Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Communications Engineering (36^{th} cycle)

Efficient Deep Learning Inference: A Digital Hardware Perspective

Evaluating and improving performance and efficiency of artificial and spiking neural networks hardware accelerators

Fabrizio Ottati

Supervisors
Professor Luciano Lavagno, Supervisor
Professor Mario Roberto Casu, Co-supervisor

Doctoral Examination Committee:
Dr. Chiara Bartolozzi, *Referee*, Istituto Italiano di Tecnologia, Italy
Dr. Alexandre Levisse, *Referee*, École Polytechnique Fédérale de Lausanne, Switzerland
Prof. Jordi Cortadella, Universitat Politècnica de Catalunya, Spain
Prof. Alex Yakovlev, Newcastle University, United Kingdom
Prof. Claudio Passerone, Politecnico di Torino, Italy

Politecnico di Torino
February 2024

I hereby declare that the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Fabrizio Ottati
Turin, February 2024

Summary

From smartphones to televisions and cars, deep learning (DL) has become pervasive in our daily lives. Many modern DL models, especially large language models (LLMs), recommender systems, and vision transformers (ViTs) require huge amounts of power and energy for both training and inference. For instance, ViT-G/14, a top performing transformer model in object recognition, requires 2.86 GFLOP for a single inference on ImageNet, and around 159 MW h of energy to train on specialized tensor processing units (TPUs) running on 220 W.

The human brain, instead, has been trained during the evolution of humankind. Since the brain power budget is around 20 W, one could say that using only this power it is able to perform multiple actions at once and model fine-tuning (*i.e.*, learn new things).

Beyond training, DL models inference costs prove to be a serious problem: for instance, processing an user prompt using OpenAI LLM ChatGPT costs 0.04 $ in terms of energy and hardware (*i.e.*, graphic processing units (GPUs) used for the inference.). Hence, efficient hardware implementations and neural network models should be considered also for when DLs models are deployed.

Given the extraordinary efficiency of the brain in cognitive tasks, researchers are trying to take inspiration from biology when designing new artificial intelligence (AI) models and hardware; in this context, spiking neural networks (SNNs) are arising as a possible alternative to reach energy efficient AI. In this thesis, the inference of artificial neural networks (ANNs) and SNN models on digital hardware is investigated, analyzing state-of-the-art (SOTA) digital hardware accelerators targeting deep neural networks (DNNs) from both the spiking and artificial domains, on vision and audio workloads.

A C++ library for the deployment of spiking convolutional neural networks (CNNs) to field-programmable gate arrays (FPGAs) using high level synthesis (HLS) tools, is presented. The hardware architecture proposed is a dataflow one, and SOTA techniques for activations buffering are exploited to minimize the memory footprint of the neural network model, in order to target edge-class FPGAs with limited computational resources.

The library targets event-based vision tasks, in which new vision sensors inspired by the human retina are exploited to retrieve visual information from a scene. These sensors provide luminance variation events in output instead of full-frame images, contrarily to conventional RGB sensors. This data format naturally fits the spike-based processing of SNNs. The choice of this application is justified in the analysis of DL models digital hardware acceleration, in which it is shown that SNNs are not competitive with conventional DL accelerators on static frame-based vision tasks, such as object recognition, when it comes to energy consumption per inference and accuracy reached on the selected task. Using CNNs implemented in hardware by the library, one can target multiple event-based vision tasks, such as object recognition, object

detection, optical flow estimation etc.

An FPGA accelerator produced using the proposed HLS library is used as controller for autonomous small drones equipped with event cameras, in collaboration with Delft University of Technology. In the application considered, a CNN is interfaced with an event-based camera to compute optical flow from raw events, and its output used by the motor controller to drive the drone in some actions autonomously. The hardware accelerator shows a $10\times$ reduction in the power consumption when compared to a neuromorphic processor baseline on the same task, with no performance loss in terms of accuracy using the same spiking neural network model.

Acknowledgements

I think that the best word I can use to describe my doctorate is *resilience*, even though I am not a fan of it. It all started in 2020: I graduated during a pandemic, defending my master thesis during the lockdown, without having the occasion to be with my friends to celebrate my achievement. Later that year, I started my Ph.D., after having failed *two* times in a row the admission process. Then, many personal, work and health obstacles came along the way, but I can safely say that, thanks to my best friend and the amazing people I was lucky to meet in my last year of doctorate, those problems are a thing of the past. Now, here I am at the end of this beautiful journey.

I want to sincerely thank Professor Luciano Lavagno and Professor Mario Roberto Casu for accepting me as Ph.D. student in an "unconventional" situation, and teaching me how meaningful and serious research should be conducted.

I would like to thank Dr. Chiara Bartolozzi and Dr. Alexandre Levisse for agreeing to review my thesis: thanks to their insights and comments, I have been able to greatly improve the initial manuscript and provide a proper scientific perspective around it.

I would like to thank Mario Cignoni, my best friend. He has always been there, during the (many) lows and highs. We have started this journey together on completely unrelated topics, but we have always encouraged each other and provided technical feedback, somehow. In the end, we were also able to bootstrap ourselves, encouraging each other, and now we are both happy and love what we do.

I would like to thank Teodoro Urso, Filippo Minnella, Usman Jamal and Giovanni Brignone for welcoming me in the group during a very difficult period of my life and doctorate. They have supported me and taught me a lot, together with Roberto Bosio and Lorenzo Lagostina who have joined later. Thanks to *all* of them, I have been able to feel the joy of being part of a *team*. A special mention goes to Filippo for teaching me a lot on deep learning hardware acceleration, digital hardware design and piedmontese red wine and food. I want to thank Usman and Weixi for teaching me a lot about eastern food and drinking, and for always being there when I wanted to go to Vinarium. I want to thank also Giovanni in particular, for his TeX, `siunitx`, `acro`, `tikz` wisdom, for reviewing this manuscript, and helping me in figuring out the black magic of high level synthesis on FPGAs.

I want to thank Gregor Lenz for bringing me in the event-based vision world, and providing me an exciting research topic to investigate, helping me at each step. He has been my open-source mentor, teaching me how to collaborate with a large pool of people, and setting up projects that are actually used by someone else.

I want to thank also Jason Eshraghian for introducing me to the deep learning side of SNNs,

allowing me to enlarge my network, teaching me to always invest my energies in what actually makes sense. I can say without hesitation that he has been the third supervisor of my Ph.D., beyond being a friend.

Another special mention goes to Tiziano: you have been a key figure of the last year-and-a-half. Without your support and help, I would have never achieved all the amazing things I was lucky to do in my last year of doctorate, and I would have never recovered from that situation. A final, heart-felt thank you goes to you.

Now Artù, Merlino and I are up for a new adventure, in another country, having fun with hardware and deep learning, and I can't wait to start.

猿も木から落ちる.

Contents

Abbreviations

AEE average endpoint error

AI artificial intelligence

ANN artificial neural network

ASIC automatic speech recognition

ASIC application specific integrated circuit

BRAM block static random access memory

CMOS complementary metal oxide semiconductor

CNN convolutional neural network

CPU central processing unit

DL deep learning

DNN deep neural network

DRAM dynamic random access memory

DSP digital signal processor

DVS dynamic vision sensor

EDA electronic design automation

ELM expressive leaky memory

EVS event-based vision sensor

FIFO first in first out queue

FLOP floating point operation

FPGA field-programmable gate array

GNN graph neural network

GPU graphic processing unit

GRU gated recurrent unit

HDL hardware description language

HDR high dynamic range

HLS high level synthesis

IMC in-memory computing

KWS keyword spotting

LIF leaky integrate and fire

LLM large language model

LSTM long short-term memory

LUT look-up table

MAC multiply and accumulate

mAP mean average precision

ML machine learning

MOS metal-oxide-semiconductor

NLP natural language processing

NoC network-on-chip

PE processing engine

ReLU rectified linear unit

RF register file

RNN recurrent neural network

RTL register-transfer level

SIMD single instruction multiple data

SIMT single instruction multiple threads

SNN spiking neural network

SNU spiking neural unit

SoC system-on-chip

SOTA state-of-the-art

SRAM static random access memory

SSL self supervised learning

sSNU smooth spiking neural unit

SynOP synaptic operation

TPU tensor processing unit

VAD voice activity detection

ViT vision transformer

Chapter 1

Introduction

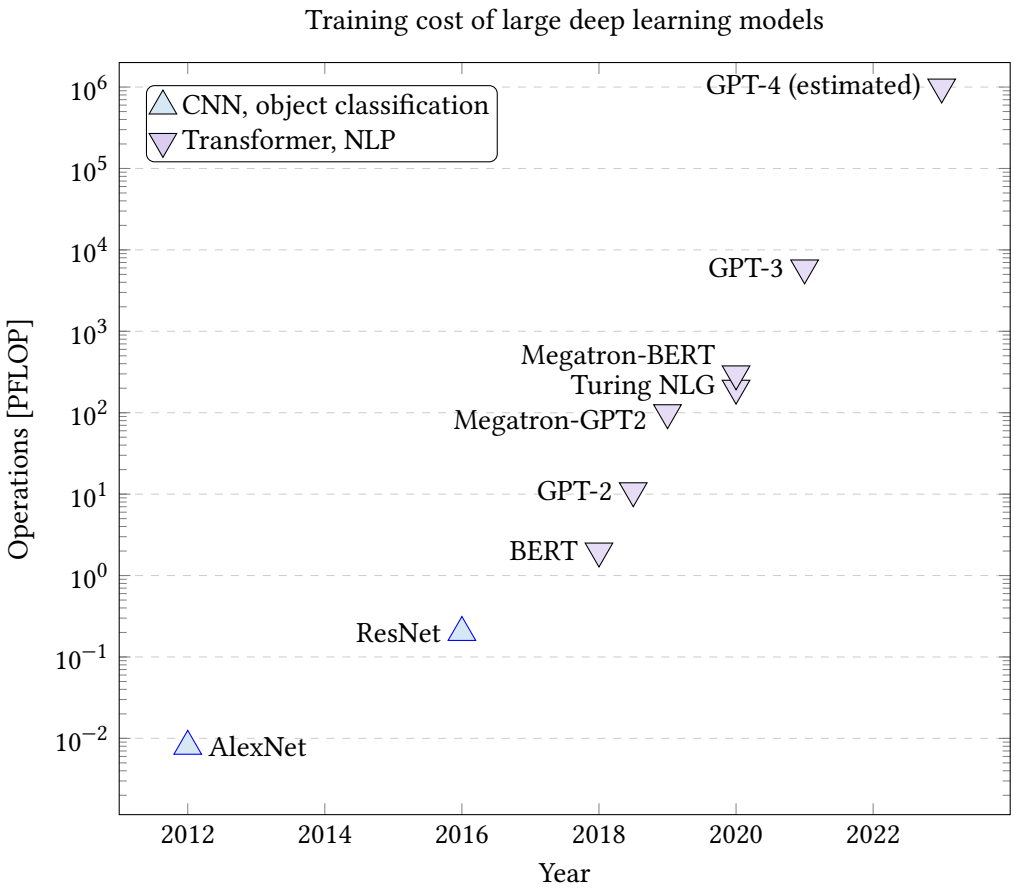


Figure 1.1: Cost of training large deep learning (DL) models across the years, measured in floating point operations (FLOPs), from object recognition models to large language models (LLMs). Credits: William J. Dally, Nvidia [22].

From smartphones to televisions and cars, deep learning (DL) has become pervasive in our

daily lives. Many modern DL models, especially large language models (LLMs) [7], recommender systems [77], and vision transformers [29] require huge amounts of power and energy for both training and inference, deriving from the large amount of floating point operations (FLOPs) needed for each iteration through the model. Moreover, the complexity keeps increasing exponentially, as it is shown in Fig. 1.1. For instance, ViT-G/14 [113], a top performing model in object recognition, requires 2.86 GFLOP for a single inference on ImageNet [26]. The model contains 184.3 billions of parameters, and optimizing these parameters has also a non negligible environmental impact [36]. In fact, Vit-G/14 has a training time of $3 \cdot 10^4$ TPUv3 days [113]; given that a TPUv3 consumes an average of 220 W[18], the energy consumption for training can be estimated to be 159 MW h.

When it comes to the human brain, training has been performed across its evolution, since humans appeared on Earth. One could say, however, that the human brain is able to fine-tune its learning algorithms throughout life on a 20 W power budget, running multiple tasks at once, while a vision transformer (ViT) needs to be finetuned on a very large dataset such as ImageNet for a long time before being able of performing object recognition only.

Beyond training, DL models inference costs prove to be a serious problem: for instance, processing an user prompt using OpenAI LLM ChatGPT costs 0.04 $ in terms of energy and hardware (*i.e.*, graphic processing units (GPUs) used for the inference). Hence, efficient hardware implementations and neural network models should be considered also for when DLs models are deployed.

Tackling these challenges requires more efficient neural network models and hardware. Many neuromorphic engineers are looking at how the brain might offer us a blueprint for making DL more efficient [92]. This is because the brain is capable of causal reasoning, integrating various sensory modalities in executing long-term planning, and providing sensorimotor feedback to enable us to engage with our environments actively. Moreover, it does so far more efficiently than any combination of large-scale DL models currently available. The pursuit of brain-inspired computing has triggered a surge in the popularity of spiking neural networks (SNNs) [32] with the ultimate goal of realizing efficient artificial intelligence (AI). For instance, SpikeGPT [117], the first spiking LLM, is estimated to use 22 × fewer operations in its execution (*i.e.*, a forward pass) when compared to its artificial counterpart.

When the first SNN accelerators emerged [3], analog-domain computation showed to be a promising substrate for the deployment of brain-inspired hardware. In fact, the metal-oxide-semiconductor (MOS) transistor in the sub-threshold regime can emulate the diffusion-based dynamics of neuronal ion channels. At the same time, memristive technologies reproduce key features of biological neurons [30, 85]. However, analog designs suffer from scalability issues due to transistor mismatch and noise, and long design time due to reduced electronic design automation (EDA) tool support [31]. In contrast, in deep sub-micron technologies, digital designs benefit from strong EDA support and reliable operation. While research on analog computation and in-memory processing has flourished over the past decade, digital accelerators are easier to design and deploy in the immediate short term, beyond being more reliable than the mixed-signal counterparts since they do not suffer from the variability issues that affect analog circuits. Moreover, the design time required by mixed-signal hardware is much longer than the one of the digital counterpart; this is mainly due to the lack of advanced EDA software in the analog domain. For these reasons, this thesis focuses on digital hardware implementations of SNNs accelerators.

Together with brain-inspired models of computation, some new sensors that emulate the human retina emerged in recent years: event-based vision sensors (EVSs) [38]. Misha Manowald, during her studies at Caltech under the supervision of Carver Mead, developed the first stereo vision device that emulates how information is conveyed from the retina to the visual cortex of the human brain. These new sensors are now being investigated to provide a more efficient machine vision, with potential applications ranging from autonomous driving to robotics.

The key characteristic of this new class of cameras is that, differently from conventional cameras, each pixel *independently* senses the light in front of the camera and registers only *variations*. Every pixel operates *asynchronously* with respect to the others, sending information out in form of *events*. This kind of information elaborations resembles the one of SNNs, and automatically captures the *sparsity* in the scene: where the scene does not vary in time, no data is provided in output by the corresponding pixels in the sensor.

At the current moment, there are no hardware platforms specialized to run SNN inference in an efficient way. The Intel Loihi chip [23], for instance, is still far from commercial availability, and the prototypes provided to researchers are not able to fit modern deep neural networks (DNNs) [52]. This might be due to the fact that the all-to-all neural connectivity design choice pursued in Loihi allows any topology to be mapped to the chip but, at the same time, routing large networks on the chip is a very difficult task. While all modern DNN accelerators [13, 64, 75, 59, 100, 84] choose a *forward* architecture, *i.e.*, the network is processed layer by layer, treating the activations, weights and neurons as matrices, network-on-chip (NoC) chips like Loihi treat the neurons as single entities, being updated independently everytime an input to them arrives. This seems a good design choice, in theory, if one supposes that the input is *very sparse*; moreover, it allows extreme flexibility in the neurons connectivity. However, the actual routing on the chip results to be really difficult, and accessing single neurons parameters from memory (even local ones) might worsen the energy consumption of the system [71].

Startups such as Synsense [99] and Innatera [55] are arising to provide efficient digital and mixed-signal hardware platforms to deploy SNNs. However, at the moment, these are not available to researchers and the prototypes cannot fit DNNs such as large as ResNet [52]; the reason why deep neural networks are needed is that these are capable of learning more complex representations, and one can expect the same to hold true also for SNNs, even though the number of layers in these is limited by current training methodologies and algorithm, that emulate very closely the ones employed for artificial neural networks (ANNs). Moreover, on tasks such as optical flow estimation using event cameras, deep SNNs are required to get performance comparable with ANNs. For this reason, the research community and the market need well established hardware platforms that are more efficient than GPUs when it comes to quantized models but, at the same time, are reliable and easily purchasable.

There are cloud facilities such as SpiNNaker [37] and BrainScaleS [87]. SpiNNaker is a custom chip consisting of multiple ARM general purpose cores interconneted *asynchronously* on a NoC. A general purpose processor allows for a great degree of flexibility and solid software pipelines for their programming; however, this comes at the cost of high inefficiency in running DNNs, even when compared with non-special purpose hardware as GPUs. BrainScaleS is a custom analog CMOS chip that tries to emulate the biological neurons as close as possible using the intrinsic characteristic of MOSFETs. However, BrainScaleS is meant to be used in neuroscience applications to explore brain models, rather than for DL applications. It cannot be used by the common DL practitioner to test machine learning (ML)-oriented models and

deploy a small system "in the wild".

Field-programmable gate arrays (FPGAs) might be a good candidate to fit these needs, as the implementation of custom hardware on these might provide very efficient implementation of DNNs without resorting to designing an application specific integrated circuit (ASIC). By *efficiency* we mean the number of operations (or, in the case on DNNs, inferences) per unit of energy consumed by the accelerator. Moreover, very few hardware platforms [28] target event based vision, which could be one of the applications most compatible with SNNs, given the event-based, temporal characteristics of the information provided by an event camera.

The objective of this thesis is to provide a critical and objective evaluation of SNNs hardware implementation, with a focus on digital hardware. The analysis is used to identify tasks that more suitable for SNNs, and provide guidelines for the research community.

Then, state-of-the-art (SOTA) design techniques for digital hardware architectures targeting DL workloads are analyzed and exploited to produce an accelerator infrastructure based on high level synthesis (HLS) for runnning SNNs on digital hardware (in particular, FPGAs) is proposed and implemented. The objective of the infrastructure is to allow designers to easily implement SNNs and deploy these to an FPGA board to test it on the field.

Moreover, differently from what is done in the SNN literature at the current moment, in this thesis SOTA techniques for conventional DL accelerators are explored and analyzed, and then applied to the accelerator infrastructure to maximize performance and minimize memory usage when running the neural network in hardware.

The network architecture family targeted is the convolutional neural networks (CNNs) one, for event-based vision applications. In this thesis, is shown that SNNs are not competitive with conventional (artificial) DNNs on spatial data, such as images; for this reasons, the accelerator targets and is tested on an event-based vision task: optical flow estimation out of event-based cameras data. The neural network still underperforms with respect the artificial counterpart in terms of accuracy, but the energy efficiency is improved by a factor $10\times$ with respect a neuromorphic processor baseline, which shows the advantage of bringing SOTA architectural design techniques from the artificial DL world to SNN design.

The outline of the thesis is as follows:

- in Chapter 2, a background on SNNs, event-based vision, digital hardware accelerators for DNNs and HLS is provided. This forms the base for the SOTA DL hardware design techniques being applied to SNNs in Chapter 4.

- in Chapter 3, the landscape of digital hardware accelerators for SNNs is analyzed, benchmarking the chips on spatial tasks, such as image classification, and spatio-temporal (*i.e.*, with data evolving through time in a single input sample) tasks, such as audio processing. An energetic model for a CNN layer being run on a reference digital architecture is provided for SNNs and ANNs, showing that SNNs are inherently less efficient than ANN on spatial tasks, and the model is validated by the data available in the literature. This motivates the choice of targeting only event-based vision workloads in the hardware architecture for SNN inference proposed in Chapter 4.

- in Chapter 4, a C++ HLS library the accelerator infrastructure for running SNNs on FPGAs is presented, with the corresponding HLS library written in C++. An accelerator architecture based on [75] is proposed, with the design choices needed to efficiently map SNNs to

hardware. An accelerator generated in the infrastructure is then validated on an event-based vision optical flow estimation task, during a project carried out in collaboration with Professor Charlotte Frenkel and Professor Guido de Croon, at Delft University of Technology.

- in Chapter 5, the thesis is concluded with a discussion on future work and a perspective of the current research landscape.

Chapter 2

Background

In this chapter, the concept of SNN is introduced, starting from the description of the most used neuron model in the literature, the leaky integrate and fire (LIF) one.

Then, an introduction to the design of digital hardware accelerators for DL applications is provided, with a classification of the main architectures available in literature and a discussion on their differences. These concepts, such as *dataflow* and *spatial* architectures, are then used as guidelines to design the hardware architecture of the SNN accelerator presented in Chapter 4.

Finally, the concept of HLS of digital circuits is introduced, since this represent an effective tool to speed up the design time of complex digital systems, such as DL models accelerators. HLS is exploited to produce an accelerator generator for SNNs, which is described in Chapter 4.

