Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Analyzing Frameworks and Strategies for Converting Neural Networks for Neuromorphic Processors

Sreelakshmi Rameshan

Course of Study:	M.Sc Computer Science
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Dr. Gerrit Ecke, Dr. Pascal Hirmer
Commenced:	Febraury 15, 2024
Completed:	August 15, 2024

Abstract

Neuromorphic computing employs innovative algorithms to mimic how the human brain interacts with the world, aiming to achieve capabilities that closely resemble human cognition. Neuromorphic processors are based on an entirely new computing paradigm and they come with new Machine Learning (ML) algorithms. Programming a neuromorphic processor often entails creating a Spiking Neural Network (SNN) that closely mimics the biological neural networks and can be deployed to the neuromorphic processor. Neuromorphic processors leverage these asynchronous, event-based SNNs to achieve substantial increase in power and performance over conventional architectures. However, training such networks is difficult due to the non-differentiable nature of spike events.

This thesis investigates the frameworks and strategies for converting standard neural networks, particularly Deep Learning (DL) models, to SNNs suitable for neuromorphic processors. Focusing on three neuromorphic platforms—Brainchip Akida, Intel Loihi, and SynSense—the thesis aims to develop a standardized conversion pipeline, address current limitations, and conduct metric-based analyses of the model developed using different frameworks. The thesis utilizes the PilotNet model, a Convolutional Neural Network (CNN) designed for autonomous driving, to evaluate the conversion processes and performance on the selected neuromorphic framework.

Results demonstrate the varying degrees of efficiency and challenges associated with each neuromorphic processor, providing insights for optimizing the conversion process and further advancing neuromorphic computing for practical applications such as autonomous driving, robotics, and edge computing. The findings emphasize the need for continued development in conversion techniques and the optimization of neuromorphic hardware to fully harness the potential of AI-driven systems.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence. 13

ANN Artificial Neural Network. 29

BPTT Backpropagation Through Time. 62

CNN Convolutional Neural Network. 3

CPU Central Processing Unit. 13

DL Deep Learning. 3

GPU Graphics Processing Unit. 13

IoT Internet of Things. 28

ML Machine Learning. 3

MSE Mean Squared Error. 15

PRISMA Preferred Reporting Items for Systematic Reviews and Meta-Analyses. 19

SDNN STDP-based spiking deep neural network. 47

SNN Spiking Neural Network. 3

STDP Spike-Timing-Dependent Plasticity. 29

1 Introduction

The rapid advancement of Artificial Intelligence (AI) has resulted in a notable need for more specialized and efficient technology that can manage the computational intricacy of neural networks. Neuromorphic processors, which are made to resemble the structure and operations of the human brain, are gradually replacing conventional processors like Central Processing Unit (CPU)s and Graphics Processing Unit (GPU)s. The neuromorphic processors are ideal for implementing AI models in resource-constrained contexts because they offer advantages in terms of energy consumption, speed, and real-time processing capabilities [1].

While the GPU, Tensor processors, and DL accelerators of today focus on dense matrix-based computation at a very high throughput, neuromorphic processors focus on sparse event-driven computation that minimizes activity and data movement. Although neuromorphic processors are not yet mainstream commercial products, they have received increasing research and development focus in the recent years, with an accelerating pace of progress [2].

Neuromorphic processors are based on SNN, which significantly differ from the traditional neural networks. SNN mimic a neurobiological approach to Neural Network computation through the spiking behavior of biological neurons. Contrary to traditional artificial neurons that produce real-valued outputs, spiking neurons receive input spikes and integrate them into the state of the neuron, which is called the membrane potential. Upon reaching a defined threshold, the neuron fires a spike, which will further propagate through the network, and resets its membrane potential. This binary format of SNNs output allows for energy-efficient computations on neuromorphic hardware, resulting in no requirement for multiplications [3]. Specifically, neuromorphic hardware can exploit the sparsity and binary nature of the output produced by SNNs. However, one of the key challenges to the wide application of neuromorphic processors lies in the conversion of existing neural network architectures, which were designed for conventional hardware, into their brain-inspired counterparts.

1.1 Problem Statement

The challenges in running DL models on neuromorphic chips lies in the intrinsic differences between traditional neural networks and the architecture of used for neuromorphic computing. Traditional DL models are usually constructed with CNN or fully connected layers and rely on continuous activations and gradient-based learning methods like backpropagation. The SNNs, however, are driven by discrete spikes, and their processing is event-driven instead of continuous. This creates the need for a conversion process in a way that the continuous activations of conventional neural networks could be translated into the spiking domain without losing much information or accuracy [4].

On the other hand, the conversion process is not straightforward, as it involves redefining how neurons communicate and process information. Moreover, maintaining the performance of the original DL model after conversion is challenging, as the spiking model must approximate the behavior of the original network under different computational constraints. Ensuring that the converted SNN retains the accuracy and robustness of the original model, especially in safety-critical applications like autonomous driving, is a significant problem that needs to be addressed [4].

Another critical problem is the training of DL models specifically for neuromorphic hardware due to the non-differentiable nature of spike events. Traditional training methods are not directly applicable to neuromorphic systems due to the lack of a direct equivalent to backpropagation in SNNs. While some neuromorphic frameworks support training through surrogate gradients or other approximations, these methods are still in their infancy and can lead to suboptimal model performance [5].

The problem is compounded by the limited toolchains and frameworks available for neuromorphic computing, making it difficult to experiment with and fine-tune models for these specialized processors. This creates a barrier to entry for researchers and developers, slowing the adoption of neuromorphic technology in practical applications. Therefore, developing robust methods for training and fine-tuning DL models specifically for neuromorphic chips is a crucial area of research that needs to be addressed to advance the field [5].

One approach is the translation process from traditional DL models to (SNNs).Here, we need to carefully consider on maintaining the performance and accuracy of the original model after conversion as it posts challenges due to the fundamental differences in architecture and operation between conventional neural networks and SNNs [6] [7] [8].

1.2 Objective

The primary objective of this research is to leverage and compare different software frameworks for the analysis of Standard Neural Network to (SNN) conversion processes. This involves developing a standardized pipeline, identifying and addressing current limitations, and utilizing these insights for metric-based analysis and evaluation of neuromorphic processors. By focusing on the specific challenges associated with converting DL models for neuromorphic architectures such as Akida, Intel Loihi, and SynSense, this thesis aims to facilitate the practical conversion of DL models for neuromorphic chips. Additionally, it contributes to the broader field of neuromorphic computing by providing an comparative analysis of the different neuromorpic chips in terms of their efficiency in performance and challenges that can be encountered in the conversion process. Ultimately, this research lays the groundwork for more widespread adoption of neuromorphic processors in real-world applications, particularly in domains such as autonomous driving, robotics, and edge computing. By addressing these challenges and leveraging the advantages, my thesis aims to contribute to the utilization of neuromorphic processors for AI tasks.

1.3 Methodology

- **Identifying the Deep Neural Network:** The initial step includes the identification of the deep neural network. For the thesis, I have chosen PilotNet, which is a simple Convolutional Neural Network (glscnn) developed by NVIDIA. PilotNet takes pixels as input and produces a desired vehicle trajectory as output. This network is particularly useful for autonomous vehicle applications.
- **Translate the Deep Neural Network:** The next step is to translate or train the network according to the requirements of specific neuromorphic hardware. In this thesis, I focus on three platforms: Brainchip Akida, Intel Loihi, and Synsense. Each of these platforms has unique requirements and constraints that need to be addressed during the translation process.
- **Conversion Process Analysis:** This step involves analyzing the conversion process in detail. The goal is to identify key parameters and conditions that affect the conversion of ANNs to SNNs within each framework. This analysis helps in understanding the challenges and optimizing the conversion process.
- **Performance Metric Evaluation:** Finally, performance metrics are defined to evaluate the effectiveness of the conversion. For this study, Mean Squared Error (MSE) is used as a key performance metric. This metric helps in quantifying the accuracy and performance of the converted networks.
- **Conclusion:** By following this methodology, the thesis aims to systematically convert and analyze deep neural networks for neuromorphic processors, providing insights into their performance and potential improvements.

2 Structure and Outling of the Thesis

2.1 Thesis Structure

The **Literature Review** chapter begins by outlining the systematic literature review process, where the methodology for selecting and analyzing relevant studies is detailed. It continues with a discussion of related work, providing an overview of existing research in the field. This chapter sets the foundation for the study by identifying gaps in the literature and presenting the research questions that guide the research.

The **Theoretical Framework** chapter delves into the foundational concepts necessary for understanding the research. It starts with an introduction to ML and neural networks, explaining the basic principles that underlie these technologies. The chapter then focuses on CNN (glscnns) and (SNNs), highlighting their relevance to neuromorphic computing. Finally, we discuss the concept of neuromorphic computing, which is central to the research, providing the theoretical context for the subsequent chapters.

In the **Neuromorphic Hardware and Software Frameworks** chapter, the thesis explores the specific neuromorphic platforms used in the thesis. It covers three key processors: Brainchip Akida, Intel Loihi, and Synsense. Each section provides a detailed description of the architecture, capabilities, and relevance of these platforms to the thesis. This chapter is crucial for understanding the deails of the neuromorphic processors in which the study is conducted.

The **Implementation** chapter describes the practical aspects of the thesis. It begins with the identification of the neural network used in the study, specifically focusing on PilotNet. The chapter then details the model conversion pipeline for the three different neuromorphic chips, explaining the steps taken to adapt the DL models to each platform. This chapter bridges the theoretical concepts with practical application, demonstrating how the research is put into practice.

The **Results** chapter presents the findings of the thesis. It includes the performance metrics, accuracy evaluations, and any other relevant outcomes from the model conversion processes. This chapter is critical for assessing the effectiveness of the different frameworks and identifying key factors that influence performance.

Finally, the **Conclusion and Outlook** chapter summarizes the key findings of the research, reflecting on the implications for the field of neuromorphic computing. It discusses the limitations encountered and provides suggestions for future research, highlighting potential areas for further exploration. This chapter ties together the entire report, offering a coherent conclusion and setting the stage for future research in this area.

This structure ensures a logical flow from the background and theoretical foundations through to the practical implementation and analysis, providing a final conclusion that reflects on the research as a whole.

3 Literature Review

To ensure a comprehensive review of the literature, a systematic approach was adopted following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines [9].The PRISMA statement gives detailed guidance on how to formulate the research question and set inclusion and exclusion criteria, conduct an exhaustive search in the literature, screen relevant studies, and synthesize the results. The process involved several key steps as detailed below:

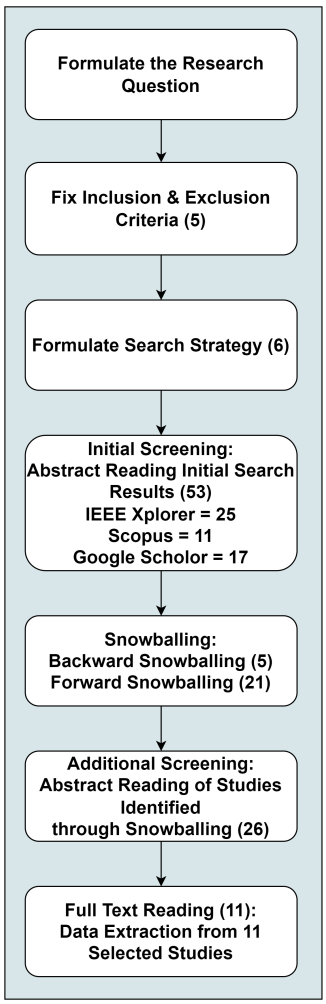

Figure 3.1: PRISMA Flow Diagram for a Systematic Literature Review

3.1 Systematic Literature Review Process

1. **Formulation of Research Question:**
   The first step involved clearly defining the research question to guide the systematic review. This step ensured that the scope of the review was well-defined and focused on addressing the specific objectives of this study.
2. **Fixing Inclusion and Exclusion Criteria:**
   Inclusion and exclusion criteria were established to filter the studies relevant to the research question. This step was crucial to ensure that only studies meeting specific quality and relevance standards were included in the review.
3. **Formulating the Search Strategy:**
   A robust search strategy was created to identify relevant studies on various databases. The search terms and keywords were carefully formulated to capture the breadth of the research topic. Proper steps were taken to ensure all-inclusiveness of the search result.
4. **Initial Screening:**
   An initial screening was conducted by reading the abstracts of the search results. The search covered three major databases:
   - **IEEE Xplore:** 25 studies
   - **Scopus:** 11 studies
   - **Google Scholar:** 17 studies

   In this phase, studies that did not meet the inclusion criteria or were deemed irrelevant to the research question were excluded.
5. **Snowballing:**
   To ensure that no significant studies were overlooked, snowballing techniques were employed. This involved:
   - **Backward Snowballing:** Reviewing the references of the selected studies to identify additional relevant literature.
   - **Forward Snowballing:** Identifying newer studies that cited the selected papers.

   This process resulted in the identification of an additional 26 studies, further enriching the review.
6. **Additional Screening:**
   An additional screening of the studies identified through snowballing was performed. The abstracts of these studies were reviewed to determine their relevance and adherence to the inclusion criteria.
7. **Full-Text Reading and Data Extraction:**
   Following the additional screening, full-text reading was conducted on 11 selected studies. Detailed data extraction was performed from these studies to gather insights and key findings relevant to the research question.

This systematic approach provided a rigorous and structured framework for the literature review, ensuring that the most relevant and high-quality studies were considered. The resulting review offered a comprehensive overview of the current state of research in the field, identified gaps, and provided a solid foundation for the subsequent sections of the thesis.

While performing a systematic literature review, especially in a technical area like neuromorphic computing or DL, there is a need to set up inclusion and exclusion criteria. These criteria are of very importance in making sure that the studies to be included in the literature review are relevant, have high quality, and are directly applicable to the research question.

The first criterion of inclusion in this systematic review was that the studies had relevance to the thesis topic. The second filtering criterion refered to the types of studies to be included. Only those studies that are directly connected with neuromorphic computing, in particular, those related to the conversion of DL models into SNNs, have been included. Moreover, studies applying either SNNs or DL models in autonomous systems, robotics, or edge computing have been taken into consideration since they are relevant for the research focus. We also took into consideration studies that mention or include specific neuromorphic platforms, such as Brainchip Akida, Intel Loihi, and SynSense.

In terms of the type of research, priority was given to peer-reviewed journal articles and conference papers, as these typically undergo rigorous evaluation. Publication date was another crucial factor. The focus was on recent publications from the last few years to ensure that the information is current and reflects the latest advancements in the field. However, if applicable, older seminal papers that have significantly influenced the field and are frequently cited were also included.

Accessibility of the full text is another consideration. Only studies where the full text is available should be included, as this allows for a comprehensive evaluation of the methodology, results, and conclusions. Additionally, the review should be limited to studies published in English to ensure that the research can be fully understood and critically evaluated.

Exclusion criteria are equally important and begin with irrelevance to the core research focus. Studies that only tangentially relate to the research question, such as those focusing on unrelated AI models, unrelated hardware, or non-technical applications of neuromorphic computing, were excluded. Similarly, studies that focused on platforms or hardware that were not relevant to the thesis (e.g., GPUs or TPUs not involved in neuromorphic processing) were excluded.

