POLITECNICO DI TORINO

Master Degree course in Computer Engineering

Master Degree Thesis

LIF-based Legendre Memory Unit: neuromorphic redesign of a recurrent architecture and its application to human activity recognition

Advisors
Gianvito Urgese
Vittorio Fra

Candidate
Benedetto Leto

October 2024

Abstract

Nowadays traditional artificial neural networks (ANNs) present a significant challenge in edge devices and embedded systems due to their high power consumption. This inefficiency in standard machine learning (ML) and deep learning (DL) solutions requires the exploration of more efficient alternatives. Spiking neural networks (SNNs), inspired by the interconnections of human brain neurons, constitute a valid alternative to address this challenge. However, the advancement of these brain-inspired models is constrained by the limited accessibility to dedicated neuromorphic hardware capable of integrating brain-inspired computational principles. Despite this low hardware availability, SNNs represent a step forward in achieving more efficient, brain-inspired computation for edge devices and embedded systems.

The goal of neuromorphic computing is to emulate the intricating complexities that have evolved over millions of years to fine-tune neural processing. Driving inspiration from the behavior of biological neurons, neuromorphic models trace a unique path to computational efficiency and adaptability, making a significant break from the rigid, power-hungry architectures of conventional computing. Instead, they adopt a more flexible, energy-efficient, and inherently parallel paradigm. Unlike the traditional neural network, these brain-inspired models activate neurons only in response to certain events. This event-based approach simulates brain information processing. As a result, this leads to a more sparse neuron activation, decreasing the amount of energy consumption.

The core part of this work is transforming a recursive cell architecture, the Legendre Memory Unit (LMU), into its neuromorphic version, named the LIF-based Legendre Memory Unit (L^2MU). This architecture redesign reconfigures the internal states of the original architecture into populations of neurons that are interconnected through synapses, allowing components to communicate via synaptic currents and facilitating information flow through neuronal spikes in response to changes in current and voltage.

As a benchmarking stage for the L^2MU architecture, a comprehensive comparative analysis is presented focusing on the task of human activity recognition (HAR); including various alternatives built from the L^2MU cell and other network architectures both in the neuromorphic field and in the non-spiking domain. This also includes an analysis of model compression techniques to keep spiking neural

networks lightweight and efficient for low-power environments.

The Thesis hence offers a comparative study with classical artificial neural networks, highlighting the advantages and trade-offs in terms of computational efficiency, energy consumption, and processing speed. Real-world scenarios are also considered, demonstrating the potential of L^2MU for various edge computing applications such as real-time data processing.

Contents

List of Figures 9

List of Tables 14

1 Introduction 17

2 Background 21
2.1 Spiking Neural Network . . . 21
2.1.1 Biologic Foundation of SNNs . . . 21
2.1.2 Spike Generation . . . 22
2.1.3 Spikes Transmission . . . 23
2.1.4 Spikes Encoding . . . 24
2.1.5 Neuromorphic Hardware and SNNs . . . 24
2.1.6 Neurons Dynamic as Synaptic Operations . . . 25
2.2 Human activity recognition . . . 26
2.3 Network Architectures . . . 28
2.3.1 RNN . . . 28
2.3.2 LSTM . . . 29
2.3.3 Legendre Memory Unit . . . 30
2.3.4 Focus on LMU . . . 32
2.4 snnTorch . . . 33
2.5 NeuroBench . . . 35
2.5.1 Metrics . . . 35
2.6 Edge Devices . . . 38
2.6.1 Edge Devices Characteristics . . . 38
2.6.2 Potential benefits of SNNs on Edge Devices . . . 38
2.7 Model compression . . . 39
2.7.1 Pruning . . . 39
2.7.2 Quantization . . . 42

3 Materials and methods 43
3.1 Encoding module . . . 44

3.1.1 Single Encoder . . . 44
3.1.2 Stacked Encoder . . . 44
3.2 LIF-based LMU (L^2MU) . . . 46
3.3 Activities selection and segmentation . . . 49
3.4 Hyperparameter optimization . . . 51
3.5 Selection of specific hyperparameters . . . 54
3.6 Model Statistics . . . 54
3.7 Model Compression . . . 55
3.7.1 Granular magnitude pruning . . . 55
3.7.2 Quantization . . . 57
3.8 Deployment on hardware . . . 58

4 Results and discussion 61
4.1 Baseline . . . 61
4.1.1 LMU insights . . . 62
4.2 L^2MU . . . 63
4.2.1 Leaky . . . 63
4.2.2 Synaptic . . . 64
4.2.3 Leaky vs. Synaptic Model . . . 66
4.2.4 L^2MU vs LMU . . . 66
4.3 Encoded L^2MU . . . 67
4.3.1 Leaky . . . 67
4.3.2 Synaptic . . . 69
4.3.3 Leaky vs Synaptic . . . 69
4.3.4 Encoded L^2MU vs L^2MU . . . 70
4.4 Multi-Encoded L^2MU . . . 71
4.4.1 Leaky . . . 71
4.4.2 Synaptic . . . 71
4.4.3 Leaky vs. Synaptic . . . 73
4.4.4 Multi-Encode L^2MU vs. Encoded L^2MU . . . 74
4.5 Deployment of Multi-Encoded L^2MU on Edge Devices . . . 74

5 Conclusion 77

A Conversion of RNN to LIF-based RNN 79
A.1 Conversion Process of an RNN . . . 79
A.2 Results and Discussions . . . 79
A.2.1 L-RNN . . . 79
A.2.2 Encoded L-RNN . . . 81
A.2.3 Multi-Encoded L-RNN . . . 82

B Learning Curves 85
B.1 LSTM . . . 86
B.2 LMU . . . 86
B.3 RNN . . . 87
B.4 L^2MU . . . 87
B.5 Encoded L^2MU . . . 88
B.6 Multi-Encoded L^2MU . . . 89
B.7 L-RNN . . . 90
B.8 Encoded L-RNN . . . 91
B.9 Multi-Encoded L-RNN . . . 92

C Confusion Matrices 95
C.1 LSTM . . . 95
C.2 LMU . . . 96
C.3 RNN . . . 96
C.4 L^2MU . . . 97
C.5 Encoded L^2MU . . . 98
C.6 Multi-Encoded L^2MU . . . 99
C.7 L-RNN . . . 100
C.8 Encoded L-RNN . . . 101
C.9 Multi-Encoded L-RNN . . . 102

D Hyperparameters 103
D.1 Hyperparameters' Description . . . 103
D.1.1 Common Hyperparameters . . . 103
D.1.2 Output Hyperparameters for SNNs . . . 103
D.1.3 Encoder Hyperparameters . . . 104
D.1.4 RNN . . . 105
D.1.5 LSTM . . . 105
D.1.6 LMU . . . 105
D.1.7 L-RNN . . . 105
D.1.8 L^2MU . . . 106
D.2 Hyperparameters' Value . . . 106
D.2.1 LSTM . . . 106
D.2.2 LMU . . . 106
D.2.3 RNN . . . 107
D.2.4 L^2MU . . . 107
D.2.5 Encoded L^2MU . . . 108
D.2.6 Multi-Encoded L^2MU . . . 109
D.2.7 L-RNN . . . 110
D.2.8 Encoded L-RNN . . . 110
D.2.9 Multi-Encoded L-RNN . . . 111

Bibliography 115

List of Figures

1.1 This illustration presents a comprehensive representation of spiking neurons and artificial neurons, along with their networks and corresponding hardware. On the top left is the spiking neuron model inspired by its biological structure, and on the top right is an abstract representation of the artificial neuron. Spiking Neural Networks on the middle left are shown as sparse configurations of neurons communicating through discrete spikes, whereas Artificial Neural Networks on the middle right, are dense, fully connected networks where neurons are continuously interacting. On the bottom left, neuromorphic chips, such as TrueNorth, Loihi, Tianjic, and Spek, constitute the design to emulate spiking neural networks in hardware. The right side features the purpose for which chips are generally utilized regarding artificial neural networks. 19

2.1 Distribution of signal values captured by both the accelerometer and gyroscope sensors along their respective X, Y, and Z axes. 28

2.2 RNN Unit architecture. 29

2.3 LSTM Unit architecture. 30

2.4 LMU Unit architecture. 31

2.5 Effect of pruning techniques on the synapse weights of the original model. 39

2.6 The image illustrates a structured pruned neural network. In structured pruning, entire neurons and their associated connections are removed, rather than just setting individual weights to zero. This is indicated by the shadowed connections and neurons in the image. The red circles symbolize active neurons, while the absence of shadowed circles and lines where neurons and connections have been removed illustrates the network after pruning. The solid blue lines represent the remaining active connections. 40

2.7 The image illustrates a pruned neural network graph where the dashed lines represent connections that have been pruned. Pruning in this context means that the weights associated with these connections have been set to zero in the weight matrix, effectively eliminating their influence on the network's computations. The solid lines are likely to represent the active connections with non-zero weights. This graphical representation emphasizes the distinction between active and pruned connections within a neural network's architecture after pruning has occurred. 41

3.1 This diagram illustrates the real-time human activity tracking process of a smartwatch. Utilizing onboard gyroscope and accelerometer sensors, the smartwatch collects data over 2 seconds, equivalent to 40 samples. These data are then transmitted to an artificial neural network. The neural network processes the sensor data and classifies the wearer's activity such as clapping, dribbling, playing catch, brushing teeth, writing, typing, or folding clothes. 43

3.2 Illustration of a spiking single layer encoding module consisting of channel-specific neurons. On the left, the "6-axis input"represents raw time-series data from the accelerometer and gyroscope sensors. In the middle, "channel-specific neurons"are shown receiving synaptic currents corresponding to each sensory axis. On the right, the processed signals converge in a spike-based recurrent unit. 45

3.3 Illustration of a spiking multi-layer encoding module consisting of channel-specific neurons, fusion, and harmonization neurons. On the left, the same structure of the single encoder is used, the "6-axis input"connected to the "channel-specific neurons". These neurons communicate with the "fusion neurons"which compress the information received by the previous neuron layer. Finally, the "harmonization neurons", receiving the spike signal from the previous layer, fine-tune the signal trying to minimize any discrepancies raised by the previous layers. At this point, this layer emits spike trains which converge to a spike-based recurrent unit. 46

3.4 Illustration of the L^2MU where each component is represented with a population of neurons. The input, denoted as $x_{t,spk}$, feeds into the model at time step t, which influences the **u** and **h** population neurons. The recurrent information flow between the **m** population, as well as the **h** population. The output neurons, gather information from the hidden neuron population to generate the final output. . . 47

3.5 Random sample of 2 seconds recorded by the smartwatch on the 6 IMU sensors for the 7 classes in the "hand-oriented activities related to general tasks"subset of the WISDM dataset. 50

3.6 This image represents a machine learning workflow using the Lightning AI framework and Neural Network Intelligence (NNI) for hyperparameter optimization in human activity recognition tasks. The left side shows a neural network model, which is fed time-series data from a human activity dataset. In the center, the "Trainer"configuration includes checkpoints, callbacks, and logging mechanisms for model training. The "Neural Network Intelligence"block indicates the use of NNI to automate the search for optimal hyperparameters, configured by a "Search Space"and an "Experiment Config"that specifies the tuner, assessor, etc... The result of this process is a comprehensive database containing trial results, a checkpoint of the most effective model, the finest-tuned parameters, and detailed logs. . . . 51
3.7 STM32MP157F-DK2 board. 59
3.8 Raspberry Pi 3B+ board. 59
3.9 Raspberry Pi 4B board. 60

4.1 Histogram plots representing the inference time probability distribution. Analyses were carried out on the three boards running the models with either `Leaky` (left) or `Synaptic` (right) neurons. 75

B.1 Learning curves for the full-precision Long Short-Term Memory (LSTM) model retrained with 10 different seeds 86
B.2 Learning curves for the full-precision Legendre Memory Unit (LMU) model retrained with 10 different seeds 86
B.3 Learning curves for the full-precision Recurrent Neural Network (RNN) model retrained with 10 different seeds 87
B.4 Learning curves for the full-precision LIF-based LMU (L^2MU) model with Leaky neurons retrained with 10 different seeds 87
B.5 Learning curves for the full-precision LIF-based LMU (L^2MU) model with Synaptic neurons retrained with 10 different seeds 88
B.6 Learning curves for the full-precision Encoded LIF-based LMU (Encoded L^2MU) model with Leaky neurons retrained with 10 different seeds 88
B.7 Learning curves for the full-precision Encoded LIF-based LMU (Encoded L^2MU) model with Synaptic neurons retrained with 10 different seeds 89
B.8 Learning curves for the full-precision Multi-Encoded LIF-based LMU (Multi-Encoded L^2MU) model with Leaky neurons retrained with 10 different seeds 89
B.9 Learning curves for the full-precision Multi-Encoded LIF-based LMU (Multi-Encoded L^2MU) model with Synaptic neurons retrained with 10 different seeds 90

B.10 Learning curves for the full-precision LIF-based RNN (L-RNN) model with Leaky neurons retrained with 10 different seeds 90
B.11 Learning curves for the full-precision LIF-based RNN (L-RNN) model with Synaptic neurons retrained with 10 different seeds 91
B.12 Learning curves for the full-precision Encoded LIF-based RNN (Encoded L-RNN) model with Leaky neurons retrained with 10 different seeds 91
B.13 Learning curves for the full-precision Encoded LIF-based RNN (Encoded L-RNN) model with Synaptic neurons retrained with 10 different seeds 92
B.14 Learning curves for the full-precision Multi-Encoded LIF-based RNN (Multi-Encoded L-RNN) model with Leaky neurons retrained with 10 different seeds 92
B.15 Learning curves for the full-precision Multi-Encoded LIF-based RNN (Multi-Encoded L-RNN) model with Synaptic neurons retrained with 10 different seeds 93

C.1 Confusion matrix for the full-precision Long Short-Term Memory (LSTM) model. 95
C.2 Confusion matrix for the full-precision Legendre Memory Unit (LMU) model. 96
C.3 Confusion matrix for the full-precision Recurrent Neural Network (RNN) model. 96
C.4 Confusion matrix for the full-precision LIF-based LMU (L^2MU) model with Leaky neurons 97
C.5 Confusion matrix for the full-precision LIF-based LMU (L^2MU) model with Synaptic neurons 97
C.6 Confusion matrix for the full-precision Encoded LIF-based LMU (Encoded L^2MU) model with Leaky neurons 98
C.7 Confusion matrix for the full-precision Encoded LIF-based LMU (Encoded L^2MU) model with Synaptic neurons 98
C.8 Confusion matrix for the full-precision Multi-Encoded LIF-based LMU (Multi-Encoded L^2MU) model with Leaky neurons 99
C.9 Confusion matrix for the full-precision Multi-Encoded LIF-based LMU (Multi-Encoded L^2MU) model with Synaptic neurons 99
C.10 Confusion matrix for the full-precision LIF-based RNN (L-RNN) model with Leaky neurons 100
C.11 Confusion matrix for the full-precision LIF-based RNN (L-RNN) model with Synaptic neurons 100
C.12 Confusion matrix for the full-precision Encoded LIF-based RNN (Encoded L-RNN) model with Leaky neurons 101

C.13 Confusion matrix for the full-precision Encoded LIF-based RNN (Encoded L-RNN) model with Synaptic neurons 101
C.14 Confusion matrix for the full-precision Multi-Encoded LIF-based RNN (Multi-Encoded L-RNN) model with Leaky neurons 102
C.15 Confusion matrix for the full-precision Multi-Encoded LIF-based RNN (Multi-Encoded L-RNN) model with Synaptic neurons 102

List of Tables

2.1 Summary information of the WISDM dataset. 26
2.2 18 activities represented in the WISDM dataset 27

3.1 Dataset Sizes for Training, Validation, Testing, and Calibration . . 50

4.1 Baseline Recurrent Artificial Neural Networks Models 61
4.2 LMU Model Metrics 62
4.3 L^2MU (Leaky) Performance 64
4.4 L^2MU (Synaptic) Performance 65
4.5 Encoded L^2MU (Leaky) Performance 68
4.6 Encoded L^2MU (Synaptic) Performance 70
4.7 Multi-Encoded L^2MU (Leaky) Performance 72
4.8 Multi-Encoded L^2MU (Synaptic) Performance 72
4.9 Results achieved deploying the L^2MU on commercial edge devices to solve the HAR task through encoding-free classification with two different neuron models. 75

A.1 L-RNN (Leaky) Metrics 80
A.2 L-RNN (Synaptic) Metrics 80
A.3 Encoded L-RNN (Leaky) Metrics 81
A.4 Encoded L-RNN (Synaptic) Metrics 81
A.5 Multi-Encoded L-RNN (Leaky) Metrics 82
A.6 Multi-Encoded L-RNN (Synaptic) Metrics 82

D.1 Base Hyperparameters for all Architectures 103
D.2 Hyperparameters Output Layer (Leaky/Synaptic) 103
D.3 Hyperparameters Single Encoder Layer (Leaky/Synaptic) 104
D.4 Hyperparameters Stacked Encoder Layer (Leaky/Synaptic) 104
D.5 Hyperparameters for RNN Architecture 105
D.6 Hyperparameters for LSTM Architecture 105
D.7 Hyperparameters for LMU Architecture 105
D.8 Hyperparameters for L-RNN (Leaky/Synaptic) Architecture 105
D.9 Hyperparameters for L^2MU (Leaky/Synaptic) Architecture 106

D.10 Hyperparameters value for LSTM 106
D.11 Hyperparameters value for LMU 106
D.12 Hyperparameters value for RNN 107
D.13 Hyperparameters for L^2MU (Leaky) 107
D.14 Hyperparameters for L^2MU (Synaptic) 107
D.15 Hyperparameters for Encoded L^2MU (Leaky) 108
D.16 Hyperparameters for Encoded L^2MU (Synaptic) 108
D.17 Hyperparameters for Multi-Encoded L^2MU (Leaky) 109
D.18 Hyperparameters for Multi-Encoded L^2MU (Synaptic) 109
D.19 Hyperparameters for L-RNN (Leaky) 110
D.20 Hyperparameters for L-RNN (Synaptic) 110
D.21 Hyperparameters for Encoded L-RNN (Leaky) 110
D.22 Hyperparameters for Encoded L-RNN (Synaptic) 110
D.23 Hyperparameters for Multi-Encoded L-RNN (Leaky) 111
D.24 Hyperparameters for Multi-Encode L-RNN (Synaptic) 111

List of Algorithms

1 Activation Sparsity 35
2 Connection Sparsity 36
3 High-Level Calculation of Synaptic Operations 37
4 ValidationDeltaStopping Callback 53

Chapter 1

Introduction

In recent years, there has been a significant paradigm shift in the deployment of machine learning models from centralized cloud environments to more distributed architectures that are located closer to sensors and edge devices.

This evolution facilitates real-time data processing, enabling faster response times and improved decision-making across various applications, including healthcare, fitness, and environmental monitoring. However, running artificial neural networks (ANNs) locally on devices introduces new challenges. The energy consumption associated with executing these models locally can be substantial, as the devices must manage not only the data acquisition but also the computational load of the neural networks.

As the demand for intelligent edge computing grows, the energy constraints of running complex models on mobile and wearable devices have become increasingly apparent. Traditional ANN architectures, while effective, often require significant computational resources and power, which can lead to battery drain and reduced operational lifetime for mobile devices. This situation emphasizes the urgent need for alternative computational paradigms that can balance performance with energy efficiency.

To address this energy challenge, neuromorphic hardware has emerged as a promising alternative. Neuromorphic systems leverage brain-inspired computing architectures that emulate the way biological neurons operate. This paradigm allows for the execution of models using significantly less energy than traditional ANN frameworks, making them well-suited for low-power applications. Spiking neural networks (SNNs), a subset of neuromorphic models, stand out due to their ability to process information asynchronously and sparsely, mirroring the dynamics of biological neural processing. While both ANNs and SNNs are designed for learning and recognizing patterns, the key difference lies in their information encoding: ANNs typically use continuous signals, whereas SNNs rely on discrete spikes to convey information.

Smart devices, wearable sensors, and edge computing can be conceptualized

as a fire triangle for efficiently implementing miniaturized, intelligent body sensor networks (BSNs) across diverse domains [1, 2, 3, 4, 5, 6]. This broad range of potential applications is often encapsulated in the definition of human activity recognition (HAR). HAR systems typically analyze data collected from wearable devices equipped with inertial measurement units (IMUs)[7, 8]. They serve not only as a practical framework for prototyping solutions but also as a means to explore the effectiveness of algorithms designed for time-varying signals. A variety of machine learning techniques, including deep learning models, are commonly employed to enhance HAR accuracy[9, 10, 11, 12].

Moreover, spiking neural networks (SNNs)[13] can be considered as valuable alternatives, offering a dual perspective. On one hand, they present a fresh approach to traditional models by drawing inspiration from the neuromorphic paradigm through biologically plausible neuron models. On the other hand, SNNs can be deployed on specialized chips[14, 15, 16] designed for asynchronous and sparse computations, providing advantages over conventional hardware. However, one major challenge when interfacing SNNs with real-world data is the encoding step, which converts continuous data into spike trains suitable for network input.

In this work, we focus on addressing the HAR task using a neuromorphic approach on commercial edge devices, without the need for encoding continuous data from the Wireless Sensor Data Mining (WISDM) smartphone and smartwatch activity and biometrics dataset [17, 18]. Our proposed L^2MU is a fully spiking implementation of the Legendre memory unit (LMU), where each block consists of leaky integrate-and-fire (LIF) neuron populations. This framework allows us to adopt the neuromorphic approach directly on raw sensor data, demonstrating a viable path for utilizing neuromorphic models on non-dedicated edge devices without prior data encoding.

This is one of the first attempts to bridge the gap between the neuromorphic paradigm, characterized by its discrete, sparse, and asynchronous properties, and the analog world of digital edge devices designed for synchronous computation. The independence from a spike-encoding step opens up possibilities for real-time applications in the Internet of Things (IoT) domain, where quick decisions and energy efficiency are paramount.

The implications of this work extend beyond the theoretical; they also address practical challenges in deploying intelligent systems in real-world environments. By reducing the need for complex preprocessing steps, we can streamline the integration of machine learning models into wearable devices, enhancing user experiences and expanding the capabilities of these technologies.

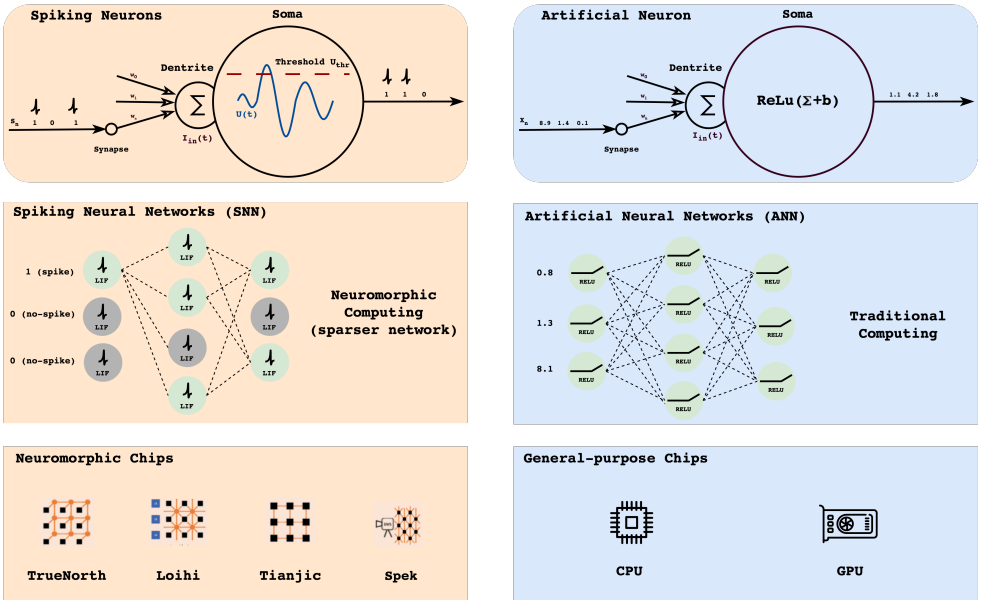

Figure 1.1: This illustration presents a comprehensive representation of spiking neurons and artificial neurons, along with their networks and corresponding hardware. On the top left is the spiking neuron model inspired by its biological structure, and on the top right is an abstract representation of the artificial neuron. Spiking Neural Networks on the middle left are shown as sparse configurations of neurons communicating through discrete spikes, whereas Artificial Neural Networks on the middle right, are dense, fully connected networks where neurons are continuously interacting. On the bottom left, neuromorphic chips, such as TrueNorth, Loihi, Tianjic, and Spek, constitute the design to emulate spiking neural networks in hardware. The right side features the purpose for which chips are generally utilized regarding artificial neural networks.

Innovations proposed by this work

- **Direct Use of Raw Sensor Data:** Developing a framework that operates directly on continuous sensor data without requiring spike encoding, thereby simplifying the data processing pipeline.

- **Implementation of L^{2}MU:** Proposing a fully spiking implementation of the Legendre memory unit (LMU) that leverages leaky integrate-and-fire (LIF) neurons for enhanced temporal pattern recognition.

- **Application to HAR:** Demonstrating the feasibility of using neuromorphic computing for human activity recognition in real-world scenarios, with an

emphasis on real-time processing.

- **Real-time IoT Applications:** Exploring potential applications in real-time Internet of Things (IoT) environments, facilitating rapid response to environmental changes and user activities.
- **Energy Efficiency Techniques:** Investigating model compression methods, such as pruning and quantization, to enhance energy efficiency across various devices, making them more suitable for battery-operated systems.
- **Scalability and Adaptability:** Evaluating the scalability of the proposed models across different platforms and contexts, ensuring adaptability to various applications beyond HAR.
- **User-Centric Design:** Focusing on user-centric applications, ensuring that the developed solutions meet the practical needs of end-users, ultimately driving wider adoption of wearable technologies.

Chapter 2

Background

2.1 Spiking Neural Network

Spiking Neural Networks (SNNs), a biologically inspired computational paradigm, fundamentally reimagines how we approach artificial intelligence. Unlike traditional artificial neural networks, which are built on continuous-valued activations and operate through parallel processing, SNNs are designed to closely mimic biological neural networks' temporal and spiking dynamics.

At the core of SNNs lies the concept of spiking behavior. In biological brains, neurons communicate through discrete electrical impulses or "spikes". SNNs replicate this process. Each neuron in an SNN emits spikes when its membrane potential reaches a certain threshold, and these spikes carry information in the form of precisely timed events. This temporal coding is a fundamental departure from the rate-based encoding used in traditional ANNs.

Another crucial characteristic of SNNs is their synaptic dynamics. SNN synapses are typically modeled to capture the biological phenomena of synaptic efficacy changes over time. One of the most significant factors influencing these changes is spike-timing-dependent plasticity (STDP) [19], which strengthens or weakens synapses based on the precise timing of pre-synaptic and post-synaptic spikes. This process enables SNNs to learn and adapt to dynamic input patterns.

The applications of SNNs span across various domains, including but not limited to pattern recognition, sensory processing, robotics, and neuromorphic hardware. Their ability to process information in a time-sensitive and energy-efficient manner has opened new horizons in creating intelligent systems that closely resemble the capabilities of biological organisms.

2.1.1 Biologic Foundation of SNNs

The brain is an extraordinarily complex structure with roughly 100 billion neurons [20]. It is structurally arranged by billions of interconnected synapses. Within this

neural network, information is transferred through electrical impulses known as spikes. The strength of the synaptic connection between a transmitting or presynaptic neuron and a receiving neuron determines the impact of a spike as it travels between these neurons. Both the synaptic strengths and the patterns of connections among neurons play a pivotal role in shaping the information-processing capabilities of the nervous system. Researchers have been deeply motivated by the brain's remarkable capacity to tackle complex problems, driving them to explore its processing functions and mechanisms of learning.

Artificial Neural Networks (ANNs) draw inspiration from the biological nervous system and have found successful applications in a wide range of domains [21], [22], [23]. However, they are considerably more abstract compared to their biological counterparts [24], and they struggle to capture the intricate temporal dynamics observed in biological neurons. As a response to this limitation, a new realm of artificial neural networks has emerged, focusing on more biologically plausible neuronal models, known as Spiking Neural Networks (SNNs).

SNNs are gaining prominence due to their exceptional ability to replicate the rich and intricate dynamics exhibited by biological neurons. They stand out for their capacity to represent and integrate diverse information dimensions, including time, frequency, and phase. This feature positions SNNs as a promising paradigm in the realm of computation, potentially enabling the modeling of complex information processing in the brain [25], [26], [27], [28], [29], [30], [31].

One of the key advantages of SNNs is their ability to handle substantial volumes of data, and they do so by employing trains of spikes to represent and carry information [30]. Furthermore, SNNs offer a notable advantage in terms of energy efficiency, making them well-suited for implementation on low-power hardware platforms. This multifaceted appeal of SNNs places them at the forefront of neural network research and opens up exciting possibilities for addressing complex computational challenges while staying aligned with the principles of biological neural processing.