2.1 Neuromorphic computing

In the definition of neuromorphic computing used in this thesis, both ANNs and SNNs are included. This is due to the fact that while SNNs are more strictly inspired by biology, with the emulation of neuron spikes and potentials, several concepts in ANNs have been inspired by the human body, such as the many-layers representation [65], convolutions, recurrent connections and more.

Hence, by neuromorphic computing, the computational models inspired by the brain are defined, such as artificial and spiking neural networks. SNNs emulate biology using spikes, while ANNs use conventional computer architecture data formats to encode information.

2.1.1 Biologically inspired neurons

In biological neural networks, neurons communicate by means of spikes: these activation voltages are then converted to currents through the synapses, charging the membrane potential of the destination neuron. SNNs emulate these behaviors: voltage spikes are simulated via single bits in digital circuits, and actual spikes in analog ones. Different neuron models are available in the literature [32]; the model described in the following is the LIF one, which is among the most computationally efficient in its hardware implementation.

The destination neuron is denoted as *post-synaptic* neuron, with the index i, while the input neuron under consideration is denoted as *pre-synaptic* neuron, with the index j. We denote the

input spike train incoming from the pre-synaptic neuron with $s_j(t)$:

$$s_j(t) = \sum_k \delta(t - t_k)$$

where t_k are the spike timestamps of the spike train $s_j(t)$.

The synapse connecting the pre-synaptic neuron with the post-synaptic neuron is denoted with w_{ij}. A synapse is described by a numeric value, called weight. All the incoming spike trains are then integrated by the post-synaptic neuron membrane; the integration function is modeled by a first-order low-pass filter, denoted with $\alpha_i(t)$:

$$\alpha_i(t) = \frac{1}{\tau_{u_i}} e^{-\frac{t}{\tau_{u_i}}}$$

The spike train incoming from the pre-synaptic neuron, hence, is convolved with the membrane function; in real neurons, this corresponds to the input currents coming from the pre-synaptic neurons that charge the post-synaptic neuron membrane potential, $v_i(t)$. The sum of the currents in input to the post-synaptic neuron is denoted with $u_i(t)$ and modeled through the following equation:

$$u_i(t) = \sum_{j \neq i} w_{ij} \cdot (\alpha_v * s_j)(t)$$

Each pre-synaptic neuron contributes with a current (spike train multiplied by the w_{ij} synapse) and these sum up at the input of the post-synaptic neuron. Given the membrane potential of the destination neuron, denoted with $v_i(t)$, the differential equation describing its evolution through time is the following:

$$\frac{\partial}{\partial t} v_i(t) = -\frac{1}{\tau_v} v_i(t) + u_i(t)$$

In addition to the input currents, we have the neuron leakage, $\frac{1}{\tau_v} v_i(t)$, modeled through a leakage coefficient $\frac{1}{\tau_v}$ that multiplies the membrane potential.

Such a differential equation cannot be solved directly using discrete arithmetic, as it would be processed on digital hardware; hence, we need to discretize the equation. This discretization leads to the following result:

$$v_i[t] = \beta \cdot v_i[t-1] + (1 - \beta) \cdot u_i[t] - \vartheta \cdot s_i[t]$$

where β is the decay coefficient associated to the leakage. We embed $(1-\beta)$ in the input current $u_i[t]$, by merging it with the synapse weights as a scaling factor; in this way, the input current $u_i[t]$ is normalized regardless of the decay constant τ_v value.

Notice that the membrane reset mechanism has been added: when a neuron spikes, its membrane potential goes back to the rest potential. There are different approaches to modelling the reset mechanism: in the *hard* reset one forces to zero the state in the same timestamp a spike is generated; in the *soft* mechanism, instead, the threshold ϑ is subtracted from $v_i(t)$. In this model, the spike is modeled with a Heaviside discontinuos function $s_i[t]$:

$$s_i[t] = \begin{cases} 1 & \text{if } v_i[t] \geq \vartheta \\ 0 & \text{otherwise} \end{cases}$$

$s_i[t]$ is equal to 1 at spike time (*i.e.*, if at timestamp t the state $v_i[t]$ is larger than the threshold ϑ) and 0 elsewhere.

The input current is given by:

$$u_i[t] = \sum_{j \neq i} w_{ij} \cdot s_j[t]$$

Notice that since $s_i[t]$ is either 0 or 1, the input current $u_i[t]$ is equal to the sum of the synapses weights of the pre-synaptic neurons that spike at timestamp t.

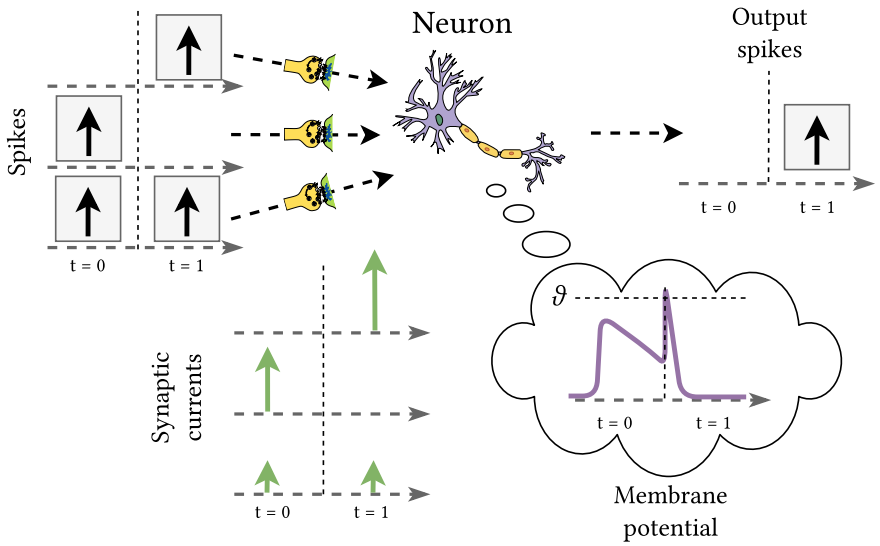


Figure 2.1: Model of a LIF spiking neuron, characterized by a leakage of the membrane potential in time. When the input currents, generated by the input spikes being applied to the synapses, charge the membrane potential above the spiking threshold ϑ, an output spike is generated. The neuron and synapses images are taken from Wikicommons (`https://commons.wikimedia.org/wiki/Neuron`).

When considering multiple layers of neurons, a pedix l can be introduced to denote the l-th layer, and the state update equations can be modified as follows:

$$v_{l,i}[t] = \beta \cdot v_{l,i}[t-1] + u_{l,i}[t] - \vartheta \cdot s_{l,i}[t-1] \tag{2.1}$$

$$u_{l,i}[t] \stackrel{\Delta}{=} \sum_j w_{l,ij} \cdot s_{l-1,j}[t] \tag{2.2}$$

(2.1) requires three inputs to update the state $v[t]$: the weights w_{ij}, the previous value $v[t-1]$ and the input spikes $s_j[t]$. This means that a SNN hardware processing engine (PE) needs access to 3 memory structures to retrieve the inputs, the state, and the weights.

2.1.2 The artificial neuron

The artificial neuron does not produce spikes but discrete values, consisting of floating point or integer values when implemented in digital hardware. The most common artificial neuron

model [49] is state-less, *i.e.*, it does not preserve any state between successive inputs. A particular class of ANNs, namely recurrent neural networks (RNNs), employ special gates that introduce a state in the neuron [49]. Here, the stateless ANN neuron, derived by the one proposed by McMulloc and Pitts [72], is analyzed. The artificial neuron model is described by (2.3).

$$z_{l,i} = \varphi(\sum_{j} z_{l-1,j} \cdot w_{l,ij} + b_{l,i}) \tag{2.3}$$

The weights are multiplied by the inputs from the previous layer, $z_{l-1,j}$, and accumulated; a bias term $b_{l,j}$ can be added. An activation function, $\varphi(x)$, is applied to the accumulated value. The specific choice of activation function often depends on the application, layer, and the type of neural network [49].

Distinct from (2.1), only two inputs are needed in (2.3): the previous layer activations $z_{l-1,j}$ and the current layer weights $w_{l,ij}$. This implies that only two memory structures are needed for the ANN model being implemented in hardware. This represents an advantage over SNN implementations in both area occupation of the circuit, and energy consumption, since memory accesses are the most energy-intensive operations in modern digital hardware architectures [71]. One could point out that in-memory computing accelerators do not suffer from this problem, since computation and data are co-located; however, ANNs still have an advantage in this regard, as the memory locations dedicated to the state of an SNN could be repurposed for weights. Of course, in the context of stateful networks, there is no disadvantage to state storage as this is needed for the task.

When considering recurrent topologies, different *cells* can be employed, such as the long short-term memory (LSTM) and gated recurrent unit (GRU) ones [49]. A *vanilla* version of a recurrent cell is reported in (2.4):

$$\begin{aligned} h[t] &= \sigma_h(U_h \cdot x[t] + V_h \cdot h[t-1] + b_h) \\ o[t] &= \sigma_o(W_o \cdot h[t] + b_o) \end{aligned} \tag{2.4}$$

$h[t]$ represents the state of the cell, while $o[t]$ its output. The functions σ_h and σ_o are non-linear activation functions. The remaining quantitities are the cell parameters, which are learned through training and determine how the cell state is updated through time and how the output is determined by the state at a given timestamp. Hence, in the case of artificial RNNs, a state is kept between computations. The most important difference with SNNs is that the activations produced are non-binary.

Of course, the LIF and artificial models are not the only ones available. One could include, more accurate imitations of the biological neuron, such as the Izhikevic [56] one, and other neurons inspired by biology. For instance, the expressive leaky memory (ELM) [97] neuron is a phenomenological model of a cortical neuron with extremely interesting properties regarding long-range temporal tasks. The models explored in this thesis are the most efficient from a computational perspective, and the most employed in the literature. A comprehensive overview of hardware accelerators for these models is provided in Chapter 3.

2.2 Neuromorphic vision

Event-based cameras [38] have been recently introduced as alternative vision sensors to frame-based, RGB cameras. While conventional cameras sample light intensity on every pixel of

the sensor at a fixed frequency (usually, around 30 Hz) to produce an image, event cameras measure the variation in light intensity at the single pixel level. In particular, when a pixel records a variation in the luminance, depending on the slope of this variation (increasing or decreasing), it produces an event, denoted with e_i, that contains the timestamp of the luminance variation measured with respect to a reference, t_i, the pixel position of the frame, (x_i, y_i), and the polarity of the variation (positive or negative), p_i. Hence, an event can be described as a tuple $e_i \triangleq (x_i, y_i, t_i, p_i)$.

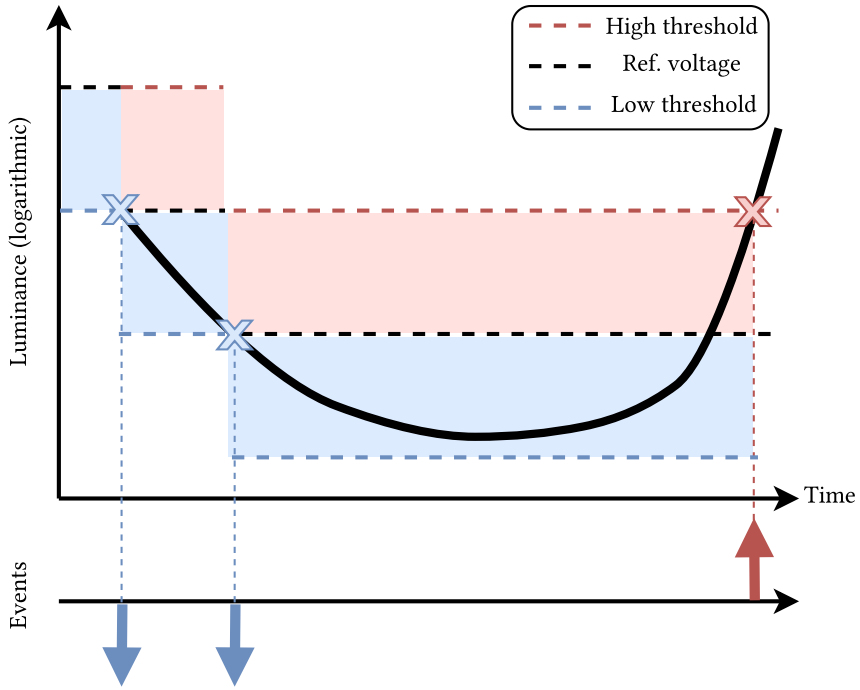


Figure 2.2: How an EVS works. The sensor works with two varying thresholds: a high one and a low one. When the input signal, which is the luminance, crosses one of these thresholds, an output event is generated, which polarity is determined by the threshold being crossed. Then, the sensor reference voltage and thresholds move in the direction of the event polarity to sense other variations in the input luminance. This image has been adapted from Sony: `https://www.sony-semicon.com/en/technology/industry/evs.html`.

In Fig. 2.2, the working principle of an EVS is presented. The logarithmic luminance variation is measured through time by the pixel; this is achieved by varying the two thresholds of the comparator of the sensor: when the luminance crosses the high threshold, a positive event is generated by the pixel and the comparator threshold is increased; viceversa, when the luminance crosses the low threshold, a negative event is generated and the threshold is lowered.

2.2.1 Event-based vision applications

Event cameras are characterized by sub-millisecond latency in event generation and outstanding high dynamic range (HDR) (around 120 dB). Moreover, the output of an event camera is

naturally sparse, since only the pixels that register a luminance variation produce data, while the others remain silent; this is in stark contrast with frame cameras, in which all pixels values are sensed at a fixed clock frequency, and the resulting output is a dense tensor being provided. This difference leads to an important difference in the information representation, which requires a change in the way this is processed by the algorithms.

Event-based object detection

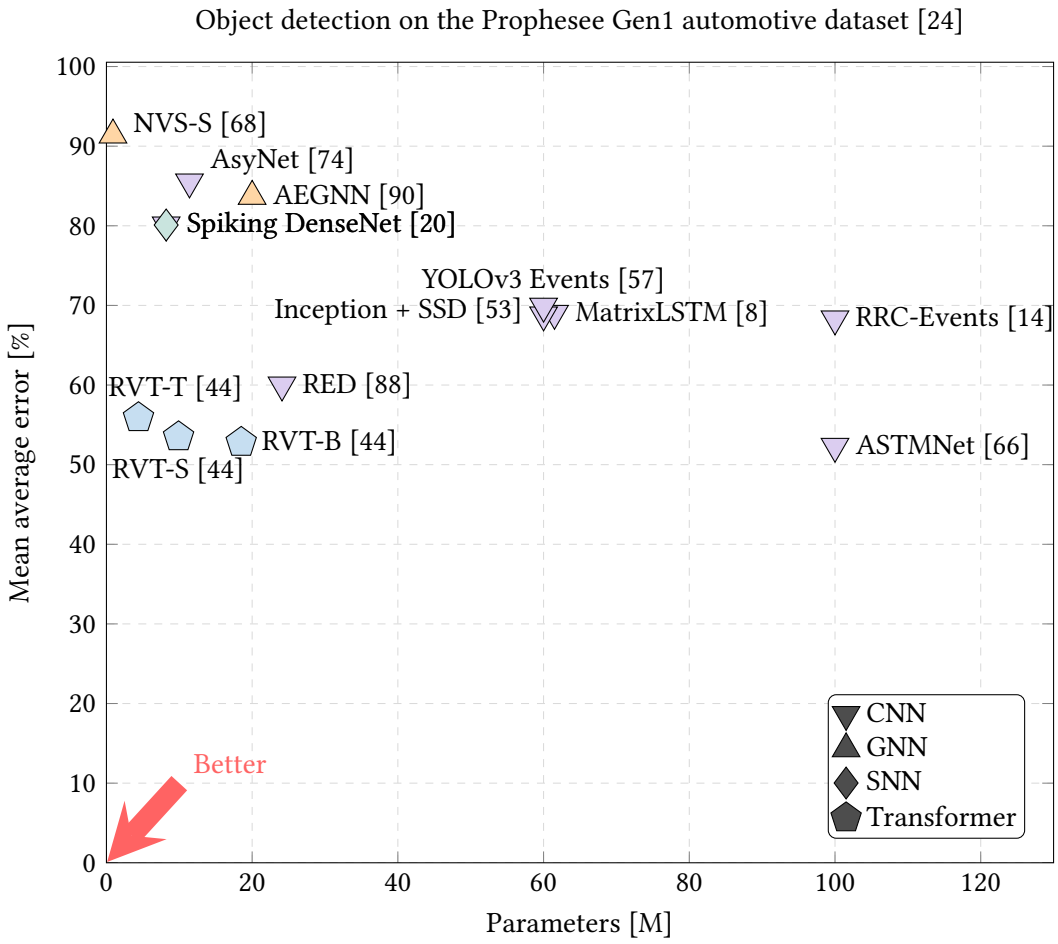


Figure 2.3: Mean average error (defined as the complement of mean average precision on object detection of DL models with respect to modes size on the Gen1 automotive event dataset. Data are taken from Gehrig et al. [44]

SOTA DL algorithms for the processing of event cameras data for different task, such as object recognition and detection, still rely on dense representations: events are converted to tensors just like the ones provided by frame cameras, that are then processed by deep CNNs. However, in this way, the processing is not event-based anymore, and the inherent sparsity provided by event cameras is lost.

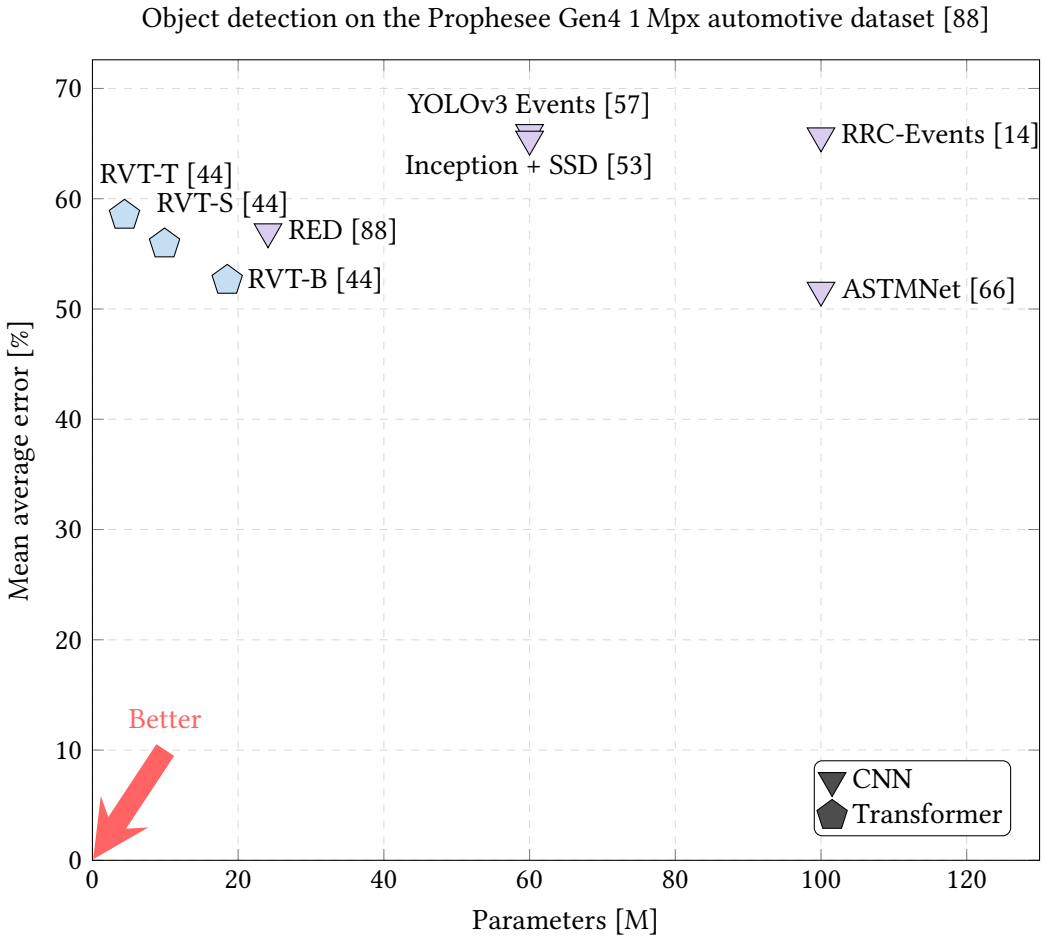


Figure 2.4: Mean average error on object detection of DL models with respect to modes size on the 1 Mpx automotive event dataset. Data are taken from Gehrig et al. [44]

In Fig. 2.3 and Fig. 2.4, the performance on object detection of different DL models is shown. Two datasets are considered, both event-based: in Fig. 2.3, the Prophesee Gen1 automotive dataset [24], which consists of camera recordings of a road environment, taken using the Gen1 Prophesee EVS with a frame resolution of 304×240 px, is used; in Fig. 2.4, the Prophesee Gen4 automotive dataset is considered, which is similar to the Gen1 dataset with the difference that a 1280×720 px sensor is used. These two datasets represent the most complex event-based vision tasks to date.