Duplicate studies or papers that present the same data set or findings without significant new analysis or results were excluded to avoid redundancy. Studies where only the abstract is available, without access to full methodology and results, were excluded.

Snowballing techniques were employed to ensure that no significant studies were overlooked. In backward snowballing, the references of included studies are reviewed to identify additional relevant literature. Focus was placed on identifying other highly cited works that have significantly impacted the field. Forward snowballing involved looking at more recent studies that have cited the included papers, applying the same inclusion criteria to determine relevance.

During the final screening for full-text reading, the relevance of studies selected for full-text reading were reconfirmed to ensure that they provided meaningful contributions to the thesis. This stage focused on extracting data related to performance metrics, conversion methodologies, and challenges specific to neuromorphic computing platforms.

Finally, data extraction criteria included key metrics such as MSE or other relevant accuracy measurements. Specific challenges discussed in the literature and any proposed solutions were identified and documented.

By applying these inclusion and exclusion criteria, the literature review was focused, rigorous, and directly aligned with the objectives of the thesis. These criteria ensured that we included the most relevant, high-quality studies while systematically excluding those that do not meet the necessary standards.

3.2 Research Questions

The thesis is guided by the following research questions, which are central to understanding the challenges and performance considerations in converting standard Neural Networks (NNs) to (SNNs):

- **How do existing frameworks perform in terms of accuracy when converting the standard Neural Networks to (SNNs), and what are the primary factors affecting their performance?**

  This question aims to evaluate the accuracy of various frameworks used in the conversion process from traditional NNs to SNNs. It also seeks to identify the key factors that influence the performance of these conversions, providing insights into the effectiveness and reliability of different approaches.

- **What are the current limitations and shortcomings in these conversion processes, and how can these insights be employed for metric-based analysis and evaluation of neuromorphic processors?**

  This question addresses the existing challenges and constraints in the conversion processes. It further explores how understanding these limitations can contribute to the development of metric-based analyses, ultimately aiding in the evaluation and optimization of neuromorphic processors.

3.3 Related Work

Some of the relevant work helped in building the foundation for the thesis research. The analysis of the related work aimed to help develop a standardized pipeline for ANN-to-SNN conversion and evaluate its performance on neuromorphic processors.

3.3.1 ANN-to-SNN Conversion Challenges and Techniques

The conversion of ANNs to SNNs presents several challenges, including accuracy loss, prolonged inference time, and the efficient handling of batch normalization layers. Li et al. [10] identified significant accuracy losses when converting ANNs with batch normalization layers to SNNs without proper calibration. They proposed a layer-wise calibration technique that adjusts weights, biases,

and initial membrane potentials to mitigate these issues. Their method significantly improved SNN accuracy, especially in models with batch normalization, making it a critical consideration for developing a standardized conversion pipeline.

Similarly, Ding et al. [11] introduced a Rate Norm Layer to facilitate the conversion of ANNs to SNNs by approximating the ReLU activation function with a rate-code. This method minimizes the loss of information during conversion and enhances the accuracy and speed of SNN inference. They also optimized inference speed by quantifying the fit between the ANN activation and the SNN firing rate, demonstrating that optimized SNNs can achieve fast and accurate inference, which is vital for real-time applications on neuromorphic hardware.

3.3.2 Energy Efficiency and Neuromorphic Hardware

Energy efficiency is a critical advantage of deploying SNNs on neuromorphic processors. Chandarana et al. [12] discussed the importance of energy-efficient deployment of ML models, especially in edge computing scenarios. They demonstrated that SNNs deployed on Intel's Loihi neuromorphic processor can achieve significant reductions in power and energy consumption compared to traditional deep neural network accelerators. This work highlights the potential benefits of SNNs in energy-constrained environments, supporting the exploration of neuromorphic processors in the thesis.

The study by Zhang et al. [13] also emphasized energy efficiency, particularly in the context of Physics-Informed Neural Networks (PINNs). By converting PINNs to SNNs, the authors leveraged the energy-efficient computation capabilities of SNNs for scientific ML applications. Their approach is particularly relevant for this research, as it explores the potential of SNNs in scenarios requiring both high computational efficiency and accuracy.

3.3.3 Benchmarking Neuromorphic Hardware

The deployment of SNNs on various neuromorphic edge devices has been explored in several studies. Ziegler et al. [14] benchmarked the performance of SNNs on the Dynapglscnn from SynSense, the Akida from BrainChip, and Intel's Loihi2. Their results showed significant differences in accuracy, run-time, and energy efficiency across these devices, providing valuable insights into the capabilities and limitations of current neuromorphic hardware. This benchmarking is essential for evaluating and comparing different neuromorphic processors, which is a key objective of the thesis.

In another study, Li et al. [10], there was a comparison of various ANN-to-SNN conversion techniques, including threshold balancing, weight normalization, and SpikeNorm, showing that their proposed calibration method outperforms existing approaches in terms of accuracy and inference latency. Such comparisons are crucial for selecting the most effective conversion framework for deployment on neuromorphic hardware.

3.3.4 Direct Training vs. Conversion of SNNs

The training of SNNs can be approached in two ways: direct training using spike-based learning rules or converting pre-trained ANNs to SNNs. The paper by Ziegler et al. [14] discusses these approaches, focusing on the use of surrogate gradients to train SNNs directly, which is necessary to approximate gradients in non-differentiable spike functions. Direct training often yields better accuracy but requires more computational resources, making it less suitable for resource-constrained environments compared to conversion-based approaches.

3.4 Overview

Despite the advances in ANN-to-SNN conversion and neuromorphic hardware, several challenges prevail. The papers reviewed highlight issues such as the accuracy gap between SNNs and traditional ANNs, especially in complex tasks like object detection. There is also a need for further optimization of SNN architectures and training algorithms to fully exploit the potential of neuromorphic hardware. Future research should focus on bridging these performance gaps and developing more robust and scalable SNN frameworks [14] [10].

The existing literature provides a solid foundation for the research presented in this thesis. The reviewed works illustrate the current state of ANN-to-SNN conversion techniques, the benefits of SNNs in terms of energy efficiency, and the challenges of deploying these networks on neuromorphic hardware. This thesis builds on these insights to develop a standardized pipeline for ANN-to-SNN conversion, addressing the identified limitations and contributing to the advancement of neuromorphic computing.

4 Theoretical Framework

4.1 Machine Learning

ML is a branch of AI that focuses on developing algorithms and statistical models that enable computers to perform specific tasks without being explicitly programmed. This ability to learn from data and improve over time is what makes ML particularly powerful. ML algorithms have widespread applications in various fields, including data mining, image processing, and predictive analytics. The primary advantage of ML is that once an algorithm has learned from the data, it can perform tasks automatically, making it highly efficient for handling large datasets and complex patterns [15].

There are several types of ML algorithms, each suited to different kinds of tasks. Supervised learning algorithms, such as decision trees and support vector machines, rely on labeled training data to learn a function that can predict outputs for new inputs. In contrast, unsupervised learning algorithms, like k-means clustering and principal component analysis, find hidden patterns or intrinsic structures in input data without labeled responses. Semi-supervised learning, which combines aspects of both supervised and unsupervised learning, can be particularly useful when labeled data is scarce but unlabeled data is abundant. Additionally, reinforcement learning focuses on how agents should take actions in an environment to maximize cumulative rewards, often used in scenarios requiring decision making over time, such as robotics and game playing [15].

Recent advancements in ML have significantly impacted various sectors, from healthcare to finance, driven by the development of sophisticated algorithms and the availability of large datasets. One notable area of progress is the rise of DL, particularly deep neural networks, which have revolutionized fields such as computer vision, speech recognition, and natural language processing. These DL systems utilize large-scale neural networks with multiple layers, capable of learning intricate patterns from vast amounts of data. The integration of parallel computing architectures, like GPUs, has enabled the training of these complex models on extensive datasets, leading to remarkable improvements in performance and accuracy [16]. While ML provides the algorithms and models for AI, neuromorphic computing offers a promising hardware foundation that can make these algorithms more efficient and applicable to new domains, particularly those requiring low power and real-time processing [17].

As ML continues to evolve, these advancements promise to further enhance its applications and integration into various industries.

4.2 Neural Networks

Neural networks, a cornerstone of modern ML, acts as a computational models inspired by the human brain's architecture. These models comprise of interconnected layers of nodes or neurons that process data in a manner similar to biological neural networks. Neural networks are proficient at handling a variety of tasks, including classification, regression, and pattern recognition. The ability to learn from data and generalize from examples makes them powerful tools for tasks such as image and speech recognition, natural language processing, and complex decision-making processes [18].

The fundamental components of an NN include nodes (neurons), connections (synapses), weights (synaptic strength), and activation functions, which control the neuron's output. These networks can be configured in various architectures, including feedforward networks where connections flow in one direction from input to output layers, and recurrent networks with connections that form cycles, allowing for feedback and temporal data processing [19].

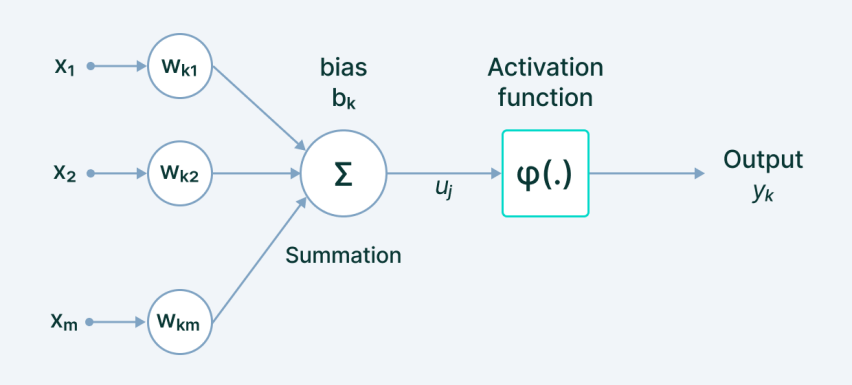

Figure 4.1: Neural Network Structure

In a neural network, the structure typically consists of several key components:

1. **Inputs (x)**: These are the initial data points fed into the network. Each input is represented by a variable, such as $x_1, x_2, \ldots, x_m$.
2. **Weights (w)**: Each input has an associated weight $(w_0, w_1, \ldots, w_m)$ which determines the importance of that input in the calculation of the output. The weight w_0 is often called the bias.
3. **Net Input Function** (Σ): This function computes the weighted sum of all the inputs. Mathematically, it is represented as:

$$\text{Net Input} = \sum_{i=0}^{m} w_i \cdot x_i$$

   Here, w_i are the weights and x_i are the inputs.
4. **Activation Function**: After calculating the weighted sum, the result is passed through an activation function. The activation function applies a non-linear transformation to the input, helping the network to learn complex patterns. Common activation functions include sigmoid, tanh, and ReLU (Rectified Linear Unit).

5. **Output**: The final value after applying the activation function is the output of the neuron. This output can then be used as an input to other neurons in subsequent layers or as the final prediction of the network.

This structure allows neural networks to process input data, learn from it through adjusting weights, and make predictions or classifications based on the learned patterns.

4.3 Convolution Neural Network

CNN are a class of deep neural networks primarily used for image and video recognition tasks. They are designed to automatically and adaptively learn spatial hierarchies of features from input images. This is achieved through the use of multiple layers, each comprising neurons that process and transform the input data.CNNs are particularly well-suited for image processing because they can capture spatial hierarchies through the use of local connections and weight sharing [20] [21].

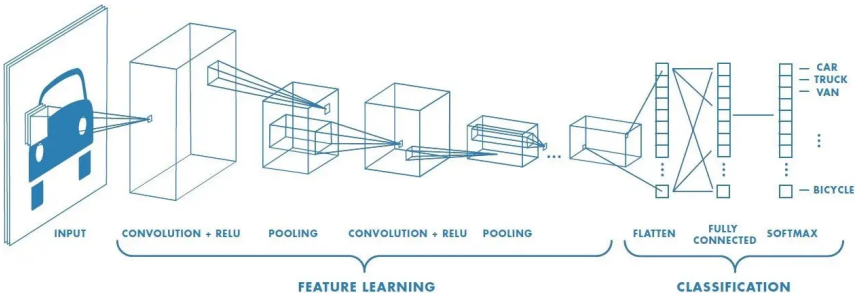

Figure 4.2: Convolutional Neural Network for Classification Task

A CNN typically consists of three main types of layers: convolutional layers, pooling layers, and fully connected layers. The convolutional layer is the core building block, where a set of filters (or kernels) is applied to the input image to produce feature maps. Each filter is convolved across the width and height of the input, computing the dot product between the entries of the filter and the input, and producing an activation map. This allows the network to learn different features at various levels of abstraction. Pooling layers are then used to reduce the dimensionality of the feature maps, which helps to decrease the computational load and prevent overfitting. Finally, fully connected layers are used at the end of the network to perform classification based on the extracted features [20].

Neuromorphic computing, inspired by the structure and function of the human brain, aligns closely with the principles of CNNs. Neuromorphic systems aim to replicate the neural architecture and synaptic connectivity of the brain in hardware, leading to more efficient and adaptive computing systems. These systems use specialized hardware to mimic the operations of neurons and synapses, which can significantly enhance the computational speed and energy efficiency of tasks such as real-time image processing and autonomous navigation. By integrating CNN principles into neuromorphic hardware, it is possible to create powerful, brain-like computational models that excel at complex pattern recognition and decision-making tasks [21].

4.4 Neuromorphic computing

The fields of DL and ML have experienced enormous growth over the past few decade, tackling complex problems and achieving groundbreaking results across various domains. From natural language processing to medical diagnosis, the influence of ML algorithms has been profound. However, traditional computing architectures used to implement these algorithms have inherent limitations, particularly noticeable in applications such as wearable devices and bioimpedance-based sensory systems. The separation of processing units and memory in these conventional architectures results in significant energy consumption and communication delays, which are critical issues for these applications.

Neuromorphic computing presents a new approach aimed at creating more energy-efficient and faster computing systems by emulating the human brain's information processing and storage methods. This approach marks a departure from the traditional Von Neumann architecture, which has been the foundation of modern computing for decades. In the Von Neumann architecture, the separation of memory and processing functions leads to communication inefficiencies and high energy consumption [22]. In the human brain, information processing and storage are integrated into a single entity, enabling fast and energy-efficient processing. Neuromorphic computing aims to replicate this integration using novel hardware and software systems designed to mimic the functions of neurons and synapses in the brain [23]. A key difference between traditional computing and neuromorphic computing lies in how information is stored and processed. Traditional computing stores data in memory, which the processor accesses to perform computations. This constant communication between memory and processor consumes significant energy and introduces delays. In contrast, neuromorphic computing systems store and process data in a single physical location, eliminating the need for continuous communication between memory and processor. This results in a more energy-efficient system capable of faster computations compared to traditional methods.

Additionally, neuromorphic computing excels under low-power conditions, making it ideal for applications where power consumption is a concern. These systems can employ energy-efficient algorithms that process and store information in parallel, conserving energy. This makes neuromorphic systems particularly suitable for portable and embedded devices, such as mobile phones, laptops, wearable technology, and Internet of Things (IoT) devices, where power consumption and size are critical factors [24].