2.1.2 Spike Generation

Spike generation is a pivotal process in Spiking Neural Networks (SNNs), emulating the behavior of biological neurons and forming the foundation of SNN computation. In SNNs, information is encoded and transmitted through discrete electrical events known as spikes or action potentials. Let's get into the intricate process of spike generation in SNNs:

1. Membrane Potential Dynamics: At the core of spike generation lies the membrane potential of artificial neurons. This potential, akin to the electrical charge across the cell membrane of biological neurons, plays a central role. It constantly evolves in response to incoming signals and determines the neuron's excitability.

2. Input Signals Integration: SNN neurons receive input signals from other neurons or external sources, and these signals influence the membrane potential. Excitatory signals depolarize the membrane potential, gradually bringing it closer to the spike threshold. Inhibitory signals, conversely, hyperpolarize the membrane potential, making it less likely for the neuron to spike.

3. Spike Threshold Crossing: Spike generation occurs when the membrane potential crosses a predefined threshold. This event mimics the biological phenomenon where the voltage difference across a neuron's membrane reaches a critical level, causing the neuron to "fire".

4. Action Potential Initiation: When the membrane potential surpasses the threshold, the neuron initiates an action potential. This is the moment when a spike is "generated". It is an all-or-nothing event; either the neuron spikes or it does not. This binary nature of spiking is a distinctive feature of SNNs.

5. Spike Propagation: Once a spike is generated, it propagates down the neuron's axon and communicates with other neurons. In SNNs, this spike transmission is typically achieved through synapses, which modulate the strength and timing of spike delivery to target neurons.

6. Refractory Period: After generating a spike, the neuron enters a refractory period, during which it is incapable of firing another spike. This refractory period aligns with the biological neuron's recovery phase and ensures that neurons do not spike too rapidly.

2.1.3 Spikes Transmission

Spike transmission is a pivotal process in Spiking Neural Networks (SNNs) and is instrumental in understanding how information is communicated within the network. In SNNs, neurons communicate through discrete electrical events called spikes or action potentials, enabling the encoding and transmission of information. Here, we highlight the intricacies of spike transmission within SNNs:

1. Synaptic Connections: Neurons in an SNN are interconnected through synapses, forming a complex network. Each synapse links a presynaptic neuron to a postsynaptic neuron, and it plays a crucial role in transmitting spikes.

2. Spike Generation: The process of spike transmission initiates with the generation of a spike in a presynaptic neuron. When the membrane potential of a presynaptic neuron crosses a predefined threshold, it triggers the generation of a spike.

3. Synaptic Weights: Each synapse has an associated synaptic weight, often referred to as synaptic efficacy. This weight determines the strength of the connection between the presynaptic and postsynaptic neurons. It influences how much the postsynaptic neuron responds to incoming spikes.

2.1.4 Spikes Encoding

Spike encoding is a crucial concept in Spiking Neural Networks (SNNs), representing how information from the external world or other neural networks is transformed into the language of spikes. SNNs use spikes, or action potentials, as their fundamental unit of communication and computation. Here, we explore the process of spike encoding in SNNs:

1. Signal Transformation: Spike encoding begins with the transformation of external signals or data into spike trains. These signals can be in the form of sensory data, real-world events, or the output of other neural networks.

2. Rate Encoding: One common method of spike encoding is rate encoding. In this approach, the frequency or rate of spikes within a given time window represents the strength or magnitude of the input signal. Higher rates of spikes indicate more intense input, while lower rates represent weaker input [32].

3. Time-to-First-Spike Encoding: An alternative approach is time-to-first-spike encoding. In this method, the timing of the first spike relative to the onset of the stimulus carries the encoded information. Early spikes represent one aspect of the input, while later spikes encode another aspect [32].

4. Temporal Encoding: Spike encoding in SNNs relies heavily on precise timing. The exact timing of each spike carries information, and the relative timing of spikes between neurons conveys relationships and patterns in the data [32].

2.1.5 Neuromorphic Hardware and SNNs

Neuromorphic hardware and Spiking Neural Networks (SNNs) share a close relationship, as neuromorphic hardware is designed to mimic the structure and function of the human brain, and SNNs are a biologically inspired neural network model.

One of the primary motivations behind the development of neuromorphic hardware is its potential for extreme energy efficiency, mimicking the brain's energy-efficient operation. SNNs naturally fit this paradigm because they use spikes, which are events that consume very little energy compared to the continuous computations in traditional ANNs.

Neuromorphic hardware operates on an event-driven or spike-based paradigm, closely mirroring the communication pattern of SNNs. Instead of continuously processing data, these systems respond only when there is a significant change or event, which is analogous to the sparse activation patterns in SNNs.

The human brain processes information in a massively parallel fashion. Neuromorphic hardware and SNNs both leverage this parallelism. In neuromorphic hardware, specialized hardware components can process multiple spikes in parallel, while SNNs inherently support parallelism as spike events can be processed independently by neurons.

Neuromorphic hardware is often used for real-time processing and control applications, much like the rapid event processing required by SNNs. SNNs can excel in tasks that demand low-latency response and real-time adaptation, making them a natural fit for neuromorphic hardware platforms.

Neuromorphic hardware platforms like IBM's TrueNorth [33], SpiNNaker [34], or Intel's Loihi [35] are designed to support the execution of SNNs efficiently. This allows researchers to leverage the advantages of SNNs while capitalizing on the energy-efficient and parallel processing capabilities of neuromorphic hardware. In summary, the synergy between neuromorphic hardware and Spiking Neural Networks is evident in their shared emphasis on biological plausibility, energy efficiency, event-based processing, parallelism, and real-time adaptation. The combination of neuromorphic hardware and SNNs holds promise for addressing a wide range of applications, including sensory processing, robotics, and edge computing, where energy-efficient, brain-inspired computation is of paramount importance. This intersection represents a frontier in both artificial intelligence and computational neuroscience, with the potential to unlock new possibilities in machine learning and neuromorphic computing.

2.1.6 Neurons Dynamic as Synaptic Operations

During spike generation, the neuron modifies its behavior because of the incoming spikes, updating its membrane potential. This concept of "neuron updating"is still vague from what emerges from the literature. From a spiking point of view, the cost associated with this operation could be considered negligible, because it is the emission of spikes that has more weight. Still, from a hardware point of view, we should consider the fact that updating the membrane potential requires setting a variable in the CPU register. Hence, this consumes energy for the needed computational resources. In this second case, the cost to generate a spike increases more considering the allocation/setting of a variable into the hardware. As highlighted in [36] synaptic operations in neuromorphic hardware often overlook stateful processes, such as memory access and updates.

Focusing on the hardware specifics of Loihi, we can see that one of the core architecture features called "dendrites", is responsible for updating the state variables. These variables include synaptic response current and the membrane potential, integral components of the neural computation process. This perspective raises the question of considering the membrane update as part of the synaptic operation to be considered to generate a spike.

2.2 Human activity recognition

In recent years, the task of human activity recognition (HAR) using data collected from sensors embedded in wearable smart devices has obtained significant attention. The availability of diverse datasets in this domain continues to expand, followed by a continuous stream of research publications exploring various deep-learning methodologies.

HAR entails the classification of signals generated by human actions, and the specific sensor or sensor network employed for data acquisition plays a pivotal role in defining different categories of HAR tasks. Among these sensors, those relying on Inertial Measurement Unit (IMU) data have gained prominence due to the increasing adoption of wearable devices for non-invasive motion monitoring.

In this work, we have selected the WISDM dataset [17] among all the available datasets [37, 38, 39] commonly used in the field of HAR studies. The WISDM dataset comprises data collected from devices (sensor-equipped) worn by 51 subjects, each instructed to perform 18 tasks lasting 3 minutes each. The data is collected from both a smartwatch (LG G Watch) worn on the subjects' dominant hands and a smartphone (Samsung Galaxy S5 or Google Nexus 5) placed in their pockets. The sensors capture accelerometer and gyroscope readings at a sampling frequency of 20 Hz, resulting in data samples with 6 dimensions corresponding to the readings along the three axes of the IMU sensors. A comprehensive summary of the dataset is shown in table 2.1.

Table 2.1: Summary information of the WISDM dataset.

Number of subjects	51
Number of activities	18
Minutes collected per activity	3
Sensor polling rate	20Hz
Smartphone used	Google Nexus 5/5x or Samsung Galaxy S5
Smartwatch used	LG G Watch
Number raw measurements	15,630,426

The 18 activities shown in table 2.2 are classified into three main categories:

- **Non-hand-oriented activities** include walking, jogging, and standing.
- **Hand-oriented activities related to general tasks**, such as writing, typing, or brushing teeth.
- **Hand-oriented activities tied to eating actions**, like consuming pasta or soup.

Table 2.2: 18 activities represented in the WISDM dataset

ACTIVITY	LABEL
Walking	A
Jogging	B
Stairs	C
Sitting	D
Standing	E
Typing	F
Brushing Teeth	G
Eating Soup	H
Eating Chips	I
Eating Pasta	J
Drinking from Cup	K
Eating Sandwich	L
Kicking (Soccer Ball)	M
Playing Catch w/Tennis Ball	O
Dribbling (Basketball)	P
Writing	Q
Clapping	R
Folding Clothes	S

Compared to its earlier version [40], the dataset in this release demonstrates a more balanced distribution of samples across the 18 activities. Figure 2.1 illustrates the distribution of samples across the classes. It appears that the accelerometer data holds the majority of the discriminative information necessary to categorize each sample. However, analysis of the distribution reveals significant overlap among nearly all classes. This issue is even more pronounced in the data from the gyroscope.

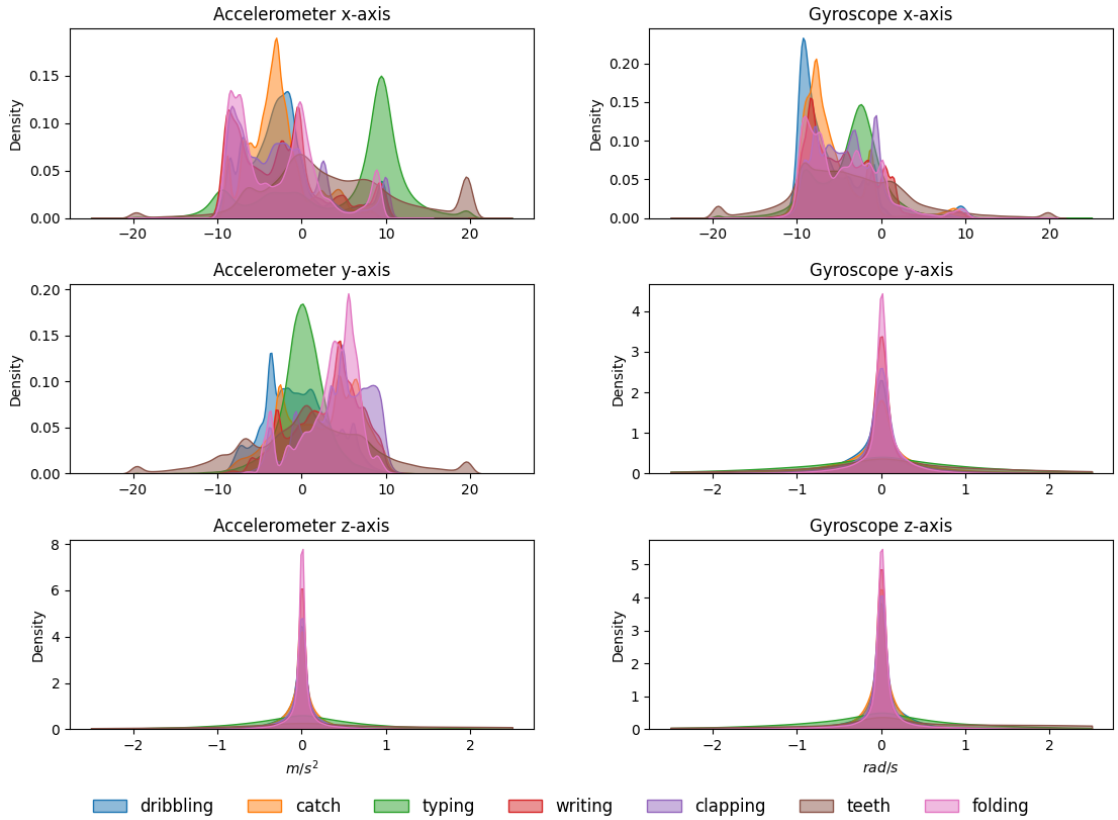

Figure 2.1: Distribution of signal values captured by both the accelerometer and gyroscope sensors along their respective X, Y, and Z axes.

2.3 Network Architectures

The ideal architecture for understanding temporal dynamics in time series data is a recurrent neural network-based architecture. These architectures serve as the fundamental foundation for our investigation because of their capacity to store a learned representation of previously learned knowledge and their capacity to analyze any input sequences using the internal state.

2.3.1 RNN

Recurrent Neural Networks (RNNs) [41] are a class of neural networks designed to handle sequential data. Unlike feed-forward neural networks, RNNs have the unique capability of maintaining internal memory, which allows them to process inputs both singly and sequentially. The network uses loops to accomplish this, whereby a neuron's output from one time step is stored and sent back into the same neuron in later time steps. RNNs can display temporal dynamic behavior thanks to

this looping process, which makes them ideal for use in time series prediction and classification, language modeling, and speech recognition, among other applications.

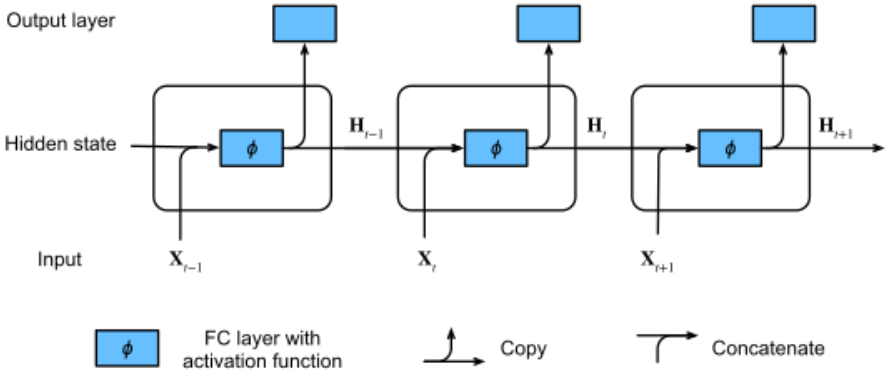

Figure 2.2: RNN Unit architecture.

Credits: `https://d2l.ai/chapter_recurrent-neural-networks/rnn.html`

2.3.2 LSTM

The development of Long Short-Term Memory networks (LSTMs) [42] was done in response to the vanishing and exploding vanishing problem presented by the RNNs. LSTM is a different type of RNN that can recognize long-term dependencies [43].

The main innovation in LSTMs is the cell state, which acts as a sort of conveyor belt and passes horizontally through the top of the LSTM cell. By transporting relevant information up and down the processing chain, it alleviates the vanishing gradient problem that is frequently encountered in traditional RNNs.

The input gate, forget gate and output gate are the three sophisticated gates that LSTMs use to accomplish this capability. By controlling the information flow into and out of the cell state, these gates enable the network to efficiently store or discard data for extended periods. The forget gate permits the cell to forget its past memories, the output gate regulates how much of the value in the cell state is utilized to calculate the output activation of the LSTM unit, and the input gate controls how much of a new value flows into the cell state.

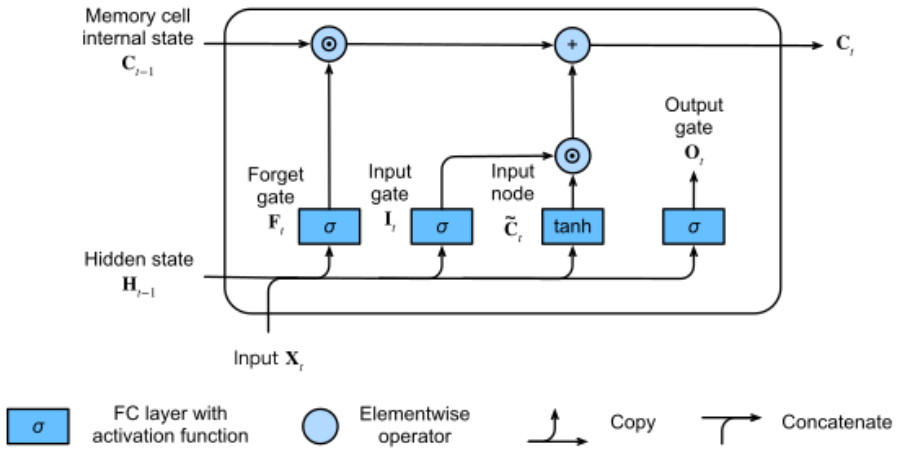

Figure 2.3: LSTM Unit architecture.

Credits: `https://d2l.ai/chapter_recurrent-modern/lstm.html`

2.3.3 Legendre Memory Unit

Within the realm of recurrent neural networks (RNNs), the Legendre Memory Unit (LMU)[44] emerges as an up-and-coming model, particularly for tasks involving the classification of time-based signals. This potential has been discovered in a recent study by Fra et al.[45], highlighting how the LMU distinguishes itself from conventional RNNs characterized by a high number of parameters. The LMU addresses this issue by reducing the number of required parameters, resulting in expedited training and memory-efficient deployment.

The principal component of the LMU network is the memory cell that can orthogonalize the continuous time history of the input $u(t)$ over a moving window of length θ. Over time, the memory cell assumes different behaviors reacting to the input signal. Equation 2.1 represents the linear transfer function that describes the behavior of the memory cell.

$$F(s) = e^{-\theta s} \tag{2.1}$$

This continuous-time behavior of the memory cell is approximated and implemented thanks to d coupled ordinary differential equations (ODEs). Equation 2.2 describes how a system changes over time based on its current state and inputs.

$$\theta \dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t) \tag{2.2}$$

where $\mathbf{m}(t)$ represents the state vector. The ideal state-space matrices $\mathbf{A}$ and $\mathbf{B}$ are determined using the Padé approximants as derived through the expressions provided in Equation 2.3 and 2.4.

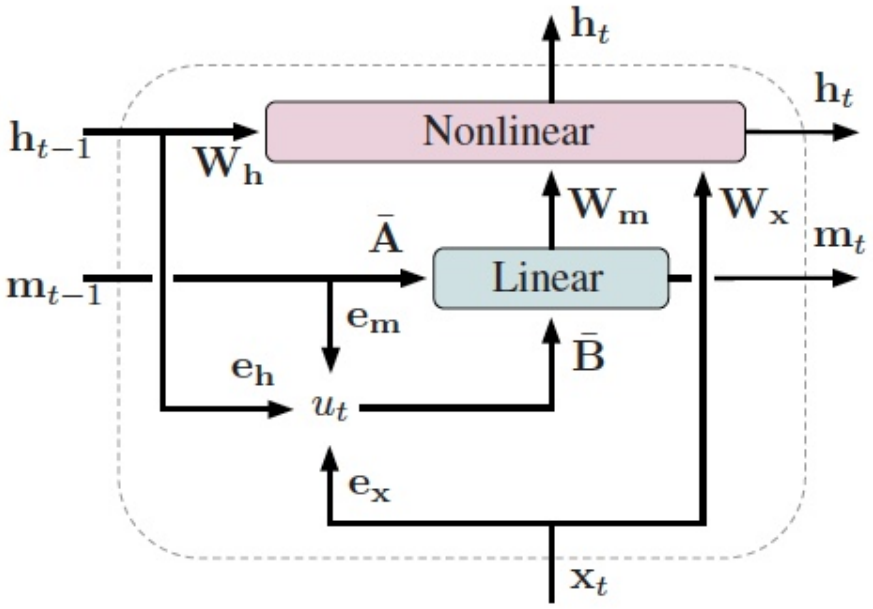

Figure 2.4: LMU Unit architecture.

Credits: [44]

$$\begin{aligned} &\mathbf{B} = [b]_i \in \mathbf{R}^{d-1} \\ &b_i = (2i+1)(-1)^i, \quad i, j \in [0, d-1] \end{aligned} \tag{2.3}$$

$$\begin{aligned} &\mathbf{A} = [a]_{ij} \in \mathbf{R}^{d \times d} \\ &a_{ij} = (2i+1) \begin{cases} -1 & i < j \\ (-1)^{i-j+1} & i \geq j \end{cases} \end{aligned} \tag{2.4}$$

The essential characteristic of the LMU network is the exceptional decoding performance of a signal with a temporal delay, denoted as $u(t - \theta')$, contained in the sliding time window of duration θ. The input signal $u(t)$ is projected into a high-dimensional manner and orthogonalized using shifted Legendre polynomials to accomplish the decoding operation. The mathematical expression for the i-th shifted Legendre polynomial in this context is provided by Equation 2.5

$$P_i(x) = (-1)^i \sum_{j=0}^{i} \binom{i}{j} \binom{i+j}{j} (-x)^i \tag{2.5}$$

and it is effectively employed to introduce a delay to the input signal, as described by the Equation 2.6

$$u(t-\theta') \approx \sum_{i=0}^{d-1} P_i\left(\frac{\theta'}{\theta}\right) m_i(t) \tag{2.6}$$

where the highest order, denoted as $d-1$, within the series expansion is directly correlated with the state vector's dimension, denoted as $\mathbf{m}(t)$.

2.3.4 Focus on LMU

This work primarily focuses on the Legendre Memory Unit cell. This choice is because LMU is based on the state-space model representation, a physical system where inputs, outputs, and internal state variables, which evolve over time, are described by differential equations. This approach aligns with the intention to explore brain-inspired architectures in neuromorphic computing [46, 47]. In contrast, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are effective for handling temporal dependencies, but they often require a large number of parameters and may struggle with issues like vanishing or exploding gradients.

The general form of the continuous-time state-space model is given by the following equations [46]:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}u(t) \tag{2.7}$$

$$y(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}u(t) \tag{2.8}$$

where:

- $\mathbf{x}(t)$ is the state vector representing the system's internal memory at time t,
- $u(t)$ is the input signal,
- $y(t)$ is the output signal,
- $\mathbf{A}$ is the state matrix that governs the dynamics of the state $\mathbf{x}(t)$,
- $\mathbf{B}$ is the input matrix that describes how the input $u(t)$ affects the state,
- $\mathbf{C}$ is the output matrix that maps the internal state to the output $y(t)$,
- $\mathbf{D}$ is the feedforward matrix that describes the direct influence of the input on the output.

This state space model formulation, which is based on LMU, allows storage and retrieval of temporal information capturing long-term dependencies using "minimal" computation resources, making it suitable for neuromorphic-based architecture.

This is crucial in applications like Human Activity Recognition (HAR), where activities often consist of different durations and require continuous memory over extended periods.

In summary, the LMU's use of a state-space approach, where the system's dynamics are represented through differential equations, provides a more computationally efficient model for capturing temporal dependencies, which is why we prioritize it in this work [48].

2.4 snnTorch

snnTorch is an open-source Python library built upon the popular PyTorch framework [49]. With snnTorch, it is possible to directly design an SNN, leveraging PyTorch's flexibility and efficiency for tensor computations and automatic differentiation. The library provides a comprehensive set of tools, including various spiking neuron models such as leaky integrate-and-fire (LIF) neurons and adaptive threshold neurons. It empowers the ability to design and train SNN architectures, simulate spiking dynamics, and explore spike-based coding and decoding schemes. snnTorch further enhances the PyTorch ecosystem by allowing users to seamlessly transition between conventional artificial neural networks and SNNs, enabling a unified platform for research and experimentation in neural network models. A range of spiking neuron types, provided by the library, are at the user's disposal, conveniently regarded as activation units within the PyTorch framework. This allows each layer of spiking neurons to seamlessly interface with diverse components, such as fully connected layers, convolutional layers, and residual connections, without requiring specific adaptations.

For the L^{2}MU, by working within the `snnTorch` framework, we selected two versions:

`Leaky`

First-order LIF model. The input is considered to be a current injection and the neuron's membrane potential is modeled in time through an exponential decay. Spike emission is regulated by Equation 2.9, where U_t denotes the membrane potential in time. If such quantity overcomes the threshold potential U_{thr}, a spike is emitted.

$$S_t = \begin{cases} 1, & if \quad U_t > U_{thr} \\ 0, & otherwise \end{cases} \tag{2.9}$$

The choice of reset mechanism, whether by subtracting a value from the membrane potential or setting it to a baseline, further refines its behavior. The activation of this mechanism occurs when $S_t = 1$, while it remains inactive when $S_t = 0$.

In the first case (subtracting mechanism) U_t, whenever the neuron generates a spike, the threshold value is subtracted from its membrane potential, as shown in Equation 2.10.

$$U_t = \beta U_{t-1} + I_{in,t} - S_t U_{thr} \tag{2.10}$$

Otherwise (zero mechanism), U_t will be set to 0 whenever a spike is generated, as shown in Equation 2.11

$$U_t = \beta U_{t-1} + I_{in,t} - S_t(\beta U_{t-1} + I_{in,t}) \tag{2.11}$$

`Synaptic`

Second-order LIF model. In this model, the synaptic current experiences a sudden increase upon the arrival of a spike, leading to a corresponding jump in the membrane potential. Both the synaptic current and the membrane potential subsequently decay exponentially, with respective rates denoted as alpha and beta. With this model, the temporal dynamics of the neuron's membrane potential account for an additional term corresponding to a synaptic conductance. Similarly to the membrane potential, such quantity decays in time with exponential behavior. As for the `Leaky` model, the emission of spikes is governed by Equation 2.9, in fact, they both have the same reset mechanism, with the substantial difference that synaptic current, I_{syn}, and membrane potential U_t, are both decayed by two different factors, α and β respectively.

The equations (2.12) and (2.13) show the behavior of the neuron in case the threshold value is subtracted from its membrane potential.

$$I_{syn,t} = \alpha I_{syn,t-1} + I_{in,t} \tag{2.12}$$

$$U_t = \beta U_{t-1} + I_{syn,t} - S_t U_{thr} \tag{2.13}$$

Equations (2.12) and (2.14) represent the scenario where the membrane potential is set to zero.

$$U_t = \beta U_{t-1} + I_{syn,t} - S_t(\beta U_{t-1} + I_{syn,t}) \tag{2.14}$$

2.5 NeuroBench

NeuroBench is an open-source benchmark tool for neuromorphic computing algorithms and systems [50]. It is composed of a system track for fully deployed solutions and an algorithm track for evaluation that is independent of hardware.

The goal of the algorithm benchmark track is to assess algorithms independently of systems, separating algorithm performance from particular implementation characteristics.

2.5.1 Metrics

The algorithm track defines primary metrics that are independent of the type of solution and are typically applicable to all kinds of solutions, including spiking and artificial neural networks (ANNs and SNNs).

Activation Sparsity

The average sparsity of neuron activations throughout execution is defined as follows: 0 denotes no sparsity (i.e., all neurons are always activated), and 1 denotes the scenario in which all neurons have zero output. This information is collected for all timesteps of all examined samples. See Algorithm 1 for the implementation.

Algorithm 1 Activation Sparsity

Require: A neural network model, predictions, and dataset

function ActivationSparsity($model, preds, data$)
  Initialize $total\_spike\_num \leftarrow 0$ and $total\_neuro\_num \leftarrow 0$
  for each *hook* in $model.activation\_hooks$ **do**
    for each *spikes* in $hook.activation\_outputs$ **do**
      $spike\_num \leftarrow$ count of non-zero values in *spikes*
      $neuro\_num \leftarrow$ total number of values in *spikes*
      $total\_spike\_num \leftarrow total\_spike\_num + spike\_num$
      $total\_neuro\_num \leftarrow total\_neuro\_num + neuro\_num$
    end for
  end for
  if $total\_neuro\_num = 0$ **then**
    $sparsity \leftarrow 0.0$
  else
    $sparsity \leftarrow \frac{total\_neuro\_num - total\_spike\_num}{total\_neuro\_num}$
  end if
  return $sparsity$
end function

Connection Sparsity

The connection sparsity for a certain model is calculated by dividing the total number of weights, accumulated over all layers, by the number of zero weights. One represents complete sparsity (no connections), while zero denotes no sparsity (completely connected). This statistic takes into consideration sparse network designs and intentional pruning. See Algorithm 2 for the implementation.