Different models are benchmarked on these datasets on the object detection task. The metric used to evaluate the performance is mean average precision (mAP), which measures the relative precision with which an object is detected in front of the camera. Conventional CNN models preprocess the events to a dense frame-based representation compatible with conventional convolutional layers. Other models, such as the SNN and graph neural network (GNN) ones, instead, process the raw events; in particular, the GNN models [90] update only portions

of the network depending on the events position. One can also notice how SNN models perform worse than their ANN counterparts; this is mostly due to the fact that training algorithms for SNNs are not as mature as ANN ones, and that the spiking mechanism introduces a hard quantization in the model. In fact, while the SNN state is mapped to a multi-bit quantity, allowing for a large range of values, the output of the cell is still a single bit one, which does not allow to map information to it in magnitude. Attention should be paid also to the transformer models: these result to be among the smallest models in terms of parameters, while providing outstanding performance in terms of mAP on such a complex task as the Gen4 dataset from Prophesee.

Event-based optical flow

Optical flow estimation is the problem of computing the velocity of objects on the image plane without knowledge about the scene geometry or motion. It is a computationally intensive and challenging problem, even more so when dealing with raw events. In frame-based cameras one considers two consecutive images and makes assumptions about the brightness constancy and smoothness; in this way, equations can be formulated to solve the flow at the pixel level. Raw events coming out of an EVS, instead, do not carry visual information by themselves, and one needs to aggregate and process them to compute the flow.

To estimate optical flow using the output of event cameras, the main approach adopted in the literature is to use neural networks. The events are either fed to a temporal encoder that puts them in a dense tensor with a preprocessing that encodes the temporal information in it, so that stateless neural network, such as a CNN, can compute the flow out of it, or these are provided directly to a stateful network that adds the time information on its own. The second approach is the one that better suits the event-driven operation of event cameras, and both ANNs and SNNs have been adopted with this methodology. The architecture of the network employed is a convolutional one, with the difference that a state is embedded in the activations, by either using variant of the LIF spiking neuron, or stateful neurons from the ANN domain such as GRUs and LSTMs.

Hagenaars et al. [51] modify the methodology proposed in Zhu et al. [116] to construct a fully self supervised learning (SSL) training pipeline for SNNs, and test the model on event-based optical flow datasets. In particular, the SNN is trained on the UZH-ZPV drone racing dataset [25], and validated on the MVSEC [115] dataset to test for generalization of the model.

The approach followed in the paper is to take two SOTA neural network architectures, EV-FlowNet [114] and FireNet [91], and add recurrence and spiking activations to these. The architectures are reported in Fig. 2.5. Different variations of the LIF neuron are used to benchmark different behaviors and how these affect performance. In Schnider et al. [93], another neural network architecture is proposed, inspired by Timelens [102], an event-based methodology for video frame interpolation. Schnider et al. propose a cell based on the spiking neural unit (SNU) [5], which is an extension of the LIF model that includes axonal dynamics to model neural connectivity. In particular, two variants of the SNU model are proposed: SNUo, that models axon-to-axon synaptic dynamics; SNUa, that models axon-to-soma synaptic dynamics. The SNUo model does not output spikes but analog values, *i.e.*, multi-bit units. Its functioning resembles the one of a gated unit such as GRU. The model equations are reported in (2.5), (2.6)

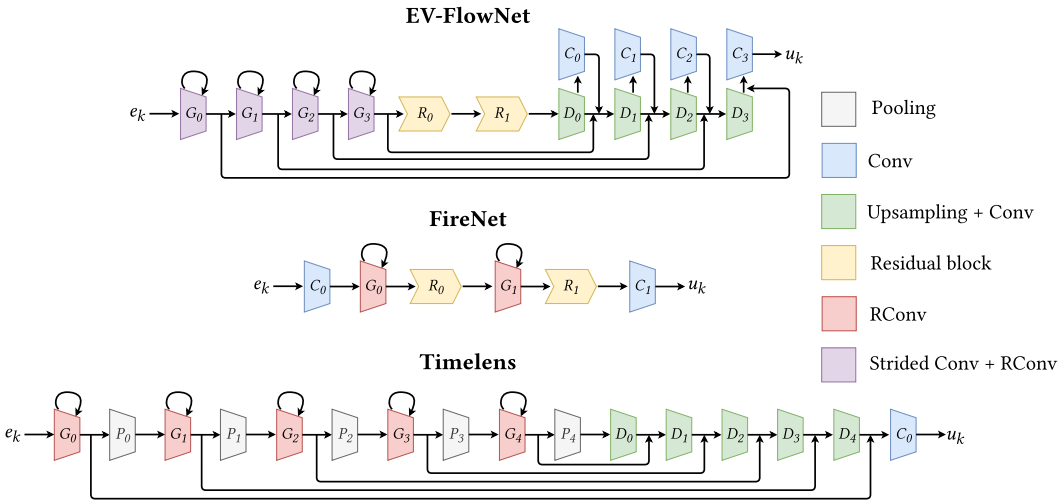


Figure 2.5: The neural networks investigated in Hagenaars et al. [51], EV-FlowNet [114] and FireNet [91]. Only the activations and recurrent cells are varied between the neural networks explored in the paper. Timelens is proposed as more performant alternative in Schnider et al. [93]. e_k denotes the input events slice, while u_k denotes the optical flow vector being produced by the network.

and (2.7).

$$h[t] = \sigma_h(W \cdot x[t] + H \cdot o[t-1] + d \cdot h[t-1] \cdot (1 - \tilde{o}[t-1])) \tag{2.5}$$

$$\tilde{o}[t] = \sigma_{\tilde{o}}(h[t] - \vartheta) \tag{2.6}$$

$$o[t] = \tilde{o}[t] \cdot \sigma_o(W_o \cdot x[t] + H_o \cdot o[t-1] + b_o) \tag{2.7}$$

$o[t]$ represents the discrete analog output of the neuron; $\tilde{o}[t]$ is its unmodulated version, which representation depends on the choice of the thresholding function $\sigma_{\tilde{o}}$ [5]. In the variant proposed in [106], a sigmoid activation is used and the corresponding model is called smooth spiking neural unit (sSNU).

One can notice that, differently from the vanilla variant proposed in (2.4), the previous values of the outputs $o[t-1]$ and $\tilde{o}[t-1]$ are used to update the state $h[t]$ of the cell and also the output $o[t]$. Hence, here two recurrent connections are exploited: one on the output ($o[t-1]$, $\tilde{o}[t-1]$) and one on the state ($h[t-1]$).

In Table 2.1, the neural network architectures reported in Fig. 2.5 are analyzed using different activation cells, from spiking ones derived from the LIF and SNU models, to conventional artificial ones such as the GRU cell. All the networks are trained on the UZH-ZPV drone racing dataset [25], and validated on the four sequences of the MVSEC [115] dataset. The metrics used to compare the networks are the average endpoint error (AEE) and the percentage of outliers, $\%_{\text{out}}$, when performing optical flow estimations, averaged over the four sequences. The lower these values, the better the performance of the network.

One can notice that the Timelens architecture always outperforms the other ones, even when considering the spiking variant SNUo. This may be associated to a variety of reasons: the spiking model employed is more complex and, hence, it is able to learn better than a plain LIF one; in the case of the ANN version, sSNU, this outperforms both the vanilla recurrent cell and the GRU one.

Table 2.1: Quantitative evaluation of the network architectures analyzed in Hagenaars et al. [51], on the MVSEC [115] dataset, on the optical flow estimation task. Each network is trained using the SSL methodology proposed in Hagenaars et al. [51]. For each sequence of the MVSEC dataset (`outdoor_day1`, `indoor_flying1`, `indoor_flying2` and `indoor_flying3`), the average endpoint error (AEE) (lower is better) in pixels and the percentage of outliers, $\%_{\text{out}}$ (lower is better), of each network are measured. Here, the average scores across the four scenarios of MVSEC are reported, represented by $\overline{\text{AEE}}$ and $\overline{\%}_{\text{out}}$ in the table. The `dt` parameter reports the displacement in time between events and active pixel sensor frames (the frames are used to estimate the ground truth). Best in **bold**, runner up underlined.

`dt = 1`	Cell	**EV-FlowNet** [51]		**FireNet** [51]		**Timelens** [93]	
		$\overline{\text{AEE}}$	$\overline{\%}_{\text{out}}$	$\overline{\text{AEE}}$	$\overline{\%}_{\text{out}}$	$\overline{\text{AEE}}$	$\overline{\%}_{\text{out}}$
ANN	ConvGRU	0.79	3.61	1.10	6.89	-	-
	ConvRNN	0.83	4.10	1.21	8.57	-	-
	sSNU	-	-	-	-	**0.77**	3.89
SNN	Leaky	0.95	5.09	1.13	8.08	-	-
	LIF	0.96	5.90	1.13	8.08	0.87	4.09
	ALIF	1.22	8.95	1.27	9.34	-	-
	PLIF	1.02	6.06	1.13	6.99	-	-
	XLIF	0.95	5.40	1.22	8.96	-	-
	SNUo	-	-	-	-	0.79	**3.44**
`dt = 4`							
ANN	ConvGRU	2.69	26.08	3.94	44.47	-	-
	ConvRNN	2.80	27.77	4.38	47.83	-	-
	sSNU	-	-	-	-	**2.66**	**25.66**
SNN	Leaky	3.34	36.41	4.07	42.91	-	-
	LIF	3.36	35.26	4.33	47.96	3.09	32.88
	ALIF	4.47	49.60	4.64	51.93	-	-
	PLIF	3.59	39.10	4.12	46.55	-	-
	XLIF	3.59	39.10	4.43	48.64	-	-
	SNUo	-	-	-	-	2.76	28.47

A more complete analysis would involve the LSTM cell, which is the SOTA cell in the RNN domain (we do not consider transformers to be recurrent since these completely unroll the input sequence). Recently, a new kind of neuron inspired by the ones in the brain cortex, the ELM neuron [97], has been introduced, that can potentially outperform an LSTM cell on long temporal dependencies. A model like this could lead to even better performance with small networks.

One can notice that SNNs *always* underperform with respect to ANNs, in both spatial and spatio-temporal tasks. For spatial tasks, this is probably due to the fact that spikes are not able to *densely* encode information as well as multi-bit activations (*e.g.*, an INT8 activation in a ViT), which leads to a high quantization error. For spatio-temporal tasks, cells such as the LSTM provide more powerful non-linear function that are applied to the state of the network;

this allows the training algorithm to deal with complex tasks, such as optical flow estimation, better. The activation function of an SNN results to be a "simple" multiplication by a leakage and a thresholding, in the LIF neuron case, which severely limits the learning capabilities of the network.

2.3 Hardware acceleration of neural networks

Due to the increasing adoption of DL models in everyday applications, researchers and companies have been developing different hardware accelerators to make DNNs deployment more efficient from an energetic perspective, but also to increase performance (*e.g.*, inference throughput). The fundamental operation of a neural network is the multiply and accumulate (MAC), and different techniques are used to parallelize the MACs in a network, since this is the only way on increasing efficiency and performance since the end of Moore's law. Moreover, in modern computer architecture the highest time and energy costs are associated to memory [71]; hence, to improve performance, one has to employ methodologies that minimize the number of accesses to the higher levels of the memory hierarchy. In the following, an overview of these techniques is provided.

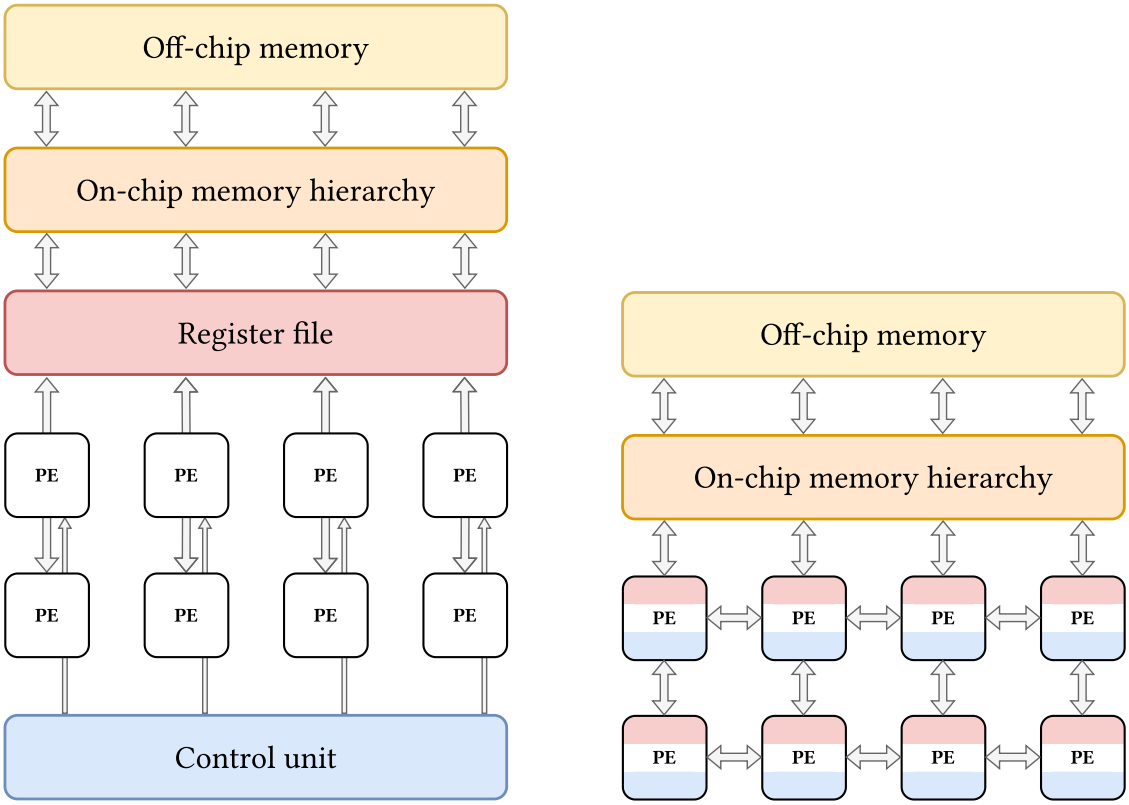
2.3.1 Hardware accelerators classification

The recent literature of DNN digital accelerators is rich of examples [96, 80, 9]. The conventional division of digital hardware accelerators for DL workloads is in *temporal* and *spatial* architectures [64]. To understand better the difference among these, consider the architecture shown in Fig. 2.6. The general architecture we are considering is made of five main components: (i) an array of PEs, which carry out the computations; (ii) a control unit, which give instruction to the arithmetic blocks in the PEs on which operation to carry out; (iii) a register file, which is used to host the inputs to the PEs and their outputs, partial and final ones; (iv) an on-chip memory hierarchy, which is made of slower and more energy-hungry hardware with respect to the register file, but with higher capacity; (v) an off-chip memory, which has a larger capacity than the on-chip one but is much more costly to access, in terms of both energy consumption and latency.

In *temporal* architectures (Fig. 2.6a), the register file and the control unit are centralized outside the PEs array. PEs do not communicate among each other: they read from and write to the register file all the data. This architecture is typical of general purpose hardware, such as central processing units (CPUs) and GPUs, in which techniques as single instruction multiple data (SIMD) and single instruction multiple threads (SIMT) are used to exploit the parallelism of the computations to reduce the processing time.

In *spatial* architectures (Fig. 2.6b), each PE may host a small control unit and a small register file. Moreover, the PEs can communicate among each other through a NoC, exchanging data: a PE can use as input the result produced in the previous clock cycle by a neighbor PE. This methodology is called *dataflow* processing, since the PEs form processing chains that pass intermediate results one to another. Usually, DNNs accelerators on ASICs or FPGAs are implemented in this way [100, 9].

Among spatial accelerators, one can define other two architecture families: *overlay-based* architectures and *fused-layer* (or *streaming*) architectures.



(a) A temporal architecture, with central register file and control unit.

(b) A spatial architecture, with decentralized register file and control unit.

Figure 2.6: Examples of temporal and spatial architectures, inspired by Sze et al. [100]. In *temporal* architectures, the PEs do not communicate among each other and perform only computations. All the PEs receive instructions from a central control unit, read their inputs from a central register file and write back their results to it. In *spatial* architectures, instead, control and memory are delocalized, and PEs can exchange data to chain computations.

Overlay-based architectures (Fig. 2.7) map single layers to the accelerator PEs at each cycle, maximizing the parallelism on it. To do this, all the activations of the previous layer in the network must be computed and saved back to memory (either the global buffer or, if the feature map is too large for the on-chip memory, an external dynamic random access memory (DRAM)). In Fig. 2.7, the computation of the product of 3 matrices A, B and C is shown. In overlay accelerators, all the PEs would be mapped to computing the first product $A \cdot B$, diving these in sub-matrices and mapping them to the single PEs. The resulting matrix, $A \cdot B$, is saved back entirely to the on-chip memory, to be read back in the next cycles and compute the final product, $(A \cdot B) \cdot C$.

Fused-layer architectures [45] (Fig. 2.8), instead, run multiple layers concurrently on the available PEs. This approach leads to lower memory footprint for the intermediate results, that can be used in the next cycle by other PEs. In Fig. 2.8, the three matrices are multiplied all together partially in multiple iterations, and the final product is obtained by merging the intermediate results from the previous iteration.

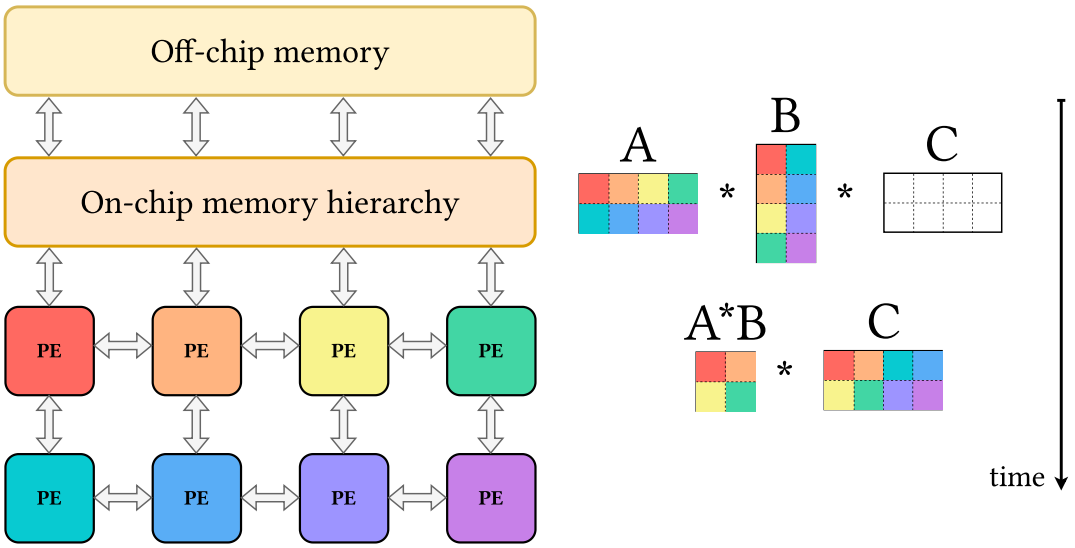


Figure 2.7: Overlay architecture. The computations are scheduled one at a time to the array of PEs. In this example, when computing $A \cdot B \cdot C$, in an iteration the product $A \cdot B$ is mapped to the entire array and the intermediate result, $A \cdot B$, is written back to memory; then, in the next iteration, it is retrieved as inputs to compute the last part of the matrix multiplication, $(A \cdot B) \cdot C$.

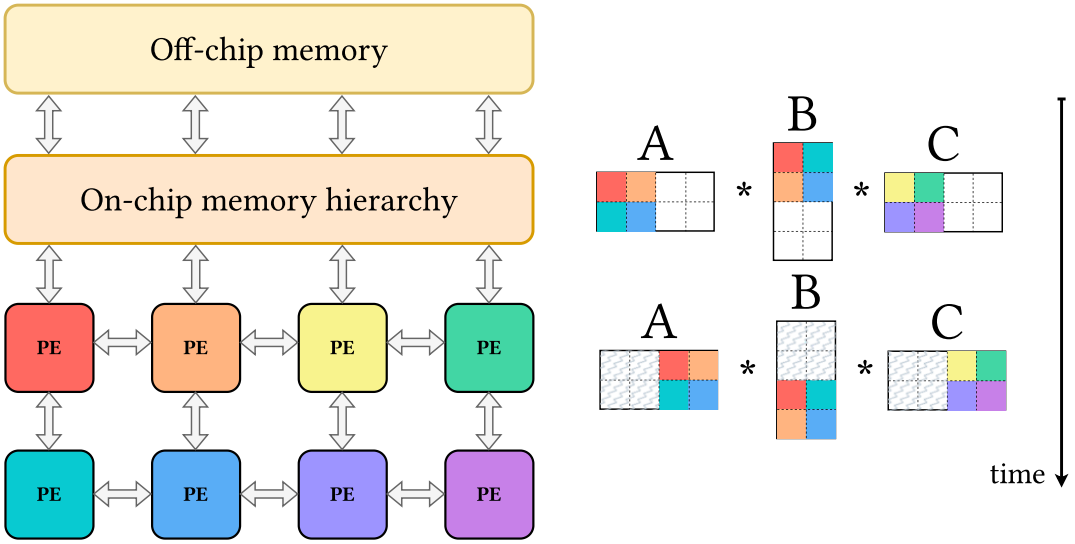


Figure 2.8: Fused-layer architecture. Here, differently from Fig. 2.7, the computations are *fused*: the matrices are partially multiplied together in the first iterations, since the entire matrices would not fit on the array of PEs; then, the remaining part of the product is finalized in the subsequent iterations. Layer fusion proves to be effective if the temporary results of the first sub-matrix multiplication can be stored entirely to on-chip buffers, without writing them back to the off-chip memory.