In neuromorphic computing systems, computation is carried out using artificial neurons and synapses, which are modeled after their biological equivalents. These artificial neurons and synapses are integrated into the same physical location and are designed to replicate the functions of their biological counterparts. Neuromorphic computing can be implemented using both digital and analog circuits. Digital neuromorphic systems employ digital circuits to realize artificial neurons and synapses, whereas analog neuromorphic systems use analog circuits. Analog neuromorphic systems are known for their lower power consumption compared to digital systems, making them ideal for low-power applications.

One of the significant challenges in neuromorphic computing is developing scalable systems capable of performing complex computations. Currently, neuromorphic systems have limited computational capabilities, and ongoing research aims to create more advanced neuromorphic systems that can handle complex tasks.

4.5 Spiking Neural Network

Numerous ML algorithms have been designed for stream learning. However, most standard models need retraining in dynamic environments and struggle to scale due to their inherent learning algorithms. Recently, Artificial Neural Network (ANN)s, inspired by the biological processes through which the brain acquires and processes sensory information, have been employed to address rapidly changing information streams. The SNN is a biologically realistic neuron model that excels at capturing the dynamics among real biological neurons, enabling more precise and powerful computation by integrating various information dimensions into a single model and handling large volumes of data [25]. SNNs are considered the third generation of ANNs and are capable of learning continuously and incrementally, making them well-suited for non-stationary and evolving environments and effective as drift detectors. Additionally, SNNs have proven their ability to capture temporal associations between variables in streaming data. They represent the latest advancement in AI, utilizing ML techniques to train models within the spike domain [26].

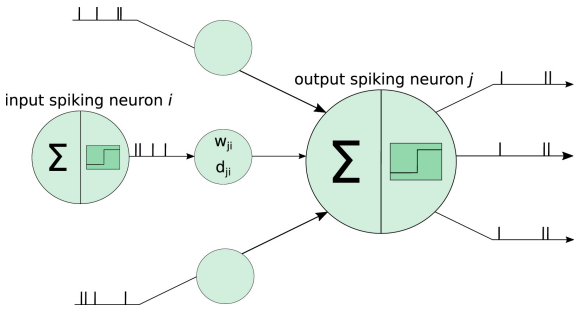

Figure 4.3: Spiking Neuron

SNNs learn from trained data in a manner similar to traditional ANNs. Although SNNs are a relatively new area of study, considerable research has been conducted to understand how biological neurons learn and apply this knowledge in training SNNs for various tasks.

In biological neurons, learning primarily occurs through the strengthening and weakening of synapses. Simply, when an incoming spike leads to an output spike, the greater the synaptic weight between the neurons, the stronger the connection. This synaptic modification is a fundamental aspect of biological learning, observed experimentally. Furthermore, research suggests that the addition or removal of synapses also contributes to the learning process.

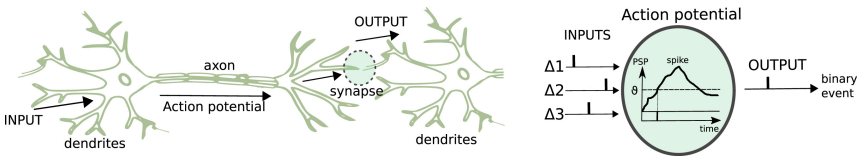

Figure 4.4: Biological Neuron and its Association with an Artificial Spiking Neuron

One model for biological learning that has garnered significant attention is Spike-Timing-Dependent Plasticity (STDP) [27].This model posits that if a presynaptic neuron fires just before a postsynaptic neuron, the connection between them will be strengthened, whereas if the postsynaptic neuron

fires before the presynaptic neuron, the connection will be weakened. Although STDP has shown promise in early research, developing a large, complex functional system using this model has proven to be more challenging than initially anticipated.

Despite these challenges, researchers have continued to advance the field of SNNs. A major development has been the application of supervised learning algorithms, such as the backpropagation algorithm, to train SNNs. This approach allows for the adjustment of synaptic weights in a manner that minimizes the error between predicted and actual outputs. One of the notable advantages of SNNs is their ability to model the brain's asynchronous and parallel information processing, making them a promising solution for real-time application such as image and speech recognition, where traditional ANNs can struggle. In addition, SNNs can be implemented in hardware much more efficiently than traditional ANNs, making them well-suited for applications where power consumption is a concern.

Data and information are represented as spikes in SNNs. Before input data can be fed into an SNN, it must first be encoded into spike trains, which are spatio-temporal patterns of spikes representing the input stimuli. This encoding process is still an open question in neuroscience, with ongoing debates about the information contained in these spiking patterns and the coding mechanisms neurons use to transmit this information. However, traditional research indicates that most relevant information is typically contained in the neurons' mean firing rate [25].

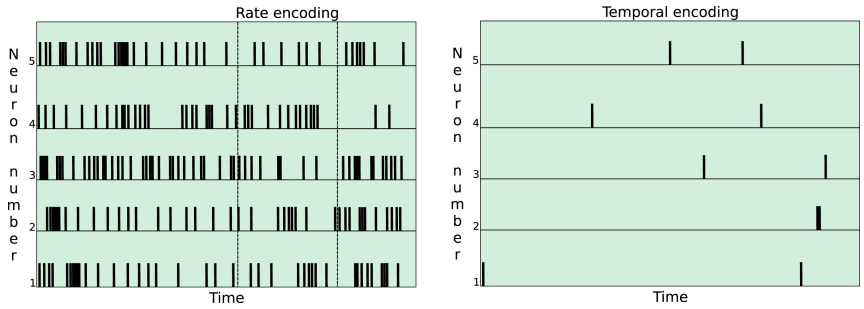

Figure 4.5: Rate-based encoding versus Temporal encoding

There are two primary encoding schemes for converting input data into spike trains: temporal encoding and rate-based encoding [28]. These are also known as temporal coding and rate coding, respectively. Temporal encoding is utilized when patterns within the encoding window convey information about the stimulus that cannot be inferred from the spike count alone. This method relies on the timing of spikes and includes approaches such as time-to-first-spike, where the timing of the first spike contains all the information about the new stimulus; phase coding, which applies time-to-first-spike encoding when the reference signal is a periodic event rather than a single event; and correlations and synchrony, which use spikes from other neurons as reference signals for spike coding [25]. In contrast, rate-based encoding is based on the spiking characteristics within a time interval, such as frequency, and encompasses three notions of mean firing rate: spike count, spike density, and population activity [28]. This scheme is employed when information is encoded in the neuron's mean firing rate rather than the timing of individual spikes.

The choice of encoding scheme depends on the specific characteristics of the input data and the task at hand and can impact the performance of the SNN.

Despite the exciting advancements in the field of SNNs, researchers still face significant challenges in training them effectively. These challenges include integrating biologically inspired learning rules, such as STDP, into the training process, and managing the high-dimensional, time-varying nature of spike data. Nevertheless, SNNs are highly esteemed within the online learning research community due to their adaptability and accurate emulation of brain-like information processing, making them well-suited for high-performance hardware platforms. Although the field of SNNs is still in its infancy, there is substantial promise for their application across a wide range of domains. Researchers will continue to leverage insights from both biological and ML studies to develop more effective training methods for SNNs, enhancing their applicability and performance.

5 Neuromorphic Hardware

ML algorithms along with DL workloads are increasingly being mapped onto neuromorphic hardware in recent years. The main aim is to use the inherent energy efficiency and low power consumption of neuromorphic systems while still performing complex computations. The use of ML and DL algorithms in domains such as image and speech recognition, autonomous driving, and natural language processing has surged over the last decade. However, these algorithms require a lot of energy and computational power which makes them very demanding, thus making it hard for conventional hardwares. Neuromorphic hardware designed to replicate the energy-efficient information processing in human brains appears a promising solution to these challenges. A number of ways have been suggested for mapping ML workloads into neuromorphic hardware each having unique objectives and tradeoffs. Notable mapping methods are Corelet [29] that maps SNNs onto TrueNorth hardware[30] while PACMAN maps SNNs onto SpiNNaker [31]. Another approach is PyNN [32], a tool for mapping SNNs onto different hardware platforms including Loihi, BrainScaleS, and Neurogrid [33] where the load is balanced across tiles respectively.

The goal of these techniques is to ensure a proper load balancing that can distribute evenly the neurons and synapses thereby satisfying the computational demands but reducing energy loss. In addition to load balancing, recent methods have investigated further purposes. For instance, PSOPART tries to minimize power utilization on common buses by adapting SNNs in neuromorphic hardware whereas SpiNeMap carries out a corresponding freeclustering of SNNs mapping them to require lower inter-tile communication [34]. TrainSNN improves cluster efficiency through SNN decomposition [35]. Making use of energy-responsive neuromorphic architectures enables carrying out intricate computations at a minimum energy level which preserves the speed of operations for ML algorithms. Each large-scale SNN simulation technique possesses its own merits and demerits. One can select the most appropriate method for a given task by looking into factors like model neuron and synapse technology used, communication topology employed, or whether there exists support for synaptic plasticity among others.

5.1 Brainchip Akida

One of the leading companies in the neuromorphic technology sector is BrainChip, known for developing Akida, a neuromorphic processor IP designed for processing sensor data with exceptional efficiency, precision, and energy savings. The Akida processor is a fully customizable, event-driven AI neural processor, capable of supporting up to 256 nodes interconnected through a mesh network [36]. Its scalable architecture and compact design result in significant performance improvements over traditional Von Neumann architectures. Central to Akida are its Neural Processing Units (NPUs), organized into nodes, each containing four NPUs with scalable and configurable SRAM. These NPUs can be configured as either convolutional or fully connected, tailored to the specific needs of the application [36]. One of the main advantages of Akida is its ability to capitalize on

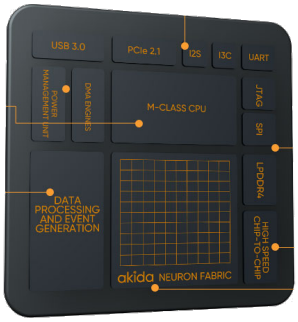

Figure 5.1: Akida AKD1000 Chip

data sparsity, activations, and weights to reduce the number of operations by at least 2X [36]. This is accomplished by processing only the essential data, rather than all data, including redundant and non-essential information. Additionally, Akida is designed for high energy efficiency because to its event-based architecture. By processing data only when an event occurs, Akida minimizes power consumption, making it particularly well-suited for applications with stringent power requirements, such as IoT devices.

Akida neuromorphic processors also feature on-chip learning, allowing the device to learn and adapt to new information in real time without relying on external computing resources. This capability is enabled by Akida's event-driven design, which facilitates efficient processing of sparse data and low-latency learning. On-chip learning in Akida opens up new possibilities for applications in areas like autonomous systems, sensory processing [37], and ML, enabling real-time adaptation and allowing devices to function in dynamic environments and respond to changing conditions. Furthermore, Akida's on-chip learning is notably energy-efficient, in contrast to traditional computing systems that require significant data storage and transfer for ML tasks, leading to higher energy consumption.

One of the primary advantages of on-chip learning in Akida is its energy efficiency. Traditional computing systems typically require substantial data storage and transfer to execute ML tasks, leading to significant energy consumption. In contrast, Akida's event-based architecture enables learning with minimal power usage, making it ideal for battery-powered or energy-constrained systems. Additionally, on-chip learning in Akida facilitates greater system integration and compactness. By performing learning locally, there is no need for external processors or memory components, reducing system size and complexity. Another benefit of Akida's on-chip learning is its ability to handle dynamic data streams effectively. Traditional computing systems often struggle with rapidly changing data, resulting in latency and inaccuracies. Akida's event-driven design, however, allows for real-time data processing, ensuring accurate and timely responses. Overall, Akida's on-chip learning represents a significant advancement in neuromorphic computing, enabling new applications and enhancing the performance and efficiency of existing ones.

One of the main challenges in mapping a compatible ML model to the Akida neuromorphic processor is ensuring that the model's structure and parameters align with Akida's event-based processing approach. Unlike traditional computing systems, where ML models typically process continuous data streams using floating-point arithmetic, Akida operates on event streams and utilizes a fixed-point arithmetic system. This difference can lead to variations in behavior and accuracy

compared to traditional systems. Another challenge is the limited memory resources available on Akida, which may necessitate reducing the model size or employing more efficient algorithms to ensure effective operation. Additionally, transitioning a ML model from software to a hardware implementation on Akida requires significant expertise in both ML and hardware design.

Mapping a ML model to Akida demands careful consideration of the model's structure, parameters, and computational needs, alongside an understanding of Akida's unique constraints and capabilities. It is also crucial to evaluate the power consumption and performance of the mapped model on Akida, as this can influence its practicality for real-world applications. Furthermore, Akida's current optimization for CNN presents another challenge, as support for other model types may be limited. The event-based processing approach of Akida might also result in different behavior and accuracy, complicating direct comparisons with traditional ML models. Despite these challenges, Akida's distinctive features, such as low power consumption and high performance, make it a promising platform for implementing ML algorithms. With advances in mapping techniques and the development of new algorithms, it may be possible to overcome these challenges and fully realize the potential of neuromorphic computing for bioimpedance sensing applications in the future.

5.2 Intel Loihi

Figure 5.2: Intel Loihi 2

Loihi is an innovative research chip developed by Intel that has the potential to revolutionize AI and ML. This advanced technology is designed to emulate the behavior of biological neurons and synapses, featuring a unique architecture that enables high-speed, low-power operation for AI and ML tasks. Loihi represents a significant leap forward in neuromorphic computing, a field focused on creating computing systems that mimic the structure and function of biological neurons and synapses [38]. The architecture of Loihi is built around a SNN, a type of ANN that uses spikes or pulse-like signals for operation. Unlike traditional Von Neumann architectures, which process information in a linear sequence, Loihi can perform computations in an event-driven manner. This allows it to respond to incoming signals in real-time and compute only when necessary, making it exceptionally energy efficient.

One of the standout features of Loihi is its capability for on-chip learning, meaning ML occurs directly on the chip itself rather than on an external computer. This allows the chip to adapt to changing inputs in real-time and enhance its performance over time. This learning ability is particularly advantageous for AI systems that must operate in resource-constrained environments,

such as IoT devices, autonomous robots, and wearable technology. In addition to its event-driven architecture and on-chip learning, Loihi boasts several other distinctive features that make it ideal for AI and ML applications. For instance, its hierarchical structure enables processing at various levels of abstraction, allowing it to perform complex computations more efficiently than traditional Von Neumann architectures. Moreover, Loihi's flexible interconnect structure allows it to be easily reconfigured for diverse applications. The combination of event-driven computation and on-chip learning makes Loihi an excellent choice for developing AI systems capable of functioning in real-world scenarios, where low power consumption and high computational efficiency are essential.