Algorithm 2 Connection Sparsity

Require: A neural network model

function GetNrZerosWeights(*module*)
    Initialize *count_zeros* ← 0 and *count_weights* ← 0
    children ← list of *module*'s children
    if *children* is empty **then**
        if *module* is in supported layer types **then**
            count_zeros ← number of zeros in *module.weight*
            count_weights ← *module.weight.numel*()
        else if *module* is a recurrent layer or cell **then**
            Compute *count_zeros* and *count_weights* for recurrent parameters
        end if
    else
        for each *child* in *children* **do**
            child_zeros, *child_weights* ← GetNrZerosWeights(*child*)
            count_zeros ← *count_zeros* + *child_zeros*
            count_weights ← *count_weights* + *child_weights*
        end for
    end if
    return *count_zeros*, *count_weights*
end function

layers ← model's layers
Initialize *total_zeros* ← 0 and *total_weights* ← 0
for each *module* in *layers* **do**
    zeros, *weights* ← GetNrZerosWeights(*module*)
    total_zeros ← *total_zeros* + *zeros*
    total_weights ← *total_weights* + *weights*
end for
$sparsity \leftarrow \frac{total\_zeros}{total\_weights}$
return $round(sparsity, 3)$

Synaptic Operations

Based on neuron activations and the corresponding fanout synapses, the average number of synaptic operations per model execution is calculated. Dense represents the number of operations required on hardware that does not support sparsity and takes into consideration all zero and nonzero neuron activations as well as synaptic connections. Eff_MACs and Eff_ACs reflect operation cost on sparsity-aware hardware by counting only effective synaptic operations and ignoring zero activations (e.g., created by the ReLU function in an ANN or no spike in an SNN). Multiply-accumulates (MACs) are synaptic operations with non-binary activation, whereas accumulates (ACs) are those with binary activation. See Algorithm 3 for the implementation.

Algorithm 3 High-Level Calculation of Synaptic Operations

Require: A neural network model, predictions, and data
Ensure: Synaptic operations (MACs for ANNs, ACs for SNNs)

Initialize counters for MACs, ACs, total synaptic operations, and total samples
function ResetCounters
    Reset MAC, AC, total synaptic operations, and total samples counters to zero
end function
function CalculateSynapticOperations($model, preds, data$)
    for each connection hook in the model **do**
        Extract input activations for the current layer
        for each set of activations in the input **do**
            Determine the type of operations (MAC or AC) for the layer
            Accumulate the operation counts based on the layer type
        end for
    end for
    Update the total number of samples processed
    return ComputeAverageOperations
end function
function ComputeAverageOperations
    if no samples processed **then**
        return default operation counts (zeroes)
    end if
    Compute average MACs and ACs per sample
    Compute total average synaptic operations per sample
    return average MACs, ACs, and total synaptic operations
end function

2.6 Edge Devices

Edge devices refer to the computational resources and sensors located at the periphery of a network, close to where data is generated. Unlike traditional centralized computing models, which rely on processing data in distant data centers (cloud computing), edge computing decentralizes computation by bringing it closer to the data source. This paradigm shift is driven by the need for low-latency processing, improved bandwidth efficiency, enhanced data privacy, and the ability to operate in real-time environments.

2.6.1 Edge Devices Characteristics

Edge devices exhibit several key characteristics that distinguish them from traditional computing systems:

Proximity to Data Sources: Edge devices are situated close to the data generation points, such as sensors, cameras, and user devices. This proximity reduces the time required for data to travel to the processing unit, thereby minimizing latency.

Resource Constraints: Typically, edge devices have limited computational power, memory, and storage compared to central servers. This necessitates efficient algorithms and lightweight models to perform computations within these constraints.

Energy Efficiency: Many edge devices operate on battery power or have strict energy consumption requirements. As a result, they prioritize energy-efficient operations to extend battery life and reduce power usage.

2.6.2 Potential benefits of SNNs on Edge Devices

Energy Efficiency: SNNs' event-driven nature aligns well with the power constraints of edge devices. Neuromorphic hardware, designed to implement SNNs, can further enhance energy efficiency by emulating the brain's low-power computations. However, the actual energy savings can depend on the specific SNN model and the hardware used.

Low Latency: Processing data locally on edge devices using SNNs reduces the need for data transmission to centralized servers, decreasing latency and enabling real-time applications. The complexity of the SNN models should also be taken into account because it could introduce additional processing delays, especially on less powerful edge devices.

Adaptability: SNNs can adapt to varying computational loads and data sparsity, making them scalable for different edge applications. Their ability to handle dynamic and temporal data is particularly beneficial for environments where real-time decision-making is essential. However, this adaptability could be reached by

increasing the design and the implementation complexity, and the advantage may vary depending on the computational resources on the edge devices.

2.7 Model compression

Model compression is a technique in the field of deep learning, which aims to reduce the size of models to optimize both latency and memory footprint, keeping the model performance reasonably comparable to the uncompressed model. This optimization is essential for deploying deep learning models on devices with limited resources, such as mobile phones and embedded systems, and for enhancing performance in terms of speed and efficiency.

Two main techniques in model compression are pruning and quantization, which, when applied effectively, can significantly reduce the computational resources required without substantially compromising the model's accuracy.

2.7.1 Pruning

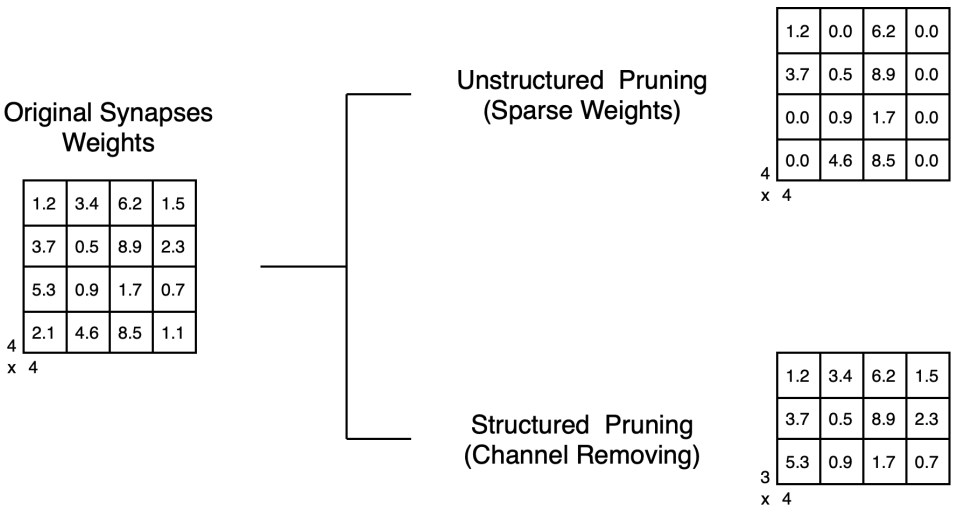

Figure 2.5: Effect of pruning techniques on the synapse weights of the original model.

Pruning is a process that simplifies neural network models by eliminating unnecessary weights, biases, or parameters, leading to a more compact and efficient model. The underlying idea is to identify and remove parts of the model that contribute minimally to its output, by creating a sparse version of the original

model. This sparsity is achieved by setting the synaptic weights to zero, effectively eliminating their contribution to the model's information flow.

Pruning can be categorized into two main types: structured and unstructured pruning, each with its own methodology and implications for model performance and storage.

Structured

Structured pruning involves the systematic removal of entire channels or layers from the neural network. This approach reduces the model's complexity by decreasing the dimensionality of the synaptic weight matrices, which, in turn, reduces the model's memory footprint and improves its latency. In structured pruning, the focus is on identifying and eliminating less significant channels or layers based on certain criteria, such as their impact on the output or their weight magnitudes.

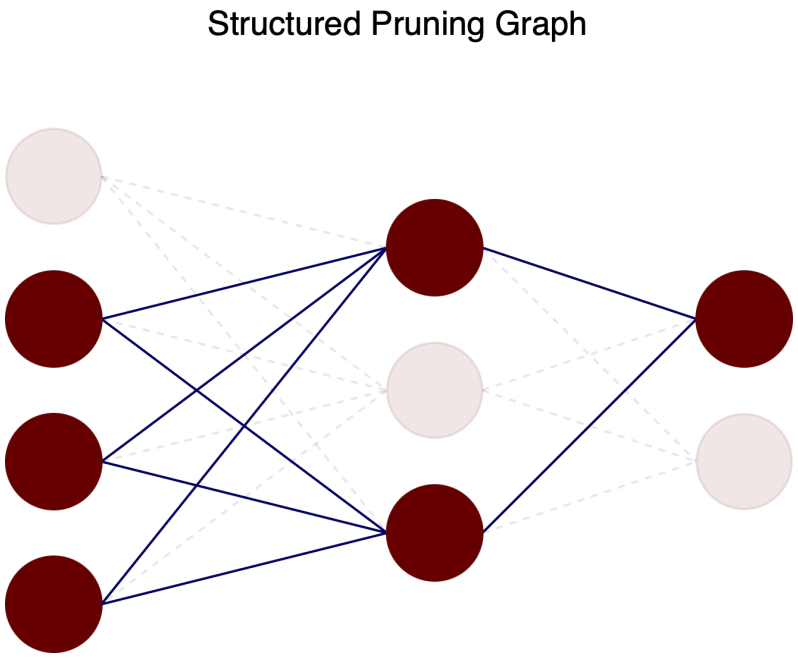

Figure 2.6: The image illustrates a structured pruned neural network. In structured pruning, entire neurons and their associated connections are removed, rather than just setting individual weights to zero. This is indicated by the shadowed connections and neurons in the image. The red circles symbolize active neurons, while the absence of shadowed circles and lines where neurons and connections have been removed illustrates the network after pruning. The solid blue lines represent the remaining active connections.

Unstructured

Unstructured pruning, on the other hand, targets individual weights within the neural network, setting them to zero to create a sparse matrix. Unlike structured pruning, unstructured pruning maintains the original dimensions of the weight matrices, which means that the computational speedup might not be as significant unless specialized hardware or software that can efficiently handle sparse matrices is used.

The advantage of unstructured pruning is its fine-grained approach, which can maintain higher model accuracy by allowing for an increased number of minor changes to the network topology. However, the irregularity of the sparsity patterns can make it difficult to maximize the computational gain, especially on hardware designed for dense matrix operations.

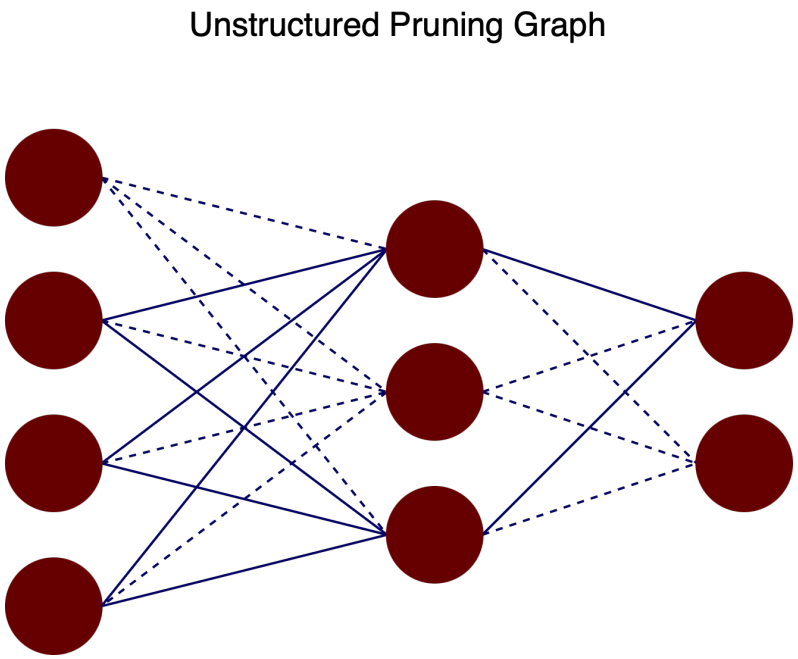

Figure 2.7: The image illustrates a pruned neural network graph where the dashed lines represent connections that have been pruned. Pruning in this context means that the weights associated with these connections have been set to zero in the weight matrix, effectively eliminating their influence on the network's computations. The solid lines are likely to represent the active connections with non-zero weights. This graphical representation emphasizes the distinction between active and pruned connections within a neural network's architecture after pruning has occurred.

2.7.2 Quantization

Quantization is a crucial technique in the field of model compression for deep learning. It involves reducing the number of bits used to represent the weights and activations of a neural network. By converting these values from high-precision (e.g., 32-bit floating-point) to lower precision (e.g., 8-bit integer), quantization aims to significantly decrease the memory footprint and computational requirements of the model. This is particularly beneficial for deploying neural networks on resource-constrained devices such as mobile phones, embedded systems, and edge devices.

Quantization in Spiking Neural Networks

When applying quantization in Spiking Neural Networks, not only the precision of the weights can be reduced, but it is also possible to quantize the internal state of the neurons, such as the membrane potential. This also leads to a reduction in terms of memory footprint, while maintaining the event-driven nature of the SNN. Quantization of SNNS is also necessary when models are deployed on neuromorphic chips. This hardware typically operates with lower precision, making quantization an essential step.

Chapter 3

Materials and methods

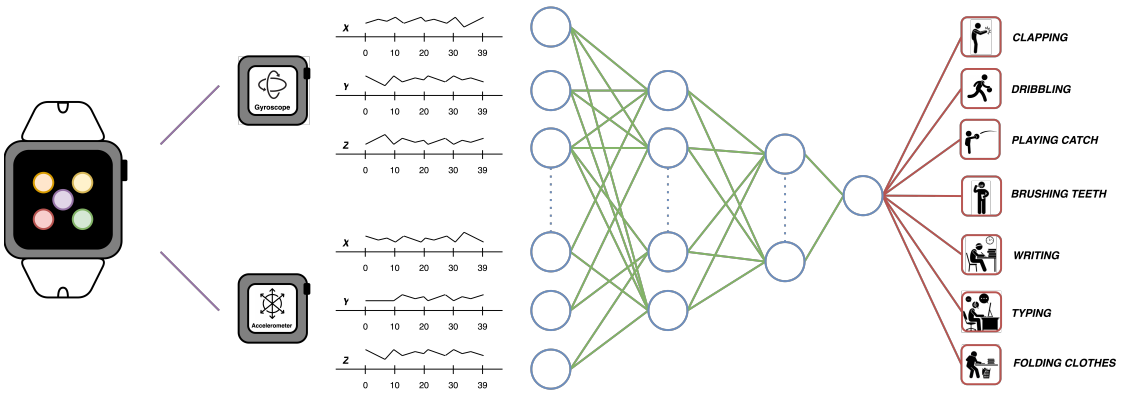

Figure 3.1: This diagram illustrates the real-time human activity tracking process of a smartwatch. Utilizing onboard gyroscope and accelerometer sensors, the smartwatch collects data over 2 seconds, equivalent to 40 samples. These data are then transmitted to an artificial neural network. The neural network processes the sensor data and classifies the wearer's activity such as clapping, dribbling, playing catch, brushing teeth, writing, typing, or folding clothes.

Building a neuromorphic model means taking inspiration from discrete and sparse computation. Nonetheless, just as the human brain must account for real-world signals, which can be considered continuous unless we enter the quantum realm, so has every SNN to interface with traditional sensors. In this section, we describe how we implemented our L^2MU and the architecture depicted in Figure 2.4 to work with raw non-spiking signals and how we employed it to solve the HAR task on commercial edge devices.

3.1 Encoding module

The signal encoding module is a crucial component within the spiking neural network architecture. Its function is to transform the raw sensor data from the accelerometer and gyroscope into spike-based signals, suitable to be processed by neuromorphic architectures, avoiding the necessity for traditional spike-encoding techniques such as latency coding or rate coding.

3.1.1 Single Encoder

The single encoder design incorporates channel-specific neurons. The rationale behind these channel neurons stands in the fact that each axis of input (accelerometer and gyroscope X, Y, and Z) holds information that is distinctive and essential for the accurate interpretation of motion and orientation. Also, each axis input has a different dynamic behavior that could cause the increase/decrease of the firing rate of some neurons to a point where they fire too much because the signal exceeds the threshold many times, or do not fire at all because the signal never overcomes the threshold.

Incorporating these channel-specific neurons, the network can independently analyze the unique characteristics of each axis and enhance the sensitivity of the network by fine-tuning the neuron's parameters such that they adapt to the signal dynamic.

`Implementation`

In the proposed single encoder shown in Figure 3.2, channel-specific neurons are aligned with corresponding specific axes from the accelerometer and gyroscope. This group of neurons receives raw time-series data in the form of synaptic current.

During this phase, the channel-specific neurons act as the first layer of processing. Their spiking responses to input current are carefully calibrated to ensure that the dynamic of the input current that depends on the raw sensory signals is preserved.

3.1.2 Stacked Encoder

The stacked encoder consists of a multi-layer structure that transforms six-axis input data from both the accelerometer and gyroscope into a series of spikes. The idea behind this encoder is based on the previously described Single Encoder but with an extension of its functionality. Specifically, the latter tries to compress and summarize the spike signals generated by the Channel-Specific Neurons, such that before reaching the spike-based recurrent unit, the signal is consistent.

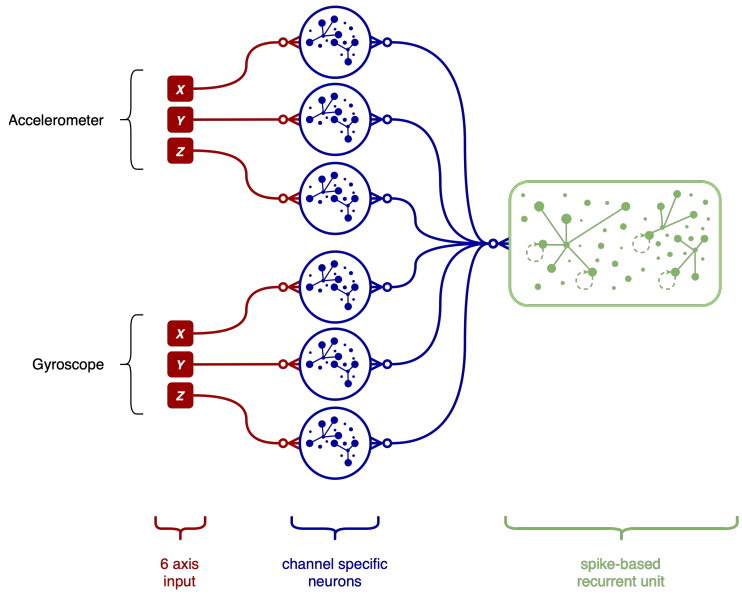

Figure 3.2: Illustration of a spiking single layer encoding module consisting of channel-specific neurons. On the left, the "6-axis input" represents raw time-series data from the accelerometer and gyroscope sensors. In the middle, "channel-specific neurons" are shown receiving synaptic currents corresponding to each sensory axis. On the right, the processed signals converge in a spike-based recurrent unit.

`Implementation`

The stacked encoder architecture is structured following this hierarchy:

Channel-Specific Neurons: At the first level, separate groups of neurons are dedicated to each of the three axes (X, Y, Z) for both the accelerometer and gyroscope. This layer is exactly like the one described before.

Fusion Neurons: The second layer involves fusion neurons that synthesize the information from the channel-specific neurons. This layer allows for inter-channel relationships and patterns to emerge. These fusion neurons facilitate the combination of data from all expanded six axes.

Harmonization Neurons: After fusion, the harmonization neurons serve to fine-tune the combined spike output, ensuring that the signal maintains coherence and minimizing any discrepancies that may arise from the fusion process. This harmonization is vital for maintaining the temporal dynamics of the input data.

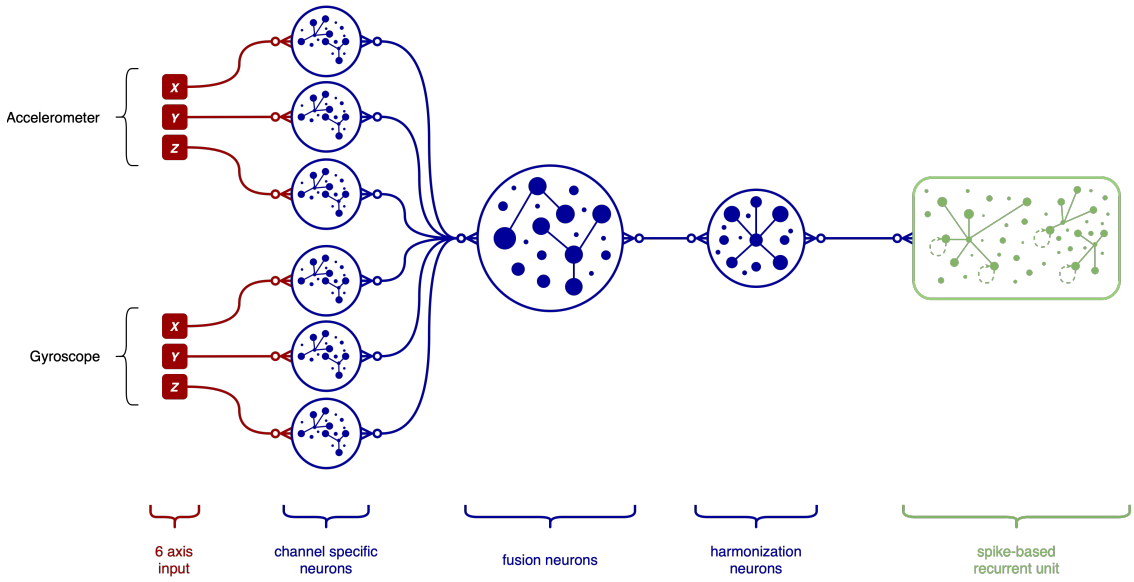

Figure 3.3: Illustration of a spiking multi-layer encoding module consisting of channel-specific neurons, fusion, and harmonization neurons. On the left, the same structure of the single encoder is used, the "6-axis input" connected to the "channel-specific neurons". These neurons communicate with the "fusion neurons" which compress the information received by the previous neuron layer. Finally, the "harmonization neurons", receiving the spike signal from the previous layer, fine-tune the signal trying to minimize any discrepancies raised by the previous layers. At this point, this layer emits spike trains which converge to a spike-based recurrent unit.

3.2 LIF-based LMU (L^2MU)

The L^2MU design draws its foundation from the original implementation, yet a noteworthy distinction exists. In this adaptation, every constituent element inherent to the original LMU, comprising the memory, hidden layer, and its internal encoding module, has transformed, thereby assuming the form of neuron populations. In this process, all the equations governing the interactions among the different components of the LMU have been adapted to manage the synaptic currents that establish connections between the individual neurons. This comprehensive reconfiguration empowers the entire framework to proficiently manage spiking signals, fundamentally aligning it with the domain of spiking neural networks. In Figure 2.4, the resulting architecture is depicted. The conversion of the primary components of the LMU is accomplished through two alternatives relying on the two neuron models `Leaky` and `Synaptic`. Importantly, these neuron types are not integrated; rather, separate networks are established for each type. It's worth noting that the

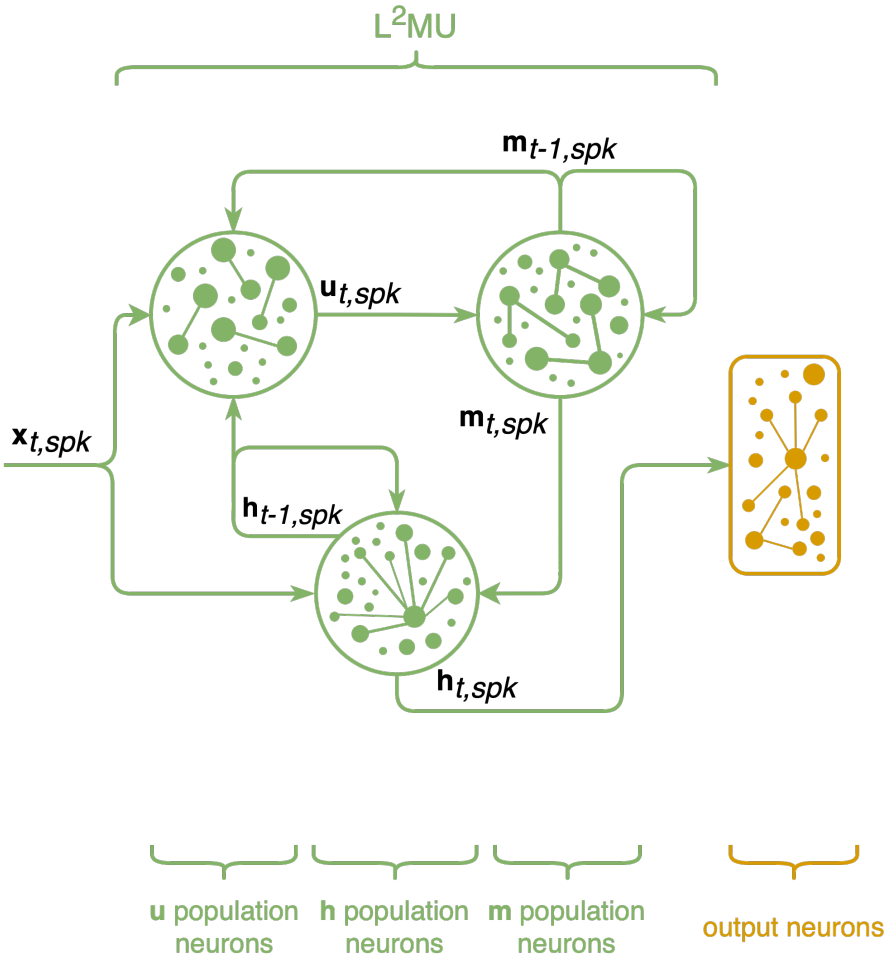

Figure 3.4: Illustration of the L^2MU where each component is represented with a population of neurons. The input, denoted as $x_{t,spk}$, feeds into the model at time step t, which influences the **u** and **h** population neurons. The recurrent information flow between the **m** population, as well as the **h** population. The output neurons, gather information from the hidden neuron population to generate the final output.

conversion of each block forming the LMU does not merely involve the transformation into a single neuron. Instead, each block is represented as a population of neurons. Directing our attention to the interconnection of these components, which have been translated into neuron populations in our scenario, a key distinction arises from the notion that each connection is conceptualized as a synapse linking neurons. Consequently, every component communicates with others under the synaptic current. This mechanism facilitates the emission of spikes by neurons, effectively facilitating the flow of information among the components.
Distinguishing inherent neuromorphic feature of the L^2MU, compared to the LMU,

is that, by definition, the connection of the different building blocks implies neural communication through spikes as a response to changes in current and voltage values.

As explained in the original paper [44], the LMU takes an input signal $\mathbf{x}_t$ and produces a hidden state $\mathbf{h}_t$, which is subsequently fed to the memory state $\mathbf{m}_t$. This progression is mediated by $\mathbf{u}_t$, which operates the transformation onto the Legendre polynomial basis. The mathematical formulation of this process is recalled by Equation 3.1:

$$\mathbf{u}_t = \mathbf{e}_x^\mathsf{T}\mathbf{x}_t + \mathbf{e}_y^\mathsf{T}\mathbf{h}_{t-1} + \mathbf{e}_m^\mathsf{T}\mathbf{m}_{t-1} \tag{3.1}$$

where $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_m$ are encoding vectors [44]. In the L^{2}MU, these steps are formulated with Equation 3.2 by modeling the synaptic current across the different populations of neurons:

$$\mathbf{u}_{t,curr} = \mathbf{e}_x^\mathsf{T}\mathbf{x}_{t,spk} + \mathbf{e}_y^\mathsf{T}\mathbf{h}_{t-1,spk} + \mathbf{e}_m^\mathsf{T}\mathbf{m}_{t-1,spk} \tag{3.2}$$

where the spike emission is ruled by Equation 3.3 as

$$\mathbf{u}_{t,spk} = \begin{cases} 1, & if \quad \mathbf{u}_{t,mem} > \mathbf{u}_{thr} \\ 0, & otherwise \end{cases} \tag{3.3}$$

being $\mathbf{u}_{thr}$ the threshold voltage to be overcome by the membrane potential $\mathbf{u}_{t,mem}$ for spike emission.

The encoded information is then written into the memory state, which can be thought of as a reservoir that maintains a history of the input. The evolution in time of the memory state, originally described through the definitions of Equation 2.2, 2.3, 2.4, and 2.6, is translated into Equation 3.4

$$\mathbf{m}_t = \overline{\mathbf{A}}\mathbf{m}_{t-1} + \overline{\mathbf{B}}\mathbf{u}_t \tag{3.4}$$

where ($\overline{\mathbf{A}}$, $\overline{\mathbf{B}}$) represent discretized matrices defined as in Equation 3.5

$$\overline{\mathbf{A}} = (\Delta t/\theta)\mathbf{A} + \mathbf{I}\ ,\ \overline{\mathbf{B}} = (\Delta t/\theta)\mathbf{B} \tag{3.5}$$

with Δt representing a time step within a window of length θ.