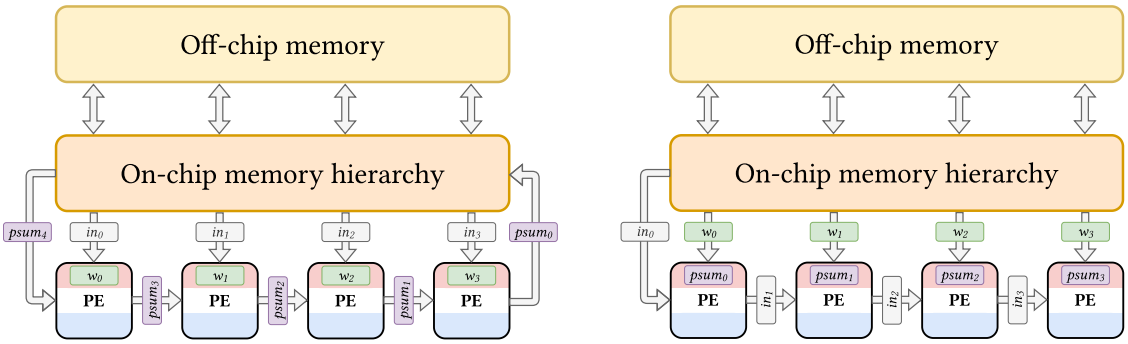
Most modern DNN accelerators [9] use an overlay-based architecture: the network is processed layer-by-layer, scheduling all the operations associated to the current layer before the next layer ones. The intermediate feature map (the matrix $A \cdot B$ in the example of Fig. 2.7) is

often too large to fit on-chip, so it is streamed off-chip during the processing of the first layer and back on-chip during the processing of the next. Layer fusion eliminates off-chip transfers of intermediate results between the fused layers, saving energy and reducing latency. To reduce on-chip buffer size, fused-layer dataflows tile the intermediate data and interleave the scheduling of producer and consumer operations. To exploit reuse, the dataflow buffers values on-chip.

2.3.2 Dataflow in deep learning hardware accelerators

An important aspect in spatial accelerators is the *dataflow*. The dataflow describes how a neural network architecture is mapped to an accelerator in *space* and *time*, *i.e.*, which data (inputs, weights and temporary outputs) gets read into which level of the memory hierarchy (space) and when these are getting processed (time).

Two main families of dataflows can be identified [12, 100, 9]: (i) *weight* stationary and (ii) *output* stationary. Some variations of these are proposed in Venkatesan et al. [103], but this general division still holds true. In Chen et al. [13], a *row* stationary dataflow is proposed, tailored for CNNs. Examples of weight and output stationary dataflows being applied to the spatial architecture proposed in Fig. 2.6b are reported in Fig. 2.9.



(a) Weight stationary dataflow.

(b) Output stationary dataflow.

Figure 2.9: Examples of weight and output stationary dataflows on a spatial architecture, inspired by Chen et al. [12]. In *weight* stationary dataflows, the weights (w_i) of the neural network are kept stationary in the PEs registers, while the inputs (in_i) are broadcasted in parallel to all the PEs and the partial results ($psum_i$, since these are the partial accumulations of the MACs being chained) are exchanged among the PEs via the NoC. In the *output* stationary case, instead, weights are broadcasted to the PEs and inputs flow from one PE to the other, while the partial results are kept stationary in the PEs registers.

In the *weight* stationary case (Fig. 2.9a), the weights of the neural network are written to the PEs register files (RFs) and kept stationary for the whole execution of the layer being mapped. The inputs are broadcasted to the PEs to be multiplied and accumulated with the results produced by neighboring PEs, indicated by $psum_i$ in Fig. 2.9a. The partial sums are exchanged by the PEs via the NoC.

In *output* stationary dataflows (Fig. 2.9b), instead, partial sums are kept in the PEs, weights are broadcasted in parallel to these and inputs are streamed through the NoC to participate in the various partial sums computation.

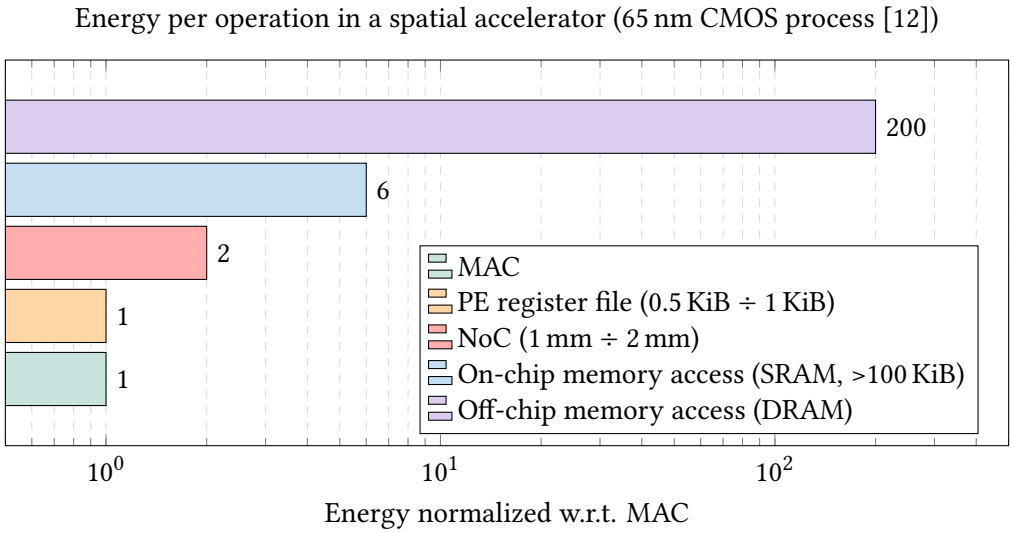


Figure 2.10: Energy consumption per operation in a spatial accelerator implemented in a 65 nm complementary metal oxide semiconductor (CMOS) process. Data are taken from Chen et al. [12]. The operations considered are a MAC computation and the memory accesses at different levels of the hierarchy (PE RF, data exchanged in the NoC, on-chip memory, off-chip memory). Energies are normalized to the minimum one, corresponding to a MAC computation.

Why does dataflow matter? Consider Fig. 2.10, where the energy cost per operation in a 65 nm complementary metal oxide semiconductor (CMOS) process for a spatial accelerator is reported. One can notice that the larger cost in terms of energy is associated to the memory accesses outside the PEs; hence, to improve the efficiency of the accelerator, one has to minimize the accesses to the higher levels of the memory hierarchy. For this reason, given a neural network architecture, the dataflow that minimizes this quantity has to be employed, analyzing which quantities (weights, partial results etc.) have to be optimized from a dataflow perspective. The choice of a dataflow given a certain neural architecture to be executed in hardware is a complex task, that depends on the hardware architecture selected (dataflow, systolic array or more), the network size and type, and requires extensive analysis and complex heuristics to be accomplished [73, 82, 103].

2.4 High level synthesis

Digital hardware accelerators are designed using hardware description languages (HDLs), such as Verilog and VHDL. However, the usage of these languages leads to a long design time, given the low level of abstraction provided by these.

Moreover, in Section 2.3 it has been shown that low level details such as efficient implementation of arithmetic units do not determine the system performance in the DL context: what matters is the dataflow and the operands precision (*e.g.*, INT8, INT4), which are determined at the higher levels of the design stack. For this reason, more abstract design methodologies are needed to speedup design time, perform design space exploration and choose the best data format, dataflow and architecture for the task at hand.

High level synthesis [58] is a design methodology that allows to use high level abstractions to perform digital hardware design. In particular, the designer describes the system functionality using a high level language, such as C or C++, and a compiler performs the translation of the source code to the register-transfer level (RTL) description that can be synthesized to digital hardware on a certain platform, e.g., FPGAs and ASICs. The designer can guide the tool to take some design decisions (*e.g.*, pipelining level, degree of parallelism in the computation) by annotating the code with C++ `pragma` statements.

The HLS compiler used in this thesis is Vitis HLS [27], which is commercialized by AMD and targets its FPGA platforms. Vitis HLS takes as input a C/C++ description of the desired functionality and produces a synthesizable RTL description optimized for FPGAs. In order to improve performance, the main objective of the HLS compiler is to extract parallelism from the source code, which is written in a sequential manner. Concurrent processes can be synchronized in HLS by means of first in first out queues (FIFOs), that in Vitis HLS are modeled using C++ objects called **`hls::stream`**.

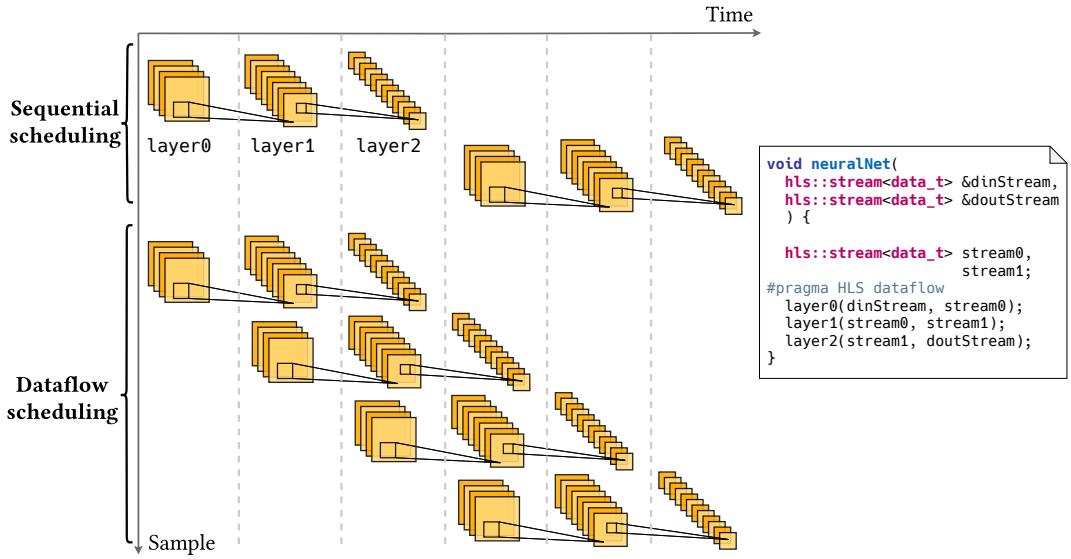


Figure 2.11: A neural network described in C++, targeting HLS compilers. To each layer, a function is assigned, and the layers are interfaced using data structures called **`hls::stream`** that emulate FIFOs. Layer execution is scheduled according to a dataflow scheme: the layers are data-driven, *i.e.*, each layer runs independently as soon its input FIFO contains data to be processed.

Figure 2.11 shows a possible C++ implementation of a neural network made of 5 layers, and the corresponding scheduling. To each layer, a C++ function is assigned, e.g. `layer0`. This function corresponds to a *task*, which is a sequence of instructions being run by custom hardware on the FPGA.

The HLS compiler is instructed to extract parallelism from the code using a `pragma` that "asks" the compiler to use a dataflow scheduling. In this, as soon as a layer produces its output, the subsequent one can consume it. This is achieved using the FIFO mechanism: when data is written to the FIFO, a flag alerts the subsequent task that there are data available and it can begin to process them. This is shown in the timing diagram of Fig. 2.11: at each timestep,

a layer produces some output that is consumed by the following one; at the same time, the layer starts processing the new incoming data without waiting for the whole neural network to finish processing the previous ones. The tasks run concurrently working on different data (*e.g.*, `layer1` consumes the data produced by `layer0` in a previous cycle, while `layer0` can start processing new inputs).

Chapter 3

Benchmarking SNNs on digital hardware

This chapter is based on our paper "To Spike or Not To Spike: A Digital Hardware Perspective on Deep Learning Acceleration", Fabrizio Ottati et. al [80], published in IEEE Journal on Emerging Topics in Circuits and Systems.

This chapter quantitatively reviews the broad landscape of digital accelerators for both SNN and conventional ANN accelerators. To extract trends and use-case driven guidelines for the use of spike-based representations, we segment the analysis into spatial and spatio-temporal (e.g., sequence learning) workloads as follows:

(i) Section 3.1 introduces the metrics and used to compare the hardware accelerators from the ANN and SNN domains, and an energetic model to approximately estimate how spatial and spatio-temporal tasks would perform on SNN and ANN hardware.

(ii) Section 3.2 analyzes and compares the hardware acceleration of convolutions for spatial vision workloads (in particular, object recognition in images), and then compares SOTA digital ANN and SNN accelerators.

(iii) Section 3.3 analyzes and compares the hardware acceleration of spatio-temporal workloads, i.e., tasks that involve sequences of inputs evolving in time.

We derive the following conclusions:

(i) Based on the current SOTA digital chips results and measurements on spatial data classification tasks, ANNs perform better than SNNs in processing efficiency and classification accuracy. Since ANNs Pareto dominate SNNs (*i.e.*, SNNs underperform in both energy efficiency and task accuracy with respect to ANNs), we claim that SNNs do not suit purely spatial tasks. A key reason is that SNN classification requires multiple timesteps, resulting in more operations than an ANN since the latter is computed in a single pass.

(ii) On spatio-temporal tasks, SNNs show energy efficiency comparable to ANN chips; furthermore, only in the SNN domain there is an example of on-chip training and learning on

spatio-temporal sequences with competitive task performance [35]; hence, further investigation in this direction is needed, together with model training techniques to improve performance (*e.g.*, classification accuracy) and system energy consumption.

(iii) Classification-based workloads employing bio-inspired sensors data, such as event cameras [38] and silicon cochleas [109], naturally fit the stateful nature of spiking neurons, but SOTA models and accelerators are not exploring this kind of tasks, which could lead to an effective advantage related to SNNs models.

(iv) The optimal solution for a given classification task might be a heterogeneous model, made of ANN and SNN parts that operate synergically together. Further research is needed in this direction.

3.1 Methodology

While evaluating the actual hardware performance, considering efficiencies in terms of operations per second per watt (OPS/W) is not enough: on spatial data, SNN would need multiple time steps to perform a classification, while ANN accelerator performs an inference in a single forward-pass. To ensure fairness, this analysis adopts the energy consumption per inference as the most balanced metric for comparison; latency and accuracy-related metrics are accounted for separately.

In ANN architectures, the most common operation is the MAC [100], which includes the input activation multiplication by the corresponding weight and the subsequent accumulation. We count each MAC as two operations, since it computes a multiplication and an addition, even if in hardware they might be merged. This is to isolate the contribution of each computation to total power consumption, and is the same approach adopted by the ANN hardware research considered in this analysis [59, 76, 104, 84]. In SNN hardware, no multiplication is performed [32]: a weight is accumulated if there is an input spike, otherwise it is not. Hence, this is considered a single operation in our analysis.

In the SNN literature, there are numerous references to the synaptic operation (SynOP) metric [4, 34]. However, there is little consensus on its formal definition. Given the ambiguous definition of this metric, it is not used to measure the efficiency of a SNN accelerator in this work. In addition to ambiguity, the SynOP metric does not tend to account for stateful operations (state access and updates).

To obtain an estimation of the energy cost of SNN and ANN architectures, two simple mathematical models are provided in Sections 3.2 and 3.3. These allow us to approximately compare ANN and SNN hardware accelerators *a priori*. While evaluating the actual hardware performance, considering efficiencies in terms of operations per second per watt (OPS/W) is not enough: on spatial data, a SNN would need multiple time steps to perform a classification, while a ANN accelerator performs an inference in a single forward-pass. To ensure fairness, this paper adopts the energy consumption per inference as the most balanced metric for comparison; latency and accuracy-related metrics are accounted for separately.

Sparsity is ignored in the energy consumption analysis performed in Sections 3.2 and 3.3, since modern ANN accelerators have complex and very efficient sparse dataflows [59, 39, 41]; however, the sparsity handling capability of SOTA accelerators is included in the energy consumption results shown in Tables 3.2 and 3.3.

3.2 Spatial tasks

(2.2) embeds one of the major advantages of SNN processing: the removal of multiplication between the input feature map (i.e., spikes) and the synaptic weights. In theory, this leads to both hardware and energy savings. However, consider the data in Table 3.1: the energy

Table 3.1: Energy consumption comparison between integer add, multiply and memory operation on on-chip static random access memory (SRAM) caches [71], targeting a 45 nm CMOS process.

	Energy [pJ]	**Energy density** [pJ/B]
Add 8 b	0.03	0.03
Multiply 8 b	0.20	0.20
Read 64 b (8 kB capacity)	10.00	2.50

consumption for reading a byte from the on-chip buffer, implemented as static random access memory (SRAM), exceeds the cost of an 8 b integer multiplication. Hence, a multiply-free SNN digital hardware accelerator is not necessarily more efficient than a ANN accelerator, given that SNNs require additional memory accesses to update the neuron states.

3.2.1 Energy model

To evaluate these mathematical models quantitatively on spatial tasks, the convolution operations shown in Algorithms 1 and 2 are used as benchmarks, since convolution is the fundamental operation employed in the large majority of current hardware accelerators for object recognition, detection and so on [84, 104, 63, 62]. more recent approaches, transformer architectures are emerging as better-performing vision models [60]; as such, a transformer accelerator [59] is considered in the final results section.

With respect to Algorithms 1 and 2, all activations, weights, and states are quantized to 8 b, while spikes are treated as unary quantities. Consider Algorithm 1:

- $C_I \cdot H_K \cdot W_K$ weights and spikes are read from memory, where C_I is the number of channels in the input feature map, and H_K and W_K represent the shape of the convolution kernel. For the sake of clarity, this quantity is denoted with N_{rd}.

  $$N_{rd} \triangleq C_I \cdot H_K \cdot W_K$$

  Assuming that 8 spikes are encoded to an 8 b memory word in the spike scratchpad (i.e., the memory structure used to host the input and output data near the PE), the energy associated with a spike memory operation (either read or write) can be approximated to $E_{rd}/8$, where E_{rd} (E_{wr}) is the energy consumption of a memory read (write) considering the 8 kB cache field in Table 3.1. The energy consumption associated with the memory accesses of weights and spikes is denoted with $E_{rd_{tot}}$:

  $$E_{rd_{tot}} = N_{rd} \cdot (E_{rd} + E_{rd}/8)$$

- $C_I \cdot H_I \cdot W_I$ additions are performed on the synaptic current. The energy associated to checking if the spike ($ifmap[c_i][h_i][w_i]$) is equal to 1 is negligible since its cost is included

Algorithm 1 SNN convolution of a single window and timestamp.

Require: S, β, ϑ ▷ Stride, leakage, threshold.
Require: *weights* ▷ Convolution kernel weights.
Require: *states* ▷ Neurons states.
Require: *ifmap*, *ofmap* ▷ Input and output feature maps.
Require: $(c_o,\ h_o,\ w_o)$ ▷ Output value coordinates.

```
I ← 0                                                    ▷ Input synaptic current.
for c_i ← 0, C_I − 1 do                                  ▷ Input channels.
    for h_k ← 0, H_K − 1 do                              ▷ Kernel height.
        for w_k ← 0, W_K − 1 do                          ▷ Kernel width.
            h_i ← h_o * S + h_k
            w_i ← w_o * S + w_k
            if ifmap[c_i][h_i][w_i] ≠ 0 then
                I ← I + weights[c_o][c_i][h_k][w_k]
            end if
        end for
    end for
end for
m ← states[c_o][h_o][w_o] * β + I                        ▷ State update.
if m ≥ ϑ then
    m ← m − ϑ
    ofmap[c_o][h_o][w_o] = 1
else
    ofmap[c_o][h_o][w_o] = 0
end if
states[c_o][h_o][w_o] ← m
```

in the memory access performed to retrieve the spikes. Hence, this energy is denoted with E_{acc}.

$$E_{acc} = N_{rd} \cdot E_{add}$$

- Next, one state has to be retrieved from memory, multiplied by the leakage, β, added to the synaptic current, thresholded (compared against the threshold, ϑ, and, if higher, reduced by ϑ via subtraction) and written back. In the worst case, this energy denoted with E_{state}, is given by:

$$E_{state} = E_{rd} + E_{mult} + E_{add} + E_{comp} + E_{sub} + E_{wr}$$

- The output spike must then be written to the scratchpad. This energy is denoted with E_{ofmap}.

$$E_{ofmap} = E_{wr}/8$$

Hence, the total energy involved in a SNN convolution, defined as E_{SNN}, is:

$$E_{SNN} = E_{rd_{tot}} + E_{acc} + E_{state} + E_{ofmap}$$

The following values are employed for numerical estimation: the shape of the convolution, i.e., of the input feature map window to be evaluated in order to compute a single output

feature map value, is $(C_I, H_I, W_I) = (512, 3, 3)$, which means the number of memory reads is $N_{rd} = 4608$. For memory operations, additions/subtractions, and multiplications, the data from Table 3.1 are used. It has to be remarked that, in this analysis, we do not consider any memory optimization (e.g., buffering on registers) since both ANNs and SNNs access the memories with the same window patterns; thus, these would bring similar advantages to both of them, without impacting the comparison. The energy consumed by a threshold comparison, E_{comp}, is assumed to be the same as that of an 8 b addition. This is a reasonable assumption, since comparison commonly employs a subtraction. This leads to:

$$E_{SNN} = 13.1\,\text{nJ}$$

The energy consumption obtained above assumes a dense input feature map, i.e., all input neurons are firing. To take into account sparse firing activity, a coefficient γ_{SNN} can be introduced that reflects the proportion of neurons in the feature map that are firing at a given time, and the hardware overhead due to the sparse data structures employed. Hence:

$$E_{SNN} = 13.1\,\text{nJ} \cdot \gamma_{SNN}$$
$$0 < \gamma_{SNN} \leq 1$$

Consider now the convolution operation performed in a ANN, which is reported in Algorithm 2.

Algorithm 2 ANN convolution of a single window.

```
1:  a ← 0                                                        ▷ Activation.
2:  for c_i ← 0, C_I − 1 do
3:      for h_k ← 0, H_K − 1 do
4:          for w_k ← 0, W_K − 1 do
5:              h_i ← h_o * S + h_k
6:              w_i ← w_o * S + w_k
7:              a ← a +
8:                weights[c_o][c_i][h_k][w_k] * ifmap[c_i][h_i][w_i]
9:          end for
10:     end for
11: end for
12: z = φ(a)                                                     ▷ Non-linear activation.
13: ofmap[c_o][h_o][w_o] = ψ(z)                                   ▷ Quantisation.
```

Following the same approach adopted for the SNN convolution:

- N_{rd} weights and inputs are read from memory:

$$E_{rd_{tot}} = 2N_{rd} \cdot E_{rd}$$

- The same number of additions and multiplications are performed, and this energy is denoted with E_{MAC}.

$$E_{MAC} = N_{rd} \cdot (E_{add} + E_{mult})$$

- The obtained value is then processed by the nonlinear activation function $\varphi(z)$. This energy is denoted with E_{act}.
- The result is quantized in order to be processed by the next layer in the network. The quantization step is modeled through a function $\psi(z)$, with an associated energy cost E_{quant}, which is estimated as an 8 b addition (for instance, the rectified linear unit (ReLU) activation is a 0-thresholding).
- Finally, the result is written to the scratchpad memory:

$$E_{ofmap} = E_{wr}$$

The total energy consumption of a ANN convolution is:

$$E_{ANN} = 24.1\,\text{nJ} \cdot \gamma_{ANN}$$

A similar sparsity coefficient can be introduced, denoted with γ_{ANN}. The approximations made for E_{quant} and E_{act} are acceptable since their contribution to the total energy is negligible, given that they are performed only on the final result of the convolution.

Hence, despite the additional memory accesses and state operations involved in SNNs, ANN convolutions consume $1.84\times$ more energy. Of course, this result depends heavily on the convolution filter depth and size, but it gives a reasonable approximation of the different costs between ANN and SNN processing, as highlighted in Section 3.2.2.

However, the energy estimation obtained for the SNN corresponds to a single time step! When dealing with spatial data, such as images, a SNN needs multiple time steps (T, for instance), since the input image pixels are encoded to multiple spikes in time [32]. Hence, the actual energy consumption associated with a convolution operation in a SNN must be multiplied by the number of time steps needed to perform an inference:

$$E_{SNN} = 13.1\,\text{nJ} \cdot T \cdot \gamma_{SNN}$$

Since $T > 1$ for any SNN, otherwise there would be no temporal evolution in the network and it would be equal to heavily quantized ANNs (*i.e.*, binary neural networks), SNNs are less efficient on spatial vision data than ANNs, if the same degree of sparsity-awareness ($\gamma_{SNN} = \gamma_{ANN}$) is taken into account. In fact, if $T = 1$, the state of an SNN would be updated at timestamp $t = 0$, but not evaluated subsequently; hence, there is no point in storing it if it is not measured after, and the SNN becomes equivalent to a binary neural network in which activations are represented on a single bit, while weights are multiple bits quantities.

This analysis, *i.e.*, the by-construction disadvantage of SNNs with respect to ANNs on purely spatial data, is validated by the data shown in Table 3.2, which includes SOTA accelerators at the time of writing. The given results account for how the hardware handles sparse feature maps, as stated by the author of the papers considered [59, 76, 104, 84, 63, 62].

3.2.2 SOTA accelerators

The literature provides many overviews of hardware accelerators for SNNs, ranging from digital hardware to in-memory computing (IMC) and mixed-signal architectures [4, 6, 1]. An up-to-date list of SNN hardware accelerators and processors can be found in [79]. In these reviews [4],

Table 3.2: DL accelerators evaluated on ImageNet [26]. The best efficiency is considered for each design, and the associated task accuracy is evaluated under the same conditions. Mixed indicates an accelerator running both SNN and ANN processing elements.

	ANN Transf.	**ANN CNN**		**SNN**	**Mixed**
Work	Keller'23 [59]	Park'22 [84]	Mo'21 [76]	SNPU'23 [63]	C-DNN'23 [62]
Process [nm]	5	7	28	28	28
Area [mm^2]	0.2	4.7	1.9	6.3	20.3
Supply voltage [V]	1.1	1.0	0.9	1.1	1.1
Clock frequency [MHz]	1760	1196	470	200	200
Data format	INT4-VSQ [59]	INT8	INT8	INT8, INT4	INT1-16, INT4/8
Network model	DeiT-Base	MobileNetTPU	ResNet50	ResNet18	ResNet50
Parameters [M]	768	3	25	12	25
Operations/inference [GOP]	35.2	17.4	13.3	61.4	7.4
Task accuracy [%]	80.5	71.7	76.9	66.8	77.1
Throughput [FPS]	56	3433	120	245	123
Power [mW]	56	5114	132	478	34
Energy/inference [mJ]	1.0	1.5	1.1	2.0	0.3

different coding schemes for SNNs, such as latency coding and phase coding [32], are explored; however, these still lack in classification accuracy performance with respect to rate coding, even if they represent an interesting approach that could provide an advantage to SNNs.

Most SNN hardware reviews are missing an important feature: an objective comparison with SOTA ANN hardware accelerators. Most SNN accelerators are benchmarked on spatial datasets; such datasets are inappropriate for proper system characterization, since (i) they are often considered trivial or "solved" datasets (e.g., CIFAR-10, MNIST) [4], and (ii) ANNs are more efficient on spatial data.

Table 3.2 reports SOTA ANN vision accelerators and the best performing SNN accelerator [63] (SNPU'23). All accelerators run the same task: object classification on ImageNet [26]. SNPU'23 [63] is chosen as the SNN reference since it is the only accelerator targeting complex vision workloads such as ImageNet. In addition to SNPU'23, C-DNN'23 [62] is analyzed. This chip employs both ANN and SNN PEs to maximize inference efficiency by inferring part of the neural network layers on the SNN hardware and part of these on the ANN hardware. In fact, mixed neural models employing both artificial and spiking backbones are arising as high performance implementations which allow to improve SNNs accuracy [95, 10] on vision tasks, beyond tackling more complex workloads such as object detection. Moreover, C-DNN'23 also provides on-chip training capabilities, which most ANN accelerators lack.

Table 3.2 considers the most efficient SNN digital hardware accelerator, SNPU'23 [63], and a mixed-topology design, C-DNN'23 [62]. These are compared to various ANN accelerators for object recognition, which target both transformer-based models [60] and convolutional neural networks [52]. Different observations can be made:

- The model employed by SNPU'23 [63], ResNet18 [52] SNN, is the lowest performing model in terms of classification accuracy: 66.8 %, against 80.5 % (Keller'23 [59]), 71.68 % (Park'22 [84]), 76.92 % (Mo'21 [76]) and 77.1 % (C-DNN'23 [104]).
- The energy per inference of SNPU'23 [63] is the highest among all the designs (1.95 mJ/inf), even considering a ANN accelerator implemented on the same 28 nm complementary metal oxide semiconductor (CMOS) node, i.e., Mo'21 [76] (0.46 mJ/inf).

Figure 3.1 and Fig. 3.2 shows the energy consumption per inference and the performance

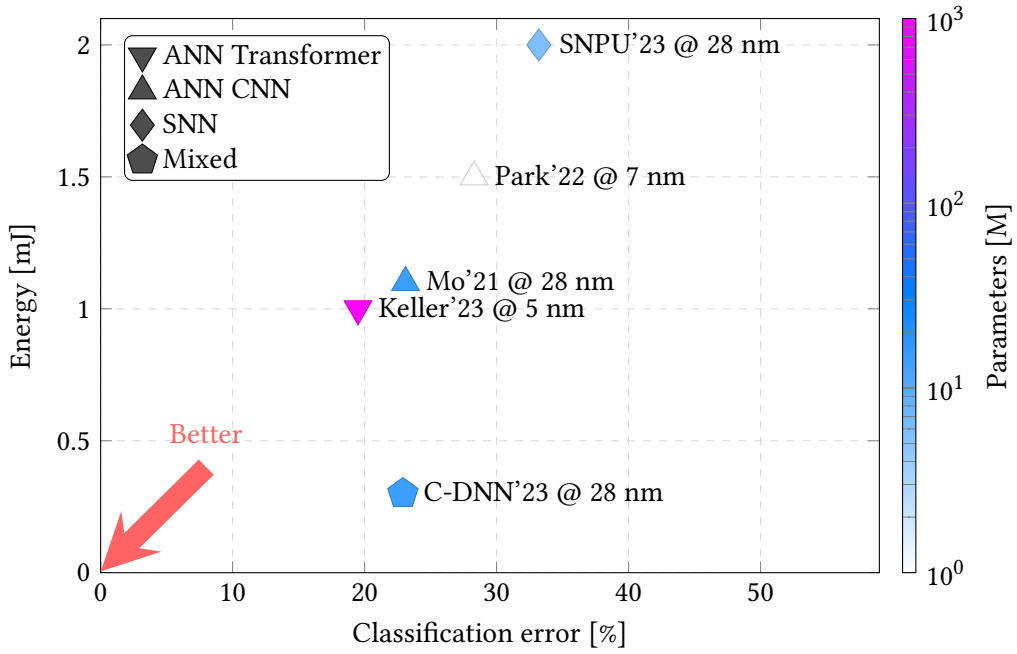


Figure 3.1: Energy consumption for single inference of DL accelerators with respect to classification error on ImageNet.

vs. the top-1% classification error on ImageNet, for each accelerator reported in Table 3.2.

The performance is expressed as the initiation interval, i.e., the ratio between the clock frequency and the throughput. It is the number of clock cycles between two inferences at the output at the steady state and it corresponds to the latency of non-pipelined accelerators. It is worth noting that this performance metric is totally architecture-dependent and allows for more immediate comparison between accelerators implemented with different technological process. While SNPU'23 is considered the SOTA SNN accelerator at the time of writing, it is nonetheless Pareto-dominated by all ANN accelerators, both in the energy vs. error (Fig. 3.1) and in the performance vs. error (Fig. 3.2) spaces. This is because SNPU'23 requires 16 timesteps to process a single input image; hence, even if the SNPU'23 model contains less than half of the parameters of the Mo'21 one, it requires 4.6 × operations per inference.

For what concerns C-DNN'23 [62], the mixed architecture leads to a very low energy consumption per inference (281 µJ), the best among all the chips despite being implemented on a CMOS node older than Keller'23 [59] and Park'22 [84] (28 nm vs. 5 nm and 7 nm, respectively). However, the model being run on C-DNN'23, the ResNet50, is much smaller than the vision transformer of Keller'23, which achieves a higher accuracy (80.5% vs. 77.1%).

The design presented in SNPU'23 is among the best-performant in the SNN domain. Accelerating ResNet family models on-chip is an impressive feat, given that many SNN accelerators are limited to smaller-scale architectures; however, our analysis of SNPU'23 against ANN accelerators highlights that spatial data is not the optimal way to demonstrate processing efficiency.

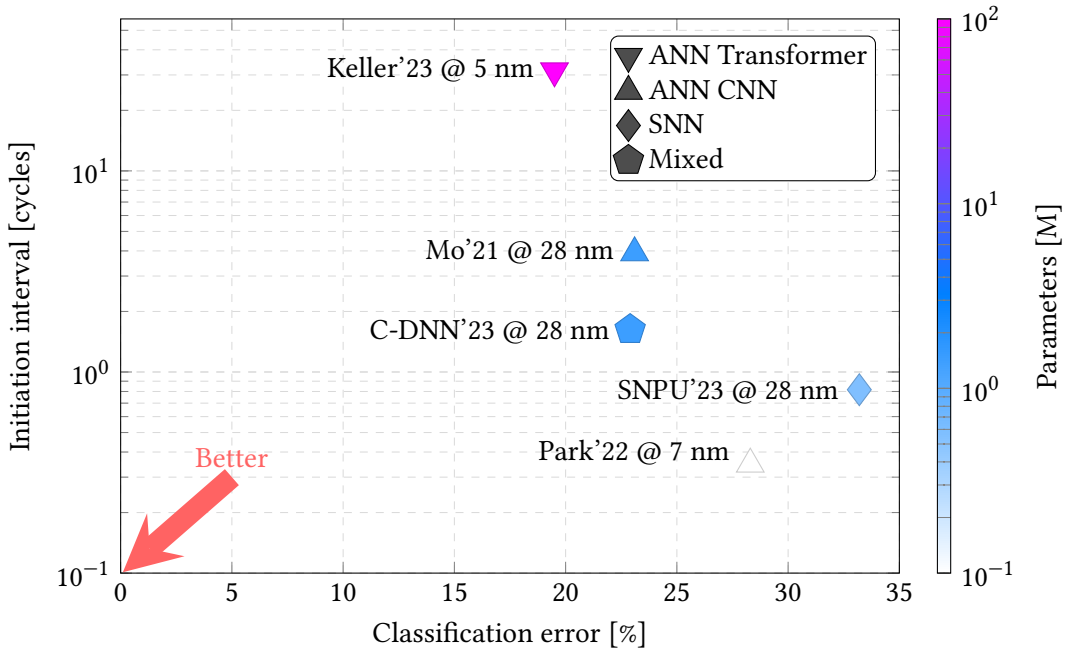


Figure 3.2: Energy v.s. classification error of recent digital SNN and ANN accelerators measured on the voice activity detection (VAD) and keyword spotting (KWS) tasks.

Beyond spatial vision tasks, a possibility is to focus on tasks that take advantage of the event-based nature of SNNs, such as dynamic vision sensor (DVS) data [38]. These sensors capture scenes in the form of 'events' that can be treated naturally as spikes, without any artificial encoding. Some alternative SNN accelerators target dynamic workloads, though only those that handle large-scale (at least on the scale of ResNets) models with spatial data are considered in the above analysis [28, 35]. These accelerators represent the minority [4], since most benchmarks are limited to the MNIST and CIFAR-10 spatial datasets.

Beyond classification, more complex event-based vision tasks are less explored by the neuromorphic community, though tend to dominate the modern computer vision ANN field [88, 20, 44, 2]. Conversely, on-chip learning solutions in the SNN domain [35, 34] are more advanced than for the ANN community. This advantage should be exploited to reduce training costs and allow for adaptive intelligence at the edge. In order to reduce the memory access burden for computation of neuron states, low-rate training techniques should be explored for efficient hardware inference [32, 34].

It has to be remarked that on-chip training solutions have been and are investigated in the ANN domain [69]; however, these approaches target general purpose platforms, such as microcontrollers, and still target simple neural networks and tasks. Of course, this is true also for SNNs: in fact, the classification accuracy of on-chip trained networks is worse than the one of off-chip models, which perform worse than the ANN counterpart. This is why in practically any application, inference-only, ANN-based models are deployed on highly efficient accelerators.

3.3 Spatio-temporal tasks

SNNs are inherently time-aware neural networks due to their statefulness (see Chapter 1). As such, they are a natural fit for sequential data processing. In video processing tasks, such as video segmentation, both non-spiking and spiking neural networks often employ convolution structures to extract features [107, 15, 16]. Given that the computational costs of spiking and non-spiking convolutional operators are addressed in Section 3.2, this section primarily concentrates on audio processing, another prominent subset of temporal tasks.

In previous work [11, 86], a fully connected feed-forward RNN targeting keyword prediction is used to compare ANNs, rate-based SNNs, and latency-based SNNs [32]. The rate-based SNN is only 9 % more efficient than the ANN due to the high input firing rate (2.5 kHz) necessary to match the ANN performance. The limited efficiency advantage makes the effort of migrating to a new type of neural network hard to justify. Conversely, the latency-based SNN is 84 % more efficient than the ANN, primarily thanks to the significantly lower firing rate that leads to a reduction in the number of the timesteps evaluated by the SNN. However, this methodology sets the target time window to 75 ms to incorporate temporal information, which is not suitable for real-time processing.

To evaluate the performance and efficiency of ANN and SNN accelerators, in the following audio processing benchmarks are considered, such as keyword spotting (KWS), voice activity detection (VAD) and automatic speech recognition (ASIC) [21, 105, 81]. In particular, we present an energy analysis similar to the one in Section 3.2. Finally, we compare SOTA digital hardware accelerators from the spiking and artificial domains.

3.3.1 RNN versus SNN

RNNs are designed for discerning patterns in sequential data, uniquely characterized by their capacity to retain memory of previous inputs within their hidden state. Variants that aim to enhance the "memory" of such neurons have also emerged, such as LSTMs and GRUs [49], and have been adopted in audio processing tasks [42, 67]. The formulation of a vanilla RNN layer is:

$$\begin{aligned} h_t &= \sigma_h(U_h \cdot x_t + V_h \cdot h_{t\text{-}i} + b_h) \\ o_t &= \sigma_o(W_o \cdot h_t + b_o) \end{aligned} \tag{3.1}$$

where x is the input, h the hidden layer, and o the output. U_h, W_o, and V_h are the weight matrices, b is the bias vector and σ is the activation function. Differently from the SNN model defined by (2.1), the next state is computed considering a bias b_h and by multiplying the previous state by a matrix V_h; in SNNs, V_h reduces to a scalar β, employed for all the neurons [32]. The equivalent synaptic current is represented by $U_h \cdot x_t$.

We compare the computational cost of these operations in the context of a speech recognition task. We consider the cost of a vanilla RNN and a SNN, which share the same mechanism of implicit recurrence through the neuron state [31, 49]. As in Section 3.2, activations, weights, and states are quantized to 8 bits, while spikes are single-bit quantities. As a use case, we consider a layer with N inputs and calculate the energy consumption of a neuron in the layer, as in Section 3.2.

With the SNN model:

(i) N weights and input spikes are loaded from memory:

$$E_{rd_{tot}} = N \cdot (E_{rd} + E_{rd}/8)$$

(ii) These values are then accumulated depending on the spike value to obtain the activation to be fed to the state:

$$E_{acc} = N \cdot E_{add}$$

(iii) The state is loaded from memory and decayed (i.e., multiplied) by a factor β; then, it is accumulated with the activation computed in the previous step, compared against the threshold ϑ, reset if needed and stored to memory:

$$E_{state} = E_{rd} + E_{mult} + E_{add} + E_{comp} + E_{sub} + E_{wr}$$

(iv) The output spike, if generated, is then stored to the scratchpad:

$$E_{ofmap} = E_{wr}/8$$

Assuming $N = 1024$ inputs and considering the data reported in Table 3.1, the total energy, E_{SNN}, is given by:

$$E_{SNN} = 2.92\,\text{nJ} \cdot \gamma_{SNN}$$

Consider now the vanilla artificial RNN layer:

(i) N weights and N inputs are read from memory, together with the hidden state and its recurrent weight:

$$E_{rd_{tot}} = (2N + 2) \cdot E_{rd}$$

(ii) the inputs and the state are multiplied by the weights and accumulated:

$$E_{MAC} = (N + 1) \cdot (E_{add} + E_{mult})$$

(iii) the state is written back to memory. As in Section 3.2, the quantization and activation energies are neglected:

$$E_{state} = E_{wr}$$

(iv) the obtained value is processed by the nonlinear activation function, quantized and written back to memory:

$$E_{ofmap} = E_{wr}$$

Hence, the total energy consumption is:

$$E_{ANN} = 5.37\,\text{nJ} \cdot \gamma_{ANN}$$

The energy analysis shows that in both vanilla RNN and spiking layers, the memory accesses for weights and states consume the majority of the energy, as in the convolution case. Notice that E_{SNN} is 1.84 × smaller than E_{ANN}: this is due to the fact that while in the ANN the inputs are on 8 b, in the SNN these are single-bit quantities. Since the memory access energy dominates the other figures, this results in a major overhead of ANN models with respect to SNN ones.

Moreover, the ANN model involves a multiplication when processing inputs, which is more energy-hungry than the addition (Table 3.1).

Differently from the convolution case, the number of timesteps needed to process the inputs is larger than 1 for both ANNs and SNN, since the data is now evolving through time.

It has to be taken into account that very simple RNN and SNN models are considered. In Section 3.3.2, digital hardware accelerator targeting more sophisticated neural network architectures are investigated.

Table 3.3: SNN and ANN accelerators evaluated on temporal tasks including VAD and KWS.