The second-generation Loihi 2 neuromorphic research chip is compatible with the Lava open-source software framework, which is designed to develop applications for neuromorphic hardware architectures. Although Lava currently runs on CPUs and Loihi chips, its compiler and runtime are extendable to other architectures. Access to Loihi 2 will primarily be provided through the Neuromorphic Research Cloud, which offers shared systems like the "Oheo Gulch," a single-chip system connected to an Aria 10 FPGA for early evaluations [39]. It will soon be accompanied by "Kapoho Point," a compact 4x4-inch, stackable 8-chip system with Ethernet connectivity. Intel has tested both Loihi 1 and 2 chips in various applications, including adaptive robot arm control, visual-tactile sensory perception, odor and gesture recognition, drone motor control with low-latency response to visual input, fast database similarity search, modeling diffusion processes for scientific computing, and solving optimization problems such as railway scheduling [40]. Additionally, the Loihi chip consumes significantly less power than standard CPU and GPU solutions, making it a promising candidate for neuromorphic AI acceleration. This could enable Loihi to provide datacenter-like capabilities for robots, autonomous vehicles, and other applications with lower power consumption and latency. The efficiency of SNNs in battery-powered sensors with integrated AI is also expected to benefit from lower-end neuromorphic chips. In various demonstrations, Loihi's power consumption has been measured at less than 1 watt, compared to the tens or hundreds of watts typically consumed by standard CPU and GPU solutions, highlighting Loihi's breakthrough in energy efficiency [40]. In many cases, these demonstrations have shown relative gains of several orders of magnitude, underscoring the vast improvements in energy efficiency that Loihi delivers. Furthermore, Loihi exhibits state-of-the-art response times to incoming data samples while continuously adapting and learning from data streams, making it ideal for the most demanding applications. The combination of low power consumption, low latency, and continuous adaptation positions Loihi to introduce new intelligent functionalities to power- and latency-constrained systems at a scale and versatility beyond what any other programmable architecture currently offers.

The Loihi 2 neuromorphic chip marks a substantial leap forward from the first-generation Loihi, offering several key enhancements. Notably, the Loihi 2 chip delivers up to 10 times faster processing capabilities, including a 2-fold improvement in simple neuron state processing, a 5-fold increase in synaptic operations, and a 10-fold enhancement in spike generation. Additionally, the Loihi 2 chip offers up to 60 times greater inter-chip bandwidth, achieved through a combination of 4 times faster inter-chip signaling speed, more inter-chip links (6 instead of 4), and over a 10-fold reduction in inter-chip bandwidth usage [40]. Furthermore, the Loihi 2 chip can support up to 1 million neurons, reflecting a 15-fold increase in resource density [40]. It is also scalable in three dimensions, with native Ethernet support and fully programmable neuron models with graded spikes. Lastly, the Loihi 2 chip features improved learning and adaptation capabilities. These significant advancements underscore the progress made in neuromorphic computing and hold great promise for future innovations in the field.

5.3 Synsense

Figure 5.3: SynSense Speck

SynSense is a neuromorphic computing platform that leverages an innovative architecture inspired by the biological structure of the human brain. Neuromorphic computing, a branch of AI, seeks to create computing systems that emulate the functioning of the human brain. What sets SynSense apart is its use of a vast array of simple processing units, each designed to replicate the behavior of a single neuron. The core component of the SynSense platform is the artificial neuron, modeled to mimic the characteristics of biological neurons. These artificial neurons receive inputs from other neurons, process them, and generate outputs that can be transmitted to other neurons. The processing done by these artificial neurons is grounded in mathematical models that draw inspiration from biological neuron behavior. SynSense employs a parallel processing architecture, enabling extensive computations to occur simultaneously [41].

One of the primary benefits of the SynSense platform is its energy efficiency. The simplicity of the individual processing units, designed to imitate biological neurons, allows them to operate with minimal power consumption. This makes the SynSense platform highly suitable for a wide range of applications, particularly in edge computing, where low power usage is critical. Moreover, the energy efficiency of SynSense makes it ideal for applications where traditional computing systems would be impractical, such as in wearable devices and IoT devices, including those for gesture recognition, face or object detection, location tracking, and surveillance [41].

Another significant advantage of the SynSense platform is its ability to handle complex and dynamic tasks. Unlike traditional computing systems, which are typically designed to perform specific tasks, the SynSense platform can adapt to new inputs and tasks in real time. This adaptability is due to the artificial neurons' ability to modify their behavior based on the inputs they receive. This feature makes the SynSense platform particularly well-suited for applications requiring real-time decision-making and problem-solving, such as in autonomous robots and vehicles.

The SynSense platform holds considerable promise for a wide range of applications in the healthcare sector. SynSense has developed a suite of hardware solutions designed for compact and energy-efficient neuromorphic bio-signal processing. These solutions are specifically optimized for ultra-low power sensory processing at the edge, typically consuming only a few milliwatts. They can continuously monitor critical body signals, such as electrocardiogram (ECG), electromyogram (EMG), and electroencephalogram (EEG), in real time from wearable devices, allowing for the instant detection of anomalies [41]. This technology could be instrumental in developing wearable devices for monitoring and diagnosing neurological disorders like epilepsy or Parkinson's disease. Additionally, the platform's capability to manage complex, dynamic tasks makes it a valuable asset for creating assistive technologies that support individuals with disabilities.

Furthermore, SynSense has successfully implemented ultra-low-power, always-on keyword and command detection using (SNNs) for auditory processing. This advanced technology is designed to process data close to the sensor, employing cutting-edge algorithms tailored to specialized processors. Overall, SynSense represents a promising platform for neuromorphic computing applications, offering energy efficiency, the ability to manage complex tasks, and versatility across a variety of applications. These qualities position SynSense as a significant advancement in and computing. While these multi-chip neuromorphic systems hold the potential to transform computing and AI, they also introduce new challenges and complexities. A key challenge is ensuring efficient communication between the various components of the system. To achieve real-time performance, data must be transferred between neuromorphic sensors and general-purpose neural network chips with minimal latency and high bandwidth.

Multi-chip neuromorphic systems can consume a substantial amount of power, posing a significant challenge for mobile and battery-powered applications. To overcome this, power-efficient design strategies, such as the use of low-power analog circuits and voltage scaling, will be crucial for creating practical and scalable neuromorphic systems. These devices have already shown the promise of neuromorphic computing across a broad range of applications, leading to the development of larger multi-chip systems. Although these systems bring new challenges, they also offer the potential for considerable advancements in computing and AI. With ongoing research and development, multi-chip neuromorphic systems could fundamentally change the way we interact with technology and each other.

6 Implementation

6.1 Identifying the Neural Network

For the conversion of a neural network to SNN, the initial step is to identify a neural network for the conversion process. Here we have chosen the PilotNet as a common model for the conversion process for three different chips. PilotNet model is a project by NVIDIA to develop a self-driving car using a convolutional neural network CNN that learns to steer directly from raw pixel data from a single front-facing camera. This approach is called ënd-to-end learning,äs it includes the CNN learning the entire processing pipeline from input (camera images) to output (steering commands) without any explicit feature extraction or human-designed rules [42].

The PilotNet Architecture

The Pilotnet model has 9 layers in total: 1 normalization layer, 5 convolutional layers, and 3 fully connected layers. It allows the car to learn how to drive directly from raw images, bypassing traditional methods like explicit lane detection or complex path planning. Instead, it learns by experience—using the data it has been trained on to make real-time decisions, just like a human driver learns from practice. Lets look into how each layers are designed.

- **Input:**
  - The model starts by taking an image of the road ahead, captured by the car's front-facing camera. This image is 66x200 pixels in size and is processed in YUV color space, which is a way of representing color that is particularly suited for images and videos.
- **Normalization Layer:**
  - The first step in the network is to normalize this image. We can think of normalization as a way to standardize the input, ensuring that no matter the lighting or weather conditions, the image is processed consistently. This layer is pre-programmed and doesn't change during learning.
- **Convolutional Layers:**
  - PilotNet then passes the image through five convolutional layers. These layers are where the network begins to understand and extract important features from the image—things like edges, shapes, and textures that are crucial for driving. Here's what each layer does:
    - The first three layers use larger filters (5x5 pixels) and strides (how much the filter moves across the image) to quickly break down the image into basic features.
    - The last two layers use smaller filters (3x3 pixels) to refine these features into something more detailed and useful for making decisions.

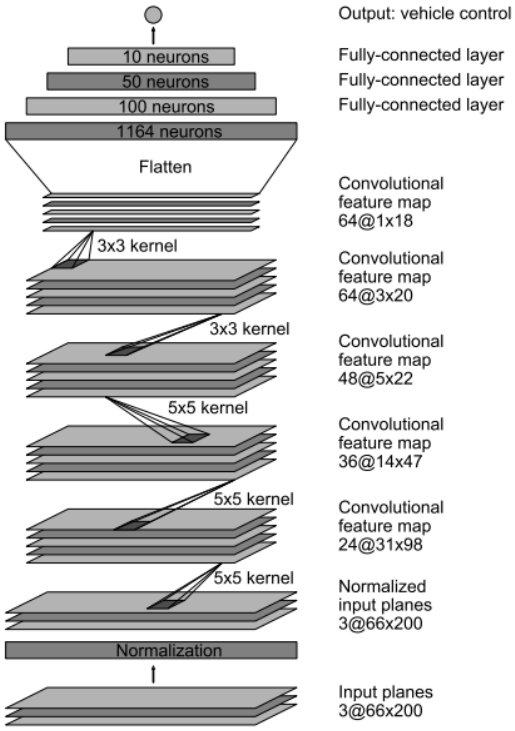

Figure 6.1: PilotNet Architecture

  - As the image moves through these layers, it's gradually reduced in size, but the information it carries becomes more and more focused on what matters for driving.

- **Fully Connected Layers:**
  - After the image has been processed by the convolutional layers, it's flattened into a single line of data and sent through three fully connected layers. These layers act like the decision-making part of the brain. They take all the extracted features and figure out the best way to steer the car.
    - The first layer has 1,164 neurons, which help in making the initial interpretation.
    - The second and third layers narrow down this information with 100 and 50 neurons, respectively, refining the steering command further.
    - The final output is a single value that represents how sharply the car should turn. This value is the inverse of the turning radius (1/r), a clever way to handle sharp turns and straight paths smoothly.
- **Output:**
  - The output from PilotNet is just one number, but it's crucial in telling car exactly how to steer. This design allows the car to navigate without needing to separately detect lanes or plan paths.
  - It has about 27 million connections and 250,000 parameters that adjust during training to improve driving.

6.2 Setting Cloud Environment for model Training

Creating an Azure VM for GPU and CUDA Installation

- This chapter describes the process that was carried out to set up an Azure Virtual Machine (VM) equipped with a GPU and install CUDA for GPU-accelerated tasks.
- The first step involved selecting a suitable Azure VM instance:
  - An N-series VM, specifically designed for GPU-accelerated tasks, was chosen.
  - It was essential to ensure that the VM used an NVIDIA GPU, as CUDA is an NVIDIA-specific technology.
  - For this implementation, the `Standard NC8as T4 v3` VM was selected.
  - The 'C' in the VM type code indicated that the VM was designed for computational tasks, making it ideal for our requirements.
  - V-series VMs, optimized for visual tasks, were deliberately avoided.
  - Due to quota restrictions and the global chip shortage, GPU-equipped VMs were carefully selected.
- Following the VM selection:
  - Ubuntu 22.04 was chosen as the operating system for its compatibility and reliability.
  - The security type was set to SStandard"to avoid potential issues with the installation of NVIDIA extensions and drivers, as Azure's default "Trustedßetting could have caused problems.
- Once the VM was operational:
  - The system was updated by executing the command `sudo apt-get update`.
  - The GNU Compiler Collection (GCC) was installed using `sudo apt-get install gcc`, as it was necessary for building CUDA applications.
  - Make, a build automation tool, was also installed with the command `sudo apt-get install make`.
- The next phase involved downloading and installing the CUDA Toolkit:
  - The CUDA Toolkit was downloaded from the official website.
  - The local installer was selected for its reliability compared to the network installer.
  - The toolkit was downloaded using the command `wget https://developer.download.nvidia.com/compute/cuc`
  - The installation was initiated with `sudo sh cuda_12.2.2_535.104.05_linux.run`.
  - During the installation, the terms were accepted, and the default settings were used.
  - This process successfully installed NVIDIA Driver version 535.104.05 and CUDA Toolkit version 12.2.2.
- To verify the installation:

- The command `nvidia-smi` was used, which provided details on the installed NVIDIA driver, CUDA version, and GPU utilization.
- The CUDA compiler installation was verified by running `nvcc --version`.
- Initially, if the `nvcc` command was not found, it indicated that CUDA was not in the system's PATH.
- To resolve this, the bash configuration file (`/etc/bash.bashrc`) was edited by adding the line `export PATH=$PATH:/usr/local/cuda/bin` at the end.
- After saving the file and rebooting the VM, the PATH update was confirmed with the command `echo $PATH`.
- Finally, the CUDA installation was rechecked by running `which nvcc` and `nvcc --version`, ensuring that the installation was successful.

This ensured that the cloud VM is setting for training our model.

6.3 Model Conversion Pipeline for Three Different Neuromorphic Chips

To deploy the base PilotNet model on three different neuromorphic chips—Intel Loihi, Synsense, and Brainchip—we must follow a conversion process specific to each chip. Below, we detail the conversion pipelines for each chip individually.

6.3.1 Intel Loihi

For deploying PilotNet on Intel Loihi, we use Lava, an open-source software framework designed for neuromorphic computing. Before diving into the conversion pipeline, it is essential to understand the key attributes of Lava and the reasons for its use in converting ANNs to (SNNs).

Introduction to Lava Framework

Neuromorphic systems, inspired by the brain's architecture, consist of neurons and synapses that operate in parallel and communicate via short, asynchronous signals known as spikes. These systems aim to replicate the brain's efficiency and computational power.

On a macro scale, neuromorphic hardware is composed of various components, from specialized neural accelerators to conventional CPUs/GPUs, sensors, and actuators. Lava's framework is designed to manage this complex, massively parallel architecture, making neuromorphic technology more accessible and user-friendly.

Currently, no other open software framework comprehensively integrates all these architectural elements. Lava fills this gap, providing an efficient, cohesive platform for neuromorphic computing [43].

In the Lava architecture, the core concept is the Process. We can think of a Process as a basic unit or building block that carries out a specific function. For example, in a neuromorphic system, one Process might represent a group of LIF (Leaky Integrate-and-Fire) neurons, which are a type of artificial neuron model.

Here is how a process works:

1. **Data (State):**
    - Every Process has its own set of data that reflects its current condition or status. We can think of it like a memory that keeps track of what is happening. For example, if we are simulating neurons, this data might represent things like the electrical charge within a neuron at any given moment.
2. **Algorithms (Behavior):**
    - A Process comes with its own set of instructions, or algorithms, that tell it how to handle its data. These instructions define what the Process does. For instance, in a simulation of neurons, the algorithm might describe how the neuron's charge changes over time based on incoming signals.
3. **Ports (Communication):**
    - Processes often need to share information with each other, and they do this through something called ports. Ports are like doors that allow data to move in and out, enabling Processes to communicate and collaborate. This is similar to how different parts of a brain send signals to each other to coordinate actions.
4. **API (Interaction):**
    - Each Process also comes with an API, which is like a control panel that users can use to interact with the Process. Through the API, you can start or stop the Process, or change its settings, making it easy to manage how the Process works.

