslam. Context-Aware Machine Learning: A Survey. *Lecture Notes in Networks and Systems*, 1154 LNNS:252–272, 2024. URL: https://link.springer.com/chapter/10.1007/978-3-031-73110-5_17, doi:10.1007/978-3-031-73110-5_{_}17/FIGURES/9. 48

-
- [217] Guang Li Huang, Arkady Zaslavsky, Seng W. Loke, Amin Abkenar, Alexey Medvedev, and Alireza Hassani. Context-Aware Machine Learning for Intelligent Transportation Systems: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 24(1):17–36, 1 2023. doi:[10.1109/TITS.2022.3216462](https://doi.org/10.1109/TITS.2022.3216462). 48
- [218] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity Recognition using Cell Phone Accelerometers. In *Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data (at KDD-10)*, Washington DC, 2010. 51, 61, 70, 90
- [219] Katarina Dedovic, Robert Renwick, Najmeh Khalili Mahani, and Veronika Engert. The Montreal Imaging Stress Task : using functional imaging to investigate the ... *Psychiatry & Neuroscience*, 30(5):319–325, 2005. 51
- [220] Giorgos Giannakakis, Dimitris Grigoriadis, Katerina Giannakaki, Olympia Simantiraki, Alexandros Roniotis, and Manolis Tsiknakis. Review on Psychological Stress Detection Using Biosignals. *IEEE Transactions on Affective Computing*, 13(1):440–460, 2022. doi:[10.1109/TAFFC.2019.2927337](https://doi.org/10.1109/TAFFC.2019.2927337). 54
- [221] Juan Camilo Bautista Cifuentes, Sergio Andrés Marín Patiño, Esteban Morales Mahecha, and Javier Alberto Chaparro. Hand Gesture Classifier Using Edge Artificial Intelligence. *2024 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pages 1–6, 11 2024. URL: <https://ieeexplore.ieee.org/document/10814802/>, doi:[10.1109/LA-CCI62337.2024.10814802](https://doi.org/10.1109/LA-CCI62337.2024.10814802). 54
- [222] Kun Xia, Jianguang Huang, and Hanyu Wang. LSTM-CNN Architecture for Human Activity Recognition. *IEEE Access*, 8:56855–56866, 2020. doi:[10.1109/ACCESS.2020.2982225](https://doi.org/10.1109/ACCESS.2020.2982225). 55
- [223] Ravi Raj and Andrzej Kos. An improved human activity recognition technique based on convolutional neural network. *Scientific Reports 2023 13:1*, 13(1):1–19, 12 2023. URL: <https://www.nature.com/articles/s41598-023-49739-1>, doi:[10.1038/s41598-023-49739-1](https://doi.org/10.1038/s41598-023-49739-1). 59, 76

-
- [224] Kieran Woodward and Eiman Kanjo. IFidgetCube: Tangible Fidgeting Interfaces (TFIs) to Monitor and Improve Mental Wellbeing. *IEEE Sensors Journal*, 21(13):14300–14307, 7 2021. doi:[10.1109/JSEN.2020.3031163](https://doi.org/10.1109/JSEN.2020.3031163). 60
 - [225] Taiwo Samuel Ajani, Agbotiname Lucky Imoize, and Aderemi A. Atayero. An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications. *Sensors 2021*, Vol. 21, Page 4412, 21(13):4412, 6 2021. URL: <https://www.mdpi.com/1424-8220/21/13/4412/html><https://www.mdpi.com/1424-8220/21/13/4412>, doi:[10.3390/S21134412](https://doi.org/10.3390/S21134412). 64
 - [226] Stefano Abbate, Marco Avvenuti, Francesco Bonatesta, Guglielmo Cola, Paolo Corsini, and Alessio Vecchio. A smartphone-based fall detection system. *Pervasive and Mobile Computing*, 8(6):883–899, 12 2012. doi:[10.1016/J.PMCJ.2012.08.003](https://doi.org/10.1016/J.PMCJ.2012.08.003). 64
 - [227] Russell Li and Zhandong Liu. Stress detection using deep neural networks. *BMC Medical Informatics and Decision Making*, 20(11):1–10, 12 2020. URL: <https://link.springer.com/articles/10.1186/s12911-020-01299-4><https://link.springer.com/article/10.1186/s12911-020-01299-4>, doi:[10.1186/S12911-020-01299-4](https://doi.org/10.1186/S12911-020-01299-4)/TABLES/5. 64
 - [228] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. 6 2017. URL: <https://arxiv.org/abs/1706.05098v1>. 64, 66
 - [229] Yu Zhang and Qiang Yang. Special Topic: Machine Learning An overview of multi-task learning. *National Science Review*, 5:30–43, 2018. URL: <https://academic.oup.com/nsr/article/5/1/30/4101432>, doi:[10.1093/nsr/nwx105](https://doi.org/10.1093/nsr/nwx105). 64
 - [230] Yu Zhang and Qiang Yang. A Survey on Multi-Task Learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 12 2022. doi:[10.1109/TKDE.2021.3070203](https://doi.org/10.1109/TKDE.2021.3070203). 64, 82
 - [231] Peter K. Ghavami and Kailash C. Kapur. The application of multi-model ensemble approach as a prognostic method to predict patient health status.