	VAD		**KWS**			
	SNN	**ANN**	**SNN**	**ANN**		
Work	Yang'19 [110]	Oh'19 [78]	Frenkel'22 [35]	Kim'22 [61]	Giraldo'22 [46]	Shan'23 [94]
Process [nm]	180	180	28	65	65	28
Area [mm^2]	1.6	-	0.45	2.03	2.56	3.6
Supply voltage [V]	0.55	0.6	0.5	0.75	0.6	0.4
Clock frequency [MHz]	0.5	0.7	13	0.25	0.25	0.2
Feature extractor	Analog	Analog	No FEx	Analog	Digital	Digital
Data format	INT1	INT4	INT8	INT8	INT8	INT1
Dataset	Aurora4 w/ DEMAND	LibriSpeech w/ NOISEX-92	Spiking Heidelberg Digits		GSCD	
Task accuracy [%]	85	90	90.7 (1-word)	86 (10-word)	90.9 (10-word)	97.8 (2-word)
Network model	FCN	FCN	Spiking RNN	GRU	LSTM	DSCNN
Parameters [k]	4.6	1.6	132	24	21.5	4.7
Power [µW]	1	0.142	79	23	10.6	0.8
Energy/inference [nJ]	10	2.3	42	285.2	169.6	23.6
Latency [ms]	-	512.0	5.7	12.4	16.0	29.5

3.3.2 SOTA accelerators

Regarding audio processing tasks, most accelerators are tested on or designed specifically for VAD [110, 78] and KWS [35, 61, 46, 94]. This is due to the fact that VAD and KWS represent less challenging problems to tackle; hence, simple neural network architectures are employed, which allow achieving higher efficiency on hardware inference.

In Table 3.3 we chose the two most efficient SNN chips [35, 110], respectively in VAD and KWS, which are compared against SOTA ANN counterparts. For the VAD task, Oh'19 [78] achieves the best efficiency measured in energy per inference, despite being implemented on an old 180 nm CMOS technology node. It results to be more efficient and to perform better, in terms of classification accuracy, than Yang'19 [110], which runs an SNN model and uses the same CMOS node.

As for the KWS task, different CMOS technologies are employed in the accelerators. These values are not normalized with Dennard scaling since all the chips have a power consumption in the order of 10 µW; in these conditions, most of the power consumption is static, while Dennard scaling gives an approximation of how dynamic power scales across technology nodes.

Also for KWS, the highest energy efficiency is achieved by an ANN chip, Shan'23 [94], which also has a lower average power consumption and significantly higher accuracy than all the other chips targeting the same task. The SNN accelerator, Frenkel'22 [35], achieves the lowest latency and is the only chip that supports online learning and on-chip training: in fact, the whole training process is performed on-chip, that is validated also on vision and autonomous

agent tasks. It should be noted that the number of parameters of the spiking RNN in [35] is 28× larger than that of the ANN chip in [94], while achieving lower accuracy. This might suggest that SNNs are still lacking in terms of classification accuracy when compared to their ANN counterparts; however, Shan'23 [94] is an inference only chip, hence the model is fine-tuned for the task, while Frenkel'22 supports arbitrary network topologies and on-chip training and can be repurposed. This is a significant advantage for chips deployed on edge devices, which model might need to be re-adapted to the environment in which the system is deployed.

Figure 3.3 shows the distribution of these accelerators in terms of energy efficiency and

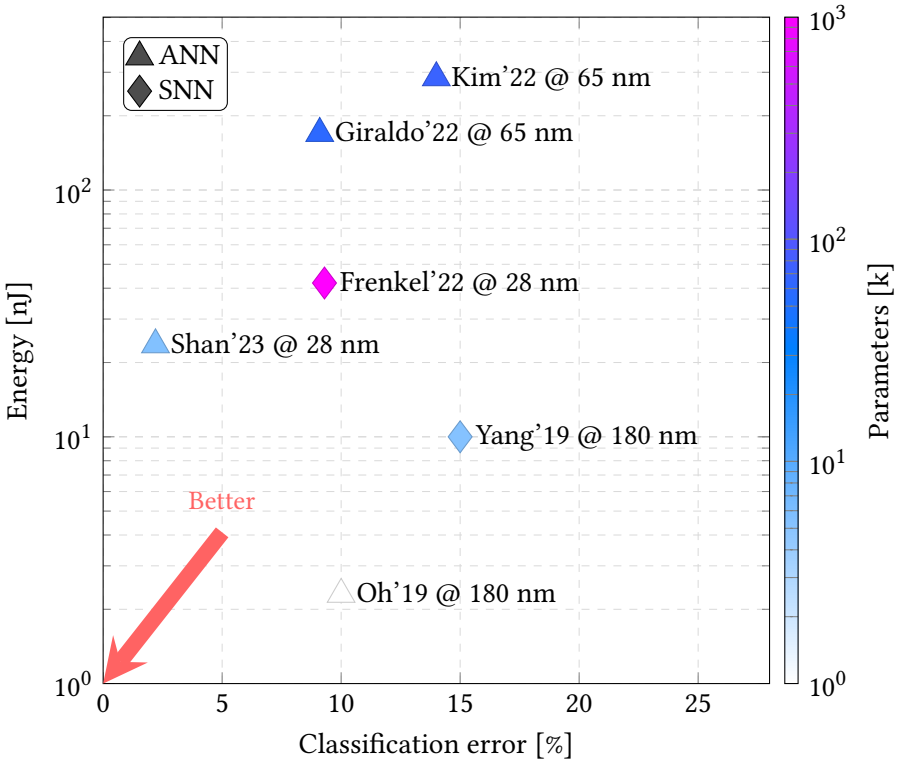


Figure 3.3: Energy v.s. classification error of recent digital SNN and ANN accelerators measured on the VAD and KWS tasks.

accuracy along with FPGA accelerators [40, 41], that achieve the highest accuracy in KWS. One can notice that Shan'23 [94] Pareto-dominates all the other chips, thanks to a highly efficient sparsity-aware chip architecture and a performant neural network model. Frenkel'22 [35] presents a very competitive efficiency, taking into account that it is the only chip with in-hardware training capabilities: in fact, the network benchmarked on KWS is trained directly on it; nonetheless, the resulting classification accuracy on the task results to be competitive even when considering most of the ANN chips analyzed.

Chapter 4

An HLS library for SNNs inference

In this chapter, a C++ library targeting HLS compilers for SNNs inference on FPGAs is presented. In particular, CNNs are targeted to perform event-based vision processing. As use case, the optical flow estimation using event cameras and SNNs for autonomous drones applications, carried out in collaboration with Delft University of Technology.

At the current moment, there are no hardware platforms specialized to run SNN inference in an efficient way. The Intel Loihi chip [23], for instance, is still far from commercial availability, and the prototypes provided to researchers are not able to fit modern DNNs [52]. While all modern DNN accelerators [13, 64, 75, 59, 100, 84] choose a *forward* architecture, *i.e.*, the network is processed layer by layer, treating the activations, weights and neurons as matrices, NoC chips like Loihi treat the neurons as single entities, being updated independently everytime an input to them arrives. This seems a good design choice, in theory, if one supposes that the input is *very sparse*; moreover, it allows extreme flexibility in the neurons connectivity. However, the actual routing on the chip results to be really difficult, and accessing single neurons parameters from memory (even local ones) might worsen the energy consumption of the system [71].

Startups such as Synsense [99] and Innatera [55] are arising to provide efficient digital and mixed-signal platforms to deploy SNNs. However, at the moment, these are not available to researchers and the prototypes cannot fit DNNs such as large as ResNet [52]; the reason why deep neural networks are needed is that these are capable of learning more complex representations, and one can expect the same to hold true also for SNNs, even though the number of layers in these is limited by current training methodologies and algorithm, that emulate very closely the ones employed for ANNs. Moreover, on tasks such as optical flow estimation using event cameras, deep SNNs are required to get performance comparable with ANNs. For this reason, the research community and the market need well established hardware platforms that are more efficient than GPUs when it comes to quantized models but, at the same time, are reliable and easily purchasable.

FPGAs might be a good candidate to explore efficient inference of SNNs, as the implementation of custom hardware on these might provide very efficient implementation of DNNs without resorting to designing an ASIC. By *efficiency* we mean the number of operations (or, in the case on DNNs, inferences) per unit of energy consumed by the accelerator. Moreover, very

few hardware platforms [28] target event based vision, which could be one of the applications most compatible with SNNs, given the event-based, temporal characteristics of the information provided by an event camera.

The hardware architecture and design methodology proposed in this chapter try to fill these gaps. An accelerator infrastructure based on HLS for running SNNs on digital hardware (in particular, FPGAs) is proposed and implemented. The objective of the infrastructure is to allow designers to easily implement SNNs and deploy these to an FPGA board to test it on the field. Moreover, differently from what is done in the SNN literature at the current moment, in this hardware architecture SOTA techniques for conventional DL accelerators are explored and analyzed, and then applied to the accelerator infrastructure to maximize performance and minimize memory usage when running the neural network in hardware.

The network architecture family targeted is the CNNs one, for event-based vision applications. In this thesis, it is shown that SNNs are not competitive with conventional (artificial) DNNs on static data, such as images; for these reasons, the accelerator targets and is tested on an event-based vision task: optical flow estimation out of event-based cameras data.

The field of robotics has benefited from the recent developments in DL, with deep ANNs achieving SOTA performance in tasks such as stereo vision [17, 50], optical flow estimation [54, 98, 101], segmentation [112, 70], object detection [47, 89, 108], and monocular depth estimation [43, 48, 111]. However, this high performance typically relies on substantial neural network sizes that require quite heavy and power-hungry processing hardware. This limits the number of tasks that can be performed by large robots, such as self-driving cars, and even prevents deployment on smaller robots with highly stringent resource constraints, like small flying drones.

SNNs and event cameras may prove to be powerful instruments to improve efficiency on drones with limited resources. Hagenaars et al. [83] implemented the first processing pipeline based on an SNN and an event camera to estimate optical flow in events and use it to pilot a drone in partial autonomy. The pipeline was deployed to the Intel Loihi chip [23], but the promises of efficiency fell short: the chip required around 1 W of power to operate, which required a large battery and, hence, drone to carry it.

In collaboration with Delft University of Technology, we have tried to improve these results, working at both the neural network architecture and hardware levels. In particular, a custom FPGA accelerator has been developed, using the infrastructure proposed in this chapter, and the larger design space (*i.e.*, more freedom in choosing the neuron model, parameters, leakage mechanisms etc.) resulting from having an application specific accelerator allowed us to simplify the network model without worsening performance, measured in AEE. The neural network still underperforms with respect the artificial counterpart in terms of accuracy, but the energy efficiency is improved by a factor $10\times$ with respect a neuromorphic processor baseline, which shows the advantage of bringing SOTA techniques from the artificial DL world to SNN design.

The hardware architecture presented in this chapter is based on the one developed by Filippo Minnella and Teodoro Urso [75], Ph.D. students at Politecnico di Torino and colleagues in the same laboratory.

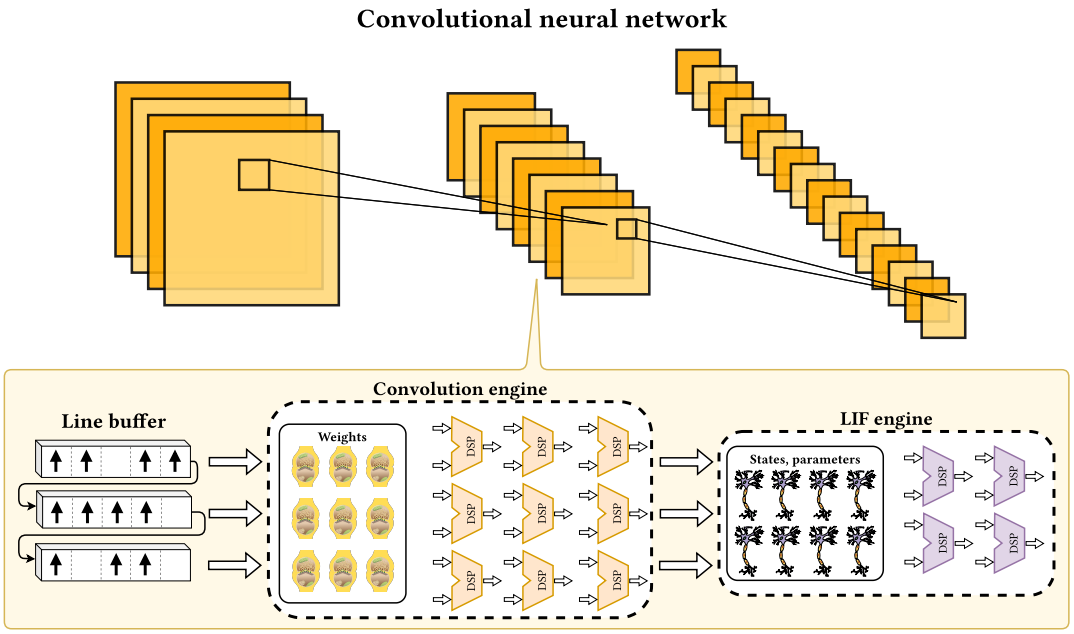


Figure 4.1: A convolutional layer being mapped to hardware in the accelerator infrastructure. To maximize computational and resource usage efficiency, the computations are mapped as much as possible to the digital signal processor (DSP) macros available on AMD FPGAs. Line buffering [75] is used to minimize memory footprint of spikes, and each layer has weights and states memory macros associated.

4.1 Hardware architecture

The C++ library developed is synthesized to RTL via HLS, using the Vitis HLS [27] compiler commercialized by AMD. The neural network architecture targeted is a CNN, using a hardware implementation based on the work of Minnella et al. [75], and it is shown in Fig. 4.1. The convolutional layer is mapped to: (i) a buffer to store the previous layer activations; (ii) a convolution engine, which consists of a memory to host the weights kernel, mapped to the FPGA block static random access memories (BRAMs) and a set of digital signal processors (DSPs) to perform the computations; (iii) a LIF engine, consisting of a memory to store the neurons states and parameters (leakages, thresholds) and a set of DSPs to perform state update and spikes generation.

Each layer is mapped to an instance of such engine, and all the layers run concurrently following a *fused-layer* scheme. This allows to minimize the intermediate feature maps memory footprint, as stated in Section 2.3.2.

4.1.1 Convolution layers dataflow

To minimize the memory occupation of the spikes, a line buffering [75] approach is adopted: to perform a convolution, only `KSZ - 1` lines per layer need to be kept in memory; the last line, instead, is provided to the convolution engine one (or more) entry/entries at time. Moreover, since the layers are executed with a dataflow scheduling, as soon as the values of the convolution window in input to a layer are ready, the convolution engine starts working. This implies

that all the layers are executed in parallel, performing computations as soon as their input data is available. This scheduling scheme is reported in Fig. 4.2.

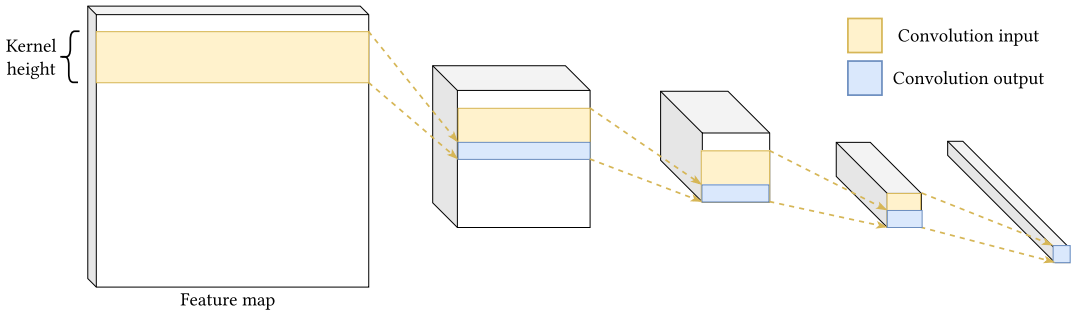


Figure 4.2: Dataflow execution of multiple convolutional layers. Layer execution is interleaved, since the output of a convolution is used by the subsequent layer as soon as a convolution window can be read.

The data structure employed for buffering the intermediate feature maps is a line buffer [75], which streams to the processing engine a convolution window of size `KSZ * KSZ` at each cycle. The principal characteristic of the line buffer is then, when a convolution filter of shape `(KSZ, KSZ)` is applied to a feature map, one needs only `KSZ` rows of the map to perform a slide of convolutions (*i.e.*, sliding horizontally the kernel across the map). The data flow adopted is a *depth-first* one, which means that the channel dimension is processed before the height and width ones. This implies that all the input channels `CI` of a window are read from the line buffer before proceeding to the next window. Considering a single output channel, the following pseudocode can be derived.

```
// Defining the line buffer object.
LineBuffer<...> lb(...);
// The first two loops slide across the feature map vertically and horizontally.
for (auto ho = 0; ho < HO; ho++){
  for (auto wo = 0; wo < WO; wo++) {
    // Reading depth-first the convolution window.
    for (auto ci = 0; ci < CI; ci++) {
      doutStream << lb.read_window(ho, wo, ci);
    }
  }
}
```

Since the neural network is hosted completely on-chip, the dataflow employed is *output stationary* [100] for the temporal scheduling of the operations: in this way, the scratchpad size used to accumulate the weights is minimized, and the line buffer is not stalled while the convolutional layer is processing its output.

In fact, consider an *input stationary* [100] dataflow, in which the input is kept inside the processing unit while the output channels and weights are re-read. In the following, the C++ code for the computation of a single convolution output is shown.

```
spad_t spad[CO];
for (auto ci = 0; ci < CI; ci++) {
```

```
  for (auto k = 0; k < KSZ * KSZ; k++) {
    for (auto co = 0; co < CO++; co++) {
      // Notice that since din is a boolean variable, the product
      // is equivalent to a bitwise AND.
      spad[co] += din[ci][k] * weights[co][ci][k];
    }
  }
}
// Sending the results to the next processing layer.
for (auto co = 0; co < CO; co++) {
  doutStream << spad[co];
}
```

One can notice that `CO` entries in the scratchpad are needed to save temporary results. If we consider a *weight stationary* dataflow, the same considerations apply. Instead, if one uses an *output stationary* dataflow:

```
// din[CI][KSZ * KSZ] is a 2D tensor representing the convolution window being processed.
// weights[CO][CI][KSZ * KSZ] is a 3D tensor representing all the convolution filters.
spad_t spad;
for (auto co = 0; co < CO; co++) {
  for (auto k = 0; k < KSZ * KSZ; k++) {
    for (auto ci = 0; ci < CI++; ci++) {
      spad += din[ci][k] * weights[co][ci][k];
    }
  }
  // Sending the results to the next processing layer.
  doutStream << spad;
}
```

Only a single entry for the scratchpad is needed. However, in this way, the line buffer has to send the convolution window to the processing engine `CO` times, as it is shown in the code snippet above; nevertheless, since spikes are represented on single bits, this does not represent a limitation for the memory bandwidth.

Allowing for reduced bitwidth in the weights leads to lots of room for hardware optimizations: since in SNNs there is no multiplication between the activations (spikes) and the weights, the product between a spike and a weight is equivalent to a bitwise AND operation, which does not lead to an increase in the result bitwidth; only when accumulating multiple products along the channel and feature map dimensions, one needs to increase the bitwidth in order to prevent overflow. In FPGAs, additions with limited bitwidth are more efficiently implemented in the logic fabric, *i.e.*, with look-up tables (LUTs), since the DSP48E2 [27] DSP macros in AMD Xilinx FPGAs perform additions on 48 b. However, these support also SIMD operation for addition, *i.e.*, multiple additions can be mapped to the same DSP in parallel.

To achieve this goal, a C++ function that maps to a SIMD DSP48E2 macro in AMD FPGAs has been developed, using Vitis HLS [27]. The DSP48E2 macro supports 3 modes of operation for the addition: (i) a single addition on 48 b, (ii) 2 additions on 24 b and (iii) 4 additions on 12 b. Vitis HLS does not support automatically the mapping of SIMD instruction to DSPs, which has been achieved using the blackbox functionality: a Verilog file describing an RTL block is

provided in conjunction to a C function that describes the same functionality at the behavioral level. This blackbox function has been inserted in a C++ `template` to parametrize it with respect to the bitwidth of the operands. When the accumulation would lead to overflow on 12 b, the 2-SIMD version of the DSP is instantiated instead, which works with a parallelism of 24 b.

Having access to smaller bitwidth for the accumulators would not represent an advantage, since in more advanced neural networks the number of channels to accumulate is really large; hence, only for few layers or in small networks, like the one presented in Section 4.2, even when using 4 b weights, it is possible to use the DSP as 12 b accumulator.