In short, a Process in Lava is a compact, independent unit that manages its own data, follows its own instructions, communicates with other Processes, and can be controlled by the user. **Lava-DL**

Since our goal is to implement a pipeline for converting our ANN model to SNN, we utilize the Lava-DL which is a library of DL tools within Lava that support offline training, online training and inference methods for various Deep Event-Based Networks. Lava-DL is a specialized library within the Lava ecosystem designed to support the training and inference of Deep Event-Based Networks. Event-based networks, such as SNNs, are particularly well-suited for neuromorphic computing, which mimics the way biological brains process information. Lava-DL offers tools for both direct training of these networks and for converting traditional ANNs into SNNs.

Deep Event-Based Networks are specialized neural networks that process information in a manner similar to the brain, using discrete events or spikes, which allows for efficient handling of sparse data and precise timing. There are two primary training approaches: Direct Training, which utilizes the exact timing of events for high accuracy and efficiency but requires substantial resources and time, and ANN-to-SNN Conversion, which involves training a conventional ANN before converting it into a SNN. This latter approach benefits from the speed of ANN training but may require increased latency in the resulting SNN. Lava-DL provides several modules to support these processes, including SLAYER for direct training, which optimizes event timing in SNNs, Bootstrap,

which accelerates the ANN-to-SNN conversion process by reducing latency, and Netx, which facilitates the training and deployment of event-based deep neural networks across traditional and neuromorphic platforms. Additionally, the DECOLLE plugin extends Lava-DL's capabilities by offering advanced training strategies using local learning rules and surrogate gradients, enhancing the biological plausibility of learning in SNNs.

Notably, Lava-DL operates independently of the core Lava library, as Lava processes cannot currently be trained directly; instead, models are trained with Lava-DL and later converted into a format compatible with Lava using the Netx module, ensuring flexibility in training and deploying Deep Event-Based Networks across various platforms.

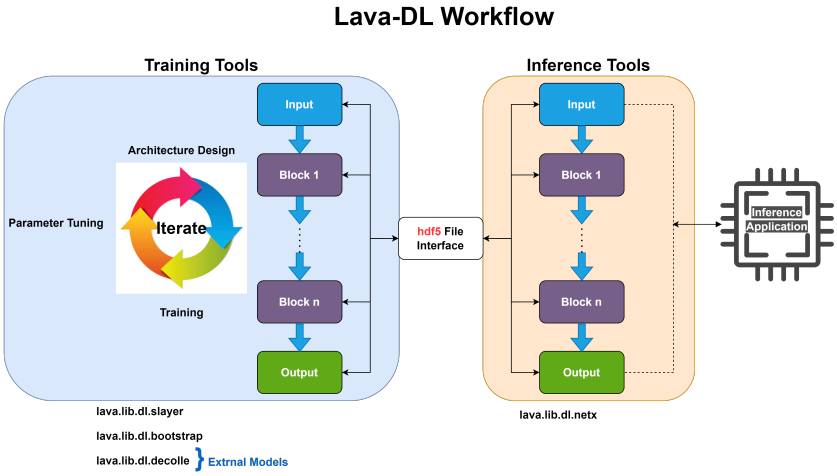

Figure 6.2: Workflow of Lava-DL

The Figure 6.2 illustrates the workflow of Lava-DL, depicting the process from training deep event-based networks to deploying them for inference. On the left side, the Training Tools section shows how the architecture of the neural network is designed, followed by the iterative process of parameter tuning and training across multiple blocks or layers of the network. Once the network is trained, it is saved in an hdf5 file, which acts as an interface between the training phase and the inference phase. The right side of the image focuses on the Inference Tools, where the trained model is loaded, and the input data is processed through a series of stages (Process 1 to Process n) to produce an output, which is then used in an Inference Application. This workflow is supported by various Lava-DL modules like SLAYER for direct training, Bootstrap for accelerated training, and Netx for deploying the trained models on different hardware platforms, including traditional and neuromorphic systems. The diagram also highlights the integration of external modules like DECOLLE for advanced local learning strategies.

With the enough background on lava, now we can look into the conversion pipeline for PilotNet using the Lava framework.

6.4 Conversion Pipeline for Intel Loihi

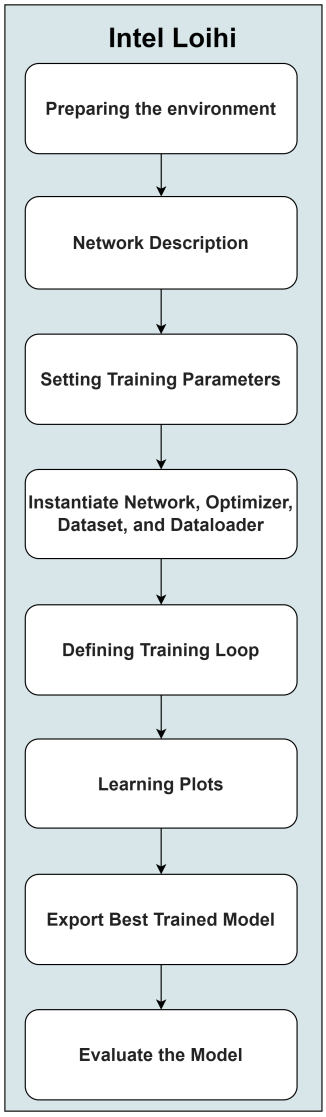

Figure 6.3: Conversion Pipeline for Intel Loihi

1. **Preparing the environment:**

   - Preparing an environment to train or evaluate a DL model using PyTorch and the Slayer library, for which they are tailored. The imports include common data science libraries (like NumPy and Matplotlib), PyTorch for DL, Slayer for translation, and custom modules specific to the dataset and utilities for the project. The manual seed is set to ensure that the experiments are reproducible.

2. **Network Description:**

   - Here we create the PilotNet neural network using SLAYER 2.0 (`lava.dl.slayer`), a library designed for (SNNs). SLAYER 2.0 provides neuron models, synapses, axons, and dendrites that support quantized training, allowing them to handle discrete rather than continuous data. To facilitate ease of use, it offers "block" interfaces that package these related components into modules, simplifying the creation of neural networks. Sigma-delta blocks are pre-configured modules (such as Dense, Conv, Pool, Input, Output, Flatten, etc.) that can be used to build various parts of this neural network. They manage tasks like convolution, pooling, and dense layers while also offering features like synaptic weight normalization and neuron normalization. These blocks support typical operations in a SNN and can be seamlessly integrated into a network using standard PyTorch procedures.

3. **Setting Training Parameters:**

   - Here we initialized key hyperparameters and settings for training the neural network, including batch size, learning rate, penalty for high event rates, and the number of training epochs. We also set up directories for saving the trained models and logs, ensuring they are created if they don't exist. Finally, we specify that the training will take place on a GPU, which is essential for handling the computational demands of DL.

4. **Instantiate Network, Optimizer, Dataset, and Dataloader:**

   - We perform four main steps here. We instantiate the network to the specified device, typically a GPU (`device = torch.device('cuda')`), which is crucial for efficient training. An optimizer is instantiated using the RAdam optimizer, a variant of the Adam optimizer with rectified learning rates that can provide more stable and faster convergence. We then prepare the two datasets using the `PilotNetDataset` class: one for training (`train=True`) and one for testing (`train=False`). DataLoaders are created for both the training and testing datasets. The DataLoader is responsible for loading data in mini-batches during training. Finally, we initialize `LearningStats`, an object to track and manage learning statistics, such as loss and accuracy, during training. The `Assistant` class from `slayer.utils` is used to manage the training process. It simplifies the workflow by integrating the network, loss function, optimizer, and statistics tracking.

5. **Defining Training Loop:**

   - We describe the training loop for a neural network using the assistant utility provided by the `slayer.utils` module at the previous step. The loop handles the process of training the model, testing its performance, adjusting the learning rate, saving the best model, and periodically saving checkpoints and statistics.

6. **Learning Plots:**

   - This step is used to visualize the learning curves of our model during or after training. These plots provide valuable insights into the training process, allowing us to assess the model's performance and make necessary adjustments if any issues are noticed. The specified figure size ensures that the plot is clearly readable and well-proportioned.

7. **Export Best Trained Model:**
    - At this step, the best-trained model is loaded from a saved state and exported to an HDF5 file. This file format is particularly useful for integration with the Lava framework, where the model can be loaded and run as a neuromorphic process. The use of HDF5 ensures that the model's parameters and architecture are stored efficiently, making it easy to deploy or reuse the trained network in future applications.
8. **Evaluate the Model:**
    - This code is used to evaluate the computational efficiency of a trained STDP-based spiking deep neural network (SDNN) by measuring neuron activity and synaptic operations on the testing dataset. By comparing these statistics with those of an ANN of the same architecture, we can assess the benefits of using SDNNs in terms of reduced computational load. The SDNN is expected to show lower neuron activity and synaptic operations due to its event-driven nature, potentially leading to more efficient computation compared to a traditional ANN. The final averaged counts provide a quantitative measure of this efficiency.
        - **Synaptic Operations:** These refer to the number of operations involving the connections (synapses) between neurons. In a SNN like an SDNN, these operations are event-driven, meaning they only occur when a neuron spikes, leading to potential computational savings.
        - **Neuron Activity:** This refers to the frequency at which neurons activate or ßpike"during the processing of inputs. Lower neuron activity in an SDNN compared to an ANN can indicate more efficient computation.

The implementation pipeline for PilotNet using Lava for Intel Loihi is successfully defined here.

6.4.1 Brainchip Akida

Implementation of the Pilotnet code on the Akida neuromorphic processor includes the usage the framework MetaTF. The Akida Development Environment (MetaTF) is a comprehensive ML framework designed for the effortless training, creation and testing of neural networks on the Akida Neuromorphic Processor Platform. MetaTF features an Akida Neuromorphic Processor IP simulator for model execution, as well as hardware implementations like the AKD1000 reference SoC. Since they are modeled after the Keras API, MetaTF offers a high-level Python API for working with neural networks. This API supports the initial design, evaluation fine-tuning, and final productization of neural network models [44]. MetaTF directly trains an ANN approximating a SNN through quantization [45].

The framework consists of three main Python packages: the Akida package, which interfaces with the Brainchip Akida Neuromorphic System-on-Chip (NSoC) and provides a runtime, a Hardware Abstraction Layer (HAL), and a software backend for simulating the Akida NSoC; the cnn2snn tool, which converts glscnn models trained with DL techniques into low-latency, low-power event-based networks compatible with the Akida runtime; and the Akida model zoo, which offers pre-built network models created using the Akida sequential API and the cnn2snn tool, including quantized Keras models [46].

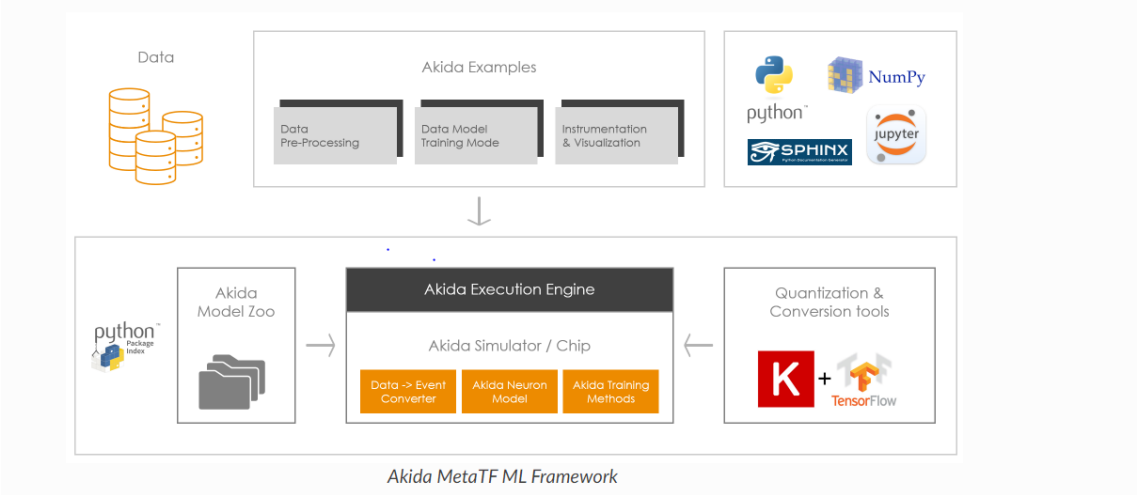

Figure 6.4: Framework of Akida MetaTF ML

The Figure 6.4 illustrates the Akida MetaTF ML Framework, showcasing its components and workflow. The framework begins with data input, which undergoes pre-processing and model training within the ”Akida Example” section. This process involves Python-based tools like NumPy, Jupyter, and Sphinx for data handling and visualization. The trained models and data are then processed through the ”Akida Execution Engine,” which uses an Akida Simulator or Chip for conversion, neuron modeling, and training methods. The framework also integrates with TensorFlow and Keras for quantization and conversion of models, while a ”Model Zoo” provides pre-trained models. This ecosystem allows efficient deployment of neural network models onto neuromorphic hardware, optimizing performance and energy efficiency.

With the understanding of the MetaTF framework, we can now check the conversion pipeline for translating PilotNet to SNN.

6.5 Conversion Pipeline for Akida Brainchip

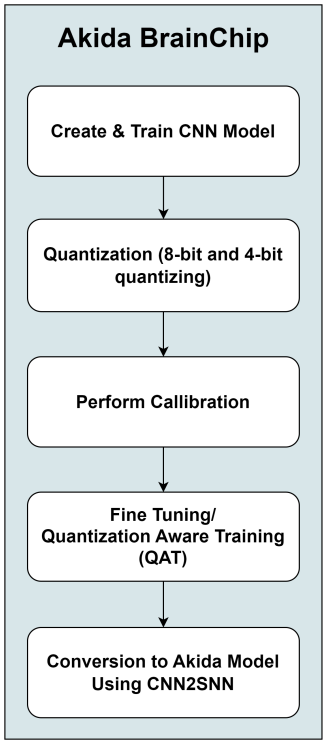

Figure 6.5: Conversion Pipeline for Brainchip Akida

The image outlines the steps involved in deploying a neural network model, such as the PilotNet model, onto the Akida Brainchip platform. Here's an explanation of each step tailored for the PilotNet model:

1. **Create and Train glscnn Model:**
    - This step involves creating and training the PilotNet model, which is a Convolutional Neural Network CNN originally designed for autonomous driving. The model is trained using a dataset of driving scenarios to predict the appropriate steering angle based on input from camera images.
2. **Quantization (8-bit and 4-bit Quantizing):**
    - After training, the PilotNet model undergoes quantization. This process reduces the precision of the model's weights and activations from their original floating-point precision (e.g., 32-bit) to a lower precision, such as 8-bit or 4-bit. This step is crucial for deploying the model on the Akida chip, which benefits from lower precision for energy efficiency and faster inference times without significantly compromising accuracy.
3. **Perform Calibration:**

- Calibration is performed to adjust the model after quantization. This step ensures that the quantized model's outputs remain accurate and aligned with the original floating-point model. Calibration typically involves running a subset of the training data through the quantized model to fine-tune the quantization parameters.