With similar arguments as for Equation 3.2 and 3.3, the fully spiking counterpart of Equation 3.4 can be defined as

$$\mathbf{m}_{t,curr} = \overline{\mathbf{A}}\mathbf{m}_{t-1,spk} + \overline{\mathbf{B}}\mathbf{u}_{t,spk} \tag{3.6}$$

with

$$\mathbf{m}_{t,spk} = \begin{cases} 1, & if \quad \mathbf{m}_{t,mem} > \mathbf{m}_{thr} \\ 0, & otherwise \end{cases} \tag{3.7}$$

where $\mathbf{m}_{t,mem}$ denotes the membrane potential and $\mathbf{m}_{thr}$ is the threshold potential.

Adopting the same approach, the hidden state defined by Equation 3.8 as

$$\mathbf{h}_t = \tanh(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_m\mathbf{m}_t) \tag{3.8}$$

is translated into a neuron population described by Equation 3.9

$$\mathbf{h}_{t,curr} = \mathbf{W}_x\mathbf{x}_{t,spk} + \mathbf{W}_h\mathbf{h}_{t-1,spk} + \mathbf{W}_m\mathbf{m}_{t,spk} \tag{3.9}$$

with

$$\mathbf{h}_{t,spk} = \begin{cases} 1, & if \quad \mathbf{h}_{t,mem} > \mathbf{h}_{thr} \\ 0, & otherwise \end{cases} \tag{3.10}$$

where, coherently with the previous cases, $\mathbf{h}_{t,mem}$ and $\mathbf{h}_{thr}$ represent the population-specific membrane voltage and threshold potential respectively.

In contrast to the original paper, wherein an activation function is applied in Equation 3.8, our version omits this step. The rationale behind this omission comes from the discrete nature of the signals emitted by neurons. As such, there is no requirement for an additional activation function, since the neuron itself can be perceived as an inherent activation function.

3.3 Activities selection and segmentation

Aligned with our goal, the dataset employed remains consistent with the version featured in the referenced paper [45]. In this version, the signals have been additionally partitioned into non-overlapping temporal windows, each consisting of a duration of 2 seconds (40 sample signals in total). Figure 3.5 shows a random sample from the selected dataset recorded for 2 seconds with the smartwatch. This selection is primarily motivated by the aim to create a system that operates effectively in real-world scenarios, where achieving low response latency is a must for delivering a consistent user experience. This dataset version constitutes a subset derived from the whole WISDM dataset, concentrating solely on hand-oriented activities captured by smartwatch devices. This specific focus aligns with the emphasis on wearable devices, which boast the potential for individualized application across an array of domains and have witnessed a notable increase in adoption. The subset consists of 36,201 samples, including raw signals without any prior feature extraction. These samples have been partitioned into training, validation, and test sets, maintaining a distribution of 60%, 20%, and 20% respectively as previously done in [45]. An analysis of their probability density, based on kernel density estimation, is shown in Figure 2.1. A 60:20:20 partition was then performed to define training, validation, and test set respectively.

Table 3.1, presents the sizes of training, validation, testing, and calibration datasets used in this work. The calibration dataset is a small subset of the training dataset used specifically for post-training quantization, to adjust the model's weights and activation to lower precision.

Table 3.1: Dataset Sizes for Training, Validation, Testing, and Calibration

Dataset	**Samples**	**Time Steps**	**Features**
Training	21,720	40	6
Validation	7,240	40	6
Testing	7,241	40	6
Calibration	100	40	6

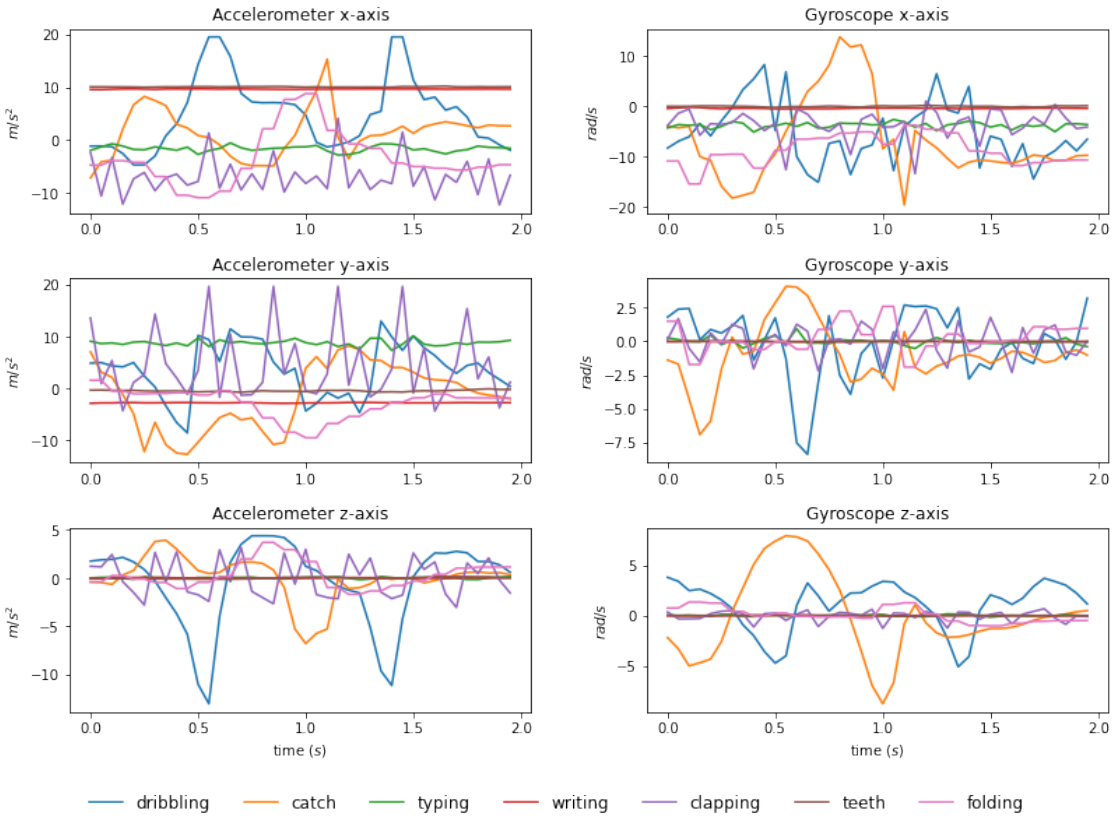

Figure 3.5: Random sample of 2 seconds recorded by the smartwatch on the 6 IMU sensors for the 7 classes in the "hand-oriented activities related to general tasks" subset of the WISDM dataset.

3.4 Hyperparameter optimization

Artificial Neural Networks (ANNs) are characterized by their network structure, composed of different layers and their interconnections, as well as hyperparameters controlling the network's behavior. To avoid unnecessary complexity and find the best configuration of the network, hyperparameter optimization (HPO) has been employed. Our HPO was conducted using the Neural Network Intelligence (NNI) toolkit, employing the Anneal algorithm. These optimization experiments consisted of multiple trials, each involving four evenly spaced random re-initializations of the tuner. This strategy aimed to mitigate the impact of local minima, which can affect annealing algorithms. Each trial consists of 300 epochs. The Adam optimizer was employed, utilizing a constant learning rate that was optimized through experiment trials. To enhance computational efficiency, we incorporated early stopping mechanisms. An "assessor" within NNI monitored intermediate results of each trial and halted computations if sub-optimal outcomes were predicted, thereby saving computational resources.

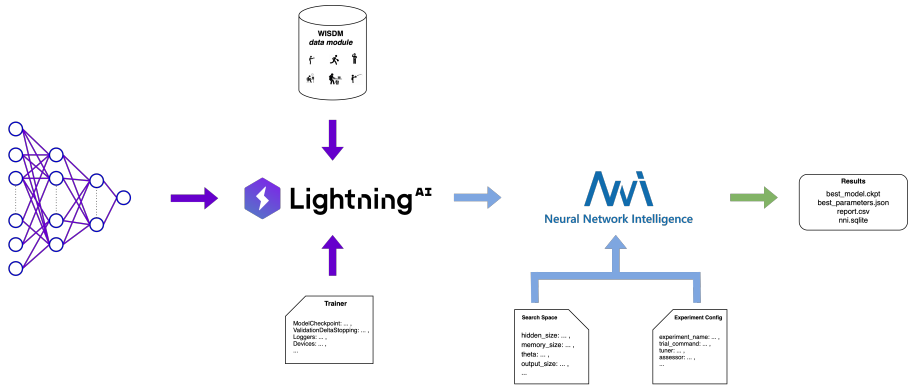

Figure 3.6: This image represents a machine learning workflow using the Lightning AI framework and Neural Network Intelligence (NNI) for hyperparameter optimization in human activity recognition tasks. The left side shows a neural network model, which is fed time-series data from a human activity dataset. In the center, the "Trainer" configuration includes checkpoints, callbacks, and logging mechanisms for model training. The "Neural Network Intelligence" block indicates the use of NNI to automate the search for optimal hyperparameters, configured by a "Search Space" and an "Experiment Config" that specifies the tuner, assessor, etc... The result of this process is a comprehensive database containing trial results, a checkpoint of the most effective model, the finest-tuned parameters, and detailed logs.

Another early stopping mechanism was implemented. This stopper, called "ValidationDeltaStopping", leverages a custom callback mechanism to dynamically halt training based on fine-grained changes in validation loss and accuracy metrics. By

monitoring the percentage variations in these metrics across consecutive epochs, our method offers a robust and adaptable stopping criterion. The callback incorporates multiple conditions, including constraints on small and significant changes in validation loss, as well as shifts in validation accuracy. These conditions are designed to capture intricate fluctuations in model performance, enabling the detection of subtle changes and avoiding prolonged training without meaningful progress. Our "ValidationDeltaStopping" callback seamlessly integrates into existing training frameworks and has demonstrated its effectiveness in various experimental settings, showcasing its potential to contribute to more efficient and effective neural network training practices.

Algorithm 4 ValidationDeltaStopping Callback

Initialize thresholds for validation loss decrease, loss increase, accuracy decrease, and accuracy increase.
On validation epoch end, update the current validation loss and accuracy.
if Current epoch ≥ 2 **then**
    Calculate the percentage change in validation loss and accuracy from the previous epoch.
    Check if the change in validation loss is less than the threshold for decrease.
    if Change in validation loss is small for a specified number of epochs **then**
        Prepare to trigger early stopping due to minor loss improvement.
    end if
    Check if the increase in validation loss exceeds the threshold for increase.
    if Validation loss increases significantly for a specified number of epochs **then**
        Prepare to trigger early stopping due to loss worsening.
    end if
    Check if the change in validation accuracy is less than the threshold for increase.
    if Change in validation accuracy is small for a specified number of epochs **then**
        Prepare to trigger early stopping due to minor accuracy improvement.
    end if
    Check if the decrease in validation accuracy exceeds the threshold for decrease.
    if Validation accuracy decreases significantly for a specified number of epochs **then**
        Prepare to trigger early stopping due to accuracy decline.
    end if
    if Any early stopping condition is met **then**
        Trigger early stopping and record the epoch number.
        Log the reason for stopping if in verbose mode.
    end if
end if

3.5 Selection of specific hyperparameters

In line with our network implementation, an input encoding module is integrated. In this context, the initial layer of this module is designed to transform the six raw input signals into spikes. These signals have varying dynamics. For instance, the gyroscope's x-axis and accelerometer's x-axis and y-axis have a range of about ± 20, while the other axes are within ± 2. These differences are important for setting hyperparameters, especially the neuron threshold, for each input channel. Notably, each of these signals has distinct dynamics. Referring to Figure 2.1, it is evident that the gyroscope's x-axis values and the accelerometer's x-axis and y-axis values have a distribution spanning approximately ± 20, while the other axes from both the gyroscope and accelerometer exhibit a distribution within $\approx \pm 2$. These differences are crucial in determining hyperparameters' search space related to the threshold of neuron populations for each input channel.

Specifically, we adjusted the search space threshold intervals based on signal characteristics: wider intervals for broader signal distributions, and narrower for others. This prevents excessive spikes from low thresholds and avoids no-spiking with high thresholds. for signals characterized by a broader distribution, we establish a wider interval in the search space of the neuron's threshold. Conversely, for other signals, the interval is more constrained. This approach is based on the rationale that an excessively low threshold could generate an excessive number of spikes that fail to accurately encode the signal. Moreover, we limit the search space, as having a threshold close to the distribution's maximum might result in neurons failing to spike altogether.

3.6 Model Statistics

Following the optimization of hyperparameters, the best model obtained was retrained with various seeds to extract insights into its training behavior. This phase was crucial not only for understanding the training dynamics but also for confirming the model's robustness. By introducing different seeds, we altered the initial configuration of the synaptic weights, offering a unique viewpoint on how these initial conditions impact the model's stability and performance.

This retraining with varied seeds is imperative in the machine learning field as it tests the model's sensitivity to the initial starting points, ensuring that the hyperparameter tuning has not led to a model that is overly dependent on a specific set of initial weights. Such an investigation is fundamental for affirming that the model's performance is consistent across different initializations, thus reinforcing the validity of the hyperparameter optimization results.

Ultimately, this rigorous approach to retraining serves as a stress test for the model, ensuring that it is not only optimized for performance but also exhibits

stability and reliability across a range of initial conditions, consequently enhancing its applicability and reliability in real-world scenarios.

3.7 Model Compression

In this section, we describe the process of model compression applied to the pre-trained model, utilizing two techniques described before, pruning and quantization.

3.7.1 Granular magnitude pruning

Granular Magnitude Pruning is a model compression technique that focuses on reducing the number of parameters in a neural network by systematically pruning less significant weights. This method leverages the observation that many weights in a trained neural network are close to zero and have minimal impact on the overall network performance. By removing these negligible weights, the model can achieve a significant reduction in size and computational complexity.

Steps of Granular Magnitude Pruning (GMP)

The GMP process involves systematically pruning weights from the model based on their magnitude. Below are the key mathematical steps corresponding to the implementation.

1. **Sensitivity Scan**
   The sensitivity scan measures how sensitive each layer in the model is to pruning by gradually increasing the sparsity of its weights. Let $\mathbf{W}_l$ denote the set of weights in layer l and $\mathbf{S}_l$ the corresponding sparsity level.

   - Define a range of sparsity levels:

     $$\mathbf{S} = \{s_1, s_2, \ldots, s_n\}, \quad s_1 = 0.1,\ s_n = 1.0,\ \Delta s = 0.05$$

     where s_i represents the sparsity level (fraction of weights to prune) for layer l.
   - For each layer l, scan through the sparsity values s_i and evaluate the validation accuracy $\mathcal{A}_l(s_i)$ after pruning to s_i sparsity. Choose the sparsity level s_l^* that results in minimal accuracy degradation:

     $$s_l^* = \arg\max_{s_i} \left(\mathcal{A}_l(s_i)\right) \quad \text{such that } |\mathcal{A}_l(s_i) - \mathcal{A}_{\text{dense}}| \leq \delta$$

     where $\mathcal{A}_{\text{dense}}$ is the accuracy of the unpruned model and δ is the allowed accuracy degradation.

2. **Fine-Grained Pruning for Each Layer**
   For each layer, l, given the chosen sparsity level s_l^*, performs magnitude-based pruning.

   - Let $\mathbf{W}_l = \{w_1, w_2, \ldots, w_m\}$ represent the weights in layer l.
   - The sparsity s_l^* indicates the fraction of weights to be set to zero. Compute the number of weights to prune:
     $$n_{\text{zeros}} = \lfloor s_l^* \cdot m \rfloor$$
     where $m = |\mathbf{W}_l|$ is the total number of weights in layer l.
   - Define the importance of each weight as its absolute value:
     $$\text{importance}(w_i) = |w_i| \quad \forall w_i \in \mathbf{W}_l$$
   - Determine the pruning threshold τ_l by selecting the n_{zeros}-th smallest weight in $\mathbf{W}_l$:
     $$\tau_l = \text{k-thvalue}(\mathbf{W}_l, n_{\text{zeros}})$$
   - Create a binary mask $\mathbf{M}_l$ where weights below the threshold are pruned (set to zero):
     $$\mathbf{M}_l = \{m_i = 1 \text{ if } |w_i| > \tau_l,\ 0 \text{ otherwise } \}$$
   - Apply the mask to the weights:
     $$\mathbf{W}_l' = \mathbf{W}_l \odot \mathbf{M}_l$$
     where $\odot$ denotes element-wise multiplication.

3. **Group-wise Pruning Across Layers**
   Perform this pruning operation for each layer in the model. The sparsity level for each layer is determined from the sensitivity scan, and the pruning process is applied independently to each layer's weight tensor.

   $$\mathbf{W}_l' = \text{Prune}(\mathbf{W}_l, s_l^*) \quad \forall l$$
   where Prune is the fine-grained pruning function defined above.

4. **Fine-Tuning the Pruned Model**
   After pruning, retrain (fine-tune) the model with the pruned weights $\mathbf{W}_l'$ to regain any potential accuracy loss.

   The fine-tuning step minimizes the loss function $\mathcal{L}$ over the remaining weights in the pruned model:
   $$\min_{\mathbf{W}'} \mathcal{L}(\mathbf{W}', \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i, \mathbf{W}'), y_i)$$
   where $\mathcal{D} = \{(x_i, y_i)\}$ is the dataset, $f(x_i, \mathbf{W}')$ is the model's prediction with the pruned weights, and ℓ is the loss function.

Fine-tuning Strategy

- **Learning Rate:** The learning rate was set to 10% of the original learning rate. This adjustment allowed for more sensitive weight updates during the fine-tuning process post-pruning, facilitating a more effective adaptation of the model's parameters.
- **Number of Epochs:** The number of training epochs was reduced to 10% of the original epochs. This reduction aimed to expedite the retraining process while still allowing the model to adjust its weights adequately after pruning. The shorter training duration was deemed sufficient to regain any potential accuracy losses incurred during pruning.

This GMP strategy not only aids in achieving a more compact model but also enhances the overall efficiency of the pruning process, making it suitable for deployment in resource-constrained environments.

3.7.2 Quantization

As previously highlighted in Section 2.7.2, quantization can be applied to different components of a spiking neural network model. Specifically, Dynamic Quantization (DQ) usually quantizes only the synaptic weights leaving the neurons' state unaffected. In contrast, Post-training Quantization (PTQ) and Quantization-Aware Training (QAT) target both the synaptic weights and the neuron states, offering a more comprehensive approach in terms of reducing the precision of the entire network model.

Dynamic Quantization

Dynamic quantization converts model weights from floating-point to integer during inference. This technique applies quantization dynamically, meaning the conversion happens on the fly as the model runs. It is straightforward to implement since it does not require changes to the training pipeline. Dynamic quantization provides a significant boost in inference speed with minimal impact on accuracy, making it suitable for deployment on CPUs where computational resources are limited.

PTQ

Post-Training Quantization (PTQ) involves quantizing a model's weights and activations from floating-point to a lower precision, such as int8, after the model has been trained. This method does not require access to the original training data and can be applied to a wide range of pre-trained models. While PTQ is easier to implement and offers reductions in model size and latency, it may result in some accuracy loss, particularly for models that are sensitive to quantization.

QAT

Quantization-Aware Training (QAT) integrates quantization into the training process by simulating the effects of quantization during forward and backward passes. This approach allows the model to adapt to quantization and learn to mitigate its adverse effects, resulting in higher accuracy compared to PTQ. QAT requires modifying the training pipeline and typically demands more computational resources. It is particularly beneficial for models where precision is critical and quantization-induced accuracy loss must be minimized.

QAT Strategy

- **Learning Rate:** The learning rate was set to 1% of the original value. This adjustment aimed to allow more gradual and precise updates to the model's weights during the quantization-aware training process, ensuring the model adapts well to the quantization effects.

- **Number of Epochs:** The number of training epochs was reduced to 10% of the original epochs such as the fine-tuning strategy after pruning.

3.8 Deployment on hardware

Given the poor availability of neuromorphic boards to use with event-based sensors or to interface with other traditional hardware, and taking into account on the other hand the enormous interest for applications of on-edge models to work with data easily collected through a variety of sensors, we focused on the deployment of our encoding-free neuromorphic model on conventional edge devices. Specifically, we identified three different hardware boards on which to deploy our trained model. These boards are commercially available and embed different ARM-based microprocessor units (MPU) oriented to edge applications.

The first is the ST Microelectronics STM32MP157F-DK2 board shown in Figure 3.7, based on the ST STM32MP157F MPU, which is composed of two ARM-A7 cores running at 800 MHz and featuring a 32-bit architecture, and accompanied by 500 MB of DDR3 RAM.

The second is a Raspberry Pi 3B+, shown in Figure 3.8, with a Broadcom BCM2837B0 MPU (four ARM-A53 cores, 1.4 GHz, 64-bit) and 1 GB of DDR2 RAM.

The last one is a Raspberry Pi 4B, shown in Figure 3.9, embedding a Broadcom BCM2711 (four ARM-A72 cores, 1.8 GHz, 64-bit) and 4 GB of DDR4 RAM.

From the software standpoint, all the boards can count on a full Linux distribution: OpenSTLinux for the ST Microelectronics one, based on the OpenEmbedded project, and Raspbian for the Raspberry Pi ones, based on Debian. This ensures

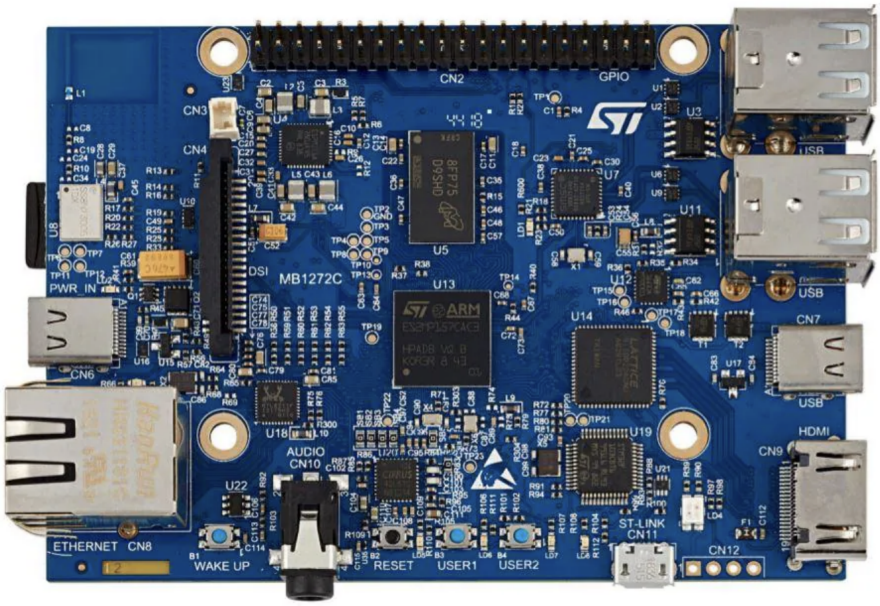

Figure 3.7: STM32MP157F-DK2 board.

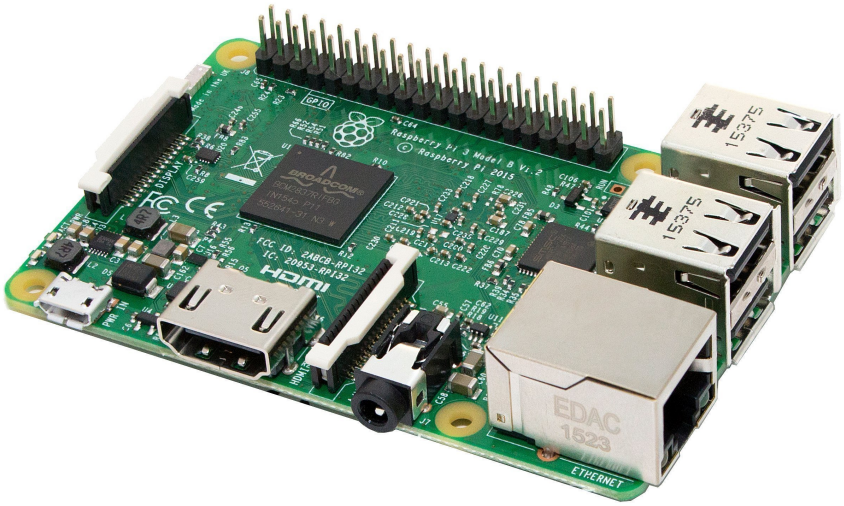

Figure 3.8: Raspberry Pi 3B+ board.

great flexibility, as Python and different AI engines are available and easy to install on them.

The first step needed to implement our model on these boards is the conversion of the models built in `snnTorch` into an ONNX model. This is necessary because the employed framework, based on PyTorch, is not available for 32-bit ARM processors,

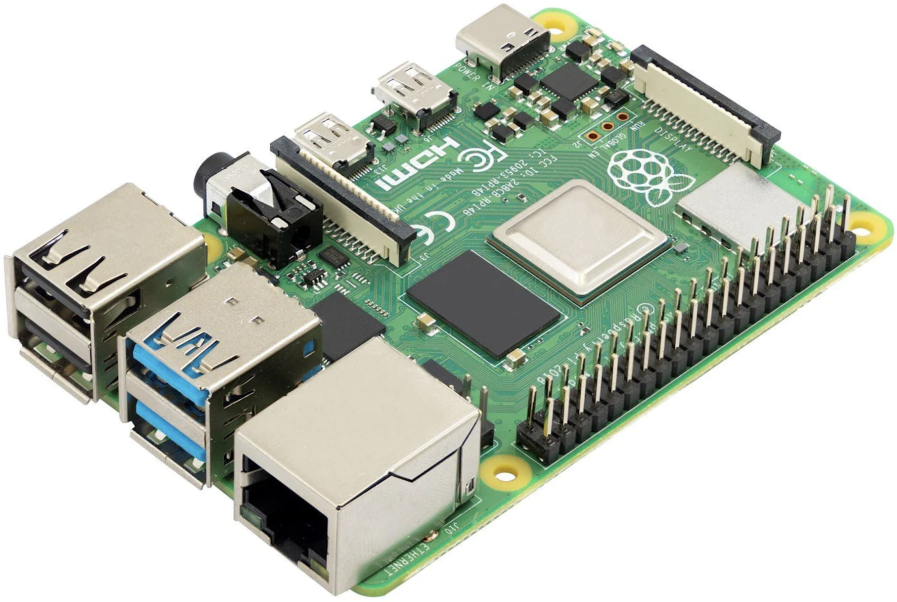

Figure 3.9: Raspberry Pi 4B board.

while the ONNX interpreter is. In addition, the ONNX engine is proven to perform better for CPU-based inferences [51]. Once the trained model is exported, inference can be performed on the target hardware using Python and the ONNX Runtime library.

Chapter 4

Results and discussion

We defined two different neuron-based models, one made by leaky neurons and the other by synaptic neurons, to classify the 7 classes of the selected dataset. For each model, we conducted many experiments and consistently obtained high levels of accuracy during the training, validation, and test phases.

4.1 Baseline

To evaluate the performance of spiking neural networks (SNNs), we first established a baseline using traditional artificial neural networks (ANNs). We adopted recurrent neural networks such as LSTM, LMU, and RNN due to their excellence in solving time-series-related tasks. As outlined in Section 2.3.4, the focus is primarily on the LMU, due to its foundation in state-space modeling, which offers an efficient method for capturing temporal dynamics.

Table 4.1: Baseline Recurrent Artificial Neural Networks Models

Model	Accuracy (%)		
	Median	Std. Dev.	Max.
`LSTM`	95.98	0.622	96.20
`LMU`	94.77	1.018	95.65
`RNN`	94.45	0.442	94.89

Through the experiments outlined in Table 4.1, we evaluated the performance of these architectures on the HAR tasks, focusing on the model accuracy. These baseline results, particularly the LMU, will serve as a reference point for assessing the potential advantages of spiking neural networks.

Table 4.2: LMU Model Metrics

Tot. Params	Conn. Sparsity	Act. Sparsity	Synaptic Operations		
			MACs	ACs	Dense
748×10^5	0.0	0.0	30.2×10^6	0.0	30.6×10^6

4.1.1 LMU insights

Table 4.2 presents key metrics for the LMU network, providing a detailed view of its structural and computational characteristics. These insights form the baseline against which we will compare LIF-based LMUs, both with and without additional encoding modules, to evaluate their performance and efficiency.

One significant observation from the table is that the connection sparsity is 0.0, indicating that the network is fully connected, with no zero-valued weights. This is typical for traditional artificial neural network (ANN) architectures, where all weights contribute to the model's operations. As a result, the LMU model performs a relevant number of synaptic operations, specifically 30.2×10^6 multiply-accumulate (MAC) operations. These operations are essential for processing input data and updating the network's internal state. Nevertheless, they can be computationally expensive due to the absence of sparsity, which could otherwise reduce the number of active computations.

In this fully connected architecture, both activation sparsity and connection sparsity are absent. This means that every artificial neuron in the network is active during the computation process, leading to a high computational load. While this is characteristic of many conventional ANN models, it becomes particularly relevant when benchmarking against spiking neural networks (SNNs), where the introduction of sparsity and event-driven computations can drastically reduce the number of active synapses and overall power consumption.