In Fig. 4.3, the whole processing pipeline is shown. At each clock cycle, the line buffer

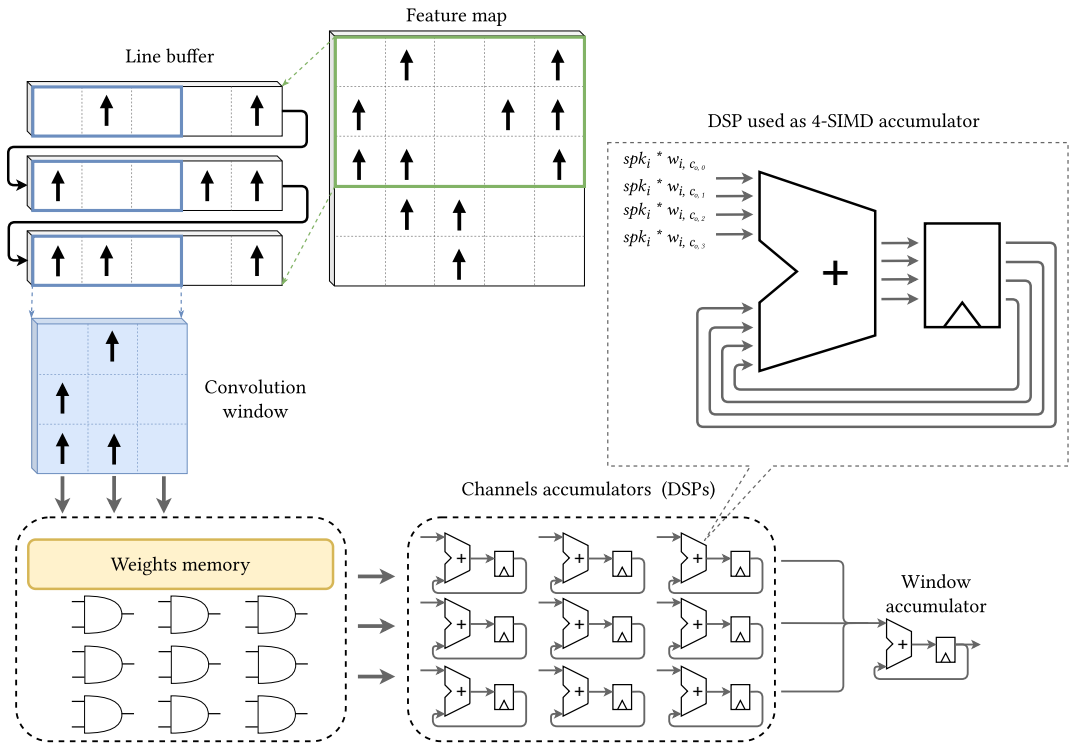


Figure 4.3: Processing of a window read from the line buffer. Each spike of the convolution window in bitwise AND-ed with the kernel weights and sent to the accumulator block. A DSP is allocated to each entry of the convolution window, and it accumulates the corresponding input channels; an output DSP is allocated to accumulate the different entries of the window once all the input channels have been processed. To fully exploit the DSPs hardware, these are configured in SIMD mode to process either 4 or 2 inputs each, which correspond to either 4 or 2 output channels of the convolution, respectively, when a single feature map is processed at time. To account for this, the weights blocks needs to multiply the inputs by 4 or 2 sets of filters at time.

provides a single-channel convolution window. This is processed by a block that multiplies each spike of the window by a filter entry; if the DSPs are used in SIMD mode, 2 or 4 filters are used for each input, depending on if the subsequent DSP accumulators are used in 2-SIMD or 4-SIMD. The output of the weight block, which is a window of activations, is then processed by `KSZ * KSZ` DSPs in parallel, one for each window entry. These accumulate activation along

the input channel dimension, and the resulting output is then accumulated along the feature map height and window by a single DSP accumulator, that will provide the final value of the convolution result.

4.1.2 The spiking neuron activation

The neuron activation is implemented as a standalone class in the C++ code. The approach used is the same as the one adopted by the major PyTorch [19] based machine learning frameworks, such as `snntorch` [33]: the neuron state update mechanism replaces the conventional ReLU activation in DNNs. In this work, the LIF neuron [32] is used as activation, and a specific class is developed for it.

The description of the LIF neuron class is as follows.

```
template <typename state_t, unsigned H, unsigned W, unsigned C>
class LIF {
 public:
  LIF(state_t leakage, state_t threshold): _leak(leakage), _thres(threshold) {}
  // ...
 private:
  const state_t _leak = 0, _thres = 0;
  state_t _state[H][W][C];
  // ...
};
```

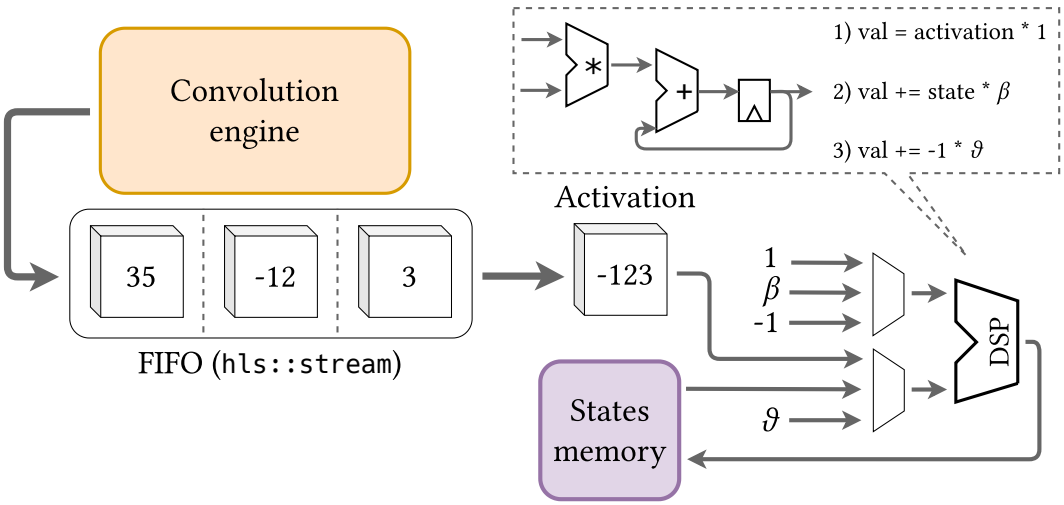


Figure 4.4: The convolution engine sends data to the LIF one via a FIFO, emulated by `hls::stream` in the C++ code. To save resources, since a product has to be performed to apply leakage to the state, a DSP is used in MAC configuration for leakage, membrane update and reset, using different multiplication factors.

The LIF layer is characterized by two hyperparameters, the spiking threshold `_thres` and the leakage coefficient `_leak`. Then, the shape of the feature map, `(H, W, C)`, is provided to

instantiate the memory resources to host the state on the FPGA chip. It runs as a concurrent kernel placed on the output on the convolutional layer, and these are interfaced through a FIFO. A single DSP per output channel is used in MAC configuration to perform the three operations: the activation is stored in the MAC register ("val" in Fig. 4.4); state leakage by multiplication with the leak factor β; if needed, soft reset of the state via subtraction of the threshold ϑ.

One could argue that, being β and ϑ constants, the multiplication by these can be trivially implemented using custom logic, without allocating a DSP for it; however, in some cases, the threshold and leakage factors vary across the channel dimension of a feature map. In this case, the value of these hyperparameters can be considered arbitrary from a hardware perspective and, hence, a multiplier has to be allocated to handle these, like it is show in Fig. 4.4.

One could notice that a complex shared circuit is implemented to make usage of the DSP for the state update. However, this results still in the best implementation possible, since using LUTs in the logic fabric would result in routing congestions for large designs, beyond higher power consumption, which had to be minimized for the drone use case presented in Section 4.2. Of course, having access to INT8 multipliers in the FPGA instead of the 27×18 provided in AMD ones, would have resulted in a more elegant and efficient implementation; however, an FPGA advantage is the possibility to completely change the design and its characteristics without fabricating a new ASIC, which leaves space to future improvements for the HLS library; hence, the drawbacks of having fixed hardware macros that can be reconfigured have to be taken into account and minimized.

4.2 Hardware acceleration of deep SNNs for optical flow estimation

Here, the drone application and use case presented in the introduction of this chapter is detailed.

4.2.1 Baseline

Figure 4.5 shows the processing pipeline designed by Hagenaars et al. [83]. Given Loihi restrictions (impossibility to route large networks with convolutional connectivity on it), instead of mapping a single SNN on the full frame of the event camera, four small SNNs are used on the four corners of the image, with each corner having shape 32×32. All the networks share the same set of weights and parameters (leakage constants and spiking threshold). This configuration is kept for our design, since training on the full feature map does not lead to a better accuracy in estimating the optical flow. Future work may consider changing the architecture of the neural network to better take advantage of the event camera capabilities.

Each corner is subsampled keeping the first 90 events, and a 5 ms time window is saved to a 2×32×32 frame (the two channels are for the positive and negative polarity events). This is then subsampled to a 16×16 shape using neighrest-neighbor and fed to the SNN.

The spikes from the four corners are then sampled and a linear transformation is applied to these, using a set of matrix multiplications in floating point precision. The resulting quantities are the optical flow vectors corresponding to the original camera corners (in Fig. 4.5, u_{00}, u_{01}, u_{10} and u_{11}).

The SNN employed has a CNN architecture, made of 4 layers, which is shown in Fig. 4.6. The first 3 are self-recurrent (*i.e.*, the output spikes are fed back to the neurons via a set of weights),

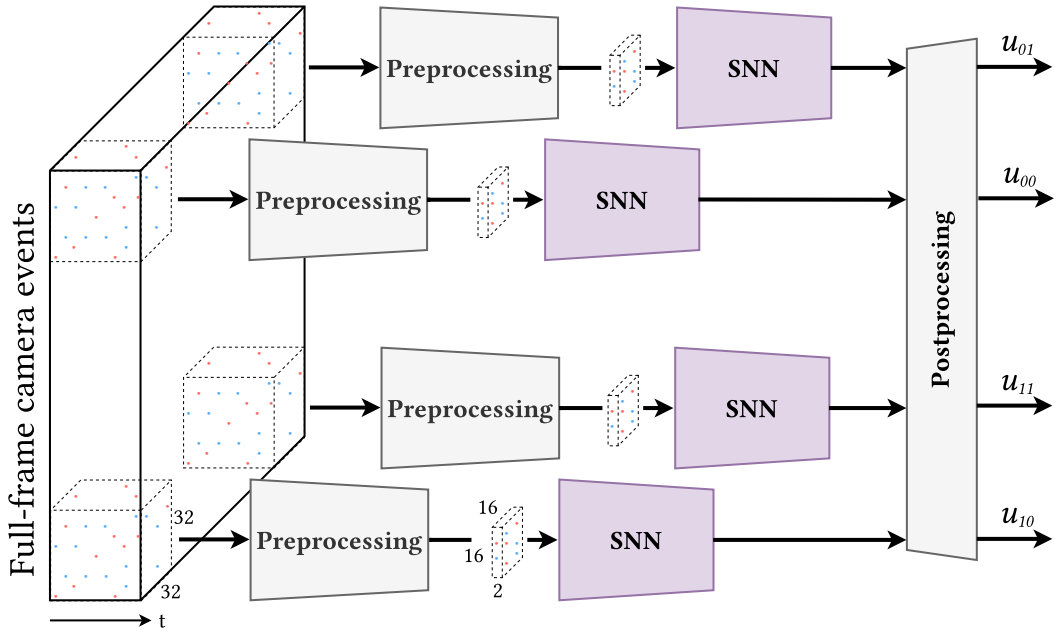


Figure 4.5: Processing pipeline to extract optical flow out of events, proposed by Hagenaars et al. [83]. Four SNNs are mapped to the corners of the event camera frame, and for each corner a vector describing the optical flow in that point is extracted from the spikes of the SNN, via a linear transformation applied as post-processing. The vectors are then fed to the drone controller for piloting it.

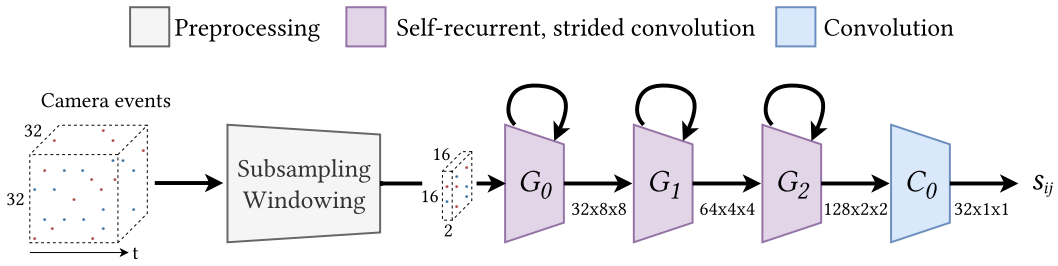


Figure 4.6: The neural network architecture employed by Hagenaars et al. [83]. The SNN is fed a 32×32 patch of the camera frame, which is then compressed to a time window of 5 ms and downsampled. Three out of four layers are self-recurrent.

and use a stride equal to 2, padding of 1 and a 3×3 kernel; the last one, is a simple convolutional one (stride of 1, no padding, 2×2 kernel), which outputs a 32×1×1 tensor of spikes, s_k which is then fed to the post-processing unit, which consists of a set of linear transformation matrices, to be converted to proper optical flow vectors for that corner. Spikes need to be post-processed due to the fact that floating point vectors representing the flow have to be provided to the motor controller.

Each SNN has an output linear layer of 32 units, which produces a vector of spikes $\vec{s_{ij}}$. The

four vectors, one per each corner, can be grouped in a matrix **S**.

$$\vec{s_{ij}} \in \mathbb{N}^{1\times 32},\ i,j \in \{0,1\}$$

$$\mathbf{S} \triangleq \begin{bmatrix} \vec{s_{00}} \\ \vec{s_{01}} \\ \vec{s_{10}} \\ \vec{s_{11}} \end{bmatrix} \in \mathbb{N}^{4\times 32}$$

The post-processing of the spikes consists in multiplying $\mathbb{S}$ by a weight matrix $\mathbb{W}_{\mathbb{post}}$ made of 32 b fixed-point parameters. The resulting matrix, $\mathbb{U}$, contains the optical flow vectors at the four corners of the camera. .

$$\mathbf{U} \triangleq \mathbf{S} \cdot \mathbf{W_{post}},\ \mathbf{W_{post}} \in \mathbb{R}^{32\times 8}$$

$$\mathbf{U} = \begin{bmatrix} \vec{u_{00}} \triangleq \begin{bmatrix} u_{00_x} & u_{00_y} \end{bmatrix} \in \mathbb{R}^{1\times 2} \\ \vec{u_{01}} \\ \vec{u_{10}} \\ \vec{u_{11}} \end{bmatrix}$$

In the baseline, the pre-processing and post-processing steps are implemented on an Intel general purpose processor available on the Loihi board, that is used also to communicate with the motor controller of the drone. The SNN is entirely mapped to Loihi. This constitutes the main source of inefficiency: since not all the steps are performed on custom hardware, the efficiency of the Loihi accelerator is masked by the CPU, which leads to the 1 W power consumption show in the paper.

Since the external world does not reason with spikes, a conversion to digital formats (*i.e.*, fixed-point, floating-point, integer) will always be needed. Regardless of the efficiency of the SNN accelerator, the preprocessing and post-processing will become the system bottleneck, just like the `x86` processor limits Loihi efficiency and analog-to-digital and digital-to-analog conversion worsen the efficiency of analog accelerators. For this reason, a "conventional" computing engine should be included in any acceleration platform, since the system efficiency counts, and not the single accelerator one.

4.2.2 Improvements

Since a custom hardware accelerator can be generated through the proposed library, a co-design opportunity arises, as most of the parameters precision and functionalities can be tuned to the use case. Given the application of our choice, *i.e.*, optical flow estimation using SNNs, we tried to simplify as much as possible the neuron model and to implement all the computations on the FPGA logic fabric, without resorting to external processors:

(i) The Loihi implementation is bounded to a CUBA-LIF [23] model with online learning capabilities. This has been changed with a LIF one, without changing the neural network architecture presented in Fig. 4.6, with an inference-only configuration.

(ii) In Hagenaars et al. [83], the Loihi neuron maps the state of the synaptic current and membrane voltage to 14 b, while in our implementation we remove the synaptic state and we use 8 b, 16 b data formats in two experiments. This allows to minimize the memory footprint associated with the neuron states and parameters, such as leakage and thresholds.

(iii) The leakage in Loihi is applied by a programmable shift, mapped to 12 b. In our accelerator, leakage is applied via a fixed-point multiplication, with the same precision as the state one, which allows for greater precision and better tunability of the neuron parameters during training. Two leakage parameters are needed in Loihi: one for the current, one for the voltage.

(iv) The spiking threshold in Loihi is mapped to 13 b. In our accelerator, the threshold has the same precision as the state.

(v) In our implementation, a different set of threshold and leakage parameters is allowed per channel of the layer, which allows for greater flexibility and more powerful learning, *i.e.*, the gradient descent based pipeline has more degrees of freedom on which to act to minimize the loss function provided.

(vi) In Loihi, reset after spike is hard (*i.e.*, the state is reset to 0); in our implementation, we chose soft reset (*i.e.*, the threshold is subtracted to the state after spike) as it shows better results in training.

(vii) The weights precision is scaled from 6 b (Loihi) to 4 b (our implementation).

(viii) The post-processing is implemented in the logic fabric of the FPGA.

(ix) The preprocessing is performed on the camera sensors itself, by programming the post-processing core available that generates frames of events transmitted through a MIPI interface to the FPGA.

A performance comparison between our implementation of the SNN accelerator and the Loihi baseline of Hagenaars et al. [83] is shown in Table 4.1. To estimate optical flow estimation precision of the network, the AEE metric is used, which is defined as the Euclidean distance between the predicted and ground-truth optical flow vector. Two FPGA devices are considered: a logic fabric only one, the AMD Spartan 7 device, and a system-on-chip (SoC) including an ARM processor (not used) alongside the logic of the FPGA, the AMD Zynq UltraScale+. The Zynq device is included even if the ARM processor is not exploited due to the fact the event-camera selected, produced by Prophesee, uses the MIPI communication protocol, which is not supported out-of-the-box in logic fabric-only FPGA devices.

Our implementation achieves around 10×lower power consumption with respect the Loihi one (945 mW Loihi, 100 mW our 16 b implementation on the fabric-only AMD Spartan 7 FPGA), with not significant performance drawbacks[1] in terms of AEE. The power consumption is measured in Vivado 2022.2 after having placed and routed the design on the FPGA device. This result is achieved by:

[1]Fine-tuned data to be confirmed after drone deployment.

Table 4.1: Comparison between our implementation and the Loihi baseline of Hagenaars et al. [83]

	Hagenaars et al. [83]	**This work**
Neuron model	CUBA LIF [23]	LIF
State precision [b]	14	8, 16
Weights precision [b]	6	4
Treshold precision [b]	13	8, 16
Leakage precision [b]	12	8, 16
Hardware implementation	NoC-based	Spatial, fused-layer
Frequency [MHz]	Asynchronous	25
Latency [ms]	<5	<2
Parameters [k]	141.6	127.7
Memory footprint [KiB]	132	69.7, 84.3
Average endpoint error	0.149	0.171, 0.148
Power consumption [mW]	945	189 (ZU1CG, 16 b) 100 (XC7S25, 16 b)

(i) lowering the operating clock frequency of the FPGA to 25 MHz, given the constraint on the latency of 5 ms required by the drone controller.

(ii) mapping the computations as much as possible to DSPs, since these are much more efficient (in the context of FPGA accelerators, the dynamic power consumption is lower) than LUTs for operations such as multiplications and additions.

It has to be remarked that most of the power consumption is due to the static power draining of the FPGA. For instance, in the case of the 16 b architecture, the Zynq device (ZU1CG) requires 162 mW and 27 mW of static and dynamic power, respectively, while the Spartan 7 device (XC7S25) requires 60 mW and 40 mW. The Zynq device uses a 16 nm CMOS process, while the Spartan 7 a 28 nm one; for this reason, the leakage power is larger for the Zynq system, while the dynamic power is lower.

The resulting latency, approximately 2 ms, is measured in co-simulation via Vitis HLS and Vivado 2022.2. To satisfy the latency requirement while using a very low operating frequency, parallelization of the computation is exploited. In particular, DSPs are used in SIMD mode. Instead of processing 4 or 2 output channels at time per DSP in a convolutional layer, as it is shown in Fig. 4.3, the spikes from the 4 corners of the camera are batched together and provided as a single input to the DSPs. Hence, each DSPs processes 4 input feature maps at time (when the required precision is above 12 b, 2 DSPs in 2-SIMD mode are used).

Moreover, the flexibility provided by our accelerator allows for more aggressive design exploration at the neural architecture and neuron model levels, leading to a smaller model in terms of memory footprint (132 KiB Loihi, 84.3 KiB our 16 b version of the network) and lower power consumption. In fact, reconfigurable hardware such as FPGAs has allowed us to explore different leakage mechanisms, different quantizations, and different dataflows until we have reached an optimum configuration for the task that minimizes power without compromising

performance. This would have not been possible with Loihi, since the neural model is fixed and results to be overparametrize, beyond not efficient, for the task at hand.

Chapter 5

Conclusions and future directions

In this thesis, the topic of brain-inspired neural network models, such as SNNs, and their hardware implementation has been discussed. One could ask: "When should we spike, and when not?", *i.e.*, when does using an SNN instead of an ANN gives an advantage in either processing efficiency or classification/regression precision. To answer this questions, some observation on the analyses and results presented in the thesis can be made:

(i) The analysis made in Chapter 3 highlights that current ANN models and digital hardware accelerators outperform their SNN counterparts on static images and object recognition tasks. One reason is that the multi-timestep processing of SNNs increases the operations per inference, leading to throughput and energy overheads. Hybrid SNN and ANN solutions might be the key to maximize task performance and efficiency. C-DNN'23 [62] shows the second-best efficiency among the accelerators analyzed, despite being implemented in a 28 nm CMOS process, while the best-performing chip takes advantage of a 5 nm CMOS technological node.

(ii) On temporal tasks, SNNs show a really competitive energy efficiency, and represent the only case in which full on-chip training and learning is performed [35]. This advantage should be exploited, together with new training strategies that promote both improvement in classification accuracy, which is still lower than the ANN counterpart, and sparse firing activity, that would lower further the energy consumption of the accelerator. We have decided to focus the HLS library on inference only, since current online learning algorithms are not powerful enough to justify a hardware implementation. Moreover, the library is meant to be used as an exploration tool for efficient inference on FPGAs.

(iii) Further investigation of efficient model and hardware solutions targeting bio-inspired sensors data, such as event cameras [38] and silicon cochleas [109], are needed to take advantage of the time-related functioning of spiking neurons and architectures. In Chapter 4, an SNN hardware acceleration library for FPGAs is proposed targeting event-based vision tasks such as optical flow estimation from events, showing very promising results in terms of processing efficiency compared to custom ASICs such as Intel Loihi [23], even when considering power-hungry platforms such as FPGAs.

In conclusion, the answer to our original question is that there are few tasks in which one

should spike, but full on-chip learning accelerators, such as ReckOn [35], and mixed architectures represent a promising research direction that should be further addressed, instead of simply replicating ANN architectures on tasks where there is no efficiency or accuracy advantage to SNNs.

For future research directions, there are different possibilities that are worth exploring:

(i) Object detection tasks with event cameras is still an unexplored topic in the hardware acceleration domain. On the neural network model size, given the large amount of data produced by these sensors, some relaxations could be made on the preprocessing step of events to allow DL models to use all this raw information to improve performance, similarly to what has been done in the natural language processing (NLP) domain.

(ii) Regarding the work presented in Chapter 4, the accelerator will be deployed on a real drone and tested on field. Moreover, small drones with limited power and weight capabilities could be used to fully take advantage of the custom hardware acceleration platform, and high-performance ANNs, such as the ones presented in Section 2.2.1, could be ported to hardware to improve the optical flow estimation performance and increase the autonomous capabilities of small drones.

(iii) Future SNN hardware should move away from simple neuron model that do not provide performant networks, such as the LIF one. In the research community, the direction addressed is to try to design algorithms that are inherently hardware friendly, and not try an infinite resources approach in which the scientist tries to come up with the best model and network for a task. If early DNN inventors, such as Geoffrey Hinton and Yan LeCun, would have done that, we would not benefit today from their ideas, since only when really performant hardware came along, *i.e.*, GPUs, we were able to witness the outstanding power of DNNs: in 2012, AlexNet was the first neural network able to largely outperform all the other handcrafted approaches on ImageNet using CNNs, and this has been possible only thanks to the fact that powerful enough GPUs were available for training the network. In neuromorphic computing case, new brain-inspired models such as the ELM one [97] seem to be promising, even if computationally prohibitive at the current moment. The hardware must evolve to allow these more powerful and interesting models to be run, and not the other way around.

(iv) We should move away from spikes. Spikes are not able to densely encode information, which is what allows us with neural network to have powerful learning models. I seriously doubt that the brain neurons work with single bit signals and I think that, instead, analog quantities are used in biology, due to fact that the world is analog, and since the birth of computer architecture we have learned to model it using multi-bit values.

Bibliography

[1] Mostafa Rahimi Azghadi et al. "Hardware implementation of deep network accelerators towards healthcare and biomedical applications." In: *IEEE Transactions on Biomedical Circuits and Systems* 14.6 (2020), pp. 1138–1159.

[2] Sami Barchid et al. "Spiking neural networks for frame-based and event-based single object localization." In: *arXiv preprint arXiv:2206.06506* (2022).

[3] Chiara Bartolozzi and Giacomo Indiveri. "Synaptic dynamics in analog VLSI." In: *Neural computation* 19.10 (2007), pp. 2581–2603.

[4] Arindam Basu et al. "Spiking neural network integrated circuits: A review of trends and future directions." In: *2022 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE. 2022, pp. 1–8.

[5] Thomas Bohnstingl et al. "Speech Recognition Using Biologically-Inspired Neural Networks." In: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2022, pp. 6992–6996.

[6] Maxence Bouvier et al. "Spiking neural networks hardware implementations and challenges: A survey." In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15.2 (2019), pp. 1–35.

[7] Tom Brown et al. "Language models are few-shot learners." In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[8] Marco Cannici et al. "Asynchronous convolutional networks for object detection in neuromorphic cameras." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2019.

[9] Maurizio Capra et al. "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead." In: *IEEE Access* 8 (2020), pp. 225134–225180.

[10] Biswadeep Chakraborty, Xueyuan She, and Saibal Mukhopadhyay. "A fully spiking hybrid neural network for energy-efficient object detection." In: *IEEE Transactions on Image Processing* 30 (2021), pp. 9014–9029.

[11] Guoguo Chen, Carolina Parada, and Georg Heigold. "Small-footprint keyword spotting using deep neural networks." In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2014, pp. 4087–4091.

[12] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. "Using dataflow to optimize energy efficiency of deep neural network accelerators." In: *IEEE Micro* 37.3 (2017), pp. 12–21.

[13] Yu-Hsin Chen et al. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks." In: *IEEE journal of solid-state circuits* 52.1 (2016), pp. 127–138.

[14] Nicholas FY Chen. "Pseudo-labels for supervised learning on dynamic vision sensor data, applied to object detection under ego-motion." In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 644–653.

[15] Qinyu Chen et al. "Reducing latency in a converted spiking video segmentation network." In: *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2021, pp. 1–5.

[16] Qinyu Chen et al. "Skydiver: A Spiking Neural Network Accelerator Exploiting Spatio-Temporal Workload Balance." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.12 (2022), pp. 5732–5736.

[17] Xuelian Cheng et al. "Hierarchical neural architecture search for deep stereo matching." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 22158–22169.

[18] Google Cloud. *System Architecture of TPU VM*. `https://cloud.google.com/tpu/docs/system-architecture-tpu-vm?hl=en#tpu_v3`. Accessed on June 27, 2023. Accessed 2023.

[19] PyTorch Contributors. *PyTorch: An open-source machine learning framework*. 2021. URL: `https://pytorch.org`.

[20] Loïc Cordone, Benoît Miramond, and Philippe Thierion. "Object detection with spiking neural networks on automotive event data." In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2022, pp. 1–8.

[21] Benjamin Cramer et al. "The heidelberg spiking data sets for the systematic evaluation of spiking neural networks." In: *IEEE Transactions on Neural Networks and Learning Systems* 33.7 (2020), pp. 2744–2757.

[22] Bill Dally. "Hardware for Deep Learning." In: *2023 IEEE Hot Chips 35 Symposium (HCS)*. IEEE Computer Society. 2023, pp. 1–58.

[23] Mike Davies et al. "Loihi: A neuromorphic manycore processor with on-chip learning." In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[24] Pierre De Tournemire et al. "A large scale event-based detection dataset for automotive." In: *arXiv preprint arXiv:2001.08499* (2020).

[25] Jeffrey Delmerico et al. "Are we ready for autonomous drone racing? the UZH-FPV drone racing dataset." In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 6713–6719.

[26] Jia Deng et al. "Imagenet: A large-scale hierarchical image database." In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[27] Advanced Micro Devices. *Vitis HLS*. 2023. URL: `https://www.xilinx.com/products/design-tools/vitis/vitis-hls.html`.

[28] Alfio Di Mauro et al. "SNE: an energy-proportional digital accelerator for sparse event-based convolutions." In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022, pp. 825–830.

[29] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale." In: *arXiv preprint arXiv:2010.11929* (2020).

[30] Rodney Douglas, Misha Mahowald, and Carver Mead. "Neuromorphic analogue VLSI." In: *Annual review of neuroscience* 18.1 (1995), pp. 255–281.

[31] Jason K Eshraghian, Xinxin Wang, and Wei D Lu. "Memristor-based binarized spiking neural networks: Challenges and applications." In: *IEEE Nanotechnology Magazine* 16.2 (2022), pp. 14–23.

[32] Jason K Eshraghian et al. "Training spiking neural networks using lessons from deep learning." In: *Proceedings of the IEEE* (2023).

[33] Jason K. Eshraghian. *snnTorch: A PyTorch library for spiking neural networks*. 2021. URL: https://snntorch.readthedocs.io.

[34] Charlotte Frenkel, David Bol, and Giacomo Indiveri. "Bottom-Up and Top-Down Approaches for the Design of Neuromorphic Processing Systems: Tradeoffs and Synergies Between Natural and Artificial Intelligence." In: *Proceedings of the IEEE* (2023).

[35] Charlotte Frenkel and Giacomo Indiveri. "ReckOn: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales." In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. IEEE. 2022, pp. 1–3.

[36] Andre Fu, Mahdi S Hosseini, and Konstantinos N Plataniotis. "Reconsidering co2 emissions from computer vision." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 2311–2317.

[37] Steve B Furber et al. "The spinnaker project." In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[38] Guillermo Gallego et al. "Event-based vision: A survey." In: *IEEE transactions on pattern analysis and machine intelligence* 44.1 (2020), pp. 154–180.

[39] Chang Gao, Tobi Delbruck, and Shih-Chii Liu. "Spartus: A 9.4 TOp/s FPGA-Based LSTM Accelerator Exploiting Spatio-Temporal Sparsity." In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–15. DOI: 10.1109/TNNLS.2022.3180209.

[40] Chang Gao et al. "DeltaRNN: A Power-Efficient Recurrent Neural Network Accelerator." In: *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '18. Monterey, CALIFORNIA, USA: Association for Computing Machinery, 2018, pp. 21–30. ISBN: 9781450356145. DOI: 10.1145/3174243.3174261. URL: https://doi.org/10.1145/3174243.3174261.

[41] Chang Gao et al. "EdgeDRNN: Recurrent Neural Network Accelerator for Edge Inference." In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 10.4 (2020), pp. 419–432. DOI: 10.1109/JETCAS.2020.3040300.

[42] Chang Gao et al. "Real-Time Speech Recognition for IoT Purpose using a Delta Recurrent Neural Network Accelerator." In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2019, pp. 1–5. DOI: 10.1109/ISCAS.2019.8702290.

[43] Ravi Garg et al. "Unsupervised cnn for single view depth estimation: Geometry to the rescue." In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*. Springer. 2016, pp. 740–756.

[44] Mathias Gehrig and Davide Scaramuzza. "Recurrent vision transformers for object detection with event cameras." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 13884–13893.

[45] Michael Gilbert et al. "LoopTree: Enabling Exploration of Fused-layer Dataflow Accelerators." In: *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE. 2023, pp. 316–318.

[46] Juan Sebastian P. Giraldo et al. "Vocell: A 65-nm Speech-Triggered Wake-Up SoC for 10-μW Keyword Spotting and Speaker Verification." In: *IEEE Journal of Solid-State Circuits* 55.4 (2020), pp. 868–878. DOI: 10.1109/JSSC.2020.2968800.

[47] Ross Girshick. "Fast r-cnn." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[48] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. "Unsupervised monocular depth estimation with left-right consistency." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 270–279.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[50] Xiaodong Gu et al. "Cascade cost volume for high-resolution multi-view stereo and stereo matching." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2495–2504.

[51] Jesse Hagenaars, Federico Paredes-Vallés, and Guido De Croon. "Self-supervised learning of event-based optical flow with spiking neural networks." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 7167–7179.

[52] Kaiming He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[53] Massimiliano Iacono et al. "Towards event-driven object detection with off-the-shelf deep learning." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–9.

[54] Eddy Ilg et al. "Flownet 2.0: Evolution of optical flow estimation with deep networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2462–2470.

[55] *Innatera*. https://www.synsense.com/. Accessed: October 22, 2023.

[56] Eugene M Izhikevich. "Simple model of spiking neurons." In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1569–1572.

[57] Zhuangyi Jiang et al. "Mixed frame-/event-driven fast pedestrian detection." In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8332–8338.

[58] Lana Josipovic. *High-level synthesis of dynamically scheduled circuits*. Tech. rep. EPFL, 2021.

[59] Ben Keller et al. "A 95.6-TOPS/W Deep Learning Inference Accelerator With Per-Vector Scaled 4-bit Quantization in 5 nm." In: *IEEE Journal of Solid-State Circuits* (2023).

[60] Salman Khan et al. "Transformers in vision: A survey." In: *ACM computing surveys (CSUR)* 54.10s (2022), pp. 1–41.

[61] Kwantae Kim et al. "A 23- μ W Keyword Spotting IC With Ring-Oscillator-Based Time-Domain Feature Extraction." In: *IEEE Journal of Solid-State Circuits* 57.11 (2022), pp. 3298–3311. DOI: 10.1109/JSSC.2022.3195610.

[62] Sangyeob Kim et al. "C-DNN: A 24.5-85.8 TOPS/W complementary-deep-neural-network processor with heterogeneous CNN/SNN core architecture and forward-gradient-based sparsity generation." In: *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE. 2023, pp. 334–336.

[63] Sangyeob Kim et al. "SNPU: An Energy-Efficient Spike Domain Deep-Neural-Network Processor With Two-Step Spike Encoding and Shift-and-Accumulation Unit." In: *IEEE Journal of Solid-State Circuits* (2023).

[64] Sehoon Kim et al. "Full stack optimization of transformer inference: a survey." In: *arXiv preprint arXiv:2302.14017* (2023).

[65] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition." In: *Neural computation* 1.4 (1989), pp. 541–551.

[66] Jianing Li et al. "Asynchronous spatio-temporal memory network for continuous event-based object detection." In: *IEEE Transactions on Image Processing* 31 (2022), pp. 2975–2987.

[67] Jinyu Li et al. "High-accuracy and low-latency speech recognition with two-head contextual layer trajectory LSTM model." In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 7699–7703.

[68] Yijin Li et al. "Graph-based asynchronous event processing for rapid object recognition." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 934–943.

[69] Ji Lin et al. "On-device training under 256kb memory." In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 22941–22954.

[70] Ze Liu et al. "Swin transformer v2: Scaling up capacity and resolution." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 12009–12019.

[71] Horowitz Mark. "Computing' s energy problem (and what we can do about it)." In: *Proceedings of the IEEE International Solid-State Circuits Conference, San Francisco, CA, USA*. 2014, pp. 9–13.

[72] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[73] Linyan Mei et al. "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators." In: *IEEE Transactions on Computers* 70.8 (2021), pp. 1160–1174.

[74] Nico Messikommer et al. "Event-based asynchronous sparse convolutional networks." In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VIII 16*. Springer. 2020, pp. 415–431.

[75] Filippo Minnella et al. "Design and Optimization of Residual Neural Network Accelerators for Low-Power FPGAs Using High-Level Synthesis." In: *arXiv preprint arXiv:2309.15631* (2023).

[76] Huiyu Mo et al. "9.2 A 28nm 12.1 TOPS/W dual-mode CNN processor using effective-weight-based convolution and error-compensation-based prediction." In: *2021 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 64. IEEE. 2021, pp. 146–148.

[77] Maxim Naumov et al. "Deep learning recommendation model for personalization and recommendation systems." In: *arXiv preprint arXiv:1906.00091* (2019).

[78] Sechang Oh et al. "An Acoustic Signal Processing Chip With 142-nW Voice Activity Detection Using Mixer-Based Sequential Frequency Scanning and Neural Network Classification." In: *IEEE Journal of Solid-State Circuits* 54.11 (2019), pp. 3005–3016. DOI: 10.1109/JSSC.2019.2936756.

[79] Fabrizio Ottati. *Awesome Neuromorphic Hardware*. https://github.com/fabrizio-ottati/awesome-neuromorphic-hw. 2023.

[80] Fabrizio Ottati et al. "To spike or not to spike: A digital hardware perspective on deep learning acceleration." In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2023).

[81] Vassil Panayotov et al. "Librispeech: an asr corpus based on public domain audio books." In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2015, pp. 5206–5210.

[82] Angshuman Parashar et al. "Timeloop: A systematic approach to dnn accelerator evaluation." In: *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE. 2019, pp. 304–315.

[83] Federico Paredes-Vallés et al. "Fully neuromorphic vision and control for autonomous drone flight." In: *arXiv preprint arXiv:2303.08778* (2023).

[84] Jun-Seok Park et al. "A Multi-Mode 8k-MAC HW-Utilization-Aware Neural Processing Unit With a Unified Multi-Precision Datapath in 4-nm Flagship Mobile SoC." In: *IEEE Journal of Solid-State Circuits* 58.1 (2022), pp. 189–202.

[85] Melika Payvand et al. "Dendritic Computation through Exploiting Resistive Memory as both Delays and Weights." In: *arXiv preprint arXiv:2305.06941* (2023).

[86] Bruno U. Pedroni et al. "Small-footprint Spiking Neural Networks for Power-efficient Keyword Spotting." In: *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. 2018, pp. 1–4. DOI: 10.1109/BIOCAS.2018.8584832.

[87] Christian Pehle et al. "The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity." In: *Frontiers in Neuroscience* 16 (2022), p. 795876.

[88] Etienne Perot et al. "Learning to detect objects with a 1 megapixel event camera." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 16639–16652.

[89] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement." In: *arXiv preprint arXiv:1804.02767* (2018).

[90] Simon Schaefer, Daniel Gehrig, and Davide Scaramuzza. "Aegnn: Asynchronous event-based graph neural networks." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 12371–12381.

[91] Cedric Scheerlinck et al. "Fast image reconstruction with an event camera." In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 156–163.

[92] Samuel Schmidgall et al. "Brain-inspired learning in artificial neural networks: a review." In: *arXiv preprint arXiv:2305.11252* (2023).

[93] Yannick Schnider et al. "Neuromorphic Optical Flow and Real-time Implementation with Event Cameras." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 4128–4137.

[94] Weiwei Shan et al. "AAD-KWS: A Sub-μ W Keyword Spotting Chip With an Acoustic Activity Detector Embedded in MFCC and a Tunable Detection Window in 28-nm CMOS." In: *IEEE Journal of Solid-State Circuits* 58.3 (2023), pp. 867–876. DOI: 10.1109/JSSC.2022.3197838.

[95] Xueyuan She et al. "Safe-dnn: a deep neural network with spike assisted feature extraction for noise robust inference." In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[96] Cristina Silvano et al. "A survey on deep learning hardware accelerators for heterogeneous hpc platforms." In: *arXiv preprint arXiv:2306.15552* (2023).

[97] Aaron Spieler et al. "The ELM Neuron: an Efficient and Expressive Cortical Neuron Model Can Solve Long-Horizon Tasks." In: *arXiv preprint arXiv:2306.16922* (2023).

[98] Deqing Sun et al. "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8934–8943.

[99] *Synsense*. https://www.synsense.ai/. Accessed: October 22, 2023.

[100] Vivienne Sze et al. "Efficient processing of deep neural networks: A tutorial and survey." In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

[101] Zachary Teed and Jia Deng. "Raft: Recurrent all-pairs field transforms for optical flow." In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer. 2020, pp. 402–419.

[102] Stepan Tulyakov et al. "Time lens: Event-based video frame interpolation." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 16155–16164.

[103] Rangharajan Venkatesan et al. "Magnet: A modular accelerator generator for neural networks." In: *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2019, pp. 1–8.

[104] Yang Wang et al. "A 28nm 27.5 TOPS/W approximate-computing-based transformer processor with asymptotic sparsity speculating and out-of-order computing." In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. IEEE. 2022, pp. 1–3.

[105] Pete Warden. "Speech commands: A dataset for limited-vocabulary speech recognition." In: *arXiv preprint arXiv:1804.03209* (2018).

[106] Stanisław Woźniak et al. "Deep learning incorporating biologically inspired neural dynamics and in-memory computing." In: *Nature Machine Intelligence* 2.6 (2020), pp. 325–336.

[107] Kai Xu et al. "Spatiotemporal CNN for video object segmentation." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 1379–1388.

[108] Mengde Xu et al. "End-to-end semi-supervised object detection with soft teacher." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 3060–3069.

[109] Minhao Yang et al. "A 0.5 V 55μW 6x 2 Channel Binaural Silicon Cochlea for Event-Driven Stereo-Audio Sensing." In: *IEEE Journal of Solid-State Circuits* 51.11 (2016), pp. 2554–2569.

[110] Minhao Yang et al. "Design of an Always-On Deep Neural Network-Based 1- μ W Voice Activity Detector Aided With a Customized Software Model for Analog Feature Extraction." In: *IEEE Journal of Solid-State Circuits* 54.6 (2019), pp. 1764–1777. DOI: 10.1109/JSSC.2019.2894360.

[111] Weihao Yuan et al. "New crfs: Neural window fully-connected crfs for monocular depth estimation." In: *arXiv preprint arXiv:2203.01502* (2022).

[112] Yuhui Yuan et al. "Segmentation transformer: Object-contextual representations for semantic segmentation. arXiv 2019." In: *arXiv preprint arXiv:1909.11065* ().

[113] Xiaohua Zhai et al. "Scaling vision transformers." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12104–12113.

[114] Alex Zihao Zhu et al. "EV-FlowNet: Self-supervised optical flow estimation for event-based cameras." In: *arXiv preprint arXiv:1802.06898* (2018).

[115] Alex Zihao Zhu et al. "The multivehicle stereo event camera dataset: An event camera dataset for 3D perception." In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2032–2039.

[116] Alex Zihao Zhu et al. "Unsupervised event-based learning of optical flow, depth, and egomotion." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 989–997.

[117] Rui-Jie Zhu, Qihang Zhao, and Jason K Eshraghian. "Spikegpt: Generative pre-trained language model with spiking neural networks." In: *arXiv preprint arXiv:2302.13939* (2023).