4. **Fine Tuning/Quantization Aware Training (QAT):**
   - In this step, the PilotNet model undergoes fine-tuning with Quantization Aware Training (QAT). QAT involves training the model while simulating the effects of quantization during the forward pass. This helps the model learn to maintain its performance despite the lower precision, leading to better accuracy post-quantization.
5. **Conversion to Akida Model using cnn2snn:**
   - Finally, the quantized and fine-tuned PilotNet model is converted into a format compatible with the Akida chip using the cnn2snn tool. This tool transforms the CNN model into a SNN, which can be efficiently executed on the Akida neuromorphic hardware. This conversion is essential for leveraging the low-power, high-performance capabilities of the Akida Brainchip for tasks like real-time autonomous driving.

6.5.1 Synsense

For converting the PilotNet model to SNN for Synsense, there is an involved usage of Sinabs framework. This framework is a DL library based on PyTorch with a focus on simplicity, fast training and extendability. Sinabs works well for Vision models because of its support for weight transfer.

Sinabs is built to complement PyTorch by adding features that support the dynamics of SNN. It includes a collection of activation layers that simulate spiking neuronal behavior. Due to the temporal characteristics of SNNs, several layers in Sinabs maintain internal states.

These Sinabs layers enhance PyTorch's activation functions and are designed to seamlessly integrate with various PyTorch layers, such as AvgPool2d, Linear, and Conv2d layers [47].

6.6 Conversions Pipeline for SynSense

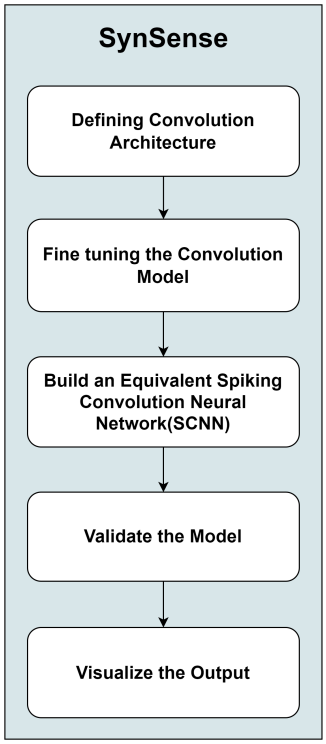

Figure 6.6: Conversion Pipeline for SynSense

1. **Defining Convolution Architecture:**
    - This is the initial step where we design the PilotNet architecture of the convolutional neural network CNN that we want to convert to an SNN. The architecture includes defining layers, types of layers (e.g., convolutional, pooling, fully connected), the number of filters, kernel sizes, etc. In Sinabs, we are using PyTorch to define the model. The architecture should be designed with conversion to SNN in mind, ensuring compatibility with spiking neuron models later on.
2. **Finetuning the Convolution Model:**
    - After defining the architecture, we trained the model on the dataset. This step involves optimizing the weights using backpropagation and a chosen optimizer (like Adam or SGD). Training is done until the model achieves satisfactory performance on the task. This trained model will serve as the baseline before converting it to an SNN.
3. **Build an Equivalent Spiking Convolutional Neural Network (SCNN):**
    - This step involves converting the trained CNN into an SNN. The main task here is to translate the continuous activation functions used in CNNs into spiking neuron models that work with discrete events (spikes). Sinabs provides tools to help with this conversion. We would map the activations of the trained CNN to spiking neurons,

adjust the thresholds, and ensure that the temporal dynamics of spiking neurons are well-represented. The goal is to preserve the learned features of the CNN while adapting them to a spiking framework.

4. **Validate the Model:**
   - Once the SNN is built, we need to test it to ensure that it performs well on the task, ideally with accuracy comparable to the original CNN. This step is crucial as the conversion process might introduce some performance degradation. We would run the SNN on a validation set and compare its performance metrics with those of the original model. Adjustments to the spiking neuron parameters may be necessary to achieve optimal performance.
5. **Visualize the Output:**
   - Finally, we analyze and visualize the outputs of the SNN to understand how it processes information and makes decisions. Visualization can help in interpreting the model's behavior and debugging any issues. We can use various tools and techniques to visualize spikes over time, activation patterns, or compare the output distributions of the CNN and SNN. This step can involve generating plots of spike trains, response maps, or confusion matrices.

6.7 Generating Quantized Model

The Figure 6.7 shows the architecture of a quantized neural network model of PilotNet. Quantization is a process that reduces the precision of the numbers used to represent a model's parameters and activations, typically from 32-bit floating-point to lower-bit formats like 8-bit integers. This process helps in reducing the model size and computational requirements, making it more efficient for deployment on resource-constrained devices, such as mobile or embedded systems.

Parameter Type	Count
Total Parameters	4,682,199
Trainable Parameters	4,681,719
Non-trainable Parameters	480

Table 6.1: Parameter Summary of the Quantized Model

The model begins with an input layer that accepts images with dimensions of 66x200 pixels and 3 color channels (likely RGB), preparing the data for the subsequent convolutional layers. These convolutional layers (QuantizedConv2D) are designed to extract increasingly complex features from the input image, progressively reducing the spatial dimensions while increasing the depth to capture detailed patterns. Each convolutional layer is followed by a quantized ReLU activation (QuantizedReLU), introducing non-linearity while operating in a quantized space. After the convolutional layers, the model includes a flattening layer that transforms the multidimensional feature maps into a one-dimensional vector, which is then processed by a series of fully connected (dense) layers. These dense layers reduce the dimensionality step by step until a final output is produced, with dropout layers (QuantizedDropout) used to prevent overfitting. At the end of the

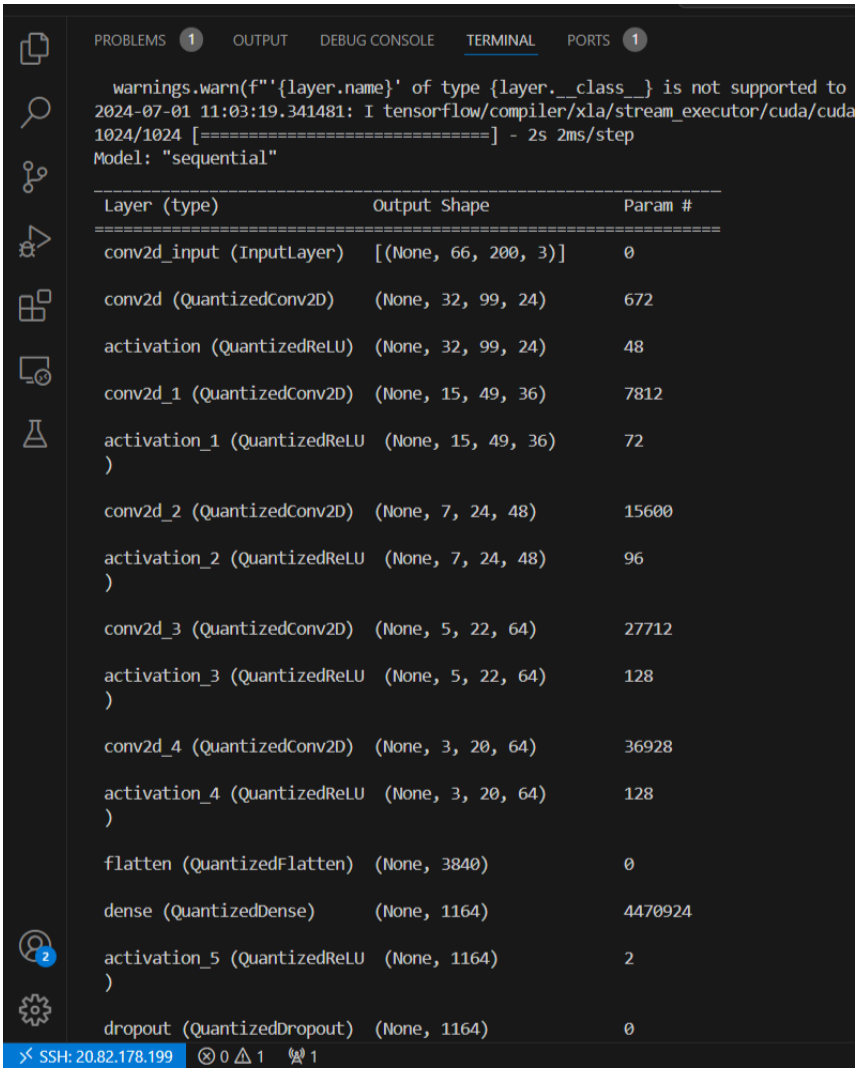

Figure 6.7: Quantized Model

model, a dequantizer layer converts the quantized outputs back to full precision. The model contains a total of 4,682,199 parameters, of which 4,681,719 are trainable and 480 are non-trainable as shown in table 6.1.

6.8 Converted model - MetaTF

The converted model starts with several convolutional layers that gradually reduce the spatial dimensions of the input while increasing the depth (number of feature maps). These layers are followed by a series of dense layers that progressively reduce the dimensionality until a single output is produced.

The dequantizer layer shows that the model's weights and activations were quantized in the previous pipeline step, but the final output is dequantized to provide a precise result. The table provides the overall structure of the converted model.

Component	Details
Input Shape	`[66, 200, 3]` This indicates the shape of the input data to the model. It typically represents an image with a height of 66 pixels, a width of 200 pixels, and 3 color channels (likely RGB).
Output Shape	`[1, 1, 1]` The output of the model is a single scalar value which is common in regression tasks like predicting a steering angle in autonomous driving.
Sequences	`1` The model processes one sequence at a time.
Layers	`11` The model consists of 11 layers.

The converted PilotNet model has been quantized, and can be effectively run on neuromorphic hardware - Brainchip Akida. The final output being dequantized for precision. The converted model balances the complexity of feature extraction through multiple convolutional layers.

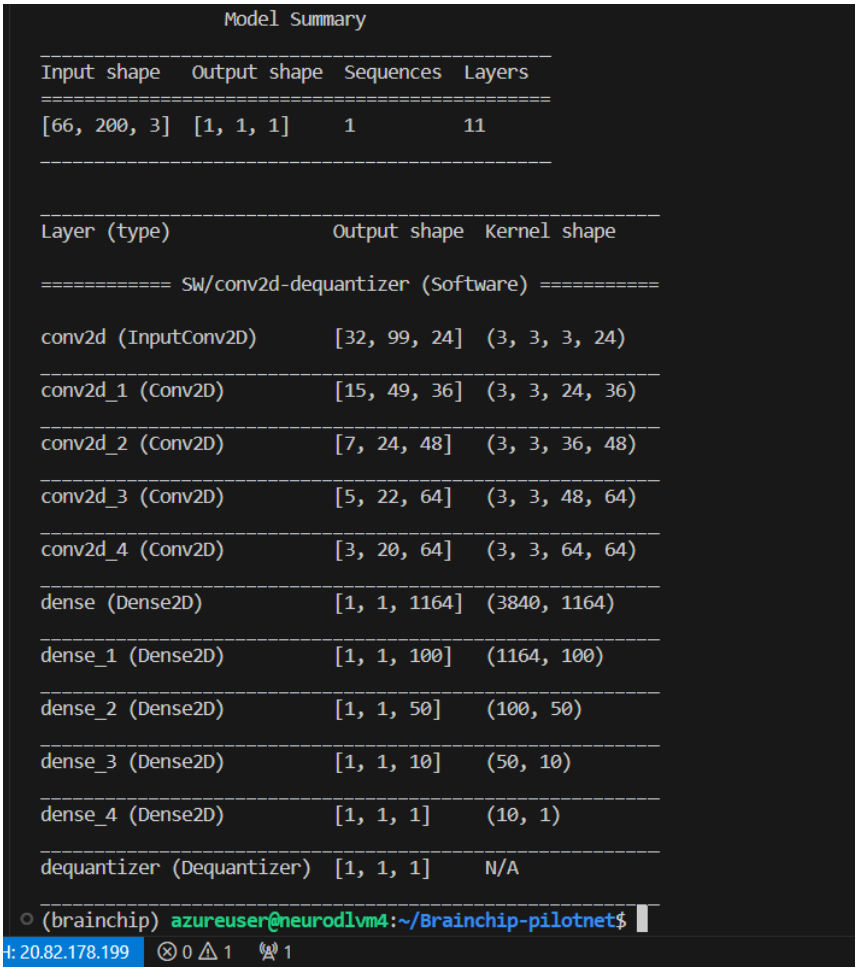

Figure 6.8: Model Summary of the converted SNN PilotNet using MetaTF

The architectural design of the model summary initiates with a sequence of convolutional layers (Conv2D). These layers progressively decrease the spatial dimensions of the input image while simultaneously increasing the depth (i.e., the number of feature maps) to capture increasingly complex features. This hierarchical feature extraction process is fundamental to the model's ability to identify intricate patterns within the input data.

Following the convolutional layers, the architecture includes a series of fully connected (dense) layers. These dense layers are responsible for further processing the extracted features, systematically reducing their dimensionality. This reduction is performed iteratively until the model outputs a single value, which is typically a scalar. The configuration of the final dense layer strongly indicates that the model is designed to predict a single, continuous quantity.

To enhance the model's generalization capability and mitigate overfitting, dropout layers are integrated into the architecture. These layers function during training by randomly dropping a fraction of the units in a given layer, ensuring that the model does not become overly reliant on specific features. The presence of a dequantizer layer in the model end confirms that the model has undergone quantization — the technique often employed in mataTF to reduce both the memory footprint and computational load, making the model more suitable for deployment on specialized hardware, such as neuromorphic processors. The dequantizer's here would to convert the quantized outputs back to full precision, thereby ensuring the accuracy and suitability of the final output for precision-critical tasks.

7 Results

7.1 Metric based Analysis

The converted PilotNet model for three different neuromorphic processors were evaluated based on the MSE. We use the MSE as a metric, as it captures the differences between the predicted and actual value. For a PilotNet model, the actual and the predicted angle need to be the same inorder to get a smooth streering angle prediction.

The initial requirement was to evaluate the orginal PilotNet model's MSE which will be used as the base value for comparing the other neuromorphic compatible PilotNet models. The orginal PilotNet model was a tensorflow implemention which was converted to equivalent Keras model and the MSE was calculated.

Similarly the MSE and the accuracy for the other models were respectively calculated.

Processor	Model	MSE	Prediction Accuracy
Server	Original PilotNet model	0.0284	97.24
Intel Loihi	Lava PilotNet model	0.0395	96.20
Akida Brainchip	MetaTF PilotNet model	0.0753	92.99

Table 7.1: MSE values for different PilotNet models on various processors

While we performed the conversion of PilotNet code for SynSense using the sinabs framework, there were complexities in calculating the MSE. The possible reason maybe that the Sinabs framework is primarily designed for (SNNs), which are often used in tasks that benefit from event-based processing and are typically more focused on classification tasks, where the output is a discrete label or class. This aligns with the strengths of spiking neurons, which are good at detecting patterns in time series or event-based data. While it might be theoretically possible to perform regression tasks using the Sinabs framework with significant customization, it is not what the framework is primarily designed for. Analysing more into this aspect, can provide a better understanding in performing Regression task using sinabs.