Furthermore, the Dense operations metrics (30.6×10^6) highlights the total synaptic workload of the LMU, accounting for the dense matrix multiplications that drive the temporal processing capabilities of the model. For neuromorphic architectures, reducing these operations through sparsity or event-driven computation could lead to more efficient processing, which is crucial for real-time applications.

By comparing these results to LIF-based LMUs, we can explore how neuromorphic principles like sparse connectivity and sparse activations can enhance the efficiency of temporal processing networks while maintaining or even improving performance. This analysis will provide a deeper understanding of how brain-inspired models such as LMUs can benefit from transitioning to spiking paradigms in neuromorphic computing.

4.2 L^2MU

This section presents the results obtained by the L^2MU with both Leaky (Table 4.3) and Synaptic (Table 4.4) neuron models. In this case, raw data are directly fed to the model without any encoding module. This also serves as a second baseline for comparison with models that incorporate signal encoding mechanisms.

From now on, we refer to the original model as the one trained with full precision following the HPO experiment. The pruned model, instead, refers to the model obtained by applying the GMP technique for weight compression. There are also different quantized models: models that have been quantized starting from the original model (quantized from original) and model quantized starting from the pruned model (quantized from GMP)

4.2.1 Leaky

The L^2MU with Leaky neurons demonstrates strong overall performance, as shown in Table 4.3. The key metrics taken into consideration for the analysis include model accuracy, memory footprint, sparsity, and synaptic operations.

Accuracy The original model achieves a high accuracy (95.63%), slightly improved by GMP pruning (95.87%). Quantization introduces a small accuracy drop when starting from the original model for QAT (93.66%) and PTQ (93.34%). However, Dynamic Quantization (DQ), nearly restores the original accuracy (95.18%). When the quantization starts from the pruned model, the accuracy remains quite stable with DQ (95.72%) and QAT (94.45%), while PTQ sees a more significant reduction (89.95%). Of course, in the case of Dynamic Quantization, only the weights are quantized and the neuron's internal states are in full precision.

Model Size The model parameters (593×10^3) are constant across all the different compressed models. For both the original and pruned models, the memory footprint (2.38 MB) remains unchanged despite the increased sparsity in the model weights, due to the injected zeros. This is because the model is stored in a dense format, where all weight values, including the zeros, are saved without any specific compression to take advantage of the sparsity. Evolving from the original 32-bit precision to 8-bit reduces the model size to 0.61 MB, making this reduction crucial for resource-constrained hardware.

Sparsity Connection sparsity increases significantly after GMP pruning (from 0.26 to 0.85 for the 32-bit precision model). Quantization applied to the original model does not have a significant impact on the connection sparsity. In contrast, when quantization is performed starting from the pruned model, it fully leverages

all the benefits of the pruning process. Activation sparsity remains high, around 0.92–0.93, meaning that only 7-8% of the neurons are active.

Synaptic and Stateful Operations A major benefit of quantization is the reduction in computational load. The original model requires 101×10^3 MACs and 961×10^3 ACs. Post-quantization, these figures drop dramatically, with DQ showing the lowest computational requirements (as low as 12×10^3 MACs and 31×10^3 ACs) for the GMP-quantized model, while for the original-quantized model, the drop remains constant for all the quantization methods (17×10^3 MACs and 39×10^3 ACs)

Another drastic reduction characterizes the dense operations, which are reduced by a factor of 171.43 in the quantized models.

Membrane updates, representing the stateful neuron operations, are affected by a substantial reduction in the quantized models, especially QAT and PTQ, while DQ maintains the same average updates, coherently with the original and pruned model.

Table 4.3: L^{2}MU (Leaky) Performance

Metrics	**Orig.**	**GMP**	**Quantized**					
			From Original			**From GMP**		
			QAT	**PTQ**	**DQ**	**QAT**	**PTQ**	**DQ**
Accuracy (%)	95.53	95.87	93.66	93.34	95.18	94.45	89.95	95.72
Params ($\times 10^3$)	593							
Precision (bit)	32		8					
Footprint (MB)	2.38		0.62		0.61	0.62		0.61
Conn. Sparsity	0.26	0.85	0.28			0.85		
Act. Sparsity	0.92	0.93	0.92			0.93		
M. Upd. ($\times 10^3$)	45	52	28	25	45	36	26	52
MACs ($\times 10^3$)	101	62	17			12		13
ACs ($\times 10^3$)	961	560	39			31	32	33
Dense ($\times 10^6$)	24		0.14					

4.2.2 Synaptic

Accuracy The original synaptic model (95.18%) performs similarly to the Leaky version. After GMP pruning, the accuracy remains almost unchanged (95.70%). However, quantization starting from the original model shows a more substantial drop in accuracy, particularly in PTQ (85.35%), though DQ performs better

Table 4.4: L^2MU (Synaptic) Performance

Metrics	Orig.	GMP	Quantized					
			From Original			From GMP		
			QAT	PTQ	DQ	QAT	PTQ	DQ
Accuracy (%)	95.18	95.70	94.02	85.35	94.78	93.97	87.60	94.96
Params ($\times 10^3$)	845							
Precision (bit)	32		8					
Footprint (MB)	3.38		0.87		0.86	0.87		0.86
Conn. Sparsity	0.37	0.81	0.45			0.81		
Act. Sparsity	0.93	0.93	0.93			0.93		
Mem. Upd. ($\times 10^3$)	51	50	31	15	50	21	18	50
MACs ($\times 10^3$)	1053	89	6	18	18	18		
ACs ($\times 10^3$)	1596	1131	73	2	71	69		
Dense ($\times 10^6$)	34		0.21	0.18	0.19	0.19		

(94.78%). Quantizing from GMP yields a somewhat better balance, with DQ maintaining accuracy at 94.96%.

Model Size The synaptic model has a larger parameter count (845×10^3) compared to the Leaky model, and its footprint is larger in the original state (3.38MB). However, after quantization, both models converge to similar footprints (around 0.86MB), still higher than the Leaky counterpart.

Sparsity GMP pruning drastically improves connection sparsity for the synaptic model (0.81 from an original 0.37). Activation sparsity, again, remains high, close to 0.93.

Synaptic and Stateful Operations The Synaptic model is initially much more computationally intensive, requiring 1053×10^3 MACs and 1596×10^3 ACs in the original state. However, quantization drastically reduces these figures, especially in the GMP-quantized models, where the MAC count drops to 18×10^3, and ACs reduce to 69×10^3.

Again, the dense operations are subject to a drastic reduction when the model is quantized (from 34×10^6 to 0.19×10^6)

Membrane updates see a substantial reduction after quantization, especially in the GMP-quantized models (as low as 15).

4.2.3 Leaky vs. Synaptic Model

Accuracy The Synaptic model has a slightly lower performance in terms of accuracy compared to the Leaky one for the original and pruned model, but its accuracy drops more significantly after quantization, particularly for PTQ. In contrast, the Leaky model remains more stable after quantization, particularly when dynamic quantization is applied. This means that The Leaky model could be more suitable to be deployed on neuromorphic hardware since all the states of the neurons are required to be on a lower precision.

Model Size and Sparsity The Synaptic model has a larger number of parameters (845×10^3 vs. 593×10^3 for the Leaky model) and a correspondingly larger footprint in the original state. However, both models achieve similar sizes after quantization. GMP pruning results in a higher connection sparsity for the Synaptic model, which may be beneficial for certain neuromorphic architectures.

Computational Load The Leaky model is more computationally efficient, requiring fewer MACs and ACs compared to the Synaptic model. Quantization drastically reduces the computational load for both models, but the Synaptic model remains more demanding in general. Additionally, the number of membrane updates is lower for the Synaptic model after quantization, which may be better for hardware where that type of operation requires more time to be performed (retrieve, check, and set are the low-level operations executed by the hardware to possibly update the value of the cell memory unit where the value is stored).

Efficiency vs. Performance Trade-offs The Leaky model offers a better balance between efficiency and performance, especially in quantized forms, making it a good choice for environments with limited computational resources. The Synaptic model is more complex in terms of operation and footprint and does not present the advantages shown by the Leaky one.

4.2.4 L^2MU vs LMU

The LMU and L^2MU represent two different paradigms for time-series classification tasks, with the LMU being the baseline for comparison. Table 4.2 and 4.1 provide insightful metrics for the baseline LMU.

Accuracy In terms of accuracy, the LMU achieves a median accuracy of 94.77%, with a maximum of 95.65%. Both the L^2MU Leaky and Synaptic models achieve competitive accuracies, with the Leaky model performing slightly better after dynamic quantization (95.18%) compared to the Synaptic model (94.96%). This demonstrates that despite the added complexity of spiking dynamics, the spiking

models can match or even surpass the performance of the LMU, especially when fine-tuned with GMP pruning and Dynamic Quantization.

Model Size and Sparsity The LMU has 748×10^3 parameters and exhibits no connection or activation sparsity (0.0). In contrast, both L^2MU models, especially the Leaky variant, benefit significantly from sparsity, with connection sparsity reaching up to 0.85 and activation sparsity of 0.93 after pruning. This sparsity enables more efficient computation, particularly in energy-constrained environments, making the L^2MU models more attractive for neuromorphic systems.

Computational Efficiency The LMU requires 30.2×10^6 MACs for synaptic operations and 0 ACs due to nonbinary activations. By comparison, before quantization, the L^2MU models (particularly the Leaky variant) require significantly fewer MACs and more ACs (more binary activations due to spike-trains data). For example, the L^2MU Leaky model operates with as few as 101×10^3 MACs and 961×10^3 ACs, the sum of these operations equal to 1.062×10^6, is still drastically lower compared to the one given by the LMU. This drastic reduction in operations for the spiking variants showcases the potential computational savings offered by the spiking architecture, especially after GMP pruning and quantization.

A key distinction between the LMU and the L^2MU models is the concept of membrane updates. The LMU, being a non-neuromorphic model, does not require these updates, while both LIF-based models rely on membrane updates to maintain their stateful operations. This introduces an additional computational overhead for the L^2MU models.

4.3 Encoded L^2MU

As with the L^2MU, the results for this section are presented in the same manner. The model under consideration is an L^2MU which features a single-layer encoding mechanism described in section 3.1.1. This model takes the name of Encoded L^2MU. Refer to Table 4.5 and 4.6, for a comprehensive overview of the performance metrics associated with both the Leaky and Synaptic versions of the Encoded L^2MU.

4.3.1 Leaky

Accuracy The original model achieves an accuracy of 94.77%, with a slight drop to 94.58% following GMP pruning. When applying QAT and PTQ, the accuracy decreases further to $\sim 90\%$ and $\sim 62\%$, respectively. Dynamic Quantization (DQ), however, recovers much of the lost accuracy, reaching 95.18%, demonstrating a minimal trade-off between quantization and accuracy.

Model Size One of the most prominent benefits of quantization is the reduction in model size. The original Leaky model, with 2.14 MB, shrinks to 0.58 MB post-quantization. The number of parameters constituting the model is equal to 533×10^3.

Sparsity The GMP pruning method improves the connection sparsity from 0.29 to 0.86, leading to a more efficient model without significantly impacting performance. Activation sparsity remains stable at around 0.92, indicating the model maintains efficiency in neural activations even after pruning.

Synaptic and Stateful Operations Quantization also brings considerable computational savings. The original model requires 12×10^3 MACs and 1291×10^3 (ACs). After quantization, ACs values are dramatically reduced, while MACs are the same as the original model when applying DQ and lower in all the other quantization cases. Such reductions make the model highly suitable for deployment in scenarios where computational resources are limited.

In terms of neuron dynamics, membrane updates decrease substantially in the quantized versions. For QAT and PTQ, updates drop (25-28) compared to the original model (48), reflecting advantages in computational efficiency.

Table 4.5: Encoded L^2MU (Leaky) Performance

Metrics	**Orig.**	**GMP**	**Quantized**					
			From Original			**From GMP**		
			QAT	**PTQ**	**DQ**	**QAT**	**PTQ**	**DQ**
Accuracy (%)	94.77	94.58	91.49	62.53	93.96	90.62	62.23	92.79
Params ($\times 10^3$)	533							
Precision (bit)	32		8					
Footprint (MB)	2.14		0.58		0.57	0.58		0.57
Conn. Sparsity	0.29	0.86	0.30			0.86		
Act. Sparsity	0.92	0.93	0.92			0.94		
Mem. Upd.($\times 10^3$)	48	42	30	25	47	28	26	41
MACs ($\times 10^3$)	12	0.88	0.98		12	7.4		8.8
ACs ($\times 10^3$)	1291	464	42			26	27	26
Dense ($\times 10^6$)	22		0.14					

4.3.2 Synaptic

Accuracy The Synaptic model starts with an original accuracy of 94.14%, which decreases to 93.61% after GMP pruning. Quantization impacts accuracy significantly, with QAT showing an accuracy of 83.70% and PTQ dropping to 71.84%. However, DQ performs relatively well, achieving 93.77% from the original model and 93.19% from the GMP model.

Model Size The original Synaptic model has a larger footprint of 3.09 MB, reflecting its higher parameter count of 770×10^3. Following quantization, the footprint decreases to 0.81 MB, comparable to the Leaky model, thus ensuring that both models remain suitable for deployment in resource-constrained scenarios.

Sparsity GMP pruning significantly enhances connection sparsity in the Synaptic model from 0.41 to 0.91. The activation sparsity remains consistently high at around 0.93, ensuring efficient parameter usage.

Synaptic and Stateful Operations In terms of computational load, the Synaptic model is more demanding than the Leaky model, requiring 7×10^6 MACs in the original state. However, post-quantization, it benefits from reduced computational requirements, indicating a more favorable balance for efficient operations in neuromorphic systems.

The Synaptic model shows a reduction in membrane updates per timestep, from 50×10^3 in the original state to 28×10^3 with QAT and 27×10^3 with PTQ. This reduction indicates improved efficiency in the stateful processing of neuronal activities.

4.3.3 Leaky vs Synaptic

Accuracy The Synaptic model displays slightly higher accuracy in the original and GMP states but suffers more significant accuracy drops post-quantization, especially with PTQ. The Leaky model remains more stable after quantization, particularly with DQ.

Model Size and Sparsity The Synaptic model has a greater number of parameters (770k vs. 533k for the Leaky model), resulting in a larger original footprint. However, both models converge to similar sizes after quantization. The Synaptic model achieves higher connection sparsity post-pruning.

Computational Load The Leaky model proves to be more computationally efficient, requiring fewer MACs and accumulations than the Synaptic model. Quantization greatly reduces the computational demands for both models, although the

Table 4.6: Encoded L^{2}MU (Synaptic) Performance

Metrics	Orig.	GMP	Quantized					
			From Original			From GMP		
			QAT	PTQ	DQ	QAT	PTQ	DQ
Accuracy (%)	94.14	93.61	83.70	71.84	93.77	82.65	65.50	93.19
Params ($\times 10^3$)	770							
Precision (bit)	32		8					
Footprint (MB)	3.09		0.82		0.81	0.82		0.81
Conn. Sparsity	0.41	0.91	0.43			0.91		
Act. Sparsity	0.93	0.95	0.93			0.95		
Mem. Upd.($\times 10^3$)	50	57	28	27	51	32	31	57
MACs ($\times 10^3$)	7	5	6		7	4		5
ACs ($\times 10^3$)	849	306	62			42	44	43
Dense ($\times 10^6$)	31		0.18					

Synaptic model generally remains more intensive. Additionally, the number of membrane updates is lower in the Synaptic model post-quantization.

Efficiency vs. Performance Trade-offs. The Encoded L^{2}MU model with Leaky neurons presents a more favorable balance between efficiency and performance, particularly in quantized forms.

4.3.4 Encoded L^{2}MU vs L^{2}MU

What stands out from the Encoded L^{2}MU is the reduction of the number of parameters, which causes a small degradation in terms of accuracy. Even though the model is more complex due to the presence of an encoder, the model size (parameters) has decreased, consequently decreasing the model footprint. In terms of Synaptic Operation for the pruned model the Encoded L^{2}MU has better performances compared to the L^{2}MU for both the Leaky and Synaptic version, this is also reflected in the GMP-quantized version.

Overall the Encoded L^{2}MU seems to exceed the expectation, particularly with the Leaky variant where there is not a substantial degradation of the accuracy.

This version proves to be well-suited for scenarios requiring a model that is not only efficient in its operations but also capable of handling resource-constrained environments. The Leaky variant, in particular, strikes an excellent balance between maintaining high accuracy and minimizing computational demands, power consumption, and memory usage. Its lower operational complexity compared to the L^{2}MU makes it a highly suitable option for edge devices where both performance

and resource efficiency are critical.

4.4 Multi-Encoded L^2MU

The model under evaluation in this section is a Multi-Encoded L^2MU, which employs a stacked encoding mechanism outlined in Section 3.1.2.

For a detailed overview of the performance metrics associated with both the Leaky and Synaptic variants of the Multi-Encoded L^2MU, please refer to Tables 4.7 and 4.8.

4.4.1 Leaky

Accuracy The original model accuracy (94.14%) is subjected to degradation in the compressed models. A slight decrement is observed in the pruned model, but whenever the quantization is applied (starting from the original or the pruned model) there is a significant degradation with QAT and PTQ, while with DQ the accuracy almost remains unchanged

Model Size At 1.09 MB for the original model and just 269×10^3 parameters, the model is reduced to as low as 0.31 MB after quantization.

Sparsity The connection sparsity of the original model (0.25) is increased through the pruning process (0.85) and the same performance effect also the quantization. Particularly advantageous is the model where the quantization started from the pruned.

Activation sparsity with 0.85 on the original model displays some improvement in the quantized models, showing that the active neurons are consistent across the original and compressed models.

Synaptic and Stateful Operations The original model shows 7×10^3 MACs, which remains consistent in most quantized versions. However, ACs drop from 1629×10^3 to as low as 50×10^3 after quantization. The dense operations, reduced from 11 million to just 0.12 million after quantization, highlight a drastic reduction in complexity post-compression, making the model far more efficient. Membrane Updates are subjected to reduction during QAT and PTQ while they are almost the same when applying DQ.

4.4.2 Synaptic

Accuracy Slightly better in certain configurations than the Leaky model, with the highest accuracy of 93.95% achieved through GMP on the original model.

Table 4.7: Multi-Encoded L^{2}MU (Leaky) Performance

Metrics	**Orig.**	**GMP**	**Quantized**					
			From Original			**From GMP**		
			QAT	**PTQ**	**DQ**	**QAT**	**PTQ**	**DQ**
Accuracy (%)	94.14	93.70	80.64	69.57	93.34	77.36	69.52	93.23
Params ($\times 10^3$)	269							
Precision (bit)	32		8					
Footprint (MB)	1.09		0.33		0.31	0.33		0.31
Conn. Sparsity	0.25	0.52	0.29			0.53		
Act. Sparsity	0.84	0.85	0.87	0.85	0.84	0.87	0.86	0.85
Mem.Upd.($\times 10^3$)	33	32	20	18	33	18	17	32
MACs ($\times 10^3$)	7	6	6		7	5		6
ACs ($\times 10^3$)	1629	1189	51	59	57	50	51	50
Dense ($\times 10^6$)	11		0.12					

Table 4.8: Multi-Encoded L^{2}MU (Synaptic) Performance

Metrics	**Orig.**	**GMP**	**Quantized**					
			From Original			**From GMP**		
			QAT	**PTQ**	**DQ**	**QAT**	**PTQ**	**DQ**
Accuracy (%)	93.48	93.95	87.28	79.6	93.12	83.96	78.52	92.72
Params ($\times 10^3$)	868							
Precision (bit)	32		8					
Footprint (MB)	3.5		0.93		0.91	0.93		0.91
Conn. Sparsity	0.18	0.80	0.25			0.80		
Act. Sparsity	0.94	0.95	0.94			0.95		
Mem. Upd.($\times 10^3$)	49	45	31	32	49	30	31	44
MACs ($\times 10^3$)	7	6	6		7	5		6
ACs ($\times 10^3$)	1659	629	73	73	74	63	62	64
Dense ($\times 10^6$)	35		0.21					

Quantization from the original model results in high accuracy drops with QAT and PTQ.

Model Size The original Synaptic model has a significantly higher memory footprint at 3.5 MB, the highest among all the analyzed models and the largest number of parameters (868×10^3). However, after quantization, it compresses down to 0.91 MB.

Sparsity The connection sparsity of the original model (0.18) increases substantially with GMP, reaching 0.80. Quantization from the original model leads to a connection sparsity of 0.25, while from GMP, the model retains the high sparsity of 0.80

The Synaptic model starts with higher activation sparsity at 0.94. This sparsity level remains almost unchanged across all quantized versions, peaking at 0.95.

Synaptic and Stateful Operations The original Synaptic model is heavier, with 7×10^3 MACs and 1559×10^3 activation counts (ACs), but compression techniques reduce ACs to as low as 62×10^3, maintaining some computational advantages while lowering complexity. Dense operations see a reduction from 35 million to just 0.21 million post-quantization, which is substantial but still more than the Leaky version. Again, the same behavior is observed for membrane updates, where there is a reduction only when applying QAT and PTQ.

4.4.3 Leaky vs. Synaptic

Accuracy The Leaky model achieves slightly lower performance in terms of accuracy across the quantized variants, particularly when compared to the Synaptic model.

Model Size and Sparsity The Leaky model is characterized by its smaller memory footprint. The model size for the Leaky variant is significantly smaller than the Synaptic variant, which reflects the trade-off between simplicity and biological realism. In terms of sparsity, both models benefit from pruning techniques, but the Synaptic model generally exhibits greater sparsity. This allows the Synaptic model to achieve higher connection sparsity after pruning.

Computational Load The Leaky model requires fewer operations overall, which translates into lower power consumption and faster inference times. This makes it a more attractive option for real-time applications where latency and energy efficiency are paramount. The Synaptic model, while providing richer and more accurate temporal representations, demands significantly more operations.

Efficiency vs. Performance Trade-offs. The Leaky model provides a highly efficient option in terms of memory usage and computational cost, making it ideal for lightweight hardware. However, this efficiency comes at the expense of reduced accuracy, especially for quantized models. The Synaptic model, while demanding more resources, offers improved accuracy and is more suited for complex tasks where performance is a priority.

4.4.4 Multi-Encode L^{2}MU vs. Encoded L^{2}MU

The comparison between the Multi-Encoded L^{2}MU and the Encoded L^{2}MU reveals clear differences in terms of complexity, memory footprint, and operational efficiency. Both models use the L2MU as a recursive network but differ in their approach to encoding layers.

The most surprising result is the Multi-Encode L^{2}MU model with Leaky neurons footprint, compared to all the others. Even though it is constituted by a different number of neuron encoding layers, it surpasses the Synaptic version and the Encoded L^{2}MU. Of course, this comes with a noticeable cost of accuracy drop for the quantized model (QAT and PTQ)

The Synaptic version instead, requires a higher computational demand in terms of operations and footprint compared to the Encoded L^{2}MU, while the model accuracy is quite stable for both the architectures.

4.5 Deployment of Multi-Encoded L^{2}MU on Edge Devices

The rationale behind choosing the Multi-Encode L^{2}MU from the different models previously described is due to several reasons.

The Multi-Encoded L^{2}MU with Synaptic neurons has the highest memory footprint and computation complexity among all the other models. This choice allows us to evaluate the performance of the most resource-intensive version of the model. If this configuration can fit and run efficiently on the target hardware, it ensures that all the other models with smaller memory footprint and computational demand will also fit and perform well.

The results achieved by the deployed models on the selected conventional hardware are presented in Table 4.9, including RAM usage, load and inference times, mean power consumption, mean energy per inference, and accuracy, for both the `Leaky` and `Synaptic` neurons. We further analyzed the inference times by running the models for 2 hours on the devices: the plots in Figure 4.1 represent the distribution of probability for the inference times of the two neuron models analyzed. Data obtained with such analyses showed that both the types of LIF neuron can be deployed on traditional hardware with good results: in particular, the Multi-Encoded L^{2}MU with the `Leaky` neurons model provided 93.89% test accuracy with reasonable power consumption, inference, and load times. Also, the model with `Synaptic` neurons revealed a satisfactory behavior, but the `Leaky` model seems a better trade-off.

Running such models with success on this hardware highlights the possibility of further integrating neuromorphic and traditional computation paradigms to enhance low-power computation.

Table 4.9: Results achieved deploying the L^2MU on commercial edge devices to solve the HAR task through encoding-free classification with two different neuron models.

Device	Neuron model	Used RAM	Mean inference time	Mean power	Mean energy per inference	Accuracy (%)
STM32MP1	`Leaky`	110 MB	0.29 s	1.6 W	230 mJ	93.89
	`Synaptic`	235 MB	27.4 s		504 mJ	93.71
Raspberry Pi 3B+	`Leaky`	135 MB	34.0 s	3.4 W	337 mJ	93.89
	`Synaptic`	245 MB	0.38 s		900 mJ	93.71
Raspberry Pi 4B	`Leaky`	135 MB	9.2 s	5.0 W	151 mJ	93.89
	`Synaptic`	245 MB	0.11 s		396 mJ	93.71

Focusing on the differences among the three boards, especially on the hardware side, it is possible to see different performances using different power values, with the Raspberry Pi 4B being the fastest one, and the STM32MP1-based board being the most low-power. The choice could depend on the final application and on the rate at which the input values are generated. The probability distribution graphs can be useful to make the best choice, as inference could last longer than the average time, and the frequency at which the input port is updated should be decided accordingly. For example, the Raspberry Pi 3B+ shows a large variance under that point of view, while values for the STM32MP157F-DK2 are generally nearer to the average one.

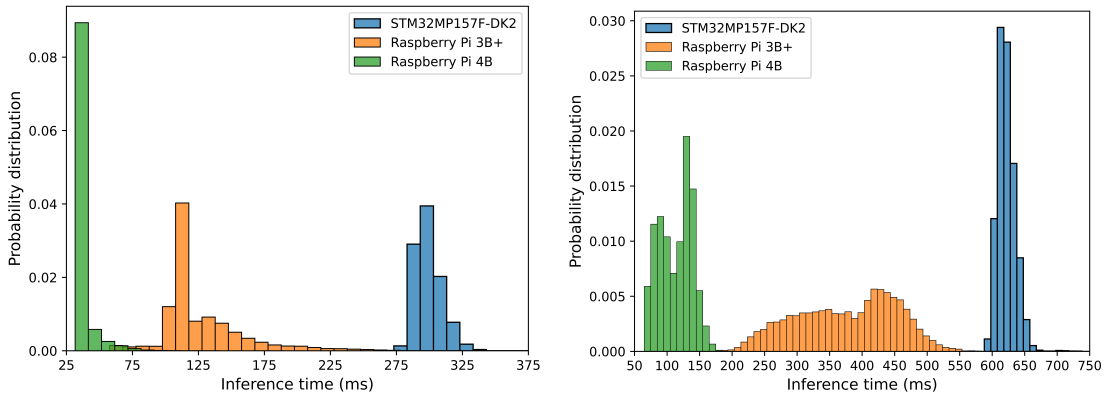

Figure 4.1: Histogram plots representing the inference time probability distribution. Analyses were carried out on the three boards running the models with either `Leaky` (left) or `Synaptic` (right) neurons.

Chapter 5

Conclusion

In this study, we chose the LMU network over various RNN architectures due to its remarkable ability to keep the constrained number of parameters while delivering performance comparable to other RNNs. This efficiency leads to a memory-efficient model, making it suitable for resource-constrained environments.

With this foundation, we converted the model into a fully spiking network referred to as L^2MU. We achieved this by adopting the LIF neurons to transform each component of the traditional LMU into a neuron population. We simultaneously designed an encoding module that aims to transform the raw signal into spikes and feed them to the L^2MU such that it can operate with discrete signals. The potential for a complete network conversion into a spiking network opens up the possibility of future deployment into neuromorphic hardware.

To evaluate our network design, we applied it to the Human Activity Recognition (HAR) classification task. The performance of the L^2MU, particularly in its leaky and synaptic variations, was thoroughly assessed against the traditional LMU. The results demonstrated competitive accuracy, with the Leaky version of the L^2MU achieving up to 95.87% accuracy in full precision, which closely rivals the performance of the more traditional model, but with a significant reduction in memory footprint and computational complexity. The Synaptic version offered a competitive performance in some configurations but at the expense of increased computational cost and memory requirements.

We also introduced both single-encoded and multi-encoded versions of the L^2MU. The multi-encoded version, which incorporates multiple encoding layers, displayed small accuracy degradation and robustness at the cost of higher memory usage and computational demands. In contrast, the single-encoded L^2MU struck a balance between performance and operational efficiency, proving that even with minimal encoding layers, the L^2MU architecture could achieve competitive results while maintaining a low memory footprint. This balance makes the encoded L^2MU well-suited for deployment in edge devices where memory and computational resources are limited.

Beyond theoretical evaluations, we deployed the full precision Multi-Encode L^2MU models on commercial edge devices, such as the STM32MP1 and Raspberry Pi boards, to solve the HAR task in a real-world, operational environment. The deployment tests were carried out for both the Leaky and Synaptic models, revealing important trade-offs between computational load, memory usage, and energy efficiency.

The results showed that the Leaky Multi-Encoded L^2MU, with its lower memory footprint, was the most efficient for deployment on edge devices. It exhibited shorter inference times, lower RAM usage, and reduced energy consumption, making it ideal for real-time applications on devices with limited hardware capabilities.

In conclusion, our fully spiking neural network (L^2MU) presents an innovative and effective approach to solving the HAR task, showing competitive accuracy while maintaining operational efficiency. The deployment of both the Leaky and Synaptic L^2MU models on commercial edge devices underscores their adaptability for real-world implementation, with the Leaky model standing out as the more efficient choice for resource-constrained applications. Furthermore, the successful transformation of the LMU into a spiking network lays the groundwork for potential future deployment on neuromorphic hardware, which could further enhance energy efficiency and scalability.

Future work could explore further optimizations of the L^2MU for specific hardware architectures and additional neuromorphic platforms. Additionally, enhancing the encoding mechanisms may lead to further performance improvements. The flexibility of the L^2MU architecture ensures that it can continue to evolve, standing the way for more widespread adoption in real-world, edge-based AI applications.

Appendix A

Conversion of RNN to LIF-based RNN

In addition to the conversion of the Legendre Memory Unit (LMU) into a LIF-based LMU, the same process was applied to a traditional Recurrent Neural Network (RNN). The reason for this additional conversion was to explore if a model that does not have its foundation in State-Space Representation, could have good performance in terms of neuromorphic efficiency.

A.1 Conversion Process of an RNN

The RNN conversion process doesn't require the conversion of any equation as it happened for the LIF-based LMU. It simply involves replacing the activation mechanism with a population of neurons, which in our case would be Leaky or Synaptic neuron population. Substituting the activation function the conversion into a spiking behavior was straightforward.

A.2 Results and Discussions

For the LIF-based RNN (L-RNN), as with the L^2MU, we tested two different neuron models (Leaky and Synaptic) and explored the use of encoding modules, including both single-layer and multi-layer encoders. In this section, we present the results of these models without applying any pruning or quantization. Consequently, all the results reported reflect the model running at full precision (32 bit).

A.2.1 L-RNN

In this configuration, the RNN has been converted into a LIF-based model, where the neurons follow the leaky integrate-and-fire dynamics. However, the input data

remains in its raw form and is not converted into spike trains.

Leaky

Table A.1: L-RNN (Leaky) Metrics

Acc.	Tot. Params	Conn. Sparsity	Act. Sparsity	Mem. Upd.	Synaptic Operations		
					MACs	ACs	Dense
94.92%	83k	0.0	0.75	280	67K	790K	3.3M

The result of L-RNN with Leaky neurons demonstrates good overall performance, achieving an accuracy of 94.92% as shown in Table A.1.

83×10^3 parameters constitute the network architecture with a 0% connection sparsity, meaning that the network maintains a fully connected architecture, which contributes to a higher number of synaptic operations (3.3×10^6 dense operations). The model activation sparsity is 0.75, suggesting 75% of the neurons are inactive during inference. This metrics suggests moderate computational savings.

This model also requires 67×10^3 multiply-accumulate operation with non-binary activation and 790×10^3 multiply-accumulate with binary activation.

Synaptic

Table A.2: L-RNN (Synaptic) Metrics

Acc.	Tot. Params	Conn. Sparsity	Act. Sparsity	Mem. Upd.	Synaptic Operations		
					MACs	ACs	Dense
95.12%	72K	0.0	0.74	270	62K	715K	2.8M

The Synaptic version of the L-RNN, presents a slightly better performance with an accuracy of 95.12%, as shown in Table A.2 and 72×10^3 parameters which are lower than the Leaky model. This model is still fully connected, as suggested by the 0.0 connection sparsity. Not a significant reduction is shown in the activation sparsity of 0.74, but this implies that fewer neurons are inactive during inference.

The synaptic operation metrics show a reduction of the synaptic model compared to its counterpart. Specifically, MACs are reduced by 5×10^3, ACs by 75×10^3, and Dense operations by 0.5×10^6. This reduction of synaptic operations, combined with the marginal improvement in accuracy, suggests that the synaptic model has better optimized performance than the Leaky one.

A.2.2 Encoded L-RNN

In this configuration, the LIF-based RNN is enhanced with an encoding neuron population layer, which transforms the input data into spike trains. This layer allows the recurrent network to operate with spike-encoded data.

Leaky

Table A.3: Encoded L-RNN (Leaky) Metrics

Acc.	Tot. Params	Conn. Sparsity	Act. Sparsity	Mem. Upd.	Synaptic Operations		
					MACs	ACs	Dense
95.41%	110K	0.0	0.93	280	7K	607K	4.4M

As we observe from Table A.3, the encoding layer increases the model complexity, leading to a total of 110×10^3 parameters which is notably higher than the one without encoding. The activation sparsity suggests that 93% of the neurons are inactive during inference, which is significantly higher than the one observed by the model without encoding. This increased activity could be attributed to the spike encoding layer, which increases the number of firing neurons.

In terms of synaptic operations MACs and ACs, respectively 7×10^3 and 607×10^3, are lower compared to the Leaky model without encoding, while the Dense operations are extremely higher, almost twice the previous value.

Synaptic

Table A.4: Encoded L-RNN (Synaptic) Metrics

Acc.	Tot. Params	Conn. Sparsity	Act. Sparsity	Mem. Upd.	Synaptic Operations		
					MACs	ACs	Dense
94.15%	103K	0.0	0.94	275	7K	595K	4.1M

The Synaptic version of the encoded L-RNN achieves slightly lower accuracy at 94.15% (Table A.4) but requires fewer parameters than the Leaky counterpart. The activation sparsity is quite similar to the Leaky version, suggesting both models activate a large proportion of neurons when working with spike-encoded data. Synaptic operations remain efficient 7×10^3 MACs and 595×10^3 ACs, except for the Dense operation, which shows a slightly better performance.

A.2.3 Multi-Encoded L-RNN

The Multi-Encoded L-RNN introduces three encoding layers made of neurons population, increasing the model complexity and ability to handle spike-encoded data.

Leaky

Table A.5: Multi-Encoded L-RNN (Leaky) Metrics

Acc.	Tot. Params	Conn. Sparsity	Act. Sparsity	Mem. Upd.	Synaptic Operations		
					MACs	ACs	Dense
94.71%	68K	0.0	0.91	280	7K	423K	2.7M

As shown in Table A.5, this architecture achieves an accuracy of 94.71%, a slight drop compared to the single encoded Leaky version. The total parameters are 68×10^3, which is lower than the L-RNN model and Encoded L-RNN in both Leaky and Synaptic neurons. This drop in parameters suggests that while the number of layers increases, the overall model complexity is managed. Activation sparsity is still high with 91% of neurons inactive.

The model maintains 7×10^3 MACs and lower ACs and Dense operations compared to the previous models. These results suggest that the introduction of a multi-layer encoding does not drastically increase the computational requirements, making the model relatively efficient.

Synaptic

Table A.6: Multi-Encoded L-RNN (Synaptic) Metrics

Acc.	Tot. Params	Conn. Sparsity	Act. Sparsity	Mem. Upd.	Synaptic Operations		
					MACs	ACs	Dense
94.48%	44K	0.0	0.81	15.7K	7K	382K	1.7M

The Synaptic version of the Multi-Encoded L-RNN performs slightly worse in term of accuracy 94.48% (Table A.6) that the Leaky counterpart, but it has an extreme reduction of parameters, 44×10^3, the lower compared to all the different L-RNN versions.

Activation sparsity of 0.81, lower than The Leaky model, suggest fewer inactive neurons during inference, which may contribute to increased efficiency in term of synaptic operations.

The model performs the same in term of MACs, compared to the Leaky counterpart and there is a reduction in terms of ACs and Dense operations, respectively 328×10^3 and 1.7×10^6, which are the lowest among all the L-RNN model versions.

Appendix B

Learning Curves

This chapter presents a series of training statistics plots that illustrate the learning behavior and performance metrics of various neural network models after optimization. Each figure depicts learning curves, including accuracy and loss metrics, of models that have been retrained with different seeds. The plots provide insights into the effectiveness of the training process, highlighting mean values and standard deviation, illustrated by solid lines and shaded areas respectively, across training epochs.

Figures are organized by model type, with each section dedicated to a specific neural network architecture. The left side of each figure illustrates the accuracy development over time, while the right side focuses on the loss metrics. This arrangement allows for a comparative analysis of model behavior under different training dynamics.

B.1 LSTM

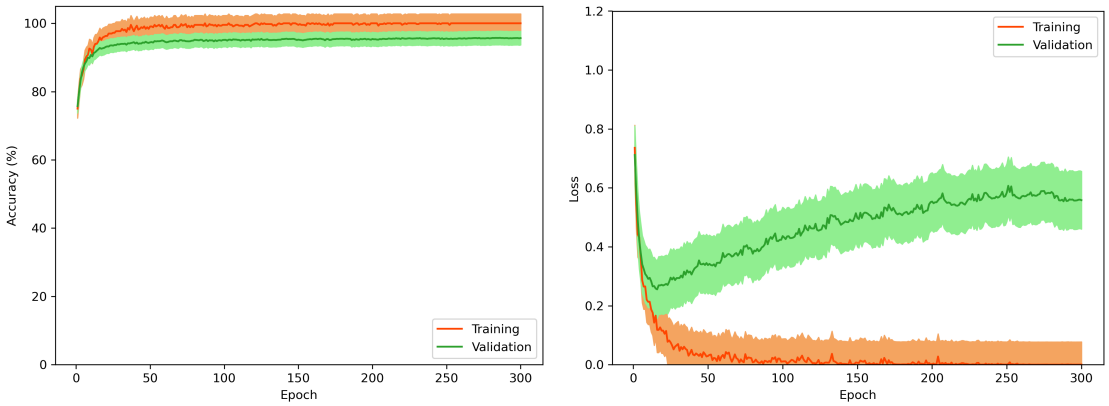

Figure B.1: Learning curves for the full-precision Long Short-Term Memory (LSTM) model retrained with 10 different seeds

B.2 LMU

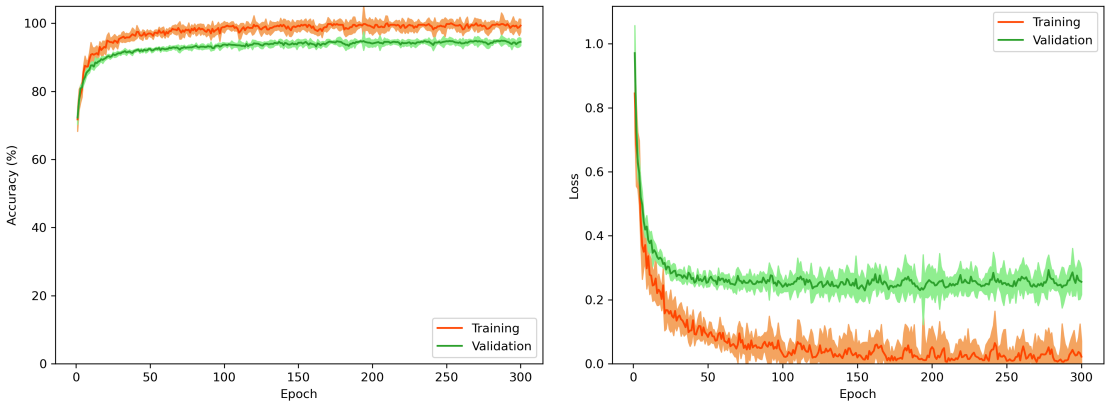

Figure B.2: Learning curves for the full-precision Legendre Memory Unit (LMU) model retrained with 10 different seeds

B.3 RNN

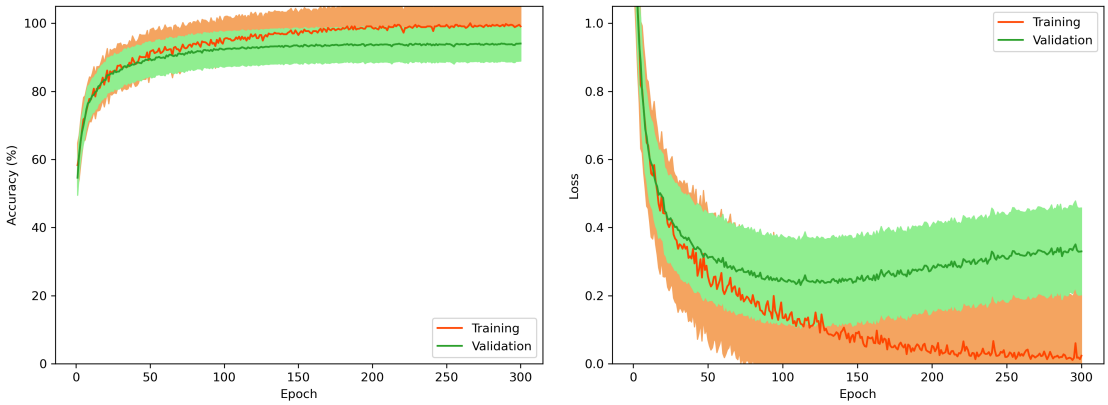

Figure B.3: Learning curves for the full-precision Recurrent Neural Network (RNN) model retrained with 10 different seeds

B.4 L^{2}MU

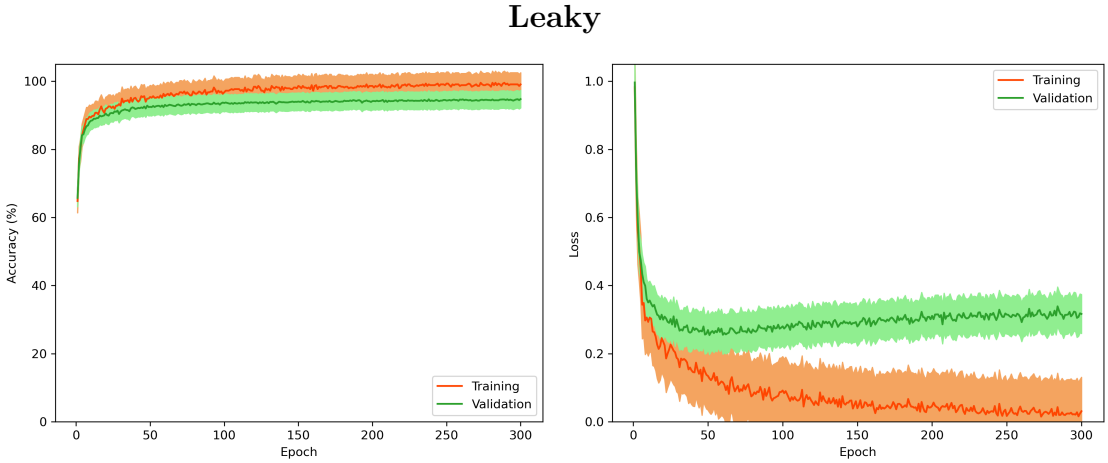

Figure B.4: Learning curves for the full-precision LIF-based LMU (L^2MU) model with Leaky neurons retrained with 10 different seeds

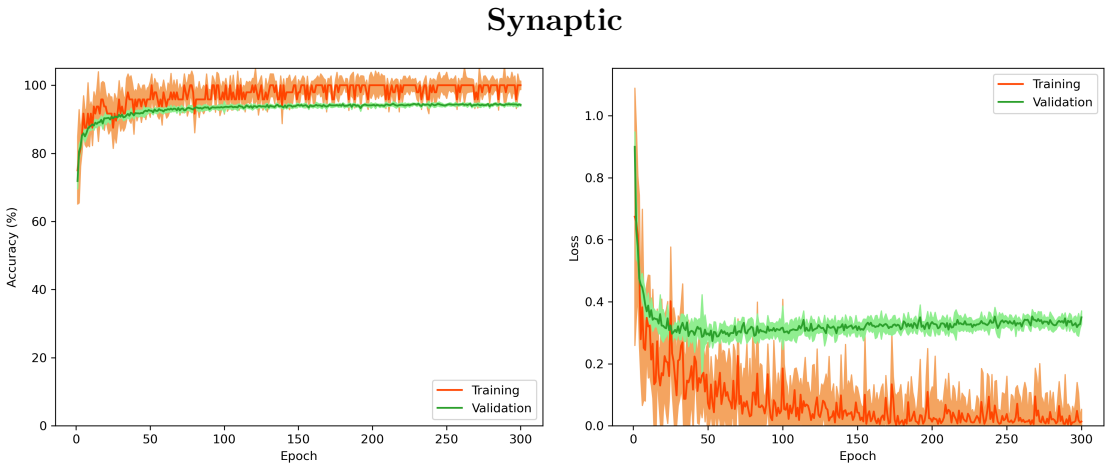

Figure B.5: Learning curves for the full-precision LIF-based LMU (L^2MU) model with Synaptic neurons retrained with 10 different seeds

B.5 Encoded L^2MU

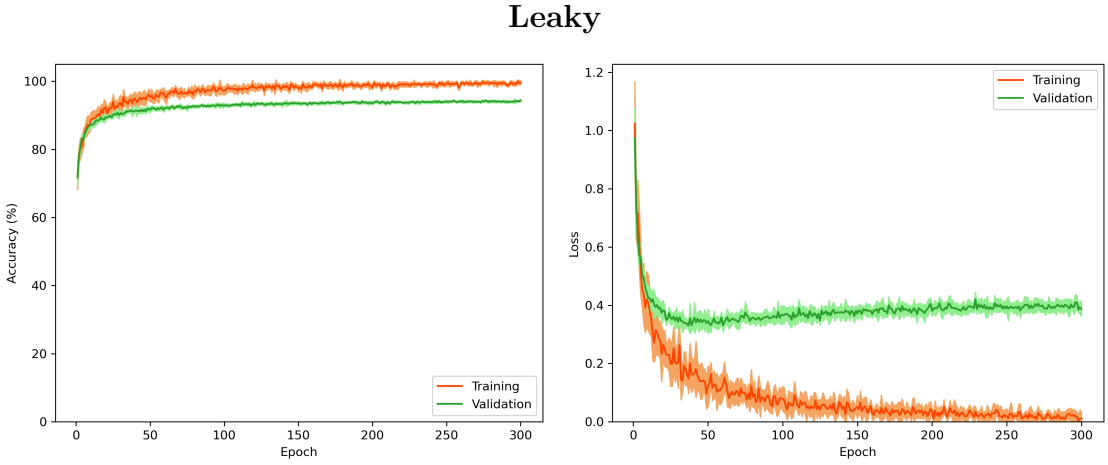

Figure B.6: Learning curves for the full-precision Encoded LIF-based LMU (Encoded L^2MU) model with Leaky neurons retrained with 10 different seeds

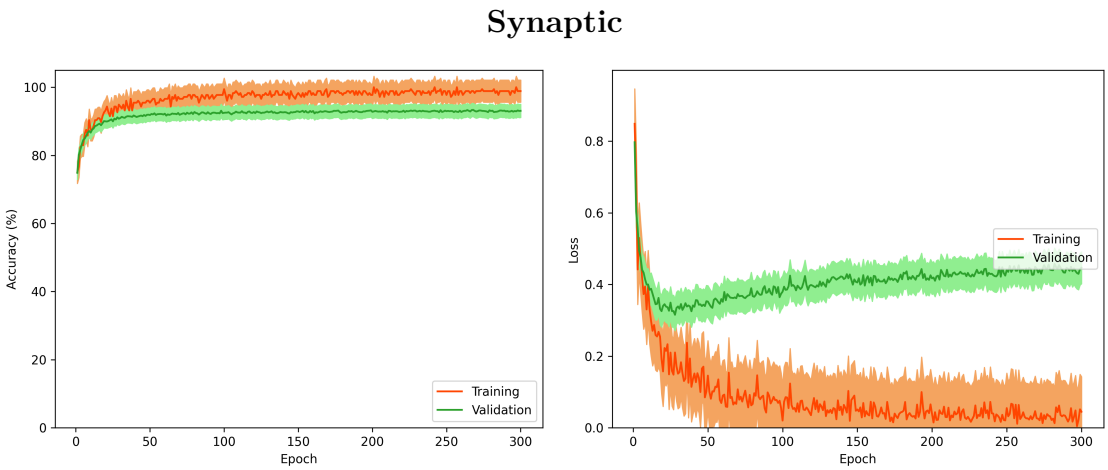

Figure B.7: Learning curves for the full-precision Encoded LIF-based LMU (Encoded L^2MU) model with Synaptic neurons retrained with 10 different seeds

B.6 Multi-Encoded L^{2}MU

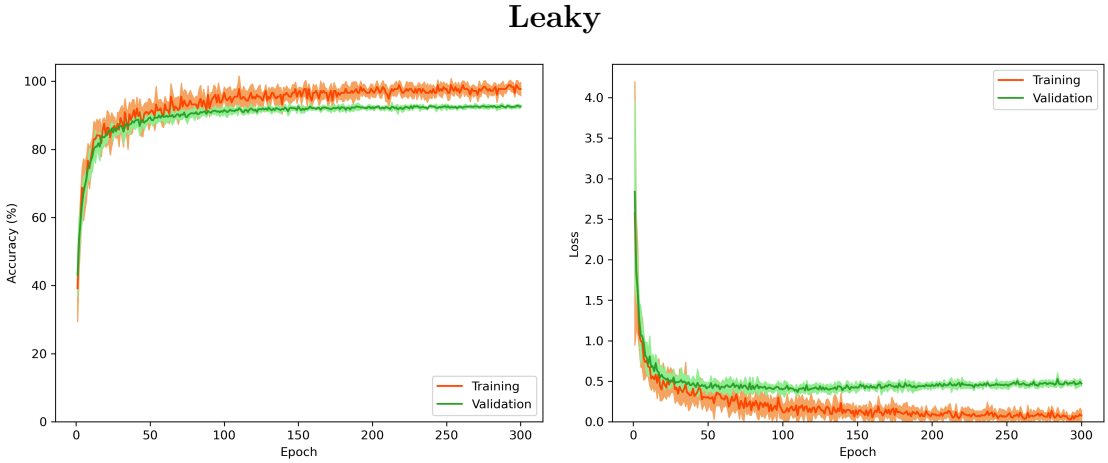

Figure B.8: Learning curves for the full-precision Multi-Encoded LIF-based LMU (Multi-Encoded L^2MU) model with Leaky neurons retrained with 10 different seeds

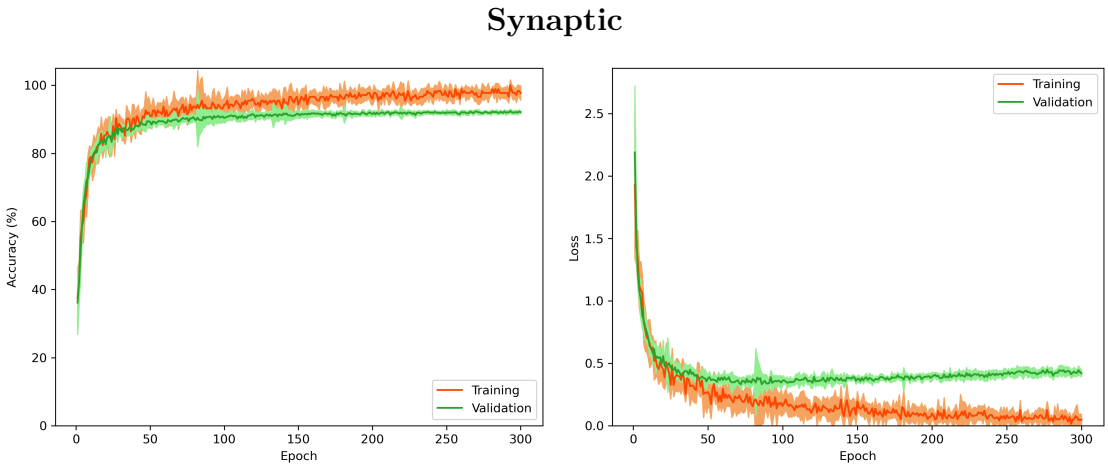

Figure B.9: Learning curves for the full-precision Multi-Encoded LIF-based LMU (Multi-Encoded L^2MU) model with Synaptic neurons retrained with 10 different seeds

B.7 L-RNN

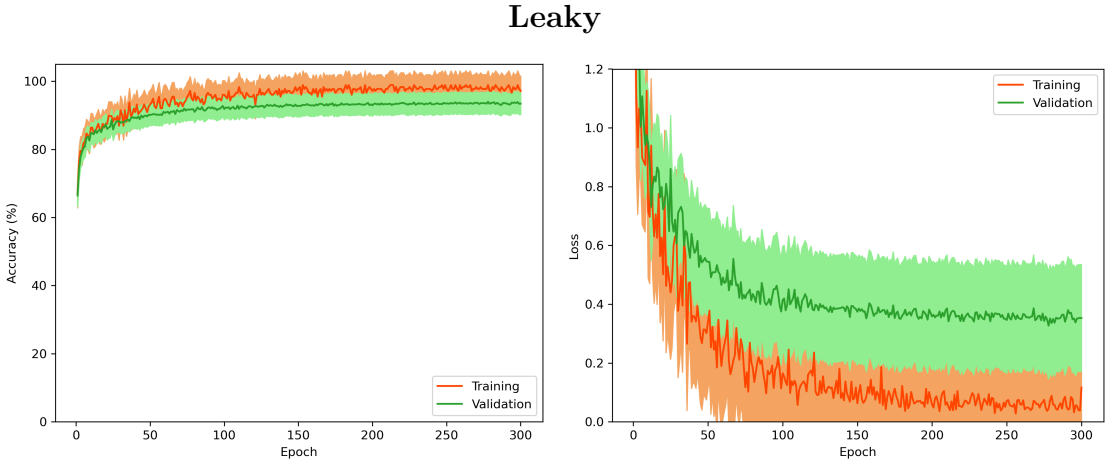

Figure B.10: Learning curves for the full-precision LIF-based RNN (L-RNN) model with Leaky neurons retrained with 10 different seeds

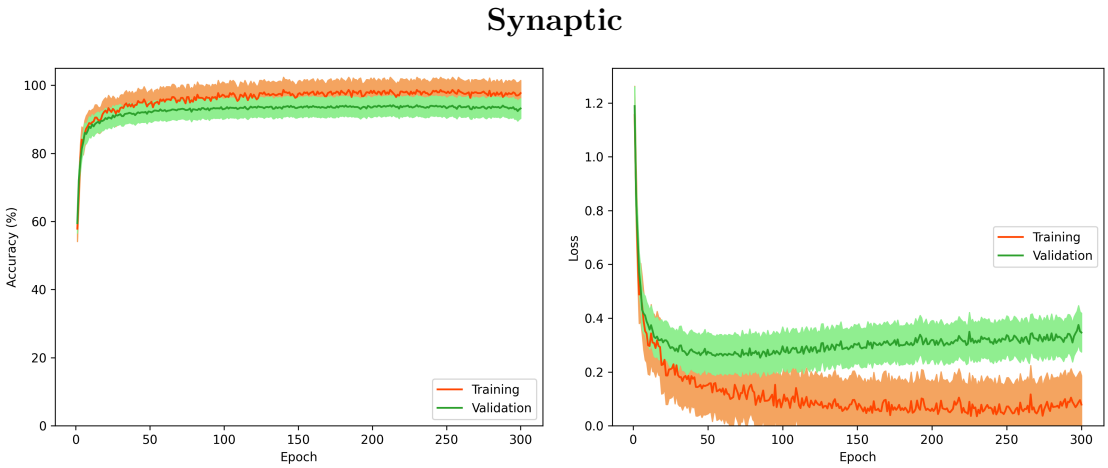

Figure B.11: Learning curves for the full-precision LIF-based RNN (L-RNN) model with Synaptic neurons retrained with 10 different seeds

B.8 Encoded L-RNN

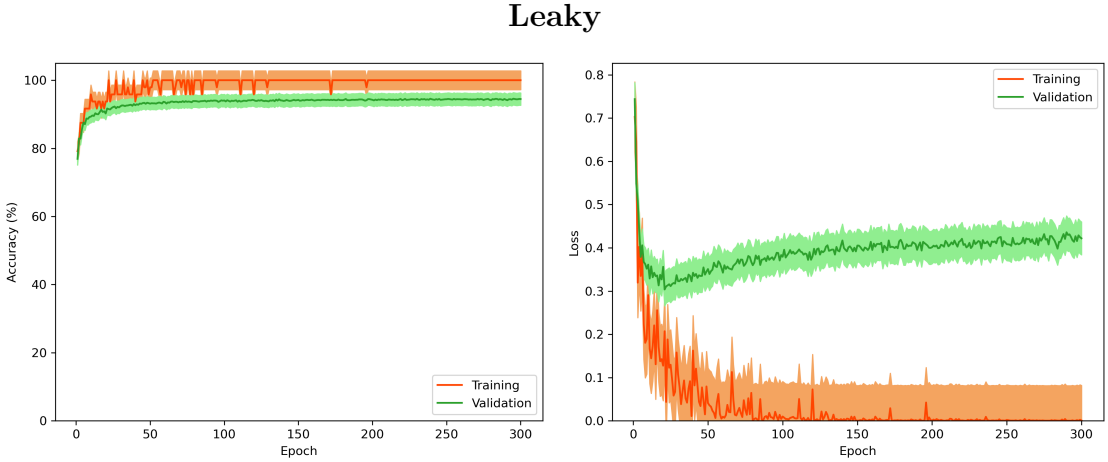

Figure B.12: Learning curves for the full-precision Encoded LIF-based RNN (Encoded L-RNN) model with Leaky neurons retrained with 10 different seeds