- PHM 2013 - 2013 IEEE International Conference on Prognostics and Health Management, Conference Proceedings*, 2013. doi:10.1109/ICPHM.2013.6621422. 65
- [232] Heyuan Shi, Xibin Zhao, Hai Wan, Huihui Wang, Jian Dong, Kun Tang, and Anfeng Liu. Multi-model induced network for participatory-sensing-based classification tasks in intelligent and connected transportation systems. *Computer Networks*, 141:157–165, 8 2018. doi:10.1016/J.COMNET.2018.05.030. 65
- [233] Terumi Umematsu, Akane Sano, Sara Taylor, and Rosalind W. Picard. Improving students’ daily life stress forecasting using lstm neural networks. *2019 IEEE EMBS International Conference on Biomedical and Health Informatics, BHI 2019 - Proceedings*, 5 2019. doi:10.1109/BHI.2019.8834624. 70
- [234] Kieran Woodward, Eiman Kanjo, and Athanasios Tsanas. Combining Deep Learning with Signal-image Encoding for Multi-Modal Mental Wellbeing Classification. *ACM Transactions on Computing for Healthcare*, 5(1), 1 2024. URL: <https://dl.acm.org/doi/10.1145/3631618>, doi:10.1145/3631618. 70
- [235] Saeedeh Zebhi. Human Activity Recognition Using Wearable Sensors Based on Image Classification. *IEEE Sensors Journal*, 22(12):12117–12126, 6 2022. doi:10.1109/JSEN.2022.3174280. 70, 76
- [236] Nidhi Dua, Shiva Nand Singh, and Vijay Bhaskar Semwal. Multi-input CNN-GRU based human activity recognition using wearable sensors. *Computing*, 103(7):1461–1478, 7 2021. URL: <https://link.springer.com/article/10.1007/s00607-021-00928-8>, doi:10.1007/S00607-021-00928-8/METRICS. 70
- [237] Jessica Permatasari, Tee Connie, and Thian Song Ong. Inertial sensor fusion for gait recognition with symmetric positive definite Gaussian kernels analysis. *Multimedia Tools and Applications*, 79(43-44):32665–32692, 11 2020. URL: <https://link.springer.com/article/10.1007/s11042-020-09438-9>, doi:10.1007/S11042-020-09438-9/TABLES/8. 71

REFERENCES

- [238] Yasin Kaya and Elif Kevser Topuz. Human activity recognition from multiple sensors data using deep CNNs. *Multimedia Tools and Applications*, 83(4):10815–10838, 1 2024. URL: <https://link.springer.com/article/10.1007/s11042-023-15830-y>, doi:10.1007/S11042-023-15830-Y/TABLES/10. 71
- [239] TensorFlow Lite for Microcontrollers. URL: <https://www.tensorflow.org/lite/microcontrollers>. 71
- [240] Giorgos Giannakakis, Kostas Marias, and Manolis Tsiknakis. A stress recognition system using HRV parameters and machine learning techniques. *2019 8th International Conference on Affective Computing and Intelligent Interaction Workshops and Demos, ACIIW 2019*, pages 269–272, 9 2019. doi:10.1109/ACIIW.2019.8925142. 76
- [241] Victoriano Montesinos, Fabio Dell’Agnola, Adriana Arza, Amir Aminifar, and David Atienza. Multi-Modal Acute Stress Recognition Using Off-the-Shelf Wearable Devices. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, pages 2196–2201, 7 2019. doi:10.1109/EMBC.2019.8857130. 76
- [242] Akane Sano and Rosalind W. Picard. Stress recognition using wearable sensors and mobile phones. *Proceedings - 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction, ACII 2013*, pages 671–676, 2013. doi:10.1109/ACII.2013.117. 76
- [243] Rajit Nair, Mahmoud Ragab, Osama A. Mujallid, Khadijah Ahmad Mohammad, Romany F. Mansour, and G. K. Viju. Impact of Wireless Sensor Data Mining with Hybrid Deep Learning for Human Activity Recognition. *Wireless Communications and Mobile Computing*, 2022(1):9457536, 1 2022. URL: <https://onlinelibrary.wiley.com/doi/full/10.1155/2022/9457536><https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/9457536><https://onlinelibrary.wiley.com/doi/10.1155/2022/9457536>, doi:10.1155/2022/9457536. 76