On the other hand, when we compare the MSE of the other models , we see the following observations as presented in the Table 6.1. The original PilotNet model running on a machine achieves the lowest MSE of 0.0284, making it the most accurate among the models tested. This means that its predictions are very close to the correct steering angles, suggesting that it would lead to safer driving behavior. The Lava PilotNet model created using the lava framework for Intel Loihi, has a slightly higher MSE of 0.0395. While it's a bit less accurate than the GPU version, it might offer advantages in energy efficiency, which is an important consideration for real-world applications like autonomous driving. Finally,we analyse the MetaTF PilotNet model developed using the MetaTF

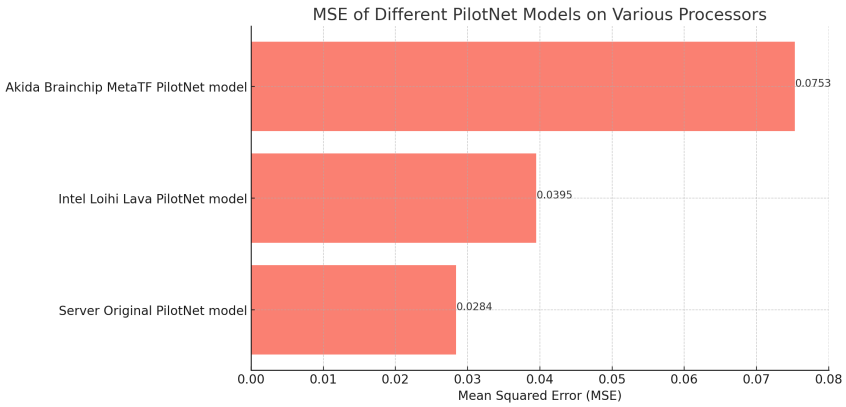

Figure 7.1: Comparison based on MSE

framework for Brainchip. Although this version is less accurate, meaning its predictions are further off from the correct steering angles, the Akida processor might still be valuable in scenarios where energy efficiency or processing speed is more critical than pure accuracy.

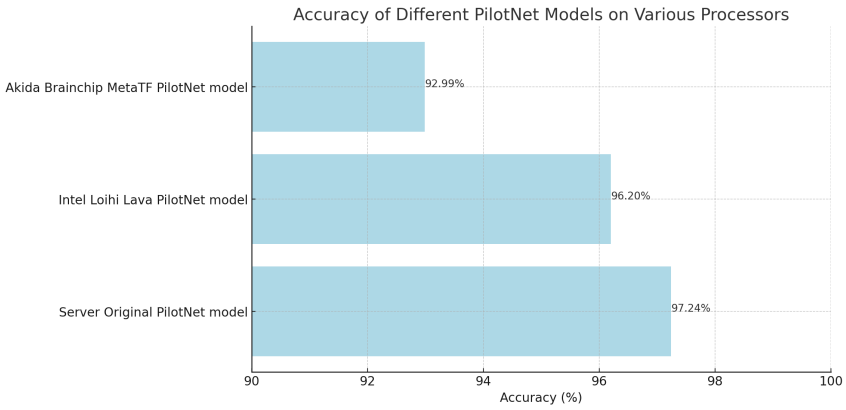

Figure 7.2: Comparison based on Accuracy

The accuracy comparison as shown in Figure 7.2 of different PilotNet models across various processors reveals notable differences in performance. The Server Original PilotNet model achieves the highest accuracy at 97.24%, indicating its strong ability to predict steering angles with minimal error. Following this, the Intel Loihi Lava PilotNet model demonstrates a slightly lower accuracy of 96.20%, which, while still high, reflects the challenges inherent in translating the model to a neuromorphic processor like Loihi. The Akida Brainchip MetaTF PilotNet model shows the lowest accuracy at 92.99%, suggesting that while the Akida processor is optimized for energy efficiency and real-time processing, it may require further optimization to match the performance of more traditional or higher-powered processors. These differences highlight the trade-offs between processing power, efficiency, and accuracy when deploying AI models on various hardware platforms.

In summary, while the orginal Neural Network provides the best accuracy, the SNN Model offer intriguing trade-offs that could be beneficial depending on the specific needs of the autonomous driving system.

Observation

The accuracy drop in SNN versions of the PilotNet model, compared to traditional neural networks, can be attributed to several factors. First, SNNs process information using discrete spikes over time, unlike Neural Networks, which use continuous values, leading to challenges in maintaining the same level of precision. The conversion from continuous activations to spike-based processing often involves approximations and quantization, which can introduce noise and reduce fidelity, resulting in lower accuracy. Additionally, training SNNs is more complex due to difficulties in applying gradient-based optimization methods, and the training process can be less effective or harder to tune. Neuromorphic hardware, while efficient for running SNNs, may also impose constraints on model complexity and performance, further contributing to the drop in accuracy. Lastly, the fine-grained detail required for tasks like predicting steering angles might be lost in the spike-based processing, making the SNN models less precise. These factors collectively explain why SNN versions of PilotNet typically experience a reduction in accuracy compared to their traditional NN counterparts.

Analysis - Akida PilotNet

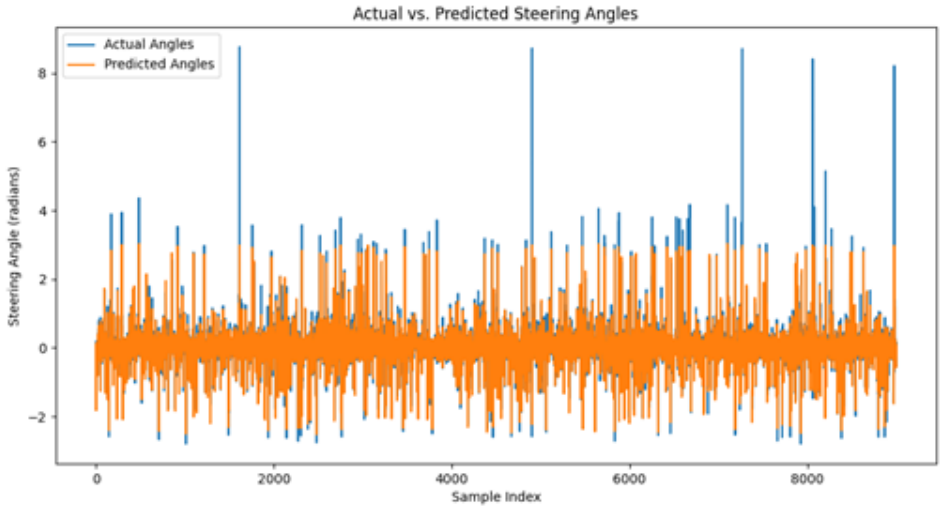

Figure 7.3: Actual vs. Predicted Steering Angles for Akida PilotNet

This Figure 7.3 illustrates the comparison between the actual steering angles (shown in blue) and the predicted steering angles (shown in orange) generated by the Akida PilotNet model. The x-axis represents the sample index, essentially different points in time or frames from the driving data, while the y-axis represents the steering angle in radians.

What stands out in the graph is that while the predicted angles generally follow the trend of the actual angles, they are more concentrated around the center, with less extreme variations compared to the actual angles. The actual steering angles show more spikes and larger deviations, which suggests that the model sometimes fails to predict sharp turns or sudden changes accurately. This discrepancy indicates that the model might be smoothing out its predictions too much, potentially leading to less responsive steering in real-world driving scenarios. The frequent underestimation or

overestimation in the predicted angles suggests that while the model captures the general direction, it struggles with precision, especially in more challenging driving situations. This could be a contributing factor to the overall higher MSE observed in this model.

Approach towards Performance Improvement in MetaTF

Inorder to achieve the accuracy to the level of the orginal model, there is a possiblility to perform Fine-tuning and Quantization-Aware Training (QAT). These are advanced techniques used in the MetaTF framework to enhance model performance, particularly when deploying on neuromorphic hardware like BrainChip's Akida. Fine-tuning involves taking a pre-trained model and continuing its training on a new dataset or specific task, allowing the model to adapt to the new data while leveraging the knowledge it has already acquired. This process typically involves freezing some layers, adjusting the learning rate, and then training on the new dataset, which helps improve performance without the need for complete retraining.

Quantization-Aware Training (QAT), on the other hand, prepares models for deployment on hardware that uses lower precision arithmetic, such as 8-bit integers. By simulating the effects of quantization during training, QAT allows the model to learn how to minimize accuracy loss due to quantization. The process involves inserting fake quantization operations during training and then continuing to train the model with these effects in place. The training happens keeping the quantization in mind, thereby reducing the performance degradation encountered after the Quantization step. This way we could improve the accuracy of our PilotNet model for Akida.

Analysis - Intel PilotNet

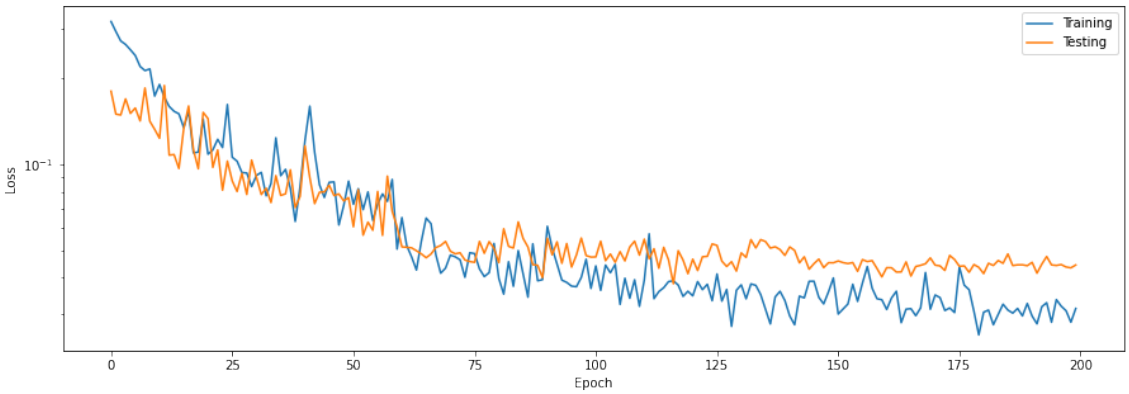

Figure 7.4: Actual vs. Predicted Steering Angles for Intel PilotNet

The Figure 7.4 illustrates the loss curve for both training and testing phases of the Intel PilotNet model running in the Lava framework. Over the course of 200 epochs, the loss gradually decreases for both training and testing, indicating that the model is learning effectively and generalizing well to new data. The consistent reduction in loss suggests that the Intel Lava framework is able to efficiently optimize the PilotNet model, allowing it to converge to a solution that accurately predicts steering angles.

This efficient training process likely contributes to the relatively high accuracy and lower MSE observed in the earlier graphs. The Intel Lava framework is designed to handle the temporal dynamics and spiking behavior inherent in neuromorphic computing, which may help the model

capture the necessary patterns for accurate predictions. Additionally, the framework's ability to effectively manage the balance between underfitting and overfitting ensures that the model remains robust during testing, leading to better performance metrics. This explains why the Intel Lava PilotNet model shows superior performance compared to other models on different processors.

7.2 Conversion Process Analysis

In this section we evaluates the ease and effectiveness of converting traditional neural networks to SNNs across three different frameworks: MetaTF, Lava, and Sinabs. Each framework has its strengths and challenges in facilitating this conversion, which is crucial for deploying models on neuromorphic hardware.

7.2.1 MetaTF Framework

Ease of Conversion and Importance of Quantization:

MetaTF framework was easier to use as it provides a user-friendly and understandable conversion process which making it easier to translate traditional ANNs into SNNs.

A key aspect to the translation in MetaTF is the emphasis on quantization. It plays a critical role for efficient deployment on neuromorphic processors. Quantization reduces the precision of the model's parameters, allowing it to run more efficiently on hardware designed for lower bit-width operations, without significantly compromising accuracy. Therefore, it is important to perform quantization for converting a model to be deployed into Akida.

Availability of Model Zoo Pre-trained Models and Compatibility with Popular Architectures:

MetaTF provides access to a Model Zoo, which includes pre-trained models that can be directly converted and fine-tuned for SNNs. This feature simplifies the conversion process, as users can leverage existing models rather than starting from scratch. Moreover, MetaTF's compatibility with popular DL architectures ensures that a wide range of models can be converted and deployed efficiently.

Comprehensive Documentation:

Another important ease of using MetaTF's framework is its comprehensive documentation, which supports developers throughout the conversion process. Good documentation is essential for understanding the intricacies of converting models to SNNs and troubleshooting any issues that arise. This availability of proper documentation makes MetaTF an accessible choice for performing researches.

7.2.2 Lava Framework

Modular Design with Layered Architecture:

Lava is designed with a modular and layered architecture, which includes the definition of neural models, learning rules, and hardware abstraction layers. This modularity allows for flexibility in designing and experimenting with different types of SNNs, providing us with control over how models are structured and how they interact with the underlying hardware. This flexibility is particularly valuable for extensively customizing the models.

Support for Various Learning Rules:

Lava supports a variety of learning rules, including STDP and Backpropagation Through Time (BPTT). STDP is a biologically inspired learning rule that adjusts synaptic strengths based on the timing of spikes, while BPTT is a more traditional method used for training recurrent neural networks. The support for these learning rules allows Lava to cater to both biologically plausible models and those that require more precise gradient-based training, making it versatile for different types of research and applications.

7.2.3 Sinabs Framework

Ease of Use due to High-level API and Integration with PyTorch:

Sinabs emphasizes ease of use, particularly due to its high-level API and integration with PyTorch which a popular DL framework. This integration simplifies the conversion process as we are already familiar with PyTorch. Sinabs therefore leverages this familiarity, allowing a conversion of existing PyTorch models into SNNs with minimal additional learning requirement.

Supports Direct Conversion and Spike-based Computation:

Sinabs also supports direct conversion of traditional neural networks into spiking models, facilitating the transition from conventional DL to SNN. This direct conversion capability, coupled with support for spike-based computation, ensures that models can be efficiently deployed on neuromorphic hardware without the need for extensive modifications or retraining.

7.3 Comparative Analysis of Neuromorphic Processors

Based on the research overview, a comparative analysis of the Neuromorphic Processors namely Brainchip Akida, SynSense, Intel Loihi were performed and formulated below:

7.3.1 Architecture

Intel's Loihi chip is designed with flexibility in mind, featuring programmable spiking neurons that make it adaptable for a wide range of applications. On the other hand, BrainChip's Akida processor is built with a specialized neuromorphic core, fine-tuned specifically for spiking neural networks SNN, making it particularly effective for edge computing tasks. Meanwhile, SynSense is crafted to prioritize ultra-low power consumption, excelling in event-driven processing, especially when it comes to tasks that involve sensory data.

7.3.2 Programming Framework

Intel Loihi utilizes the Lava framework, developed by Intel, which provides robust tools for implementing and optimizing SNNs. BrainChip Akida is supported by MetaTF, BrainChip's own framework, which is designed to facilitate the seamless translation of traditional neural networks to SNNs. SynSense employs a custom SDK tailored for event-driven processing, supporting lightweight applications that prioritize energy efficiency.