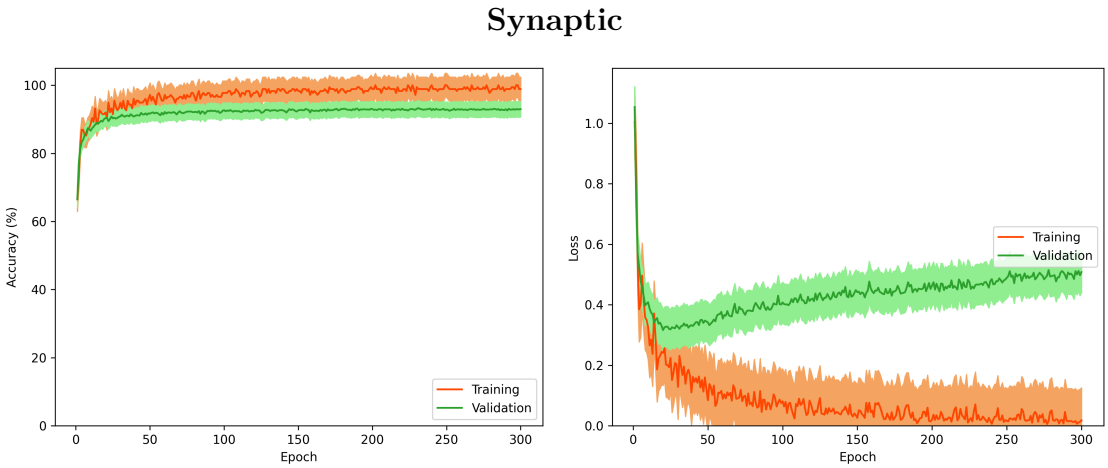

Figure B.13: Learning curves for the full-precision Encoded LIF-based RNN (Encoded L-RNN) model with Synaptic neurons retrained with 10 different seeds

B.9 Multi-Encoded L-RNN

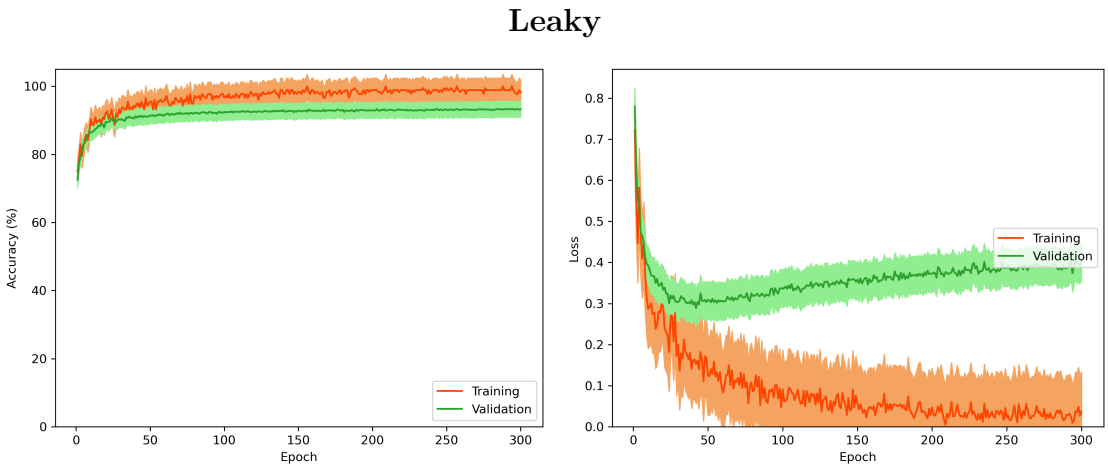

Figure B.14: Learning curves for the full-precision Multi-Encoded LIF-based RNN (Multi-Encoded L-RNN) model with Leaky neurons retrained with 10 different seeds

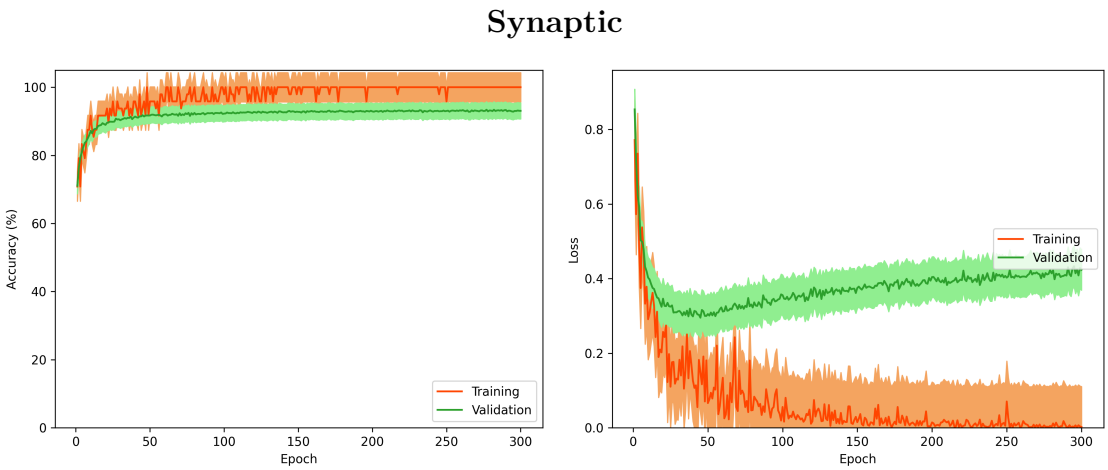

Figure B.15: Learning curves for the full-precision Multi-Encoded LIF-based RNN (Multi-Encoded L-RNN) model with Synaptic neurons retrained with 10 different seeds

Appendix C

Confusion Matrices

This chapter presents confusion matrices for various neural network models, highlighting their classification performance on test datasets. Confusion matrices are crucial for visualizing the performance of a classifier and are particularly valuable for identifying classes that are frequently misclassified. Each matrix shows the number of predictions for each class, with rows representing actual classes and columns representing predicted classes. Diagonal elements indicate correct classifications, while off-diagonal elements show misclassification.

C.1 LSTM

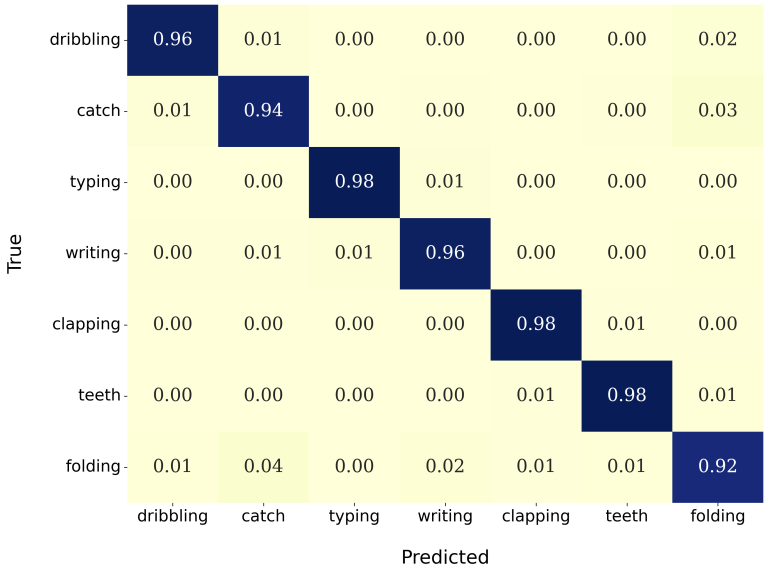

Figure C.1: Confusion matrix for the full-precision Long Short-Term Memory (LSTM) model.

C.2 LMU

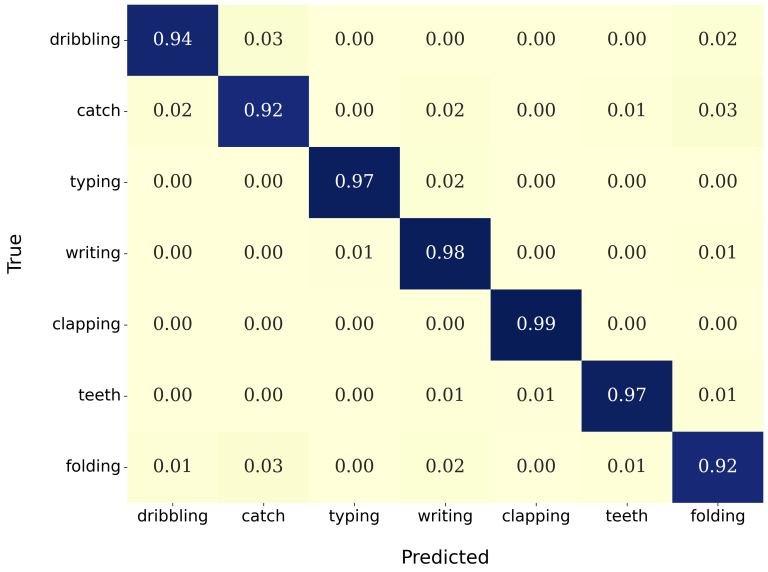

Figure C.2: Confusion matrix for the full-precision Legendre Memory Unit (LMU) model.

C.3 RNN

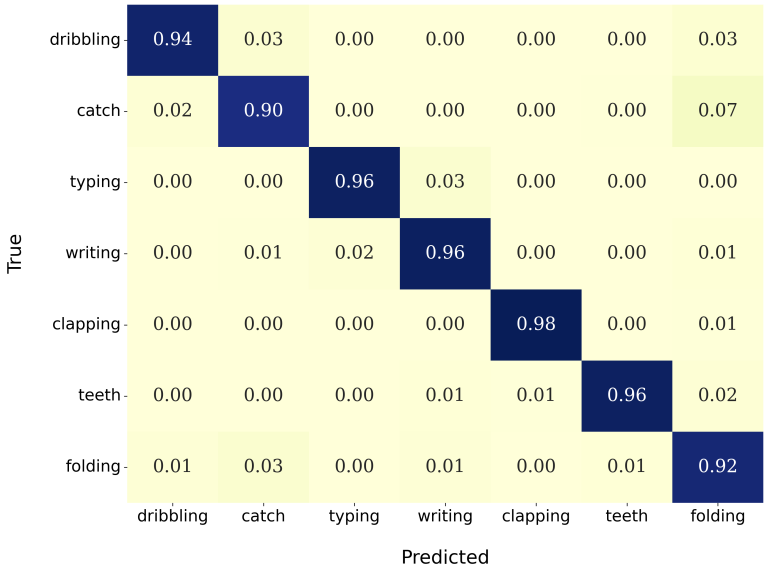

Figure C.3: Confusion matrix for the full-precision Recurrent Neural Network (RNN) model.

C.4 L^2MU

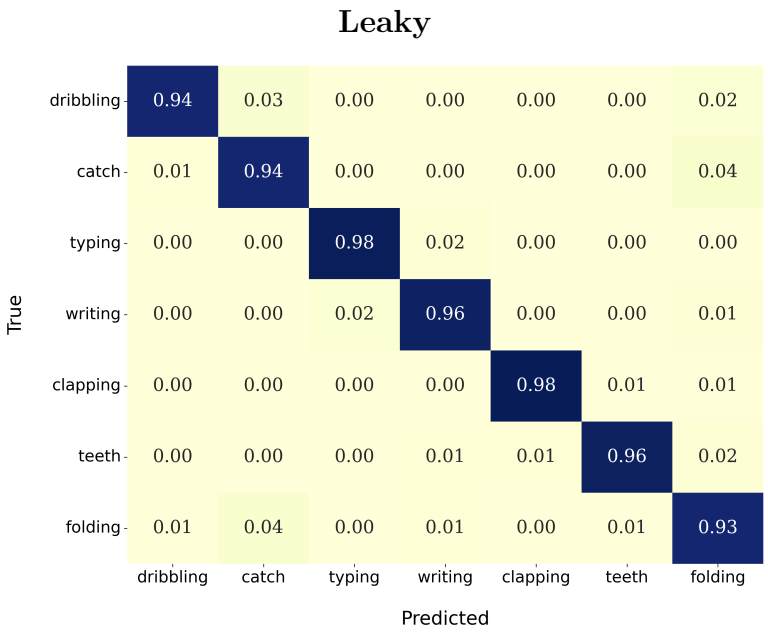

Figure C.4: Confusion matrix for the full-precision LIF-based LMU (L^2MU) model with Leaky neurons

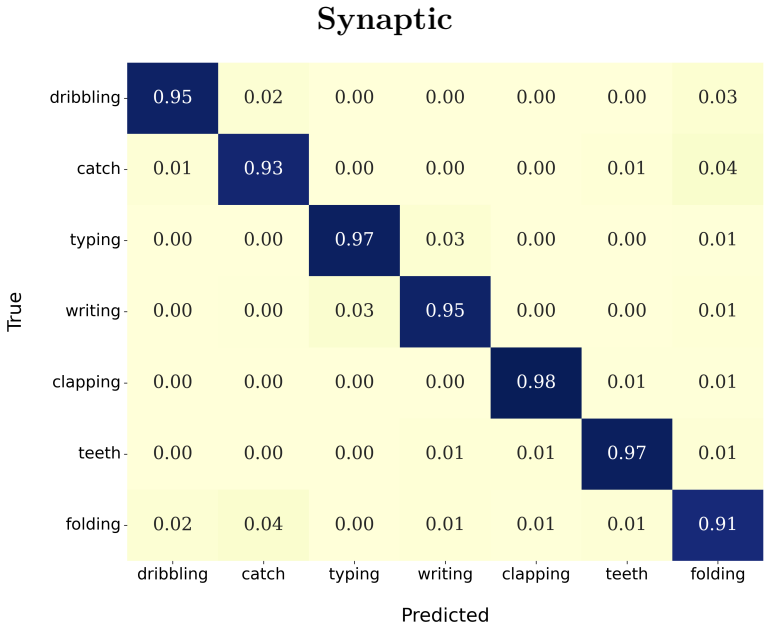

Figure C.5: Confusion matrix for the full-precision LIF-based LMU (L^2MU) model with Synaptic neurons

C.5 Encoded L^{2}MU

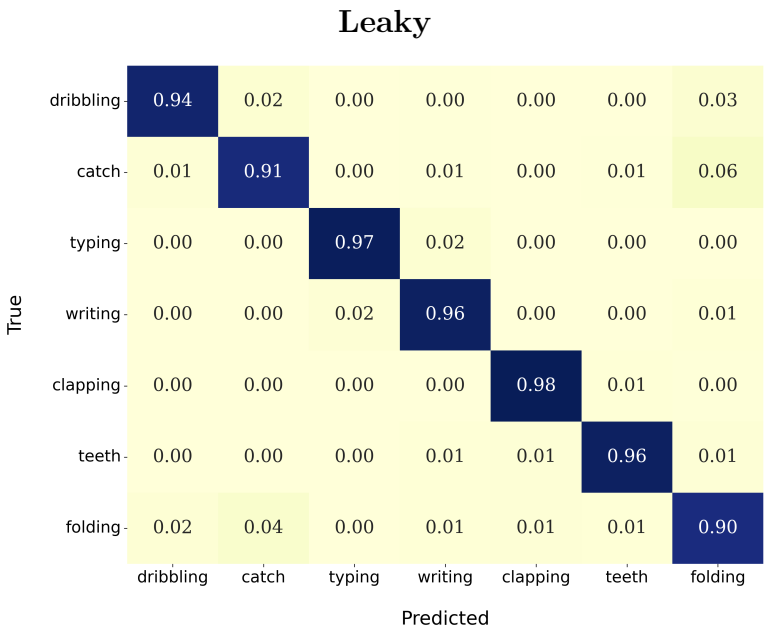

Figure C.6: Confusion matrix for the full-precision Encoded LIF-based LMU (Encoded L^{2}MU) model with Leaky neurons

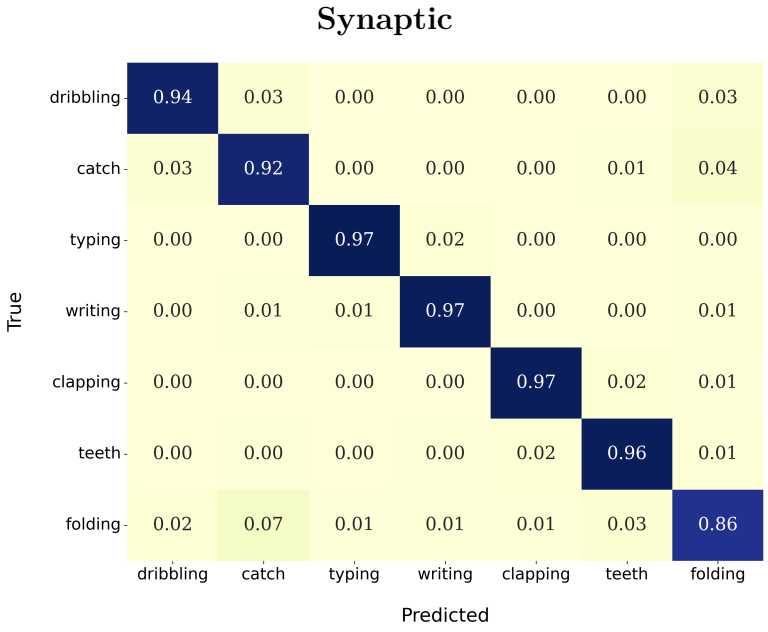

Figure C.7: Confusion matrix for the full-precision Encoded LIF-based LMU (Encoded L^{2}MU) model with Synaptic neurons

C.6 Multi-Encoded L^2MU

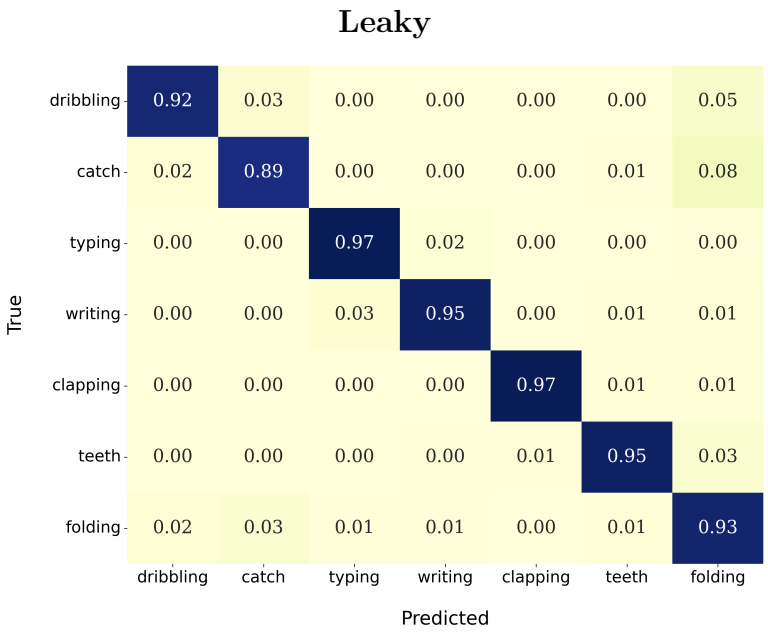

Figure C.8: Confusion matrix for the full-precision Multi-Encoded LIF-based LMU (Multi-Encoded L^2MU) model with Leaky neurons

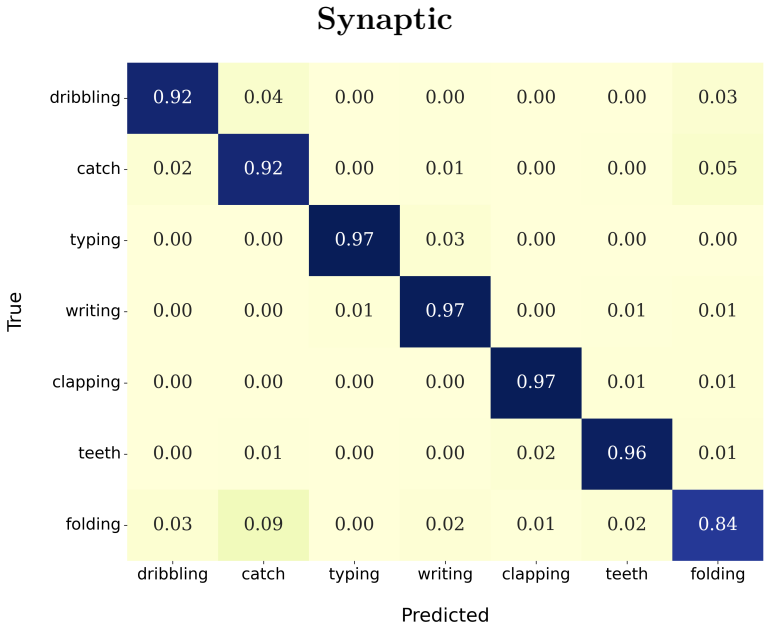

Figure C.9: Confusion matrix for the full-precision Multi-Encoded LIF-based LMU (Multi-Encoded L^2MU) model with Synaptic neurons

C.7 L-RNN

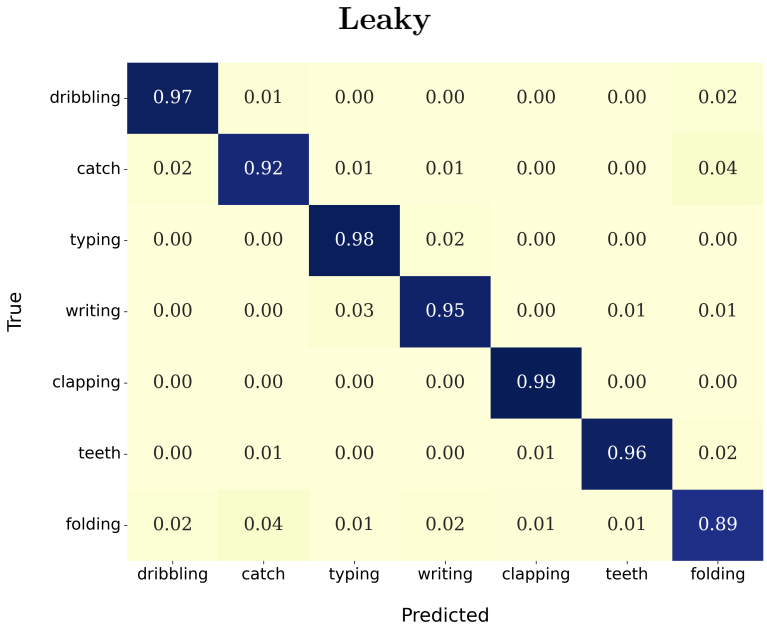

Figure C.10: Confusion matrix for the full-precision LIF-based RNN (L-RNN) model with Leaky neurons

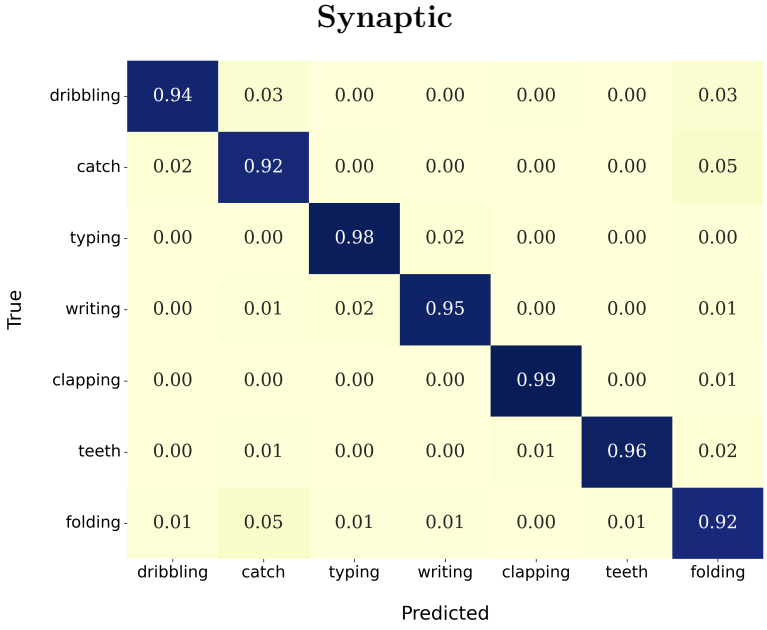

Figure C.11: Confusion matrix for the full-precision LIF-based RNN (L-RNN) model with Synaptic neurons

C.8 Encoded L-RNN

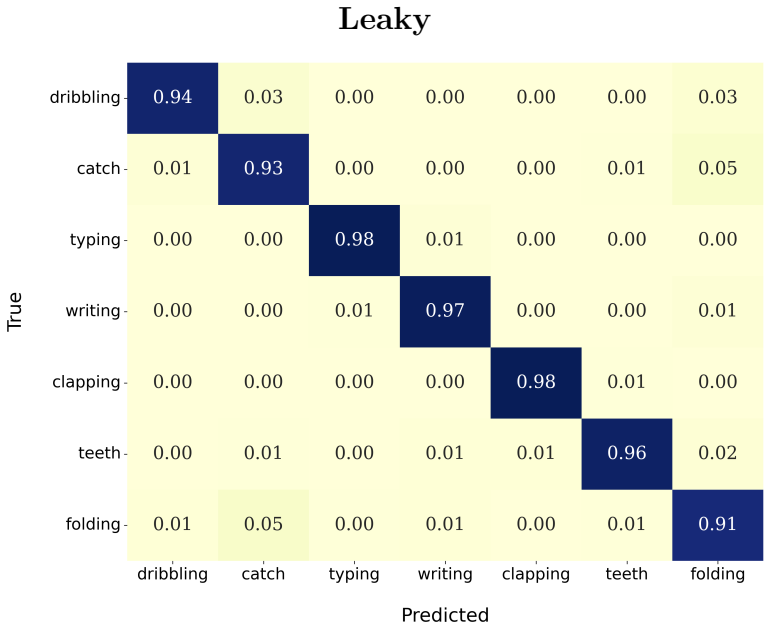

Figure C.12: Confusion matrix for the full-precision Encoded LIF-based RNN (Encoded L-RNN) model with Leaky neurons

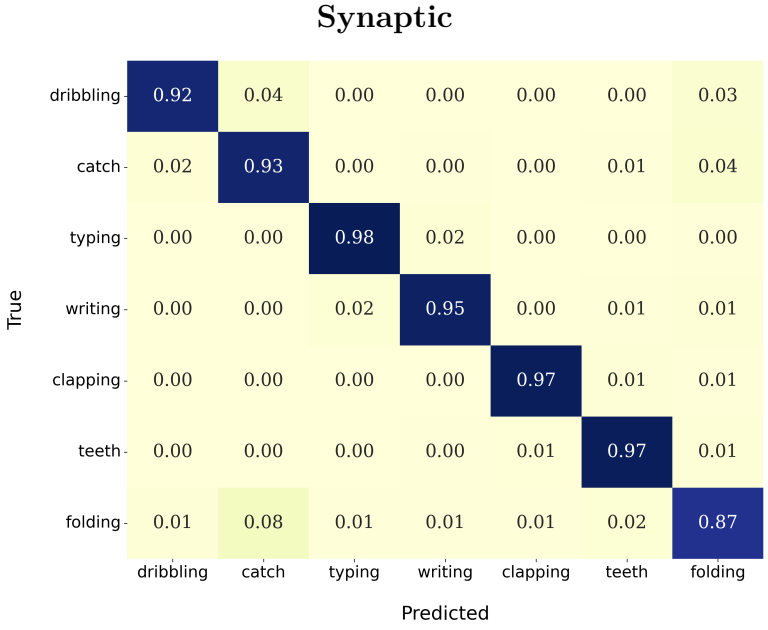

Figure C.13: Confusion matrix for the full-precision Encoded LIF-based RNN (Encoded L-RNN) model with Synaptic neurons

C.9 Multi-Encoded L-RNN

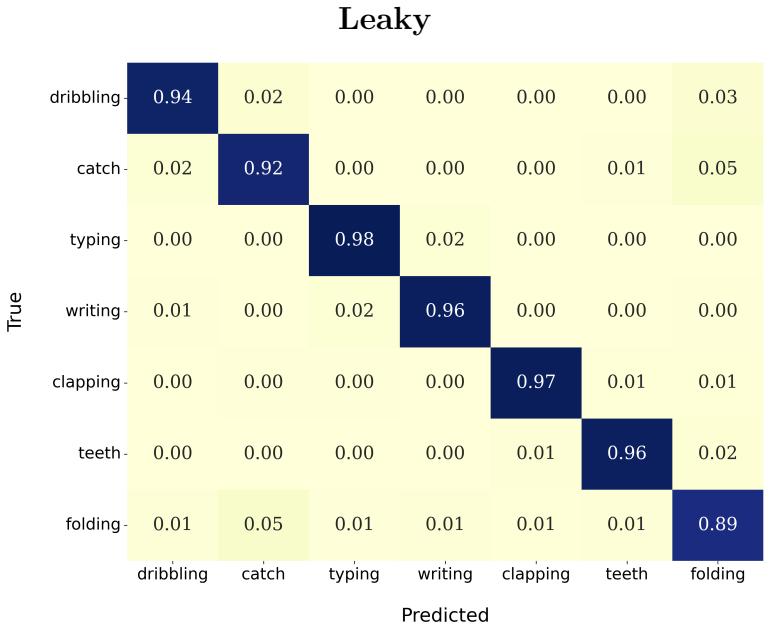

Figure C.14: Confusion matrix for the full-precision Multi-Encoded LIF-based RNN (Multi-Encoded L-RNN) model with Leaky neurons

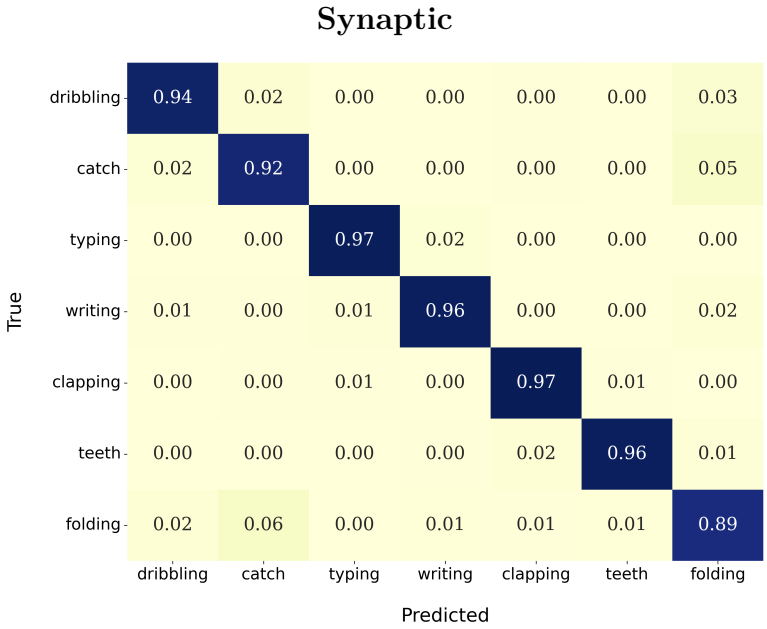

Figure C.15: Confusion matrix for the full-precision Multi-Encoded LIF-based RNN (Multi-Encoded L-RNN) model with Synaptic neurons

Appendix D

Hyperparameters

This chapter provides a comprehensive overview of the hyperparameters used for each architecture presented in this work. Hyperparameters are critical settings that control the learning process of a model and significantly impact its performance. For each architecture, we detail the key hyperparameters, their description, and display a full table with their value.

D.1 Hyperparameters' Description

D.1.1 Common Hyperparameters

Table D.1: Base Hyperparameters for all Architectures

Hyperparameter	Description
`lr`	Learning rate
`batch_size`	Batch size

D.1.2 Output Hyperparameters for SNNs

Table D.2: Hyperparameters Output Layer (Leaky/Synaptic)

Hyperparameter	Description
`beta_spk_output`	β value for output population neurons
`alpha_spk_output`	α value for output population neurons
`threshold_spk_output`	Threshold value for output population neurons

D.1.3 Encoder Hyperparameters

Table D.3: Hyperparameters Single Encoder Layer (Leaky/Synaptic)

Hyperparameter	Description
pop_size	Neuron Population size per axis
beta_spk_x_acc	β value for X-axis accel. population neurons
alpha_spk_x_acc	α value for X-axis accel. population neurons
threshold_spk_x_acc	Threshold value for X-axis accel. population neurons
beta_spk_x_gyro	β value for X-axis gyro population neurons
alpha_spk_x_gyro	α value for X-axis gyro population neurons
threshold_spk_x_gyro	Threshold value for X-axis gyro population neurons
beta_spk_y_acc	β value for Y-axis accel. population neurons
alpha_spk_y_acc	α value for Y-axis accel. population neurons
threshold_spk_y_acc	Threshold value for Y-axis accel. population neurons
beta_spk_y_gyro	β value for Y-axis gyro population neurons
alpha_spk_y_gyro	α value for Y-axis gyro population neurons
threshold_spk_y_gyro	Threshold value for Y-axis gyro population neurons
beta_spk_z_acc	β value for Z-axis accel. population neurons
alpha_spk_z_acc	α value for Z-axis accel. population neurons
threshold_spk_z_acc	Threshold value for Z-axis accel. population neurons
beta_spk_z_gyro	β value for Z-axis gyro population neurons
alpha_spk_z_gyro	α value for Z-axis gyro population neurons
threshold_spk_z_gyro	Threshold value for Z-axis gyro population neurons

Table D.4: Hyperparameters Stacked Encoder Layer (Leaky/Synaptic)

Hyperparameter	Description
...	Single Layer Hyperparameters (refer to Table D.3)
beta_spk_encoder	β value for fusion population neurons
alpha_spk_encoder	α value for fusion population neurons
threshold_spk_encoder	Threshold value for fusion population neurons
beta_spk_decoder	β value for harmonization population neurons
alpha_spk_decoder	α value for harmonization population neurons
threshold_spk_decoder	Threshold value for harmonization population neurons

D.1.4 RNN

Table D.5: Hyperparameters for RNN Architecture

Hyperparameter	Description
`hidden_size`	Size of hidden layer

D.1.5 LSTM

Table D.6: Hyperparameters for LSTM Architecture

Hyperparameter	Description
`hidden_size`	Size of hidden layer

D.1.6 LMU

Table D.7: Hyperparameters for LMU Architecture

Hyperparameter	Description
`order`	Maximum degree of Legendre polynomials
`theta`	Size of the sliding window
`hidden_size`	Size of hidden layer
`memory_size`	Size of memory layer

D.1.7 L-RNN

Table D.8: Hyperparameters for L-RNN (Leaky/Synaptic) Architecture

Hyperparameter	Description
...	RNN Hyperparameters (refer to Table D.5)
`beta_hidden`	β value for hidden population neurons
`alpha_hidden`	α value for hidden population neurons
`threshold_hidden`	Threshold value for hidden population neurons
`beta_output`	β value for the output population neurons
`alpha_output`	α value for the output population neurons
`threshold_output`	Threshold value for output population neurons

D.1.8 L^{2}MU

Table D.9: Hyperparameters for L^{2}MU (Leaky/Synaptic) Architecture

Hyperparameter	Description
...	LMU Hyperparameters (refer to Table D.7)
beta_spk_u	β value for encoder population neurons
alpha_spk_u	α value for encoder population neurons
threshold_spk_u	Threshold value for encoder population neurons
beta_spk_h	β value for hidden population neurons
alpha_spk_h	α value for hidden population neurons
threshold_spk_h	Threshold value for hidden population neurons
beta_spk_m	β value for memory population neurons
alpha_spk_m	α value for memory population neurons
threshold_spk_m	Threshold value for memory population neurons

D.2 Hyperparameters' Value

D.2.1 LSTM

Hyperparameter	Value
lr	0.0023
batch_size	256.0
hidden_size	250.0

Table D.10: Hyperparameters value for LSTM

D.2.2 LMU

Hyperparameter	Value
lr	0.00035
batch_size	256
order	9
theta	3.2
hidden_size	200
memory_size	190

Table D.11: Hyperparameters value for LMU

D.2.3 RNN

Hyperparameter	Value
lr	0.0003
batch_size	256
hidden_size	230

Table D.12: Hyperparameters value for RNN

D.2.4 L^{2}MU

Leaky

Hyperparameter	Value
lr	0.0005
batch_size	256
order	8.0
theta	13.4
hidden_size	280
memory_size	140
beta_spk_u	0.4
threshold_spk_u	0.15
beta_spk_h	0.2
threshold_spk_h	0.65
beta_spk_m	0.55
threshold_spk_m	0.9
beta_spk_output	0.7
threshold_spk_output	0.75

Table D.13: Hyperparameters for L^{2}MU (Leaky)

Synaptic

Hyperparameter	Value
lr	0.00085
batch_size	64
order	8
theta	1.1
hidden_size	230
memory_size	210
beta_spk_u	0.75
alpha_spk_u	0.55
threshold_spk_u	0.5
beta_spk_h	0.15
alpha_spk_h	0.2
threshold_spk_h	0.7
beta_spk_m	0.5
alpha_spk_m	0.4
threshold_spk_m	0.5
beta_spk_output	0.7
alpha_spk_output	0.25
threshold_spk_output	0.75

Table D.14: Hyperparameters for L^{2}MU (Synaptic)

D.2.5 Encoded L^{2}MU

Leaky

Hyperparameter	Value
lr	0.0009
batch_size	256
pop_size	50
beta_spk_x_acc	0.85
threshold_spk_x_acc	2.75
beta_spk_x_gyro	0.7
threshold_spk_x_gyro	3.95
beta_spk_y_acc	0.2
threshold_spk_y_acc	2.05
beta_spk_y_gyro	0.65
threshold_spk_y_gyro	0.45
beta_spk_z_acc	0.45
threshold_spk_z_acc	0.3
beta_spk_z_gyro	0.2
threshold_spk_z_gyro	0.2
order	9
theta	17.9
hidden_size	190
memory_size	130
beta_spk_u	0.8
threshold_spk_u	0.8
beta_spk_h	0.2
threshold_spk_h	0.8
beta_spk_m	0.35
threshold_spk_m	0.85
beta_spk_output	0.5
threshold_spk_output	0.35

Table D.15: Hyperparameters for Encoded L^{2}MU (Leaky)

Synaptic

Hyperparameter	Value
lr	0.00185
batch_size	128
pop_size	30
beta_spk_x_acc	0.15
alpha_spk_x_acc	0.55
threshold_spk_x_acc	4.0
beta_spk_x_gyro	0.15
alpha_spk_x_gyro	0.45
threshold_spk_x_gyro	3.4
beta_spk_y_acc	0.65
alpha_spk_y_acc	0.65
threshold_spk_y_acc	2.2
beta_spk_y_gyro	0.75
alpha_spk_y_gyro	0.5
threshold_spk_y_gyro	0.45
beta_spk_z_acc	0.2
alpha_spk_z_acc	0.25
threshold_spk_z_acc	0.25
beta_spk_z_gyro	0.75
alpha_spk_z_gyro	0.65
threshold_spk_z_gyro	0.3
order	8
theta	19.2
hidden_size	190
memory_size	200
beta_spk_u	0.25
alpha_spk_u	0.4
threshold_spk_u	0.5
beta_spk_h	0.3
alpha_spk_h	0.15
threshold_spk_h	0.75
beta_spk_m	0.15
alpha_spk_m	0.2
threshold_spk_m	0.85
beta_spk_output	0.35
alpha_spk_output	0.4
threshold_spk_output	0.8

Table D.16: Hyperparameters for Encoded L^{2}MU (Synaptic)

D.2.6 Multi-Encoded L^{2}MU

Leaky

Hyperparameter	Value
lr	0.0014
batch_size	128
pop_size	30
encoding_size	170
output_transformer_size	10
beta_spk_x_acc	0.55
threshold_spk_x_acc	4.7
beta_spk_x_gyro	0.6
threshold_spk_x_gyro	3.8
beta_spk_y_acc	0.2
threshold_spk_y_acc	3.95
beta_spk_y_gyro	0.25
threshold_spk_y_gyro	0.3
beta_spk_z_acc	0.3
threshold_spk_z_acc	0.2
beta_spk_z_gyro	0.3
threshold_spk_z_gyro	0.35
beta_spk_encoder	0.7
threshold_spk_encoder	0.85
beta_spk_decoder	0.2
threshold_spk_decoder	0.4
order	7
theta	13.6
hidden_size	60
memory_size	150
beta_spk_u	0.3
threshold_spk_u	0.3
beta_spk_h	0.8
threshold_spk_h	0.7
beta_spk_m	0.35
threshold_spk_m	0.25
beta_spk_output	0.35
threshold_spk_output	0.3

Table D.17: Hyperparameters for Multi-Encoded L^{2}MU (Leaky)

Synaptic

Hyperparameter	Value
lr	0.0009
batch_size	128
pop_size	30
encoding_size	180
output_transformer_size	10
beta_spk_x_acc	0.75
alpha_spk_x_acc	0.6
threshold_spk_x_acc	2.05
beta_spk_x_gyro	0.85
alpha_spk_x_gyro	0.55
threshold_spk_x_gyro	3.55
beta_spk_y_acc	0.3
alpha_spk_y_acc	0.15
threshold_spk_y_acc	1.55
beta_spk_y_gyro	0.15
alpha_spk_y_gyro	0.15
threshold_spk_y_gyro	0.1
beta_spk_z_acc	0.45
alpha_spk_z_acc	0.25
threshold_spk_z_acc	0.25
beta_spk_z_gyro	0.1
alpha_spk_z_gyro	0.5
threshold_spk_z_gyro	0.25
beta_spk_encoder	0.65
alpha_spk_encoder	0.35
threshold_spk_encoder	0.85
beta_spk_decoder	0.3
alpha_spk_decoder	0.6
threshold_spk_decoder	0.25
order	8
theta	1.8
hidden_size	180
memory_size	230
beta_spk_u	0.25
alpha_spk_u	0.25
threshold_spk_u	0.3
beta_spk_h	0.7
alpha_spk_h	0.65
threshold_spk_h	0.6
beta_spk_m	0.15
alpha_spk_m	0.75
threshold_spk_m	0.5
beta_spk_output	0.5
alpha_spk_output	0.15
threshold_spk_output	0.6

Table D.18: Hyperparameters for Multi-Encoded L^{2}MU (Synaptic)

D.2.7 L-RNN

Leaky

Hyperparameter	Value
lr	0.0017
batch_size	128
hidden_size	280
beta_hidden	0.3
threshold_hidden	0.2
beta_output	0.25
threshold_output	0.4

Table D.19: Hyperparameters for L-RNN (Leaky)

Synaptic

Hyperparameter	Value
lr	0.00075
batch_size	256
hidden_size	260
alpha_hidden	0.15
beta_hidden	0.15
threshold_hidden	0.95
alpha_output	0.3
beta_output	0.6
threshold_output	0.75

Table D.20: Hyperparameters for L-RNN (Synaptic)

D.2.8 Encoded L-RNN

Leaky

Hyperparameter	Value
lr	0.001
batch_size	64
pop_size	40
beta_spk_x_acc	0.45
threshold_spk_x_acc	1.0
beta_spk_x_gyro	0.1
threshold_spk_x_gyro	3.45
beta_spk_y_acc	0.6
threshold_spk_y_acc	3.2
beta_spk_y_gyro	0.3
threshold_spk_y_gyro	0.45
beta_spk_z_acc	0.45
threshold_spk_z_acc	0.45
beta_spk_z_gyro	0.2
threshold_spk_z_gyro	0.5
hidden_size	300
beta_hidden	0.4
threshold_hidden	0.6
beta_output	0.6
threshold_output	0.4

Table D.21: Hyperparameters for Encoded L-RNN (Leaky)

Synaptic

Hyperparameter	Value
lr	0.00075
batch_size	128
pop_size	30
alpha_spk_x_acc	0.65
beta_spk_x_acc	0.35
threshold_spk_x_acc	0.7
alpha_spk_x_gyro	0.65
beta_spk_x_gyro	0.2
threshold_spk_x_gyro	2.5
alpha_spk_y_acc	0.4
beta_spk_y_acc	0.3
threshold_spk_y_acc	2.7
alpha_spk_y_gyro	0.5
beta_spk_y_gyro	0.35
threshold_spk_y_gyro	0.25
alpha_spk_z_acc	0.5
beta_spk_z_acc	0.5
threshold_spk_z_acc	0.2
alpha_spk_z_gyro	0.4
beta_spk_z_gyro	0.4
threshold_spk_z_gyro	0.15
hidden_size	240
alpha_hidden	0.3
beta_hidden	0.4
threshold_hidden	0.7
alpha_output	0.65
beta_output	0.75
threshold_output	0.75

Table D.22: Hyperparameters for Encoded L-RNN (Synaptic)

D.2.9 Multi-Encoded L-RNN

Leaky

Hyperparameter	Value
lr	0.0007
batch_size	128
pop_size	30
encoding_size	120
output_transformer_size	26
beta_spk_x_acc	0.65
threshold_spk_x_acc	3.75
beta_spk_x_gyro	0.45
threshold_spk_x_gyro	2.55
beta_spk_y_acc	0.4
threshold_spk_y_acc	1.2
beta_spk_y_gyro	0.4
threshold_spk_y_gyro	0.15
beta_spk_z_acc	0.4
threshold_spk_z_acc	0.45
beta_spk_z_gyro	0.7
threshold_spk_z_gyro	0.25
beta_spk_encoder	0.2
threshold_spk_encoder	0.45
beta_spk_decoder	0.4
threshold_spk_decoder	0.55
hidden_size	190
beta_hidden	0.3
threshold_hidden	0.7
beta_output	0.7
threshold_output	0.9

Table D.23: Hyperparameters for Multi-Encoded L-RNN (Leaky)

Synaptic

Hyperparameter	Value
lr	0.00055
batch_size	64
pop_size	30
encoding_size	180
output_transformer_size	30
alpha_spk_x_acc	0.65
beta_spk_x_acc	0.4
threshold_spk_x_acc	4.05
alpha_spk_x_gyro	0.3
beta_spk_x_gyro	0.5
threshold_spk_x_gyro	4.8
alpha_spk_y_acc	0.35
beta_spk_y_acc	0.15
threshold_spk_y_acc	1.25
alpha_spk_y_gyro	0.25
beta_spk_y_gyro	0.6
threshold_spk_y_gyro	0.2
alpha_spk_z_acc	0.3
beta_spk_z_acc	0.25
threshold_spk_z_acc	0.3
alpha_spk_z_gyro	0.3
beta_spk_z_gyro	0.5
threshold_spk_z_gyro	0.3
alpha_spk_encoder	0.4
beta_spk_encoder	0.1
threshold_spk_encoder	0.55
alpha_spk_decoder	0.55
beta_spk_decoder	0.25
threshold_spk_decoder	0.75
hidden_size	60
alpha_hidden	0.6
beta_hidden	0.3
threshold_hidden	0.25
alpha_output	0.2
beta_output	0.35
threshold_output	0.8

Table D.24: Hyperparameters for Multi-Encode L-RNN (Synaptic)

Acknowledgements

We acknowledge a contribution from the Italian National Recovery and Resilience Plan (NRRP), M4C2, funded by the European Union – NextGenerationEU (Project IR0000011, CUP B51E22000150006, “EBRAINS-Italy”).

Bibliography

[1] Sina Dami and Mahtab Yahaghizadeh. «Predicting cardiovascular events with deep learning approach in the context of the internet of things». In: *Neural Computing and Applications* 33 (2021). DOI: 10.1007/s00521-020-05542-x.

[2] Nicole A Capela, Edward D Lemaire, and Natalie Baddour. «Feature Selection for Wearable Smartphone-Based Human Activity Recognition with Able bodied, Elderly, and Stroke Patients». In: *PLOS ONE* 10 (2015). DOI: 10.1371/journal.pone.0124414.

[3] Hoda Allahbakhshi et al. «Using accelerometer and GPS data for real-life physical activity type detection». In: *Sensors (Switzerland)* 20 (2020). DOI: 10.3390/s20030588.

[4] Enea Ceolini et al. «Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing». In: *Frontiers in Neuroscience* 14 (2020). DOI: 10.3389/fnins.2020.00637.

[5] Andrea E. Frank, Alyssa Kubota, and Laurel D. Riek. «Wearable activity recognition for robust human-robot teaming in safety-critical environments via hybrid neural networks». In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019. DOI: 10.1109/IROS40897.2019.8968615.

[6] Farzana Kulsoom et al. «A review of machine learning-based human activity recognition for diverse applications». In: *Neural Computing and Applications* 34 (2022).

[7] Oscar D. Lara and Miguel A. Labrador. «A Survey on Human Activity Recognition using Wearable Sensors». In: *IEEE Communications Surveys & Tutorials* 15 (2013). DOI: 10.1109/SURV.2012.110112.00192.

[8] Anna Ferrari et al. «Trends in human activity recognition using smartphones». In: *Journal of Reliable Intelligent Environments* 7 (2021). DOI: 10.1007/s40860-021-00147-0.

[9] Henry Friday Nweke et al. «Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges». In: *Expert Systems with Applications* 105 (2018). DOI: 10.1016/j.eswa.2018.03.056.

[10] Salwa O. Slim et al. «Survey on Human Activity Recognition based on Acceleration Data». In: *International Journal of Advanced Computer Science and Applications* 10 (2019). DOI: 10.14569/IJACSA.2019.0100311.

[11] Florenc Demrozi et al. «Human Activity Recognition Using Inertial, Physiological and Environmental Sensors: A Comprehensive Survey». In: *IEEE Access* 8 (2020). DOI: 10.1109/ACCESS.2020.3037715.

[12] Nida Saddaf Khan and Muhammad Sayeed Ghani. «A Survey of Deep Learning Based Models for Human Activity Recognition». In: *Wireless Personal Communications* (2021). DOI: 10.1007/s11277-021-08525-w.

[13] Wolfgang Maass. «Networks of spiking neurons: The third generation of neural network models». In: *Neural Networks* 10 (1997). DOI: 10.1016/S0893-6080(97)00011-7.

[14] Mike Davies et al. «Loihi: A Neuromorphic Manycore Processor with On-Chip Learning». In: *IEEE Micro* 38 (2018). DOI: 10.1109/MM.2018.112130359.

[15] Garrick Orchard et al. «Efficient Neuromorphic Signal Processing with Loihi 2». In: *IEEE Workshop on Signal Processing Systems (SiPS)*. Vol. 2021-Octob. 2021. DOI: 10.1109/SiPS52927.2021.00053.

[16] Simon F. Müller-Cleve et al. «Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware». In: *Frontiers in Neuroscience* 16 (2022). DOI: 10.3389/fnins.2022.951164.

[17] Gary M. Weiss. «WISDM Smartphone and Smartwatch Activity and Biometrics Dataset». In: *UCI Machine Learning Repository: WISDM Smartphone and Smartwatch Activity and Biometrics Dataset Data Set* 7 (2019).

[18] Gary M. Weiss, Kenichi Yoneda, and Thaier Hayajneh. «Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living». In: *IEEE Access* 7 (2019). DOI: 10.1109/ACCESS.2019.2940729.

[19] Saeed Reza Kheradpisheh et al. «STDP-based spiking deep convolutional neural networks for object recognition». In: *Neural Networks* 99 (Mar. 2018), pp. 56–67. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2017.12.005. URL: http://dx.doi.org/10.1016/j.neunet.2017.12.005.

[20] Frederico A C Azevedo et al. «Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain». en. In: *J. Comp. Neurol.* 513.5 (Apr. 2009), pp. 532–541.

[21] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: 1207.0580 [id='cs.NE' full$_name$ =' $NeuralandEvolutionaryComputing'is_active = True alt_name = None in_archive =' cs'is_general = False description =' Covers neural networks, connectionism, genetical gorithm$

[22] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. «A fast learning algorithm for deep belief nets». en. In: *Neural Comput.* 18.7 (July 2006), pp. 1527–1554.

[23] G.E. Hinton and R.R. Salakhutdinov. «Reducing the Dimensionality of Data with Neural Networks». In: *Science (New York, N.Y.)* 313 (Aug. 2006), pp. 504–7. DOI: 10.1126/science.1127647.

[24] D. Pham, Michael Packianather, and Eugine Charles. «A Novel Self-Organised Learning Model with Temporal Coding for Spiking Neural Networks». In: *Intelligent Production Machines and Systems - 2nd I*PROMS Virtual International Conference 3-14 July 2006* (Jan. 2007). DOI: 10.1016/B978-008045157-2/50057-2.

[25] Romain Brette et al. «Simulation of networks of spiking neurons: A review of tools and strategies». In: *Journal of computational neuroscience* 23.3 (2007), pp. 349–398.

[26] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge University Press, 2002.

[27] Alan Lloyd Hodgkin and Andrew Fielding Huxley. «A quantitative description of membrane current and its application to conduction and excitation in nerve». In: *The Journal of physiology* 117.4 (1952), pp. 500–544.

[28] Eugene M Izhikevich. «Which model to use for cortical spiking neurons?» In: *IEEE transactions on neural networks* 15.5 (2004), pp. 1063–1070.

[29] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2006.

[30] Nikola Kasabov et al. «Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data». In: *Neural Networks* 52 (2014), pp. 62–76.

[31] Wolfgang Maass and Anthony M Zador. «Dynamic stochastic synapses as computational units». In: *Neural Computation* 11.4 (1999), pp. 903–917.

[32] Evelina Forno et al. «Spike encoding techniques for IoT time-varying signals benchmarked on a neuromorphic classification task». In: *Frontiers in Neuroscience* (2022).

[33] Filipp Akopyan et al. «TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (2015), pp. 1537–1557. DOI: 10.1109/TCAD.2015.2474396.

[34] Christian Mayr, Sebastian Hoeppner, and Steve Furber. *SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning*. 2019. arXiv: 1911.02385 [cs.ET]. URL: https://arxiv.org/abs/1911.02385.

[35] Mike Davies et al. «Loihi: A Neuromorphic Manycore Processor with On-Chip Learning». In: *IEEE Micro* 38.1 (2018), pp. 82–99. DOI: 10.1109/MM.2018.112130359.

[36] Fabrizio Ottati et al. *To Spike or Not To Spike: A Digital Hardware Perspective on Deep Learning Acceleration*. 2024. arXiv: 2306.15749 [cs.NE].

[37] D. Anguita et al. «A Public Domain Dataset for Human Activity Recognition using Smartphones». In: *The European Symposium on Artificial Neural Networks*. 2013.

[38] Jorge Luis Reyes-Ortiz et al. «Transition-Aware Human Activity Recognition Using Smartphones». In: *Neurocomputing* 171 (2016).

[39] Friedrich Niemann et al. «LARa: Creating a Dataset for Human Activity Recognition in Logistics Using Semantic Attributes». In: *Sensors* 20 (2020). DOI: 10.3390/s20154083.

[40] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. «Activity recognition using cell phone accelerometers». In: *ACM SIGKDD Explorations Newsletter* 12 (2011). DOI: 10.1145/1964897.1964918.

[41] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. arXiv: 1912.05911 [cs.LG]. URL: https://arxiv.org/abs/1912.05911.

[42] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-term Memory». In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[43] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-term Memory». In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[44] Aaron R. Voelker, Ivana Kajic, and Chris Eliasmith. «Legendre memory units: Continuous-time representation in recurrent neural networks». In: *Advances in Neural Information Processing Systems* 32.NeurIPS (2019). ISSN: 10495258.

[45] Vittorio Fra et al. «Human activity recognition: suitability of a neuromorphic approach for on-edge AIoT applications». In: *Neuromorphic Computing and Engineering* 2 (2022). DOI: 10.1088/2634-4386/ac4c38.

[46] Carmen Amo Alonso, Jerome Sieber, and Melanie N. Zeilinger. *State Space Models as Foundation Models: A Control Theoretic Overview*. 2024. arXiv: 2403.16899 [eess.SY]. URL: https://arxiv.org/abs/2403.16899.

[47] Yu Du, Xu Liu, and Yansong Chua. *Spiking Structured State Space Model for Monaural Speech Enhancement*. 2024. arXiv: 2309.03641 [cs.SD]. URL: https://arxiv.org/abs/2309.03641.

[48] Albert Gu, Karan Goel, and Christopher Ré. *Efficiently Modeling Long Sequences with Structured State Spaces*. 2022. arXiv: 2111.00396 [cs.LG]. URL: https://arxiv.org/abs/2111.00396.

[49] Jason K. Eshraghian et al. «Training Spiking Neural Networks Using Lessons From Deep Learning». In: (2021).

[50] Jason Yik et al. *NeuroBench: A Framework for Benchmarking Neuromorphic Computing Algorithms and Systems*. 2024. arXiv: 2304.04640 [cs.AI].

[51] Domenico Stefani, Simone Peroni, and Luca Turchet. «A comparison of deep learning inference engines for embedded real-time audio classification». In: *Proceedings of the International Conference on Digital Audio Effects, DAFx*. Vol. 3. 2022.