-
- [244] K. H. Walse, R. V. Dharaskar, and V. M. Thakare. Performance evaluation of classifiers on WISDM dataset for human activity recognition. *ACM International Conference Proceeding Series*, 04-05-March-2016, 3 2016. URL: <https://dl.acm.org/doi/10.1145/2905055.2905232>, doi: [10.1145/2905055.2905232](https://doi.org/10.1145/2905055.2905232). 76
- [245] Robert Coop, Aaron Mishtal, and Itamar Arel. Ensemble learning in fixed expansion layer networks for mitigating catastrophic forgetting. *IEEE Transactions on Neural Networks and Learning Systems*, 24(10):1623–1634, 2013. doi:[10.1109/TNNLS.2013.2264952](https://doi.org/10.1109/TNNLS.2013.2264952). 78
- [246] B. Pfülb and A. Gepperth. A comprehensive, application-oriented study of catastrophic forgetting in DNNs. *7th International Conference on Learning Representations, ICLR 2019*, 5 2019. URL: <https://arxiv.org/abs/1905.08101v1>. 78
- [247] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei Ling Shyu, Shu Ching Chen, and S. S. Iyengar. A Survey on Deep Learning: Algorithms, Techniques, and Applications. *ACM Computing Surveys (CSUR)*, 51(5), 9 2018. URL: <https://dl.acm.org/doi/10.1145/3234150>, doi:[10.1145/3234150](https://doi.org/10.1145/3234150). 79
- [248] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep Learning for Anomaly Detection: A Review. *ACM Computing Surveys*, 54(2), 3 2021. URL: <https://dl.acm.org/doi/10.1145/3439950>, doi:[10.1145/3439950/SUPPL%5B%5D_FILE/PANG.ZIP](https://doi.org/10.1145/3439950/SUPPL%5B%5D_FILE/PANG.ZIP). 82
- [249] Bo Peng, Ying Bi, Bing Xue, Mengjie Zhang, and Shuting Wan. A Survey on Fault Diagnosis of Rolling Bearings. *Algorithms 2022, Vol. 15, Page 347*, 15(10):347, 9 2022. URL: <https://www.mdpi.com/1999-4893/15/10/347/html><https://www.mdpi.com/1999-4893/15/10/347>, doi:[10.3390/A15100347](https://doi.org/10.3390/A15100347). 82
- [250] Steven A. Lubitz, Anthony Z. Faranesh, Caitlin Selvaggi, Steven J. Atlas, David D. McManus, Daniel E. Singer, Sherry Pagoto, Michael V. McConnell, Alexandros Pantelopoulos, and Andrea S. Foulkes. Detection of Atrial Fibrillation in a Large Population

- Using Wearable Devices: The Fitbit Heart Study. *Circulation*, 146(19):1415–1424, 11 2022. URL: <https://www.ahajournals.org/doi/10.1161/CIRCULATIONAHA.122.060291>, doi:10.1161/CIRCULATIONAHA.122.060291/SUPPL{_}FILE/FITBIT. 82
- [251] Sigurd Løkse, Filippo Maria Bianchi, and Robert Jenssen. Training Echo State Networks with Regularization Through Dimensionality Reduction. *Cognitive Computation*, 9(3):364–378, 6 2017. URL: <https://link.springer.com/article/10.1007/s12559-017-9450-z>, doi:10.1007/S12559-017-9450-Z/FIGURES/6. 82
- [252] R Jiang, Z Wu, Y Zhang, and X Ye. ComESN: Compressive Echo State Network for Multi-task Time-series Prediction. In *2023 IEEE International Symposium on Product Compliance Engineering - Asia (ISPCE-ASIA)*, pages 1–6, 2023. doi:10.1109/ISPCE-ASIA60405.2023.10366011. 82
- [253] E. J. López-Ortiz, M. Perea-Trigo, L. M. Soria-Morillo, F. Sancho-Caparrini, and J. J. Vegas-Olmos. Exploring deep echo state networks for image classification: a multi-reservoir approach. *Neural Computing and Applications*, 36(20):11901–11918, 7 2024. URL: <https://link.springer.com/article/10.1007/s00521-024-09656-4>, doi:10.1007/S00521-024-09656-4/FIGURES/17. 83, 97
- [254] Manan Gandhi, Keuntaek Lee, Yunpeng Pan, and Evangelos A Theodorou. Propagating Uncertainty through the tanh Function with Application to Reservoir Computing. 6 2018. URL: <https://arxiv.org/abs/1806.09431v1>. 84
- [255] Rebh Soltani, Emna Benmohamed, and Hela Ltifi. Echo State Network Optimization: A Systematic Literature Review. *Neural Processing Letters*, 55(8):10251–10285, 12 2023. doi:10.1007/s11063-023-11326-w. 89
- [256] Abubakar Bala, Idris Ismail, Rosdiazli Ibrahim, Sadiq M. Sait, and Diego Oliva. An Improved Grasshopper Optimization Algorithm Based Echo State Network for Predicting Faults in Airplane Engines. *IEEE Access*, 8:159773–159789, 2020. doi:10.1109/ACCESS.2020.3020356. 89

REFERENCES

- [257] Neha Sharma, Vibhor Jain, and Anju Mishra. An Analysis Of Convolutional Neural Networks For Image Classification. *Procedia Computer Science*, 132:377–384, 1 2018. doi:[10.1016/J.PROCS.2018.05.198](https://doi.org/10.1016/J.PROCS.2018.05.198). 90
- [258] Amir F. Atiya and Alexander G. Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709, 5 2000. doi:[10.1109/72.846741](https://doi.org/10.1109/72.846741). 90
- [259] Edward N. Lorenz. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 3 1963. doi:[10.1175/1520-0469\(1963\)020<0130:DNF>2.0.CO;2](https://doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2). 90
- [260] O. E. Rössler. An equation for continuous chaos. *Physics Letters A*, 57(5):397–398, 7 1976. doi:[10.1016/0375-9601\(76\)90101-8](https://doi.org/10.1016/0375-9601(76)90101-8). 90
- [261] Anguita Davide Ghio Alessandro Oneto Luca Reyes-Ortiz Jorge and Xavier Parra. Human Activity Recognition Using Smartphones. UCI Machine Learning Repository, 2013. 90
- [262] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi:[10.1109/MSP.2012.2211477](https://doi.org/10.1109/MSP.2012.2211477). 90
- [263] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. 8 2017. URL: <https://arxiv.org/abs/1708.07747v2>. 90
- [264] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 4 2009. URL: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>. 90
- [265] Peter Steiner, Azarakhsh Jalalvand, and Peter Birkholz. Cluster-Based Input Weight Initialization for Echo State Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 34(10):7648–7659, 10 2023. doi:[10.1109/TNNLS.2022.3145565](https://doi.org/10.1109/TNNLS.2022.3145565). 91

-
- [266] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47, 1 2002. URL: <https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.74.47>, doi: [10.1103/RevModPhys.74.47](https://doi.org/10.1103/RevModPhys.74.47). 91
 - [267] Vasileios Karyotis and M.H.R. Khouzani. Malware-propagative Markov random fields. *Malware Diffusion Models for Wireless Complex Networks*, pages 107–138, 2016. doi: [10.1016/B978-0-12-802714-1.00015-3](https://doi.org/10.1016/B978-0-12-802714-1.00015-3). 92
 - [268] M. E.J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 99(suppl_1):2566–2572, 2 2002. URL: <https://www.pnas.org/doi/abs/10.1073/pnas.012582999>, doi: [10.1073/PNAS.012582999](https://doi.org/10.1073/PNAS.012582999). 92
 - [269] P Erdos, A Rényi Acta Math. Acad. Sci. Hungar, and undefined 1963. Asymmetric graphs. *static.renyi.huP Erdos, A RényiActa Math. Acad. Sci. Hungar, 1963 • static.renyi.hu*. URL: https://static.renyi.hu/~p_erdos/1963-04.pdf. 92
 - [270] Diana C.Roca Arroyo, Alexander Choquenaira Florez, Daniela Milon Flores, Roseli Romero, and Liang Zhao. Echo State Network Performance Analysis using Non-random Topologies. *2020 IEEE Colombian Conference on Applications of Computational Intelligence, ColCACI 2020 - Proceedings*, 8 2020. doi: [10.1109/COLCACI50549.2020.9248714](https://doi.org/10.1109/COLCACI50549.2020.9248714). 92, 96
 - [271] Ali Rodan and Peter Tiño. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–144, 1 2011. doi: [10.1109/TNN.2010.2089641](https://doi.org/10.1109/TNN.2010.2089641). 96, 99
 - [272] Xia Zhao, Limin Wang, Yufei Zhang, Xuming Han, Muhammet Deveci, and Milan Parmar. A review of convolutional neural networks in computer vision. *Artificial Intelligence Review*, 57(4):1–43, 4 2024. URL: <https://link.springer.com/article/10.1007/s10462-024-10721-6>, doi: [10.1007/S10462-024-10721-6](https://doi.org/10.1007/S10462-024-10721-6)/FIGURES/33. 97

REFERENCES

- [273] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018. 98, 100
- [274] Owais Mujtaba Khanday, Samad Dadvandipour, and Mohd Aaqib Lone. Effect of filter sizes on image classification in CNN: a case study on CFIR10 and Fashion-MNIST datasets. *IAES International Journal of Artificial Intelligence (IJ-AI)*, 10(4):872, 12 2021. doi:10.11591/ijai.v10.i4.pp872-878. 98
- [275] Shivam S Kadam, Amol C Adamuthe, and Ashwini B Patil. CNN model for image classification on MNIST and fashion-MNIST dataset. *Journal of scientific research*, 64(2):374–384, 2020. 98
- [276] Ji Lin, Wei-Ming Chen, Yujun Lin, john cohn, Chuang Gan, and Song Han. MCUNet: Tiny Deep Learning on IoT Devices. *Advances in Neural Information Processing Systems*, 33:11711–11722, 2020. URL: <https://tinymml.mit.edu>. 101
- [277] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in Vision: A Survey. *ACM Computing Surveys (CSUR)*, 54(10), 9 2022. URL: <https://dl.acm.org/doi/10.1145/3505244>, doi:10.1145/3505244. 108

Appendix A. Additional Echo State Network Plots

Appendix A. Additional Echo State Network Plots

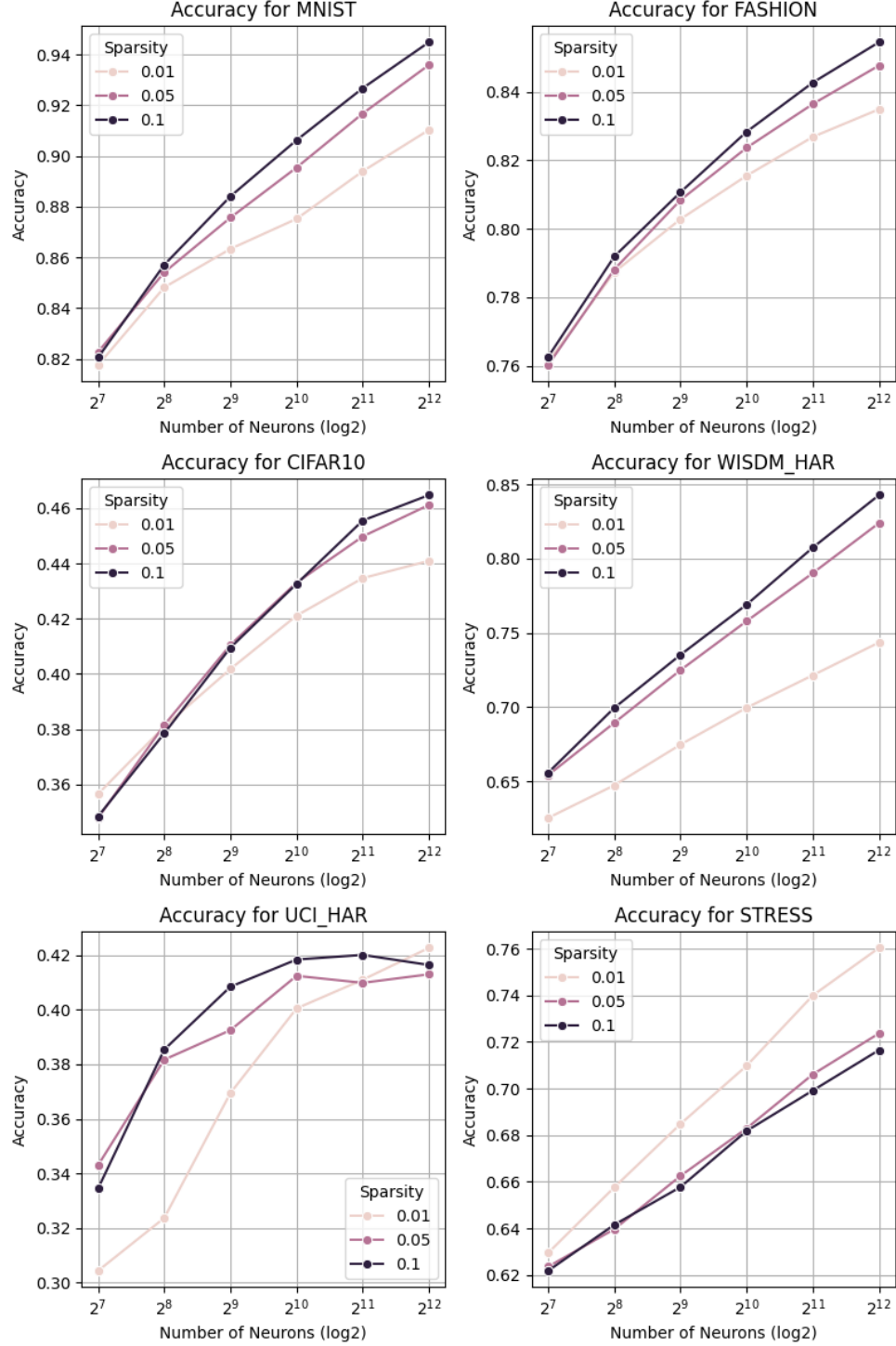


Figure 1: Plot showing the accuracy against the log-scaled number of neurons given varying sparsity levels for all classification datasets.

Appendix A. Additional Echo State Network Plots

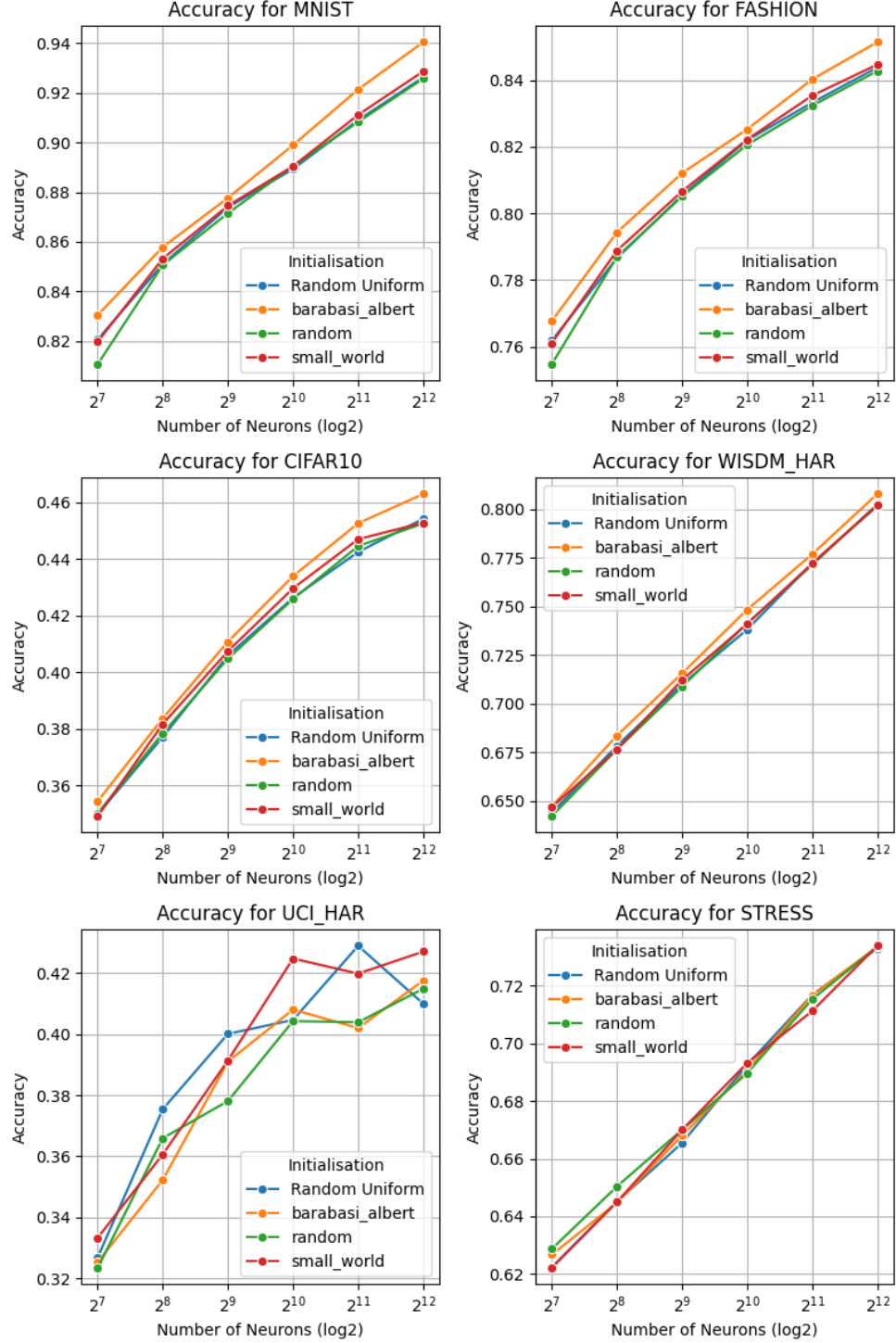


Figure 2: Plot showing the accuracy against the log-scaled number of neurons given varying initialisation methods for all classification datasets.

Appendix A. Additional Echo State Network Plots

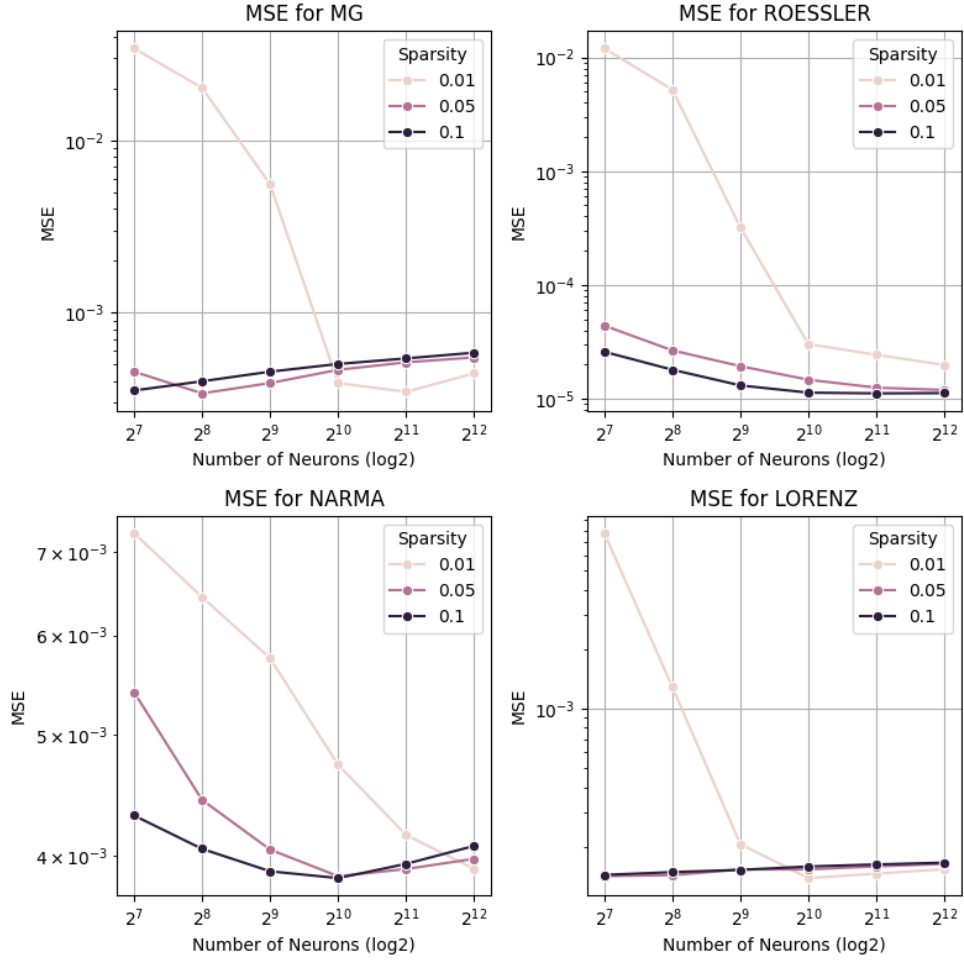


Figure 3: Plot showing the log-scaled Mean Squared Error (MSE) against the log-scaled number of neurons given varying sparsity levels for all regression-based datasets.

Appendix A. Additional Echo State Network Plots

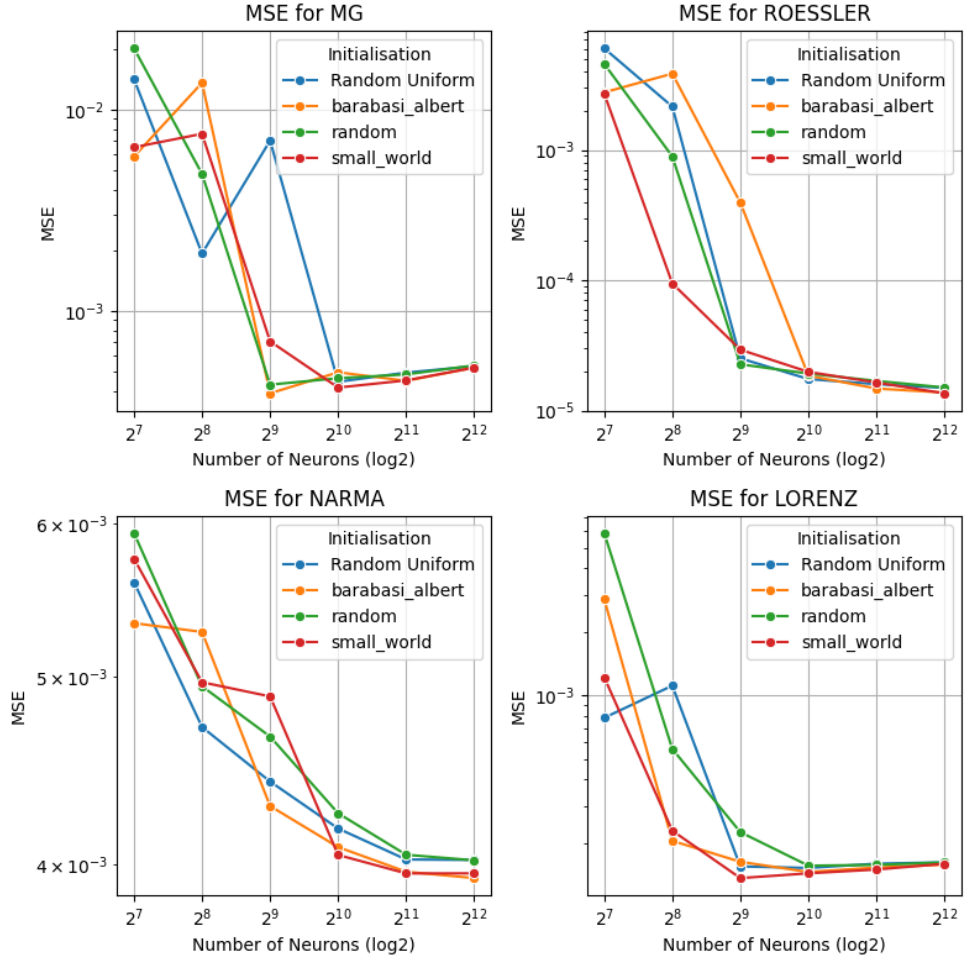


Figure 4: Plot showing the log-scaled Mean Squared Error (MSE) against the log-scaled number of neurons given varying initialisation methods for all regression-based datasets.