7.3.3 Ease of Translation

The ease of translating ANNs to SNNs on Intel Loihi is moderate, as it requires adapting traditional network architectures to a spiking format and often relies on surrogate gradient methods within Lava. BrainChip Akida offers a high ease of translation, with MetaTF providing built-in support that simplifies the process of converting Neural Networks to SNNs. On the other hand, SynSense presents a low to moderate ease of translation due to its focus on lightweight, event-driven applications, which necessitates significant adaptation for more complex SNN.

7.3.4 Training Complexity

Training complexity on Intel Loihi is high, involving specialized algorithms like surrogate gradients, which often require more iterations for the network to converge effectively. BrainChip Akida has a moderate training complexity, as MetaTF includes built-in optimizations that simplify training, although some adaptation is still needed. SynSense also exhibits high training complexity, especially since it is designed for low-power, real-time tasks, limiting its ability to train more complex SNNs.

7.3.5 Accuracy After Translation

After translation, Intel Loihi can achieve moderate to high accuracy with proper tuning, though it may require extensive optimization efforts. BrainChip Akida generally provides moderate accuracy post-translation, balancing performance with energy efficiency, particularly in real-time applications. SynSense typically results in lower accuracy after translation, as it prioritizes energy efficiency over model complexity, making it more suited to simpler tasks.

7.3.6 Energy Efficiency

Intel Loihi is highly optimized for energy-efficient spiking computation, making it a good choice for embedded systems. BrainChip Akida is known for its very high energy efficiency, particularly in real-time, edge-computing environments. SynSense excels in ultra-high energy efficiency, often at the cost of accuracy and model complexity, making it ideal for power-constrained applications.

7.3.7 Temporal Processing Strength

Intel Loihi demonstrates strong temporal processing capabilities, making it well-suited for tasks requiring complex temporal dynamics. BrainChip Akida offers moderate to strong temporal processing, especially effective in real-time applications, though it may not be as versatile as Loihi. SynSense is moderately effective in temporal processing, particularly in event-driven scenarios, but they struggle with more complex temporal tasks due to its simplified architecture.

7.3.8 Hardware Constraints

Intel Loihi presents moderate hardware constraints, requiring careful model adaptation to fit within its capabilities, but it is relatively advanced. BrainChip Akida has low to moderate constraints, generally well-balanced but still necessitating optimization for optimal performance. SynSense faces high hardware constraints, with significant limitations on model complexity and size due to its focus on ultra-low power, real-time processing.

7.3.9 Integration with Ecosystem

Intel Loihi benefits from a mature ecosystem, well-integrated with Lava and broader neuromorphic computing research, backed by strong community support. BrainChip Akida has a growing ecosystem, with MetaTF offering good support, although the ecosystem is still developing. SynSense operates within an emerging ecosystem, relying on custom SDKs with less community support, making integration more challenging.

7.3.10 Use Case Suitability

Intel Loihi is particularly suitable for complex, time-dependent tasks such as robotics and autonomous driving, where its strong temporal processing and energy efficiency shine. BrainChip Akida is ideal for real-time applications like edge computing, IoT, and moderate-complexity data processing tasks, where energy efficiency is crucial. SynSense is best suited for simple, low-power applications, especially in sensory processing and basic event-based tasks, where its ultra-low power consumption is a significant advantage.

8 Conclusion and Future Work

The research undertaken in this thesis provides a comprehensive analysis of the frameworks and strategies necessary for converting traditional neural networks, particularly CNN, into SNNs which can be efficiently deployed on neuromorphic processors. The primary motivation behind this study was to address the growing need for energy-efficient, low-latency processing in various applications, including autonomous driving, robotics, and edge computing, where traditional processors fall short in meeting the stringent power and speed requirements.

The thesis successfully demonstrated the feasibility and challenges of converting a widely used CNN, the PilotNet model, to SNNs using three different neuromorphic platform frameworks for Brainchip Akida, Intel Loihi, and SynSense. Each platform was evaluated in terms of its ability to maintain the accuracy and performance post conversion, and ease of translation of the original neural network to SNN.

The results from this thesis indicate that while neuromorphic processors like Intel Loihi offer considerable benefits in terms of energy efficiency, there are trade-offs in accuracy when compared to traditional implementations. Specifically, the Intel Loihi demonstrated a slight reduction in accuracy but can provide significant improvements in energy consumption, making it a strong option for applications requiring low-power consumption, such as autonomous vehicles and robotics. On the other hand, Brainchip Akida, which is optimized for edge computing, is known to provide superior energy efficiency but faced limitations in processing complex temporal tasks, as indicated by a higher MSE in the performance metrics.But the translation process to SNN in Akida was straight forward when compared to Intel Loihi. SynSense, although highly energy-efficient and compact, was less effective in handling complex neural network tasks due to its simpler architecture, which impacted its overall accuracy and limited its application to less demanding scenarios.

The thesis also highlighted several critical challenges that remain in the conversion process from traditional neural networks to SNNs. These challenges include the complexity of translating continuous activations of CNNs into the discrete spikes required by SNNs without significant loss of information, the difficulty in maintaining the original model's accuracy after conversion, and the limitations in existing neuromorphic frameworks that often lack robust support for complex neural network models. These findings underscore the need for continued research and development to refine the conversion processes and enhance the capabilities of neuromorphic processors.

Future Work

The deployment of neural networks on neuromorphic processors is still in its infancy, with significant potential for growth and development. Future research should focus on enhancing the conversion processes to reduce accuracy loss and improve the efficiency of SNNs on various neuromorphic platforms. This includes exploring more advanced techniques for fine-tuning and quantization-aware training, particularly in frameworks like MetaTF, which could help mitigate the performance degradation observed in the PilotNet model when deployed on Akida.

Future work can be on how to deploy the converted networks into real-time applications. This would not only optimize the neuromorphic hardware and software but also develop robust testing environments which could replicate real-world conditions to check performance, reliability, and safety for these systems. Further integration of more complex neural network architectures shall be an essential milestone to improve the usage of neuromorphic computing.

By addressing the challenges, future research can pave the way for the widespread adoption of neuromorphic computing in various high-impact domains, ultimately leading to more intelligent, efficient, and autonomous systems.

Bibliography

[1] J. Smith, A. Garcia, M. Patel. “Neuromorphic Computing: Advancements and Applications”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.5 (May 2020), pp. 1270–1283.

[2] Sumit Bam Shrestha et al. *Efficient Video and Audio processing with Loihi 2*. 2023. arXiv: 2310.03251 [cs.NE]. URL: https://arxiv.org/abs/2310.03251.

[3] A. Patterson, J. L. Hennessy. *Computer Organization and Design, Revised Printing, Third Edition*. 3rd. The Morgan Kaufmann Series in Computer Architecture and Design. Oxford, England: Morgan Kaufmann, Aug. 2004.

[4] C. Lee et al. “Efficient and Accurate Conversion of Spiking Neural Network with Burst Spikes”. In: *arXiv preprint arXiv:1804.13271* (2018). URL: https://arxiv.org/abs/1804.13271.

[5] Yuhang Li, Yufei Guo, Shanghang Zhang, et al. “Differentiable Spike: Rethinking Gradient-Descent for Training Spiking Neural Networks”. In: *NeurIPS 2021*. 2021. URL: https://openreview.net/forum?id=H4e7mBnC9f0.

[6] Peter U. Diehl, Matthew Cook. “Unsupervised learning of digit recognition using spike-timing-dependent plasticity”. In: *Frontiers in Computational Neuroscience* 9 (2015), p. 99. DOI: 10.3389/fncom.2015.00099.

[7] Bodo Rueckauer et al. “Conversion of continuous-valued deep networks to efficient event-driven networks for image classification”. In: *Frontiers in Neuroscience* 11 (2017), p. 682. DOI: 10.3389/fnins.2017.00682.

[8] Michael Pfeiffer, Thomas Pfeil. “Deep learning with spiking neurons: Opportunities and challenges”. In: *Frontiers in Neuroscience* 12 (2018), p. 774. DOI: 10.3389/fnins.2018.00774.

[9] Matthew J. Page et al. “The PRISMA 2020 statement: an updated guideline for reporting systematic reviews”. In: *BMJ* 372 (2021). Published 2021 Mar 29, n71. DOI: 10.1136/bmj.n71. URL: https://doi.org/10.1136/bmj.n71.

[10] Y. Li et al. “Converting Artificial Neural Networks to Spiking Neural Networks via Parameter Calibration”. In: *arXiv preprint arXiv:2205.10121* (2022).

[11] J. Ding et al. “Optimal ANN-SNN Conversion for Fast and Accurate Inference in Deep Spiking Neural Networks”. In: *arXiv preprint arXiv:2105.11654* (2021).

[12] P. Chandarana et al. “Energy-Efficient Deployment of Machine Learning Workloads on Neuromorphic Hardware”. In: *arXiv preprint arXiv:2210.05006* (2022).

[13] Q. Zhang et al. “Artificial to Spiking Neural Networks Conversion for Scientific Machine Learning”. In: *arXiv preprint arXiv:2308.16372* (2023).

[14] A. Ziegler et al. “Spiking Neural Network for Fast Moving Object Detection on Neuromorphic Hardware Devices Using an Event-Based Camera”. In: *arXiv preprint arXiv:2403.10677* (2024).

[15] Batta Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9.1 (2020), pp. 381–386.

[16] Michael I Jordan, Tom M Mitchell. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245 (2015), pp. 255–260.

[17] Wenjie Wei et al. *Q-SNNs: Quantized Spiking Neural Networks*. 2024. arXiv: 2406.13672 [cs.CV]. URL: https://arxiv.org/abs/2406.13672.

[18] Oludare Isaac Abiodun et al. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4.11 (2018).

[19] AD Dongare, RR Kharde, Amit D Kachare, et al. "Introduction to artificial neural network". In: *International Journal of Engineering and Innovative Technology (IJEIT)* 2.1 (2012), pp. 189–194.

[20] Keiron O'Shea, Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE]. URL: https://arxiv.org/abs/1511.08458.

[21] Zewen Li et al. "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.12 (2022), pp. 6999–7019. DOI: 10.1109/TNNLS.2021.3084827.

[22] Catherine D. Schuman et al. *A Survey of Neuromorphic Computing and Neural Networks in Hardware*. 2017. arXiv: 1705.06963 [cs.NE]. URL: https://arxiv.org/abs/1705.06963.

[23] Andrea Calimera, Enrico Macii, Massimo Poncino. "The Human Brain Project and neuromorphic computing." In: *Functional neurology* 28 3 (2013), pp. 191–6. URL: https://api.semanticscholar.org/CorpusID:30149354.

[24] Arash Fayyazi et al. "An Ultra Low-Power Memristive Neuromorphic Circuit for Internet of Things Smart Sensors". In: *IEEE Internet of Things Journal* 5 (2018), pp. 1011–1022. URL: https://api.semanticscholar.org/CorpusID:4866298.

[25] Wulfram Gerstner, Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[26] Samanwoy Ghosh-Dastidar, Hojjat Adeli. "Spiking neural networks". In: *International journal of neural systems* 19.04 (2009), pp. 295–308.

[27] Yang Dan, Mu-ming Poo. "Spike Timing-Dependent Plasticity of Neural Circuits". In: *Neuron* 44 (Oct. 2004), pp. 23–30. DOI: 10.1016/j.neuron.2004.09.007.

[28] Sander Bohte, Joost Kok, Han Poutré. "Error-Backpropagation in Temporally Encoded Networks of Spiking Neurons". In: *Neurocomputing* 48 (Feb. 2001), pp. 17–37. DOI: 10.1016/S0925-2312(01)00658-0.

[29] Shihao Song et al. "Compiling Spiking Neural Networks to Neuromorphic Hardware". In: *Proceedings of the ACM* (2020). DOI: 10.1145/3372799.3394364. URL: https://doi.org/10.1145/3372799.3394364.

[30] Michael V. DeBole et al. "TrueNorth: Accelerating From Zero to 64 Million Neurons in 10 Years". In: *Computer* 52.5 (2019), pp. 20–29. DOI: 10.1109/MC.2019.2903009.

[31] Francesco Galluppi et al. "A framework for plasticity implementation on the SpiNNaker neural architecture". In: *Frontiers in Neuroscience* 8 (Jan. 2015). DOI: 10.3389/fnins.2014.00429.

[32] Adarsha Balaji et al. *PyCARL: A PyNN Interface for Hardware-Software Co-Simulation of Spiking Neural Network*. 2020. arXiv: 2003.09696 [cs.NE]. URL: https://arxiv.org/abs/2003.09696.

[33] Ben Varkey Benjamin et al. "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations". In: *Proceedings of the IEEE* 102.5 (2014), pp. 699–716. DOI: 10.1109/JPROC.2014.2313565.

[34] Anup Das et al. "Mapping of local and global synapses on spiking neuromorphic hardware". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2018), pp. 1217–1222. URL: https://api.semanticscholar.org/CorpusID:5089160.

[35] Shihao Song et al. "Compiling Spiking Neural Networks to Neuromorphic Hardware". In: *The 21st ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES '20. ACM, June 2020. DOI: 10.1145/3372799.3394364. URL: http://dx.doi.org/10.1145/3372799.3394364.

[36] BrainChip Inc. *Akida Foundations*. 2024. URL: https://brainchip.com/akida-foundations/.

[37] Anup Vanarse et al. "A Hardware-Deployable Neuromorphic Solution for Encoding and Classification of Electronic Nose Data". In: *Sensors (Basel, Switzerland)* 19 (2019). URL: https://api.semanticscholar.org/CorpusID:207935891.

[38] 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.

[39] Andrew Lines et al. "Loihi Asynchronous Neuromorphic Research Chip". In: *IEEE Asynchronous Circuits and Systems (ASYNC)* (May 2018), pp. 32–33. DOI: 10.1109/ASYNC.2018.00018.

[40] Intel Corporation. *Taking Neuromorphic Computing to the Next Level with Loihi 2*. 2023. URL: https://www.intel.com/content/www/us/en/newsroom/resources/neuro-inspired-chips.html.

[41] SynSense Inc. *Neuromorphic Intelligence & Application Solutions*. 2024. URL: https://www.synsense.ai/neuromorphic-technology.

[42] Mariusz Bojarski et al. *Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car*. 2017. arXiv: 1704.07911 [cs.CV]. URL: https://arxiv.org/abs/1704.07911.

[43] Lava Neuromorphic Computing. *Lava Architecture Overview*. 2024. URL: https://lava-nc.org/lava_architecture_overview.html.

[44] BrainChip Inc. *BrainChip Documentation*. 2024. URL: https://doc.brainchipinc.com/.

[45] Andreas Ziegler et al. *Spiking Neural Networks for Fast-Moving Object Detection on Neuromorphic Hardware Devices Using an Event-Based Camera*. 2024. arXiv: 2403.10677 [cs.RO]. URL: https://arxiv.org/abs/2403.10677.

[46] *Fulltext Document from DiVA Portal*. 2024. URL: https://www.diva-portal.org/smash/get/diva2:1779206/FULLTEXT01.pdf.

[47] Sinabs Development Team. *Sinabs Documentation: Getting Started with Fundamentals*. Accessed: 2024-08-13. 2024. URL: https://sinabs.readthedocs.io/en/v2.0.0/getting_started/fundamentals.html.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature