Brain-inspired feature extraction for near sensor extreme edge processing with Spiking Neural Networks

Alexandra-Florentina Dobrița

Delft University of Technology

TU Delft

Brain-inspired feature extraction for near sensor extreme edge processing with Spiking Neural Networks

by

Alexandra-Florentina Dobrița

to obtain the degree of Master of Science in Computer Engineering
at the Delft University of Technology,
to be defended publicly on Friday March 29th, 2024 at 11:00 AM.

Student number:	5373301	
Project duration:	May, 2022 – March, 2024	
Thesis committee:	Prof. Dr. S. Hamdioui,	TU Delft, main supervisor
	Dr. C. Frenkel,	TU Delft
	Dr. M. Sifalakis,	IMEC the Netherlands
	Dr. A. Yousefzadeh,	IMEC the Netherlands
	Prof. Dr. S. Thorpe,	CNRS Toulouse
	Dr. A. Gebregiorgis,	TU Delft, daily supervisor

This thesis is confidential and cannot be made public until March 29, 2025.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

The work in this thesis was supported by IMEC the Netherlands. Their cooperation is hereby gratefully acknowledged.

Copyright ©2024 Alexandra-Florentina Dobrița
All rights reserved.

Abstract

Motivated by the desire to bring intelligent processing at the Edge, enabling online learning on resource- and latency-constrained embedded devices has become increasingly appealing, as it has the potential to tackle a wide range of challenges: on the one hand, it can deal with on-the-fly adaptation to fast sensor-generated streams of data under changing environments and on the other hand, it can address a variety of challenges associated with offline training in the cloud, such as incurred energy consumption of sensor data transfers and extra memory storage for the training samples, but also data privacy and security concerns. Concurrently, maintaining low-latency and power-efficient inference is paramount for edge AI computing systems, and thus learning/adapting online with minimal incurred overhead is crucial.

In this work, we propose EON-1, an Edge ONline Learning SCNN (Spiking Convolutional Neural Network) processor with 1-bit synaptic weights, 1-spike per neuron and 1-neuron updated per input, which we have benchmarked for both ASIC and FPGA platforms. Our key contribution is proposing a binary and stochastic SDTP rule which, benchmarked in an ASIC node, achieves less than 1% energy overhead for inference. To our knowledge, our solution incurs the least energy overhead for inference, compared to state-of-the-art solutions, showing a better efficiency by at least a factor of 10x. We also report 94% and 77.65% accuracy on the MNIST and Fashion-MNIST classification tasks, and we achieve 0.09pJ/SOP and 1.5pJ/SOP energy efficiency during inference and learning, respectively. We extend our solution to demonstrate a practical use-case of performing inference in real-time UHD videos while coping with streaming data and we showcase 60 FPS UHD video processing.

Alexandra-Florentina Dobrița
Delft, April 2025

Acknowledgements

I would like to begin this section by thanking my IMEC supervisors, Dr. Manolis Sifalakis and Dr. Amirreza Yousefzadeh. Throughout this thesis journey, they have been exceptional mentors, highly shaping my professional and research development. Despite their seniority and tremendously higher expertise, I always felt my ideas were equally valued, although I had much less knowledge and experience than them, and they have been encouraging and understanding even at times where I was stubborn or shortsighted about the bigger picture. They have always made time in their incredibly busy schedule, whenever I needed their input or advice. I started with very little knowledge about the Neuromorphic field, but nevertheless, they have been patient with my learning pace, and they have allowed me to feel comfortable about making mistakes and speaking out my mind, which has enabled me to learn and grow even more. For all the valuable lessons, I deeply thank them, I could not have wished for better mentors.

I would like to extend this appreciation to all my Neuromorphic team colleagues: Kanishkan, Guangzhi, Kevin, Gert-Jan, Paul and Yingfu for being an amazing team and for encouranging me with my thesis work and offering advice whenever I needed it.

Without saying, this thesis work would not have been possible without prof. Simon Thorpe, who has initiated this project. I have learned a lot from him and his contagious enthusiasm for this project has been extremely encouraging. I am very honoured for having the opportunity to collaborate with someone who's life work on rapid processing in the visual system has been pivotal in the field of Spiking Neural Networks.

I would also like to thank prof. Said Hamdioui and Dr. Anteneh Gebregiorgis, for helping me shape my work for quality scientific output.

I would like to extend my thanks to colleagues from the QCE department, Mahdi, Michael, Troya, Moritz, Arne, Abdullah, Asmae. They have made my master thesis journey much more fun, especially with epic karaoke sessions or gezellig lunches.

And lastly, I would like to thank my parents and siblings, my aunt, Cristina, and my friends, for their encouragement and patiently listening to my thesis ups and downs: Lili and Andrei, Garazi, Aniket, Varshiny, Alex, Prithvish, Luiza, Cristina, Emilio.

Alexandra-Florentina Dobrița
Delft, April 2025

List of Figures

List of Tables

Contents

1 Introduction

1.1. Motivation

1.1.1. The pillars behind AI's popularity

Everyday, artificial intelligence (AI) significantly influences our lives, by enhancing our digital experiences through various interactions with our smart portable-devices connected to the internet: from queries we make to our virtual assistants, navigation apps and language translation engines to receiving personalized social media content and recommendations for online shopping or streaming entertainment. On a less than daily basis but also at a fast-paced rate, our life quality increases exponentially through AI-powered medical advancements, smart home devices and autonomous driving. Needless to say, in our society today, AI continuously contributes to and generates wealth and progress, all of which is attributed to it's high performance in a wide range of tasks: from object detection and image classification to speech and language recognition.

AI's tremendous success, however, would not have been possible without a combination of more training data with AI networks that are deeper and have more parameters. Combining the two was propelled by a surge in computational power over the last two decades [14]. The key aspect in which massive computational power has enabled AI progress is in the reduced latency for exploration and training of the AI models. This reduction in latency comes as a result of much of the industry and research focus in the past years having been on creating or extending powerful hardware exhibiting high level of parallelism and tailored for accelerating neural networks operations (i.e., matrix multiplications). These efforts have facilitated newer and better AI models being generated at an incredibly fast rate.

Set on these pillars (i.e., more training data, higher number of parameters and more available computational power), the state-of-the-art models in AI today, Deep Neural Networks (DNNs), took-off. This is illustrated through the example of AlexNet [15], the DNN model that won the 1000-category image classification challenge, ImageNet [16]. It's performance surpassing all previous models was attributed to the depth of its network, which, in turn, was made feasible by the available computational power for training, e.g., at that time, AlexNet distributed it's model across two GPUs (Graphical Processing Units). Their achievement gave rise to the hype of DNNs, where the relation between performance and the network's depth and number of parameters, actuated by massive computational power, has been demonstrated via a number of powerful DNN models developed in the past 10 years [17], [18]. An extreme example of an ultra-powerful DNN model is the conversational agent, GPT-3, which features an impressive depth of 96 layers and has 175 billion trainable parameters [19], running on 10000 Nvidia GPUs (Graphical Processing Unit) and 285000 CPUs (Central Processing Unit) cores [20].

However, even by a layman's crunching of these numbers, it is easy to notice that current state-of-the-art AI models that we interact-with daily are large Big Data-driven networks that do not run on ordinary modern personal computers (PCs), but instead they distribute their workload over thousands of processing units clustered in supercomputers. Due to high cost and specific power and cooling requirements for using big clusters of such computing units, they can only be hosted in specialized data centers. For this reason, most AI models nowadays run in the Cloud, which is an infrastructure of software and hardware tools that facilitate access to supercomputing-rich data centers. Cloud AI, however, comes at huge costs in terms of power consumption and environmental impact. For instance, training the previously mentioned GPT-3 language model emits 27.8 times more CO_2 than an average American consumes in one year [21], [22].

1.1.2. Edge AI

Concurrent to Cloud AI growth, the advancements in semiconductors have enabled rapid developments in sensor-based environments such as autonomous driving, wearable devices and IoT (Internet-of-things) systems, where the need to bring AI smart decisioning next to the state-of-the art sensor systems has triggered the emergence of a new type of AI deployment, named Edge AI. For these applications, data transfers between Edge devices and Cloud are infeasible or undesired, on the one hand due to stringent real-time response necessary for some embedded systems and on the other hand due to data privacy and security concerns. For instance, a practical scenario where AI inference response in real-time is critical is found in autonomous vehicles, where vast amounts of multiple sensors data need to be processed almost instantly by the AI system, as any incurred delay can result in fatal accidents or missed opportunities for safe maneuvering. An example of critical data privacy and security concerns is in medical applications, an everyday scenario being wearable health monitoring devices. Another interesting example where AI can have tremendous impact but is still used with limitation due to privacy concerns is in Organs-on-Chips (OoCs) [23]. One way in which OoCs are revolutionizing the medical world is true more accurate drug discovery and research, however since OoCs often use human-derived data, integrating AI in a private and secure way is crucial. There is, therefore, ample reason to address these problems, in order to harvest the benefits of AI in sensor-based environments.

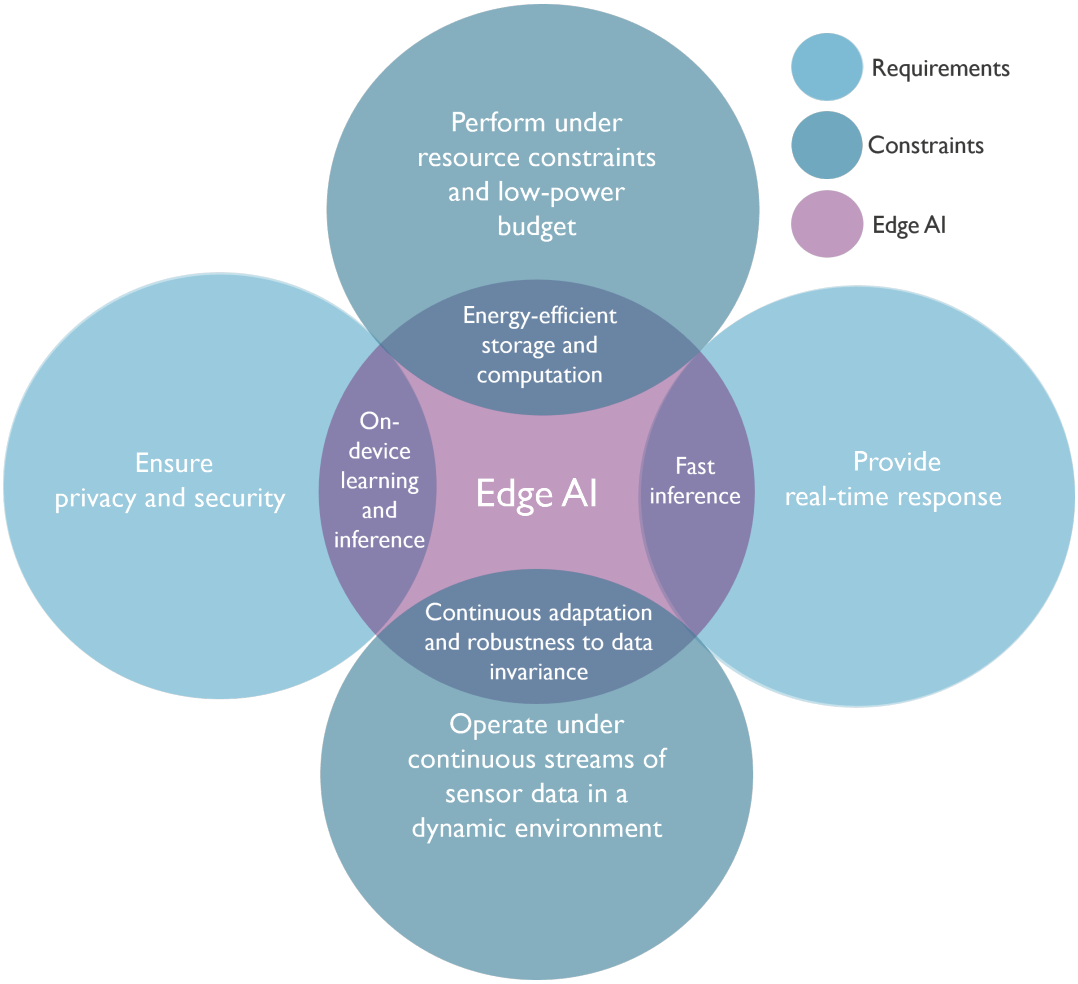


Figure 1.1: Functions and features of Edge AI at the intersection of edge-devices constraints with smart applications requirements

While the latency restrictions have been addressed to an extent for AI inference by a number of accelerators [24], [25], [26] and optimization techniques such as quantization or reducing the precision of the data [27], [28], these solutions are still not power-efficient enough for most Edge AI platforms.

Moreover, fast inference while coping with streaming data still remains a challenge, for instance in high-resolution real-time videos. At the other pole, addressing the data privacy and security concerns calls for both training and inference to be ran locally, on-device and in close proximity to data sensors. However, training on-device is not an easy task due to limited storage and computational resources and low energy consumption budget that are inherent to edge devices. Thus, batch-training over long period of times using backpropagation, the standard training flavour for DNNs, is often not feasible at the edge. Moreover, as edge devices operate in dynamic surroundings, input data distribution and user behaviour changes rapidly over-time, which brings an extra challenge to AI at the edge, i.e., that of featuring a degree of adaptation on-the-fly (i.e., during and without disturbing inference) to environment changes.

With these constraints in mind, and as emphasised in subsection 1.1.1, the current state-of-the-art DNN models, although highly performing, are heavily relying on high amounts of training data and computational power. Moreover, DNNs training flavour, backpropagation, requires intensive computations due to many gradient updates, that use floating-point parameters and complex memory requirements for storing intermediate gradient values [29], [30], which limits it's deployment for on-device learning. These DNN features renders them unsuitable for edge deployment, which calls for a divergent AI roadmap [31] in terms of methods and algorithms suitable for Edge requirements, as compared to the well-established, high-accuracy techniques used in Cloud AI.

1.1.3. Brain-inspired computing and neuromorphic hardware

At the twilight of Moore's law [32] and motivated by the high carbon footprint of conventional AI [33], [34], [22], brain-inspired computing, also referred-to as neuromorphic computing in literature, is a promising alternative to DNNs, as they aim to emulate the power-efficient learning and inference behaviour observed in the biological brain. For instance, it is estimated that the brain performs about 1 exaflop of computations per second, within 20W of power, which is 20 millions less than what one of the biggest supercomputers today, Oak Ridge Frontier [35], consumes for the same throughput [36].

The low-power consumption ability of the brain is attributed to it's neural communication via spikes in an event-driven processing manner (i.e., neurons get activated only for specific events) [37], which enables spatio-temporal sparsity. Another feature of brain-inspired computing is that memory and computation are co-located for each neuron cell, behaviour which, if emulated at a digital circuit level, addresses the memory bottleneck present in state-of-the-art computational platforms based on Von-Neumann architectures. Moreover, in brain-inspired neural networks, learning and adaptation occur online, as opposed to batch-training in DNNs. These features render brain-inspired computing a promising solution for bringing intelligence at the Edge.

To explore the brain efficiency in hardware, two main approaches have emerged [38]: one is a *bottom-up* approach, where authors replicate neurons and synapses with a high-level of bio-plausibility in general purpose neuromorphic hardware [39], [40], [41], [42], while the other is a *top-down* approach, where custom neuromorphic processors are proposed with fixed learning rules that are less bio-plausible, but borrow brain-inspired concepts to solve specific tasks with the best trade-off between accuracy and energy efficiency [43], [44], [45], [46]. The latter approach is a better fit for extreme Edge applications with specific requirements.

1.2. Problem statement

At the intersection of hard constraints of embedded devices with the desired requirements of smart sensor-based applications, seamlessly incorporating intelligence at the Edge is equivalent to a multi-objective problem, as depicted in Figure 1.1. As a consequence, performing common AI tasks such as image classification at the Edge brings many challenges for both training and inference. It is also important to note that for practical edge applications, training data can be scarce, which imposes an extra challenge on the training aspect, as satisfactory accuracy needs to be reached with a limited number of training points.

Online and continual learning [47] is an AI training method where the model weights are updated for each input data sample, as opposed to traditional batch-training which changes model weights after a number of inputs have been presented to the network. Enabling online and continual learning on

resource- and latency-constrained Edge devices has the potential to tackle a wide range of challenges: on the one hand, it can deal with on-the-fly adaptation to fast streams of data generated by sensors under changing environments and on the other hand, it can serve as a low-cost, data-secure, local and on-device training method that addresses privacy and the incurred energy consumption of data transfers for off-chip training and extra memory storage for the training samples.

Following brain-inspired computing principles, Spiking Neural Networks (SNNs), also considered the third generation of neural networks [48], inherently learn and adapt their synaptic weights in an online fashion. Moreover, they are power-efficient, exhibit high-parallelism and neuron co-located memory and processing, benefiting both training and inference phases. These aspects have made them a popular choice among researchers for exploring online learning and adaptation at the Edge [49].

In the absence of standard SNN learning rules and hardware platforms, achieving online learning for Edge devices by methods of brain-inspired computing calls for algorithm-hardware co-design efforts, where performance and hardware efficiency are intertwined. Thus, across literature, to address online learning on-chip with SNNs, authors propose SNNs with various flavours for spike encoding, learning methods and neuron model, accompanied by a custom SNN processor. However, current solutions present a number of limitations:

1. **Spike encoding:** For spike encoding, most state-of-the-art still relies on rate-coding methods, despite it being less bio-plausible and less hardware efficient than temporal coding techniques [50], [51]. Moreover, many SNNs processors for online learning perform spike encoding off-chip and don't account for the incurred energy and latency overhead for the network. This issue is aggravated by other pre-processing steps applied on the inputs before feeding them into the spike encoding layer (e.g., image downscaling and conversion to grayscale). Overall, not accounting for the cost of these steps results in a less transparent benefit of implementing the proposed solutions in practical applications with commercial impact.
2. **Neuron models:** To increase accuracy, authors opt for more bio-plausible neuron models, such as Leaky-Integrate-and-Fire (LIF) or Izhikevich models [52]. These models enable better neuron behaviour modelling, however, compared to the simpler IF model, which is the least biologically plausible, they are more computational expensive, due to extra leakage modelling or increased number of parameters. They also require additional neuron states storing. The impact this has on energy consumption is exacerbated by the fact that, during inferencing, all fan-out neurons need to update their membrane potential according to their neuron model.
3. **Learning rule:** Current solutions based on the Spike Timing-Dependent Plasticity (STDP) rule (i.e., the conventional learning method for SNNs), while less computationally-complex than backpropagation-based online learning rules [1], still suffer from increased computational complexity due to exponential terms and high-precision multiplications involved. This can be particularly prohibitive for extreme Edge applications. Moreover, STDP needs to store spike timings for each synapse, resulting in high memory area overhead [53]. Another synaptic plasticity rule is Spike-driven Synaptic Plasticity (SDSP) [40], which only utilizes the pre-synaptic spike events for synaptic updates and uses only increment/decrement operations. However, similarly to STDP, SDSP requires a frequent number of updates, for each neuron where there is a spike event. This results in a variable number of neurons being updated with each new input, incurring a non-deterministic cost on inference.
4. **Synaptic weights and neuron states memory:** In most SNN applications, storing synaptic weights [2] and neuron states occupies more than 50 - 60% of the area and is responsible for more than 80% of the total energy consumption. Despite this, many state-of-the-art solutions use high-precision weights such as fixed or floating point values. To address this, authors have explored optimizing the weights precision to 1-bit [53], [44]. However, binary weights precision has resulted in poor accuracy for practical tasks such as digit classification, particularly in combination with stochastic learning [54], which is desirable for robust performance in dynamic, noise- and variation-prone environments.

A potential solution for exploring fast online learning/adaptation in classification tasks, with low energy costs for the Edge, is based on the colour-agnostic and noise-robust Ultra-Rapid Visual Cate-

[1] Note that the quality of STDP being less computationally-expensive than backpropagation-based online learning rules comes at the cost of less performance

[2] Note that storing layer weights is a bottleneck in all neural networks and not specific to SNNs

gorization (URVC) [55] that takes place in the mammalian visual cortex. In URVC, fast analysis and classification of images is attributed to neural codes being transmitted from the retinal ganglion cells to the orientation-selective cells in the visual cortex in the shortest time possible (i.e., with the first emitted spike) via the optic nerve [56]. This observation enforces a hardware-efficient temporal spike encoding scheme of visual stimuli, where a high intensity in the stimuli corresponds to a short latency for firing a spike [57], [50], [58]. Such principles have been applied across literature for object recognition by trying to replicate the fast temporal coding in the retina via a number of convolutions with edge filters followed by max-pooling operation [45], [59], [60], [61], which capture the highest orientation-wise intensity that leads to the aforementioned spike encoding. However, processing speed and energy consumption in these solutions is still limited for real-time, edge applications, mostly due to high-precision convolutional kernel weights. Additionally, the potential of efficient computation and learning with the spikes generated by these retina models is not fully leveraged for Edge applications.

Previous work has applied URVC from an extreme computational efficiency view, by simulating URVC in large networks of Integrate-and-Fire (IF) neurons with binary weights, that completely depress or potentiate their synapses based on the first pre-synaptic spike fired [8] and has been demonstrated on simple image recognition and classification tasks such as face identification [62] and natural scene recognition [63], [7], [61]. However, the merits of these proposals for extreme spike-efficiency leveraging in URVC have not been fully explored in terms of performance on more complex tasks such as MNIST, actual processing speed benefits on modern hardware platforms such as high-end FPGAs, or energy consumption for extreme edge processing. Consequently, motivated by the previously mentioned limitations in current solutions for online learning with SNNs on edge processors and inspired from the first layers of visual processing in the brain, we define the problem statement of this work as follows:

Can we leverage an extreme, spike-efficient model of URVC, composed of simple IF neurons, with 1-bit weights and stochastic synaptic updates, to explore the following limitations in Edge AI:

- *perform energy-efficient, fast and few-shots online learning and adaptation at the Edge, with minimum incurred overhead for inference*
- *maintain inference performance in the presence of data variation due to dynamic environments (e.g., rotations or noise)*

1.3. Project goals

Taking inspiration from URVC in the first layer of the biological visual system, the aim of this project is to provide a scalable (i.e., the solution can be easily extended to more neurons in the learning layer or various input resolutions), hardware efficient solution with negligible learning overhead. As such, we want to find the boundaries of learning locally, in a single trainable layer, with extreme spike-efficiency (i.e., one spike per neuron), leaving the possibility for future extensions to multiple layers. To this aim, we explore the benefits of using a SNN of IF neurons with stochastic plasticity, binary spikes and synaptic weights, coupled with a Time-to-First-Spike encoding scheme and lateral-inhibition among orientation-selective cells, for Edge AI applications. In particular, we want to explore online learning and adaptation for typical image classification tasks (e.g., face detection or digit classification) and we want to evaluate if such an algorithm has practical applications such as object detection in high-resolution videos (e.g., 4K pixels frames at 60 FPS). We define our main goals as follows:

1. Propose a solution for online learning and adaptation at the Edge that:
    - is energy-efficient
    - is fast
    - exhibits one or few-shot learning capability
    - incurs minimum energy and latency overhead for inference
2. Benchmark and evaluate the proposed solution with respect to:
    - scalability in terms of input resolution and learning layer size
    - dataset-agnosticism
    - robustness to input variations in dynamic environments
    - fast inference in high-resolution videos

1.4. State-of-the-art

As mentioned in section 1.3, the main goal of this project is to perform fast, energy efficient online learning and adaptation at the Edge, with minimum incurred overhead for inference. The remaining goals are secondary, as they should reflect the quality and extendability of the solution that achieves the main goal. For this reason, we consider the state-of-the-art in achieving online synaptic plasticity at the edge, with custom processors for brain-inspired SNNs. In literature, there are two main streams for demonstrating the efficiency of a particular SNN with online learning on-chip: one is based on unsupervised and local (i.e., synaptic updates only require information from neighbouring layers [37]) synaptic plasticity rules, while the other uses a supervised synaptic update approach, inspired from backpropagation-like learning.

For supervised SNNs, online learning is performed via a surrogate type of gradient descent: since spikes are non-differentiable [64], updating synaptic weights based on gradient descent is not possible in a direct manner [65]. Thus, with the aim of addressing the low-accuracy yield in bio-plausible synaptic plasticity rules such as STDP (spike-driven synaptic plasticity), authors have proposed various methods for gradient approximations, e.g., with respect to spike firing times [66] or membranes potentials [67], resulting in supervised BP-STDP (Backpropagation STDP) rules. However, apart from their non-locality, these methods need multi-precision weights for gradients, they require more training data, and they are highly-sensitive to any variation in hyper-parameters [68]. As our goal is to achieve local online learning in dynamic environments, we mainly focus on literature that implements unsupervised STDP rules, suitable for stochastic learning.

For unsupervised online learning at the edge, the are two main approaches: one implements variations of STDP or SDSP, where complex computations are optimized [69], [70], [71], [43], [72], [46], while the other proposes custom synaptic plasticity learning rules that are more hardware-friendly than STDP or SDSP [44]. Overall, the state-of-the-art SNN processors embed a given online learning rule, with some processors exhibiting a degree of programmability in terms of neuron models or synaptic weights precision [40]. The main limitation in state-of-the-art solutions is that learning incurs a high energy overhead on inference, a typical value being 7 - 10 times overhead. We also make a distinction between SNN processors meant to be used as IP modules, with an address-event-representation (AER) interface for input and output spikes, versus standalone SNN processors, that can be directly plugged-in to image sensors, without additional data processing off-chip. For the former, an accurate estimation of the overall energy consumption is less straightforward. Of particular interest are also works that involve stochastic plasticity with binary synaptic weights [44], [53], since they have the potential to address the memory bottleneck presented in section 1.2 and also robustness to dynamic environments. In the following paragraphs, we elaborate on some of the methods proposed in state-of-the-art literature and their main limitation with regard to our goal.

In [69], the authors propose a SNN processor based on a variation of STDP, where the exponential is replaced by a constant, and on an adaptive computing scheme that can switch between clock-driven STDP computation (i.e., at every clock-cycle) and event-driven STDP computation (i.e., only when a pre-synaptic neuron fires). Their method, however, uses 16bit floating point synapses and multiplications in order to reach satisfactory accuracy on a common image classification task like MNIST. Despite several hardware optimizations, and although rate spike encoding and classification are performed off-chip, due to complex computations in the LIF neuron model and proposed STDP rule, their solutions results in a low-throughput, high-energy SNN edge processor, where the learning consumes 10 times more energy than the inference.

The work in [70] optimizes complex multiplications in the standard STDP rule and LIF neuron model by using approximate multipliers and they reduce inference latency by employing parallel membrane updates on an FPGA platform. Spike encoding and classification are performed off-chip. While the learning stage incurs only 12% energy overhead on inference, their method is less efficient than other solutions that achieve comparable accuracy, due to employing 16-bit fixed point weights and activations. Several other works propose low-complexity variations of STDP for online learning on chip: [71], [43], [72], [46].

To obviate the need for extra spike timing buffers, the authors in [40] propose an ASIC SNN processor with programmable neuron model (i.e., LIF and 20 Izhikevich models) and a optimized online learning implementation of SDSP. Spike encoding and classifications are performed off-chip. They feature very efficient inference when the input spikes are converted using rank-order coding, and the

neurons are modelled using LIF. By contrast, inference consumes 27 times more energy when input spikes are encoded using rate coding. Nevertheless, for the rank-order coding, learning is 7 times more expensive than inference. A neuromorphic processor with binary-weights and stochastic SDSP online learning is proposed in [53], however, the solution can not be successfully extended to more complex datasets such as MNIST, without further exploration.

In [44], the authors propose a SNN with binary synapses and a custom online learning rule, complemented by a specific processor in 65nm ASIC node. Their processor is fixed to 400 excitatory neurons with 576 synapses each (suitable for images of 24x24 pixels). Spike encoding and classification is performed off-chip. To address the high cost associated with synaptic memory access and membrane updates during inferencing, they propose using a simple IF neuron model with 1-bit synapses. This results in an energy efficient inference, however, without taking into account the spike encoding cost and the classifier cost. To address the energy consumption incurred by a high number of updates during learning, they update only the most active neuron for each new incoming input and they propose a low-complexity learning rule based on pre-synaptic spike counting of rate-coding generated spike-trains. Using the spike-counts, they define a stochastic weight update rule as follows: firstly, they generate a stochastic probability for updating weights, which they compare with a random generated number between [0,1]. Secondly, they generate a random integer that they compare with the number of spike counts in the input. They combine the two steps to decide when to turn synaptic weights on or off. While the stochastic probability for weights updated is fixed during learning, the two random numbers are generated for all the synapses in a neuron (i.e., 576 times). This, combined with the fact that they need a number of counters equal to the number of input axons (i.e., 576) results in a online learning scheme that consumes 7 times more than the inference.

1.5. Contributions

In line with the problem statement of this work and the project goals, in this thesis we propose EON-1, an Edge ONline Learning SCNN (Spiking Convolutional Neural Network) processor with 1-bit synaptic weights, 1-spike per neuron and 1-neuron updated per input [3], that is inspired from the efficient spike processing in URVC. Our contributions are as follows:

- Algorithmic contributions:
  - propose an online learning and adaptation rule, using 1-bit synaptic weights and stochastic updates of IF neurons,
  - evaluate the quality of the solution in presence of data-variations in the input but also dataset-agnosticism
- Hardware contributions:
  - end-to-end system implementation of the proposed SNN, including spike encoder and classifier
  - minimize synaptic weights memory cost by half, in terms of area and energy, through optimizing the encoding of spikes and weights
  - optimize the proposed learning scheme in hardware by proposing a sequential learning process with synaptic updates in-place and that leverages the sparsity in input spikes and synaptic weights by skipping computations.
  - hardware measurements in both FPGA and GF22 ASIC node and energy cost figures for each architecture component

We emphasize that the main scientific contribution/innovation is an online learning rule suitable for edge deployment, that has less than 1% energy overhead for inference. To our knowledge, our solution incurs the least energy overhead for inference, compared to state-of-the-art solutions, showing a better efficiency by at least a factor of 10x.

[3] independent of the training layer size, the learning cost is fixed, as only one neuron is allowed to learn for each new input

1.6. Thesis outline

The remainder of this report is organized as follows: Chapter 2 presents a general background and related work on the report topic, Chapter 3 presents the learning algorithm methodology and the proposed SCNN. Chapter 4 presents the hardware implementation methodology for the proposed SCNN. Chapter 5 presents the experimental setup for evaluating and benchmarking the proposed method in software and hardware. Chapter 6 discusses the results obtained during the experiments in the previous chapter and benchmarks the solution against similar work. Chapter 7 presents conclusions of the research work and discusses future avenues.

2
Background and related work

2.1. Spiking Neural Networks

2.1.1. Introduction

For several decades, scientists have aimed to apply the learning capabilities of biological neural networks to attain intelligent learning in practical world tasks such as image recognition or speech processing. To that extent, the biological neuron has been abstracted to computational primitives, such as *synapses* and *neurons* and integrated in a artificial neuron counterpart, which, when replicated in a structured network, forms an Artificial Neural Network (ANN). A biological neuron and it's basic artificial counterpart are depicted in sub-figures A.) and B.) of Figure 2.1. A biological neuron's cell body, the *soma*, integrates synaptic potentials from pre-neurons. The integrated result stimulates the voltage of the neuron membrane, which, when passing a certain threshold voltage, it releases action potentials to other neuron cells. These action potentials are carried by a neuron's axon, which further broadcasts the electrical signals to all it's fan-out synapses. Synapses are contact structures between a single pre-synaptic neuron with a single post-synaptic neuron.

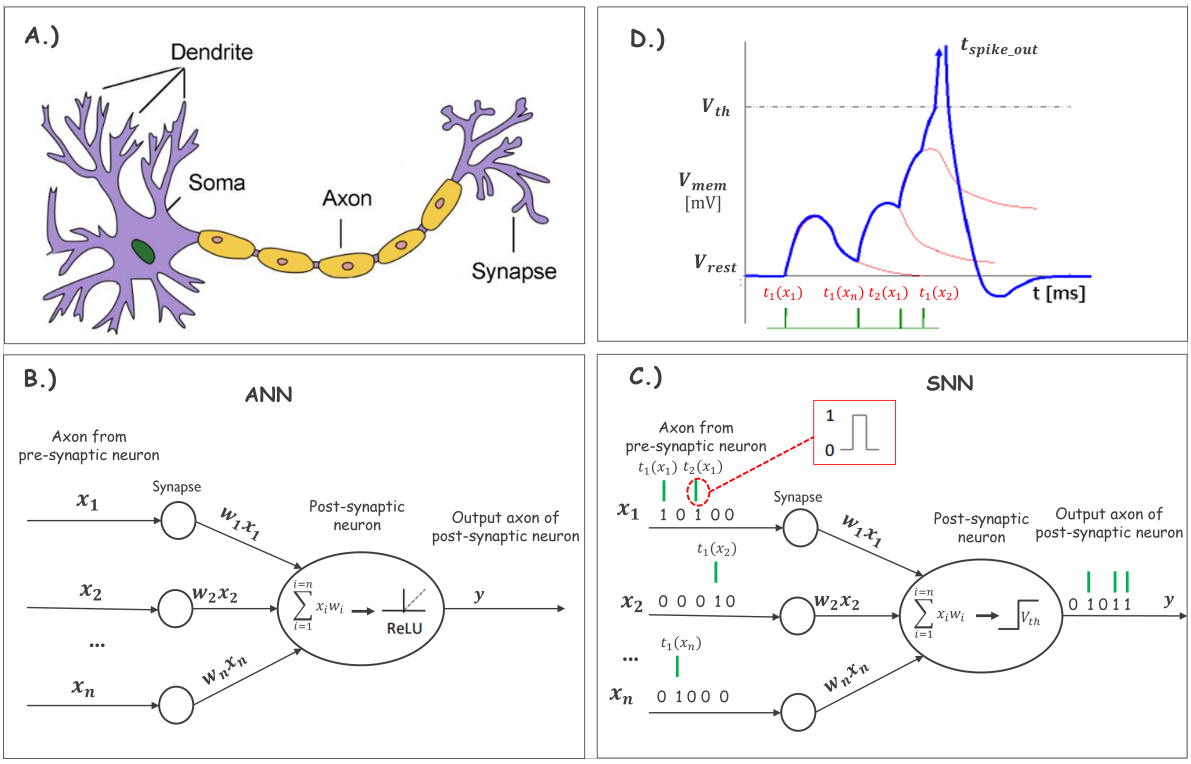


Figure 2.1: A.) Biological neuron. B.) Artificial neuron in a ANN. C.) Spiking neuron in a SNN. D.) Membrane potential of a spiking post-synaptic neuron as a function of time: incoming spikes from a pre-synaptic neuron are accumulated. When the membrane potential reaches a threshold V_{th}, the post-synaptic neuron fires a spike. Figure has been adapted from [1], [2] and [3].

Each synapse has an associated weight value that is activated by a magnitude equal to the input action potentials. In an artificial neuron counterpart, activations and synaptic weights are typically represented through floating point-numbers. The artificial neuron also integrates the incoming weighted inputs, and the result is non-linearized with an activation function, typically a Rectified Linear Unit (ReLU). The axon value at the output of a postsynaptic neuron in a ANN is given by Equation 2.1. Typically, a bias value b is added to the weighted sum performed by the neuron.

$$y = max(0, \sum_{i=1}^{i=n} w_i \times x_i + b) \tag{2.1}$$

Spiking Neural Networks (SNNs) are a third generation of neuron networks [48], in which neurons communicate through discrete spikes. Pre-synaptic neurons fire spikes that stimulate weighted synapses. Similarly to ANNs, synaptic weights can be represented through floating point numbers, however, by contrast to ANNs, the activation function of spiking neurons behaves more closely to biological neurons, such that the weighted sum activates a step function, which, based on a post-synaptic neuron's threshold potential, it outputs a value 0 or 1, i.e., it fires a discrete spike. The basic axon function in a spiking neuron is given by Equation 2.2. A visual representation of the spiking neuron and it's activation function are depicted in sub-figures C.) and D.) of Figure 2.1. In general, SNNs are composed of two layers of neurons, where the input layer encodes information to spikes and the output layer performs learning. SNNs come with several attributes such as the spiking neuron models, the type of information encoding to generate spikes for the input of the network, different synaptic plasticity rules (i.e., the method by which a neuron modulates it's synaptic weights). Some popular types of these attributes are briefly presented in the following subsections.

$$y = \begin{cases} 1, & \text{if } \sum_{i=1}^{i=n} w_i \times x_i \geq V_{th} \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

2.1.2. Information encoding to spikes

Typical tasks for neural networks are image classification or pattern recognition in input images. Input static images, for instance, are usually represented by a matrix of data-points named pixels, which indicate the contrast intensity, typically an integer between 0 and 255. However, SNNs only work with discrete spikes, and thus real input information such as pixels need to be converted to a form of spike code, in order to be processed by the SNN. There are two main spike encoding techniques used in literature: one is rate coding, and the other is temporal coding. We briefly describe the two coding schemes in the following paragraphs.

Rate coding is a spike encoding method based on the mean firing rate within a time window. The basic rate coding scheme is based on:

$$r = \frac{N_{spike}}{T}, \tag{2.3}$$

where N_{spike} represents the number of spikes fired within a stimulus time window, T. To convert pixels to discrete spikes, a common method is to use Poisson rate coding, where pixels are mapped to a firing rate that is proportional to the pixel's intensity [73], [3]. The probability of a pixel with intensity λ firing one spike in given time interval Δt is given by [3]:

$$p(N_{spike} = 1 \text{ during } \Delta t) = \lambda \Delta t (e^{-\lambda \Delta t}) \tag{2.4}$$

Temporal coding is a spike encoding scheme that uses the exact spike times to generate spikes. Multiple temporal coding schemes are used in literature, some of the most common being Time-To-First-Spike (TTFS) and Rank-Order-Coding [50]. TTFS is inspired from the biological visual system and relates the relative time of the first spike fired by a neuron cell to the amplitude of the stimuli intensity. We elaborate more on this scheme in the next section, as it is used as a fundament for the solution we propose. Rank-Order-Coding is based on the TTFS coding scheme, with the difference that it captures the order in which neurons across a population fire their first spike. This encoding scheme is depicted in Figure 2.2.

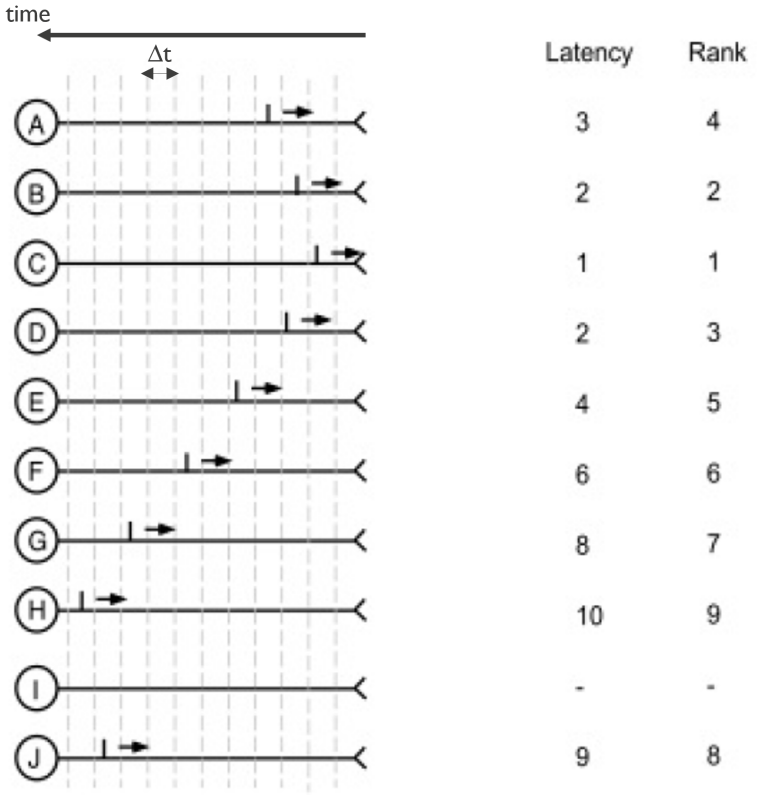


Figure 2.2: Rank-order-coding among a population of 10 neurons annotated A - J. Based on the order of the first spike fired every time window Δt, the rank-order code is C>B>D>A>E>F>G>J>H>I. Copyright: [4].

2.1.3. Neuron model

The spiking neuron models refer to the mathematical relation according to which a neuron updates it's membrane potential. Three of the most popular neuron models used in literature are Integrate-and-Fire (IF), Leaky-Integrate-and-Fire (LIF) and Izkievich model, all of which are approximations of the Hodgkin-Huxley (HH model). We briefly describe these models in the following paragraphs.

Hodgkin-Huxley (HH) model is a biologically accurate model, but also the most computationally expensive. The HH model is based on experiments on the giant axon of a squid [3], and it expresses that the action potential of the membrane is based on the current of ion channels, K^+ and Na^+, and the total synaptic current, as described in Equation 2.5:

$$C_m \frac{d\nu_m}{dt} = I_{ion}(t) + I_{syn}(t), \tag{2.5}$$

where C_m is the membrane capacitance in pF, ν_m is the membrane potential in mV and $I_{ion}(t))$ and $I_{syn}(t)$ are the total ion channels current and total synaptic current in pA, respectively.

Integrate-and-Fire (IF) is the most computationally efficient and simple neuron model [64], [3]. Compared to the HH model, it does not take into account the ion channel behaviour in the membrane potential integration. This is described by Equation 2.6

$$C_m \frac{d\nu_m}{dt} = I_{syn}(t), \tag{2.6}$$

Leaky-Integrate-and-Fire (LIF) is more biologically accurate than IF, by accounting for the leakage potential of the diffusion of ions in the absence of the cell's state of equilibrium. Compared to the HH model, it does not take into account the ion channel behaviour in the membrane potential integration. This is described by Equation 2.7:

$$C_m \frac{d\nu_m}{dt} = I_{syn}(t) - G_L(\nu_m - E_L), \tag{2.7}$$

where G_L represents the leak conductance and E_L is the leak potential [3]. When the membrane threshold ν_0 is passed, then the post-spike mebrane potential reset to ν_{reset} is given by:

$$\text{if } \nu_m \geq \nu_0, \text{ then } \nu_m \leftarrow \nu_{reset} \tag{2.8}$$

Izhikevich is a neuron model [52] that maintains the low-computational complexity of LIF, while being closer the biological plausibility of HH. Several neuron behaviours can be modeled based on the following two-dimensional system of differential equations, where a, b, c and d are constants that can model various behaviour based on [52], u is a ionic recovery variable and ν_t is the instantaneous threshold potential [3]:

$$C_m \frac{d\nu_m}{dt} = k(\nu_m - E_L)(\nu_m - \nu_t) - u + I_{syn}(t) \tag{2.9}$$

$$\frac{du(t)}{dt} = a(b(\nu_m - E_m) - u) \tag{2.10}$$

with post-spike membrane potential resetting based on:

$$\text{if } \nu_m \geq \nu_0 \text{ then } \begin{cases} \nu_m \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.11}$$

2.1.4. Learning in SNNs

As mentioned in section 1.4, there are two main learning approaches in SNNs: one is based on unsupervised and local (i.e., synaptic updates only require information from neighbouring layers [37]) synaptic plasticity rules, while the other uses a supervised synaptic update approach, inspired from backpropagation-like learning. As our goal is to achieve local online learning in dynamic environments, we only focus on the STDP rule, which is presented in the following paragraph. We mention that STDP can also be applied offline, in which weights are updated at the end of all forward passed on correlations between all stored synaptic times [64].

Spike-Time-Dependent-Plasticity (STDP) is an unsupervised, online learning rule in which synaptic weights are updated based on the order of the firing times of a pre-synaptic spike and a post-synaptic spike. When the post-synaptic spike is fired shortly after the pre-synaptic spike, weights are strenghtened, also caled Long-Term-Potentiation (LTP), and in the reverse situation, weights are weakened, also called Long-Term-Depression (LTD). The weight update in STDP is given by:

$$\Delta\omega = \begin{cases} A_+ exp(\frac{t_{pre} - t_{post}}{\tau_+}) \text{ if } t_{pre} \leq t_{post} \\ A_- exp(-\frac{t_{pre} - t_{post}}{\tau_-}) \text{ if } t_{pre} > t_{post} \end{cases}, \tag{2.12}$$

where $A_\pm$ are weight scaling factors and $\tau_\pm$ are time constants for LTP and LTD, respectively [3], [64]. Scaling factors $A_\pm$ are defined as:

$$\begin{cases} A_+(\omega) = \eta_+ exp(\omega_{init} - \omega) \\ A_-(\omega) = \eta_- exp(\omega - \omega_{init}) \end{cases}, \tag{2.13}$$

where $\eta_\pm$ are learning rates and ω_{init} refers to the initial weight value [3].

2.2. Retina models based on Spiking Convolutional Neural Networks (SCNNs)

2.2.1. Preliminary: Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are neural networks inspired from the hierarchical recognition in the mammalian visual system, commonly used for image classification and object detection tasks.

CNNs are composed of convolutional layers, followed by an activation function, and a sub-sampling or pooling layer. This structure is the equivalent of complex and simple cells neuron models in the visual pathway, that date back to the first observation of the receptive fields in the cat's visual cortex [74]. This structure is repeated several times, resulting in a hierarchical feature extraction, from simpler features to more complex features. Due to many such layers, CNNs are a subset of Deep Neural Networks (DNNs).

A typical CNN architecture is depicted in Figure 2.3. An input image in grayscale, with dimensions $M_0 \times M_0$ is fed into the network. A feature extraction process follows, for several iterations composed of two layers, a convolutional layer and a pooling layer. With each iteration, the complexity of the extracted features increases. We describe briefly the CNN steps and components.

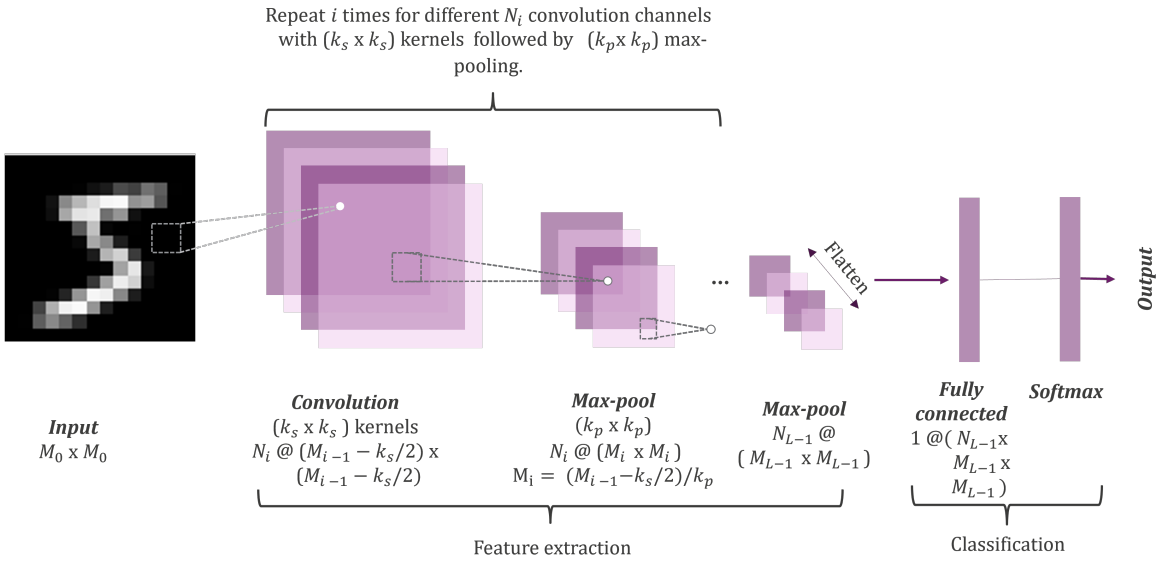


Figure 2.3: A typical CNN architecture. Adapted from [5], [6]

Convolutional layers consist of several two-dimensional (2D) convolutions between filter kernels with size $k_s \times k_s$ and each input channel. Typically, multiple convolution kernels are applied, which we denote by N_i. The kernels weights are learned during the training process. These layers are followed by an activation function, like ReLU in Equation 2.1. This step ensures the network is more robust to non-linearity in the features.

Pooling layers, also knows as sub-sampling layers, extract the most relevant features in a neighbourhood of pixels in the convolutional layer, hence reducing the dimensions of the convolution output.

Fully-connected layer is the last layer in a CNN corresponding to the one-dimensional flattened input of the last pooling layer. This layer is also followed by an activation function, for non-linearity.

Lastly, a **Softmax** layer is used for predicting the probability of each class, and the class with the highest probability is the output prediction.

2.2.2. Rapid Recognition based on Time-to-first-spike (TTFS)

In the biological visual system, it has been observed that Ultra-Rapid Visual Categorization (URVC) occurs at the visual cortex level within a very small time-window (e.g., under 150 ms) for patterns that are flashed for less than 20 ms in front of the retina [58]. Given that on the path from the retina to the visual cortex, information is carried through around 10 layers of neurons [7] that fire with a frequency of roughly 100Hz [75], Thorpe *et al.* have argued that at most one spike per neuron could be fired in such a short time-frame, ruling out the possibility of a rate-order encoding scheme [57]. Based on the assumption that ganglion cells in the retina act as intensity-to-latency converters [76] (Figure 2.4), such that they emit spikes in the order of the strength of the visual stimulation, Thorpe *et al.* have proposed a rank-order-encoding scheme [50], which is a form of temporal coding where the time-to-first-spike

(TTFS) is related to the highest contrast in a given stimuli. Their assumption caries biological plausibility, being experimentally proven for the salamander retina [77].

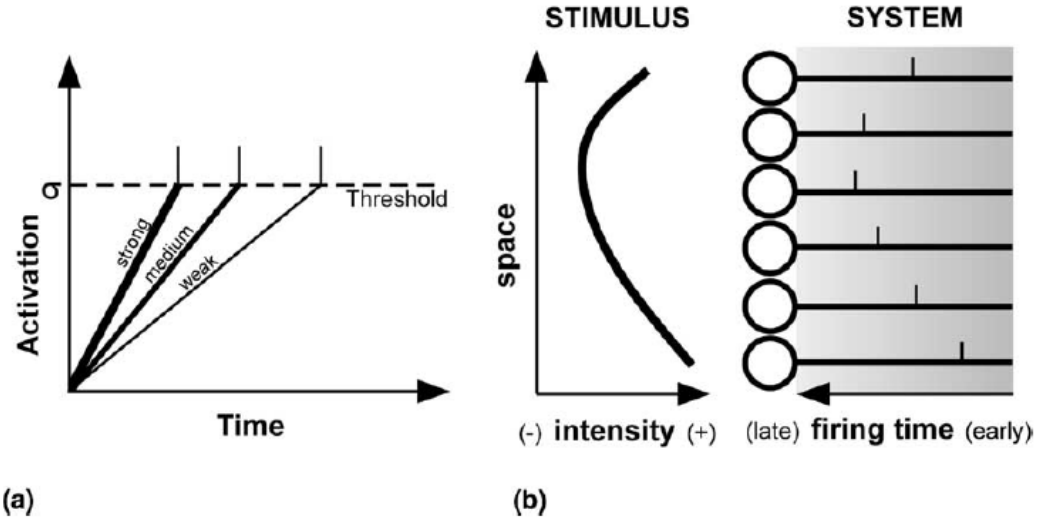


Figure 2.4: Depiction of intensity-to-latency conversion in the retinal ganglion cells: a) shows how stronger neuron activations result in earlier spikes. b) For a prototypical spatial stimulus, the neurons firing time based on the intensity at each pixel position is depicted. Figure used from [7] with the authors approval.

2.2.3. Intermezzo

This subsection presents a short intermezzo into how visual stimuli is roughly transmitted from the retina to the visual cortex. The retina is organized in three levels of neurons, namely photoreceptors, bipolar cells and ganglion cells. The flow of information is as follows [56]:

- The contrast of a visual stimuli is firstly sensed by the first-order neurons in the retina, rode and cone photoreceptors, which hyperpolarize when illuminated and depolarize when light sets off.
- Further-down the retinal organization, the neurotransmitters emitted by the photoreceptors are sensed by second-order neurons, ON- and OFF- bipolar cells, which behave as follows:
  - The OFF-bipolar cells respond to light offset or an increase in darkness, and thus they hyperpolarize when illuminated
  - The ON-bipolar cells depolarize when illuminated, which results in a sign inversion for the signal from the photoreceptors
- These ON- and OFF- signals are consequently collected by a subset of ganglion cells, that then send spikes to the visual cortex through the optic nerve.
- The first layer of the visual cortex encapsulates orientation-selective cells that, based on the information carried by ON- and OFF-signaling pathways from the ganglion cells, perform high-level feature outlining and object recognition.

2.2.4. Previous SCNN architectures simulating rapid recognition in the visual cortex

To model the URVC behaviour observed in the brain, Thorpe *et al.* [4], [61] have proposed a feed-forward SCNN that emulates the information processing from the retinal ganglion cells to the orientation-selective cells in the first layer of the visual cortex.

An early network architecture proposed by Thorpe *et al.* [8] for modelling URVC focuses on simulating a temporal 1-WTA (Winner-Takes-All) lateral inhibition scheme among eight orientation-selective cells. The basic two-layer architecture is depicted in Figure 2.5. The idea behind this mechanism is that only the orientation cell with the highest contrast fires a spike, and this spike is sufficient for high-level feature recognition in the layer that follows. This is artificially implemented for static images using a series of convolutions with edge filters, and a lateral maxpooling operation (channel-wise) to decide for which edge filter the convolution output is the highest, at each pixel point. This encoding corresponds to a N-of-M rank order encoding scheme with N = 1 and M = 8, i.e., only one spike out of eight is propagated [78].The authors have implemented and tested their ideas on a large network of asynchronous IF neurons, named SpikeNet [8]. The network has been demonstrated for a range of image classification and pattern recognition tasks such as face identification [62], classifying between faces and motorcycles [61] or pattern matching at various scales and in presence of invariance [63]. Rapid unsupervised recognition of patterns, based on the same ideas, after only 2-5 repetitions was demonstrated in [79]. SpikeNet has also been used commercially for identifying logos in sporting events such as Formula 1 [78]. Moreover, since through a sum of studies it has been shown that URVC in the brain seems to be achromatic and highly invariant to noise, contrast variation or small rotations of the input [4], [55], [56], [77], the creators of SpikeNet have demonstrated that such principles keep their validity for the artificial model they proposed [4].

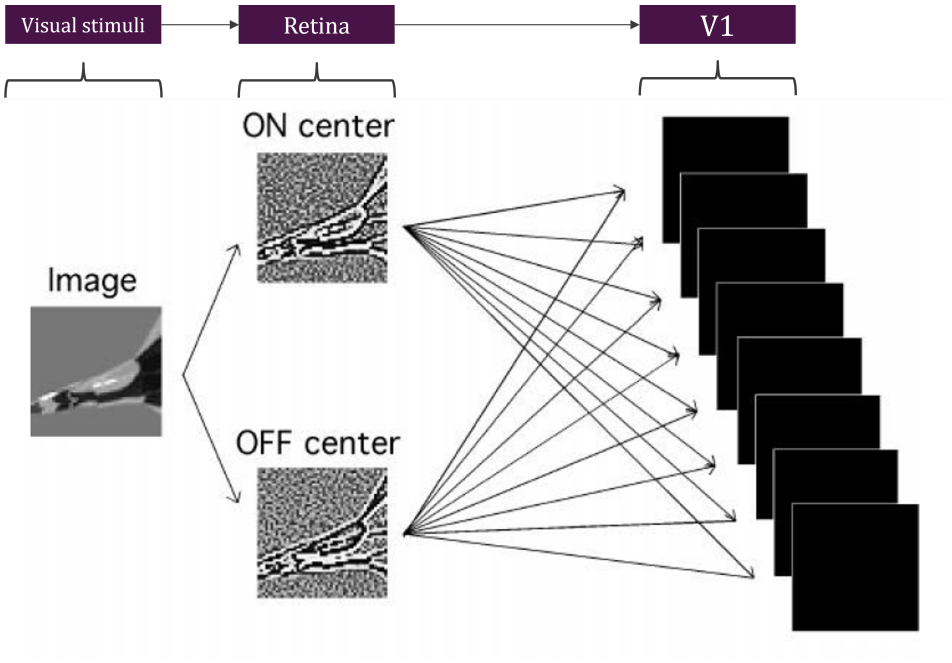


Figure 2.5: Two-layer feedforward architecture emulating the retina in [8]. Figure copyright: [9].

2.2.5. Learning in previously proposed SCNN for rapid recognition in the visual cortex

For learning based on such an encoding scheme, the authors in [8], [9] proposed a STDP rule where IF neurons act as rank-order decoders, such that their synaptic weights are driven to match the presented stimulus based on the earliest firing cells among a set of orientation-selective neurons. Thus, in their proposal, synaptic weights are basically templates of input images, however, thanks to the efficient encoding scheme based on URVC, these weights are able to infer variations of the patterns they have learned such as rotations up till $\pm 10^{\circ}$, contrast, luminance and noise variation [63]. Recognition is based on the spike fired by the IF neurons. One direct advantage of this scheme is that both spikes and synaptic weights can be represented with just one bit. This means that updating the membrane potential of neurons can be done with a bitwise AND operation followed by a popcount, which is very

efficient for hardware implementation. Another advantage of the template-matching neurons is that they can keep good inference accuracy even when only a small number of their synaptic weights are turned ON for instance, the authors propose keeping only 64 weights ON for input patterns with sizes around 30x30 points is sufficient to detect objects they have seen before, and their variations [78]. This results in very sparse synaptic weights, and thus have the potential for efficient weights-storage.

This work is further extended in [61] to embed STDP learning with learnable parameters. The learning rule is described in Equation 2.14. The network architecture consists of two sets of layers of complex and simple cells is proposed. The network is applied to simple binary classification between faces and motorcycles. The STDP rule proposed in [61] is extended in [80] to probabilistic STDP.

$$\Delta\omega = \begin{cases} A_+\omega(1-\omega), t_{post} - t_{pre} \geq 0 \\ -A_-\omega(1-\omega), \text{ otherwise} \end{cases}, \tag{2.14}$$

where $\omega \in (0, 1)$ and the amplification parameters are kept to $4/3$.

After the proposal of [61], several other SCNNs have been proposed based on URVC in the brain [59], some of which are suitable for edge learning, such as is the case of [45], which uses a supervised backpropagation-like learning in the spiking layers.

2.3. Considerations for hardware implementation

2.3.1. Algorithm-hardware co-optimization of 2D convolutions

Algorithm 1 2D_conv(img(X, Y, M), w(KX, KY, K))

Input data: A dataset with M images of $X \times Y$ elements and a set of K kernels with $KX \times KY$ weights.
Init: $conv\_out(X, Y, M, K) = 0$
for i = 0 to M - 1 **do** ▷ For each image in the dataset
  for x = 0 to X - 1 **do** ▷ For each row in the image
    for y = 0 to Y - 1 **do** ▷ For each column in the image
      for k = 0 to K - 1 **do** ▷ For each kernel channel
        for kx = -KX/2 to KX/2 **do** ▷ For each kernel row
          for ky = -KY/2 to KY/2 **do** ▷ For each kernel channel
            $conv\_out(x, y, i, k) \mathrel{+}= img(ix - kx, x - ky, i) * w(kx, ky, k)$
          end for
        end for
      end for
    end for
  end for
end for
return $conv\_out(X, Y, M, K)$

It is often desired, due to the high-level of utilization of the 2D convolution in modern applications, that the cost of performing it is minimized, while it's computation speed is enhanced. As convolution is the engine powering Deep Learning networks and modern Computer Vision application, there has been ample research and previous work on it's algorithm-hardware co-optimization and well-established techniques complemented by specialized accelerator chips [24], [25], [26] have been developed. Typically, in CNN applications, an input image from a given data-set is convolved with multiple kernels, resulting into a multi-channel convolved output. At an algorithmic level, this is achieved through a number of nested for-loops with multiply-accumulate. A pseudocode of this implementation in software is presented in algorithm 1.

Inherently, the 2D convolution is a multiply-and-accumulate operation, which exhibits increased parallelism that can be easily exploited at a hardware level. The first level of parallelism is deduced from the dependencies between the data involved in this operation, e.g., to compute one output, for a kernel with $K\_SZ \times K\_SZ$ size, K_SZ^2 dot products and one adddition with K_SZ^2 terms need to be

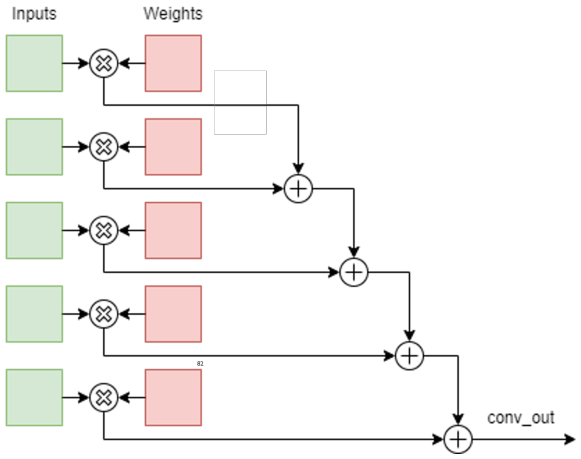


Figure 2.6: Multiply-and-accumulate pipeline for a convolution window flattened to 1D. For simplicity, a window with 5 elements is depicted. The convolution operation needs 5 pointwise multiplications and an acumulator that needs five stages for adding the 5 terms

performed. Finally, the additions are accumulated to obtain the convolved result. A simple such pipeline is depicted in Figure 2.6. Provided that all inputs and weights values can be read at once (concept which also introduces the next level of parallelism), all multiplications can be performed in parallel, however each addition is dependent on a previous intermediate result, for which reason the final accumulation needs to be performed sequentially. Nevertheless, in modern hardware fabrics, which incorporate standard cells with minimum input-output delay, it is possible to ”hide” the intermediate additions delay within one-clock cycle, at the cost of increased computation resources.

The second level of parallelism comes from how the data is made available for the computation unit. From here, several optimization techniques derive, such as temporal and/or spatial unrolling of the nested for-loops in algorithm 1. These work as follows:

- **Spatial unrolling (parallelization) -** refers to performing the computations in the body of a for-loop in parallel, in one clock cycle. For instance, in algorithm 1, the loop at line 6 can be fully parallelized, i.e., all K channels can be computed at once.
- **Temporal unrolling (data-reuse or stationarity) -** part of the data is kept stationary (e.g., the weights) and part of the data becomes available sequentially at every clock cycle. In terms of Deep Learning accelerators, two types of data-reuse can be defined in terms of specific stationarity:
  - inputs-stationary - reduces inputs data movement to-and-from memory by keeping them stationary, while weights are becoming available in parts at each time-step.
  - weights-stationary - reduces weights data movement to-and-from memory by keeping them stationary, while inputs are becoming available sequentially.

A typical digital hardware implementation of a 2D convolution is composed of memory elements (e.g., SRAM, DRAM or register files for on-chip memory and possibly extra off-chip memory) and a datapath of Processing Elements, PEs, as depicted in Figure 2.7. The datapath of PEs is also referred to in literature as a systolic array [81]. Systolic arrays are networks of interconnected processing elements (PEs) that rhythmically produce and pass data through the system, similar to the blood flow through the heart (hence, the name *systolic*). They present the advantage of local interconnections, modularity, pipelinability and concurrency, which renders them excelent candidates for VLSI implementations, targeting ASIC or FPGA platforms for their realization. Systolic array-based architectures can be employed in a wide range of applications whose functions can be divided in repetitive components and that exhibit recursivity in computing the final output (e.g., the multiply-accumulate operation can be divided into ”multiply” components, that can be accumulated *recursively* to form the final output).A famous hardware DNN accelerator employing systolic arrays for 2D convolution is Google’s TPU [25]. Since memory reads consume the most energy, maximizing data-reuse in the PE datapath is enabled by various combinations of temporal and spatial unrolling.

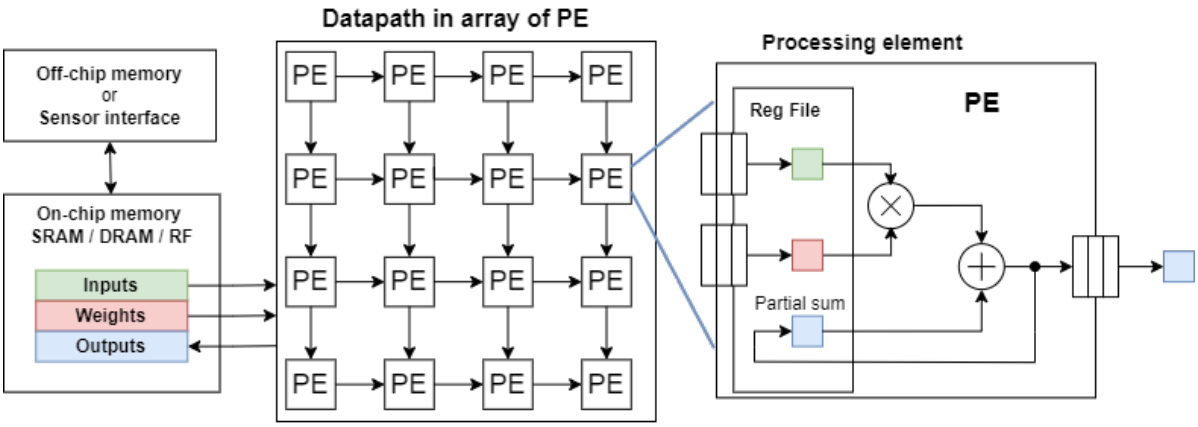


Figure 2.7: Typical hardware implementation flow for convolution via systolic array composed of a datapath of PEs. Left: data is loaded and stored to-and-from on-chip memory, which can be in turn communicating with a higher-density off-chip memory or directly to sensors interfaces. Middle: a datapath of PEs array. Right: A basic PE architecture.

Complementary to these hardware optimizations, there is also the avenue of reducing data precision and quantization of weights and inputs, e.g., weights that are floating point numbers are quantized such that a reduced precision can be used in hardware without significant performance loss [27].

2.3.2. Linear-Feedback-Shift-Register

Generating pseudo-random numbers is used in a wide range of application, some of which are Cryptography, Digital Signal Processing or Machine Learning Modelling. An efficient hardware realization of pseudo-random number generators (PRNG) is using a Linear-Feedback Shift Register (LFSR). LFSR's are implementing a primitive polynomial linear functions which ensure the LSFR will produce a maximum-length 2^{LFSR_bits} random sequence (i.e., the sequence is deterministic and repeats every 2^{LFSR_bits} clock cycles. This is achieved by a simple shift-register and by placing taps at bit positions corresponding to the degrees of the terms in the feedback polynomial. For illustration, the 4-bit LFSR in Figure 2.8 implements the primitive polynomial in Equation 2.15. Tap bits for LFSRs up to 168 bits are compiled in [82].

$$x^4 + x^3 + 1 = 0 \tag{2.15}$$

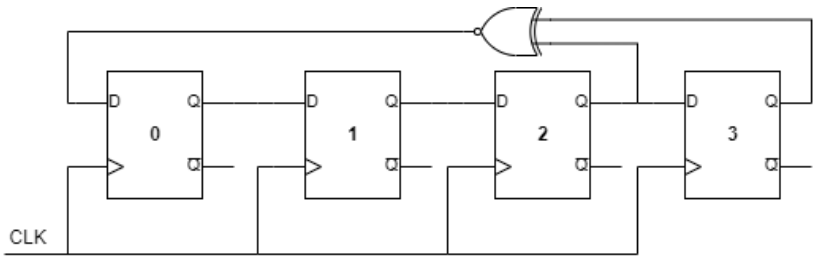


Figure 2.8: A 4-bit LFSR with taps on bits 2 and 3

2.3.3. Memory Blocks

In this subsection we briefly present the RAM choices for the synaptic memory and neuron states in the target FPGA platform and ASIC node, respectively.

The target FPGA platform used in this project is a Virtex UltraScale+ HBM VCU128 FPGA from AMD. This platform features various choices for on-chip memory: 70 Mb of Block RAM (BRAM), 270

Mb of Ultra-RAM (URAM), 36.7 Mb of distributed RAM but also 8GB of High-Bandwidth-Memory (HBM). Each of these blocks can be configured as follows:

- ***Distributed RAM*** is dedicated LUTRAM with configurable DEPTH (16 - 65k) and WIDTH (1 - 1024). It also supports various memory types, e.g., Single Port, Simple Dual Port and Dual Port. It comes with a native interface.
- ***BRAM***: There are 2016 units available with a block size of 36Kb. The default depth is 1024 words and the width is 36 bits, however the bit-width is reconfigurable and the tool can infer the amount of basic blocks needed for synthesis. BRAM can be configured to work as a Single Port, Simple Dual Port RAM or True Dual Port RAM (i.e., different addresses for read and write can be provided). The BRAM can be used with a native interface or an AXI interface.
- ***URAM***: URAM is a higher density on-chip memory, with a fixed block size of 288Kb. Each block has a fixed width of 72 bits and depth of 4096. It supports the same configurations as BRAM.
- ***HBM*** is a 3D-stacked memory of multiple DRAM Dies, placed on the same interposer with the FPGA die. The HBM used in this FPGA platform has 8GB of memory and can be accessed via an AXI-3 interface. The memory has 32 channels with an independent AXI slave available for addressing each of them. It is possible to access all AXI channels in parallel, resulting in a width of 8192 bits and depth of 2^{23} words.

The ASIC node used in this project is GF22FDX. For this node, we use the single port SRAM memory IPs with a depth of 8192 bits and a width of 32 bits.

3
Methodology - Algorithms

In this section, we describe the the bio-inspired online learning mechanism and the Spiking Convolutional Neural Network architecture used. The network architecture has one trainable and scalable layer of simple integrate-and-fire (I&F) neurons that embed the proposed local learning rule. All synapses and activations are binary and for learning, only one neuron is allowed to become selective to an input pattern.

3.1. Proposed SCNN architecture

Inspired from [9] and [61], the core feed-forward architecture used for pattern recognition is a Spiking Convolutional Neural Network (SCNN) with three layers. A visual representation of the network architecture is depicted in Figure 3.1. In line with the observation that the cells in the retina are color-agnostic [4], the network's input is a grayscale image where the contrast level is in the range $(0, 255)$, where 0 denotes the lowest intensity —black and 255 the highest intensity —white. Note that the input to the network can also be a coloured image which can subsequently be converted to grayscale levels. Additional downsizing of the input can be performed to match the number of synapses available per neuron in the trainable layer.

The first two layers of the architecture extract the orientation preference at each data point, and perform lateral inhibition among the eight orientation cells. This is done via four 5×5 convolution kernels, followed by a lateral max-pooling. The output of the Lateral Inhibition layer is mapped to 8-bit spike trains, while the output layer, which is the only trainable layer in the architecture, updates each neuron's state and triggers synaptic plasticity during the learning phase.

The core architecture can be used as stand-alone in an unsupervised manner, or can be adjusted for classification tasks, by plugging it to a classification layer and performing supervised clustering of multiple unsupervised networks.

Table 3.1: Parameters for I&F Neuron layers and synaptic plasticity

Name	**Description**
$N\_neurons$	Number of neurons
W	Number of active synaptic connections per neuron
N	Maximum number of spikes in the spike vector
M	Maximum number of spikes before lateral inhibition
T_learn	Learning threshold
T_learn0	Initial learning threshold
T_fire	Firing threshold
T_fire_rate	Firing threshold rate of learning threshold
$swap\_rate$	Rate or ratio of ineffective synaptic connections which will SWAP in each learning phase
K	Maximum number of neurons which are allowed to learn an input spike vector

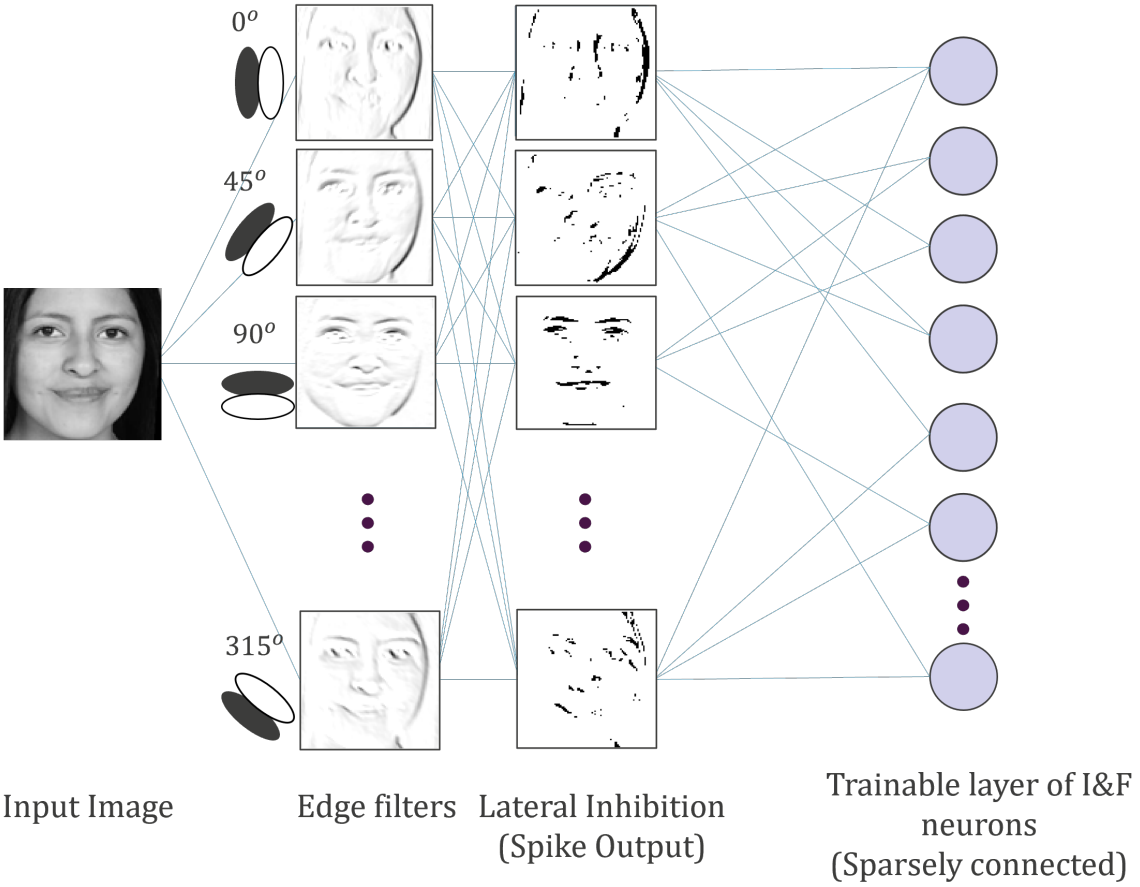


Figure 3.1: The SCNN architecture for this work. It includes a layer of edge-filtering convolutions, a lateral inhibition layer, and a layer of fully connected neurons equipped with binary STDP training.

3.2. Spike encoding

Inspired from the biological retinal ganglion cells (see subsection 2.2.2), the input layer of the SCNN maps the intensity levels of each pixel representation to orientation selective ON- and -OFF receptive fields. This is achieved by means of two-dimensional convolution with four kernels. We consider here only eight orientation angles, starting from $0°$ to $315°$, however, future experiments can consider using a wider range of angles. We employ 5×5 kernels with fixed weights (Figure 3.2), which resemble Gabor filters and are based on the work in [61]. Gabor filters are bandpass filters frequently used in image processing for tasks such as edge detection and feature extraction and are considered to approximate orientation selectivity in a bio-plausible manner, similar to cells in the visual cortex [83]. The convolution outputs are subsequently categorized under ON- and OFF- receptive fields based on the sign of the result at each pixel point: a negative sign corresponds to an OFF-center cell sensing a decrease in light, while a positive sign corresponds to an ON-center being depolarized at sensing an increase in light. By means of lateral max-pooling, the maximum absolute value from all orientations is picked channel-wise. Only the winning orientation propagates a spike, inhibiting all other orientation cells at each (X, Y) pixel location. Finding the winning kernel for one data-point is depicted in Figure 3.3. The result is a highly sparse binary spiking output that can be compressed into a spike vector, as shown in Fig.3.4.

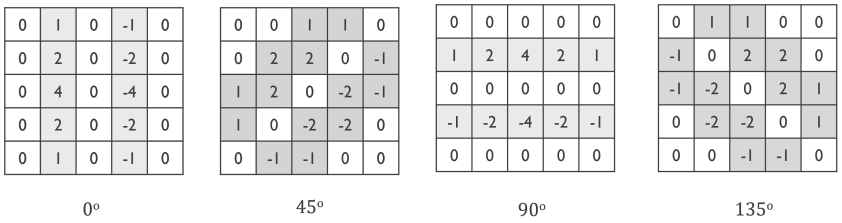


Figure 3.2: The 5×5 convolution kernels used in this work

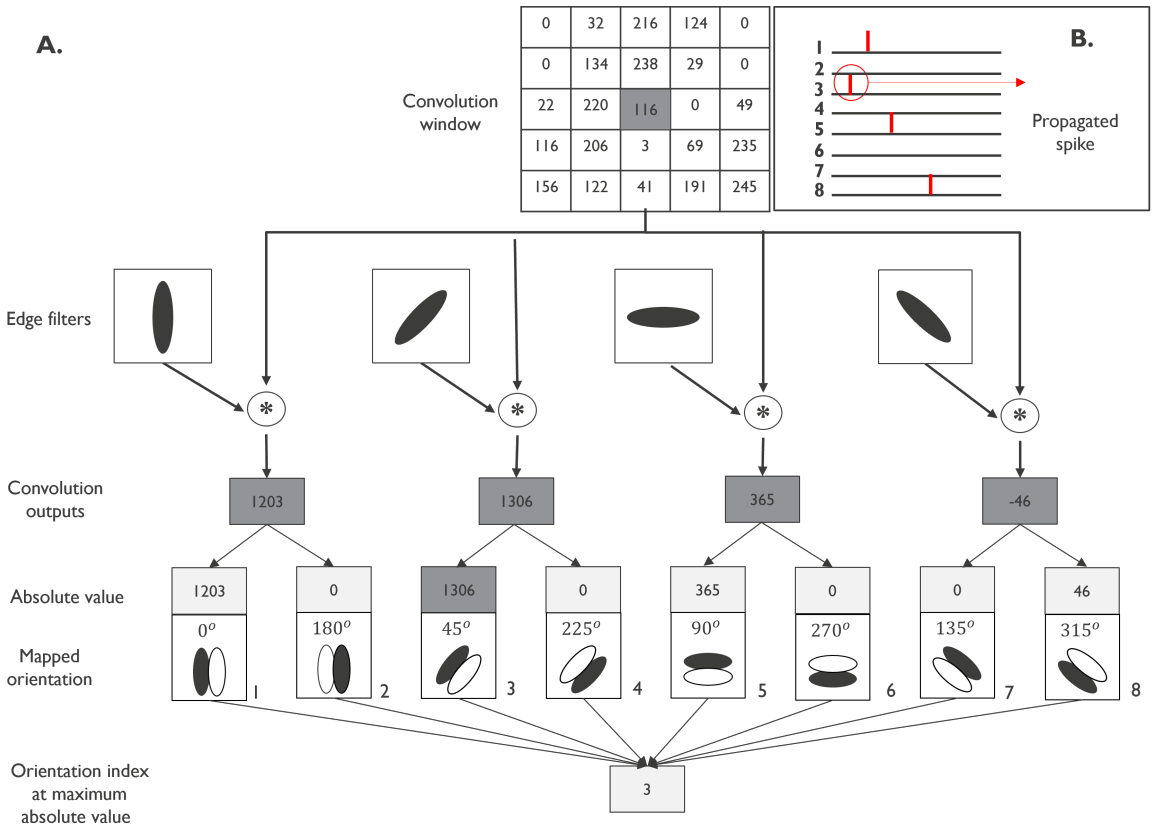


Figure 3.3: Depiction of rank-order-encoding method for $N = 1$ out of $M = 8$. **A.** A window of pixels is convolved with four edge filters, and the result is mapped to a orientation index based on the sign of the convolution output and the maximum convolution in absolute value. **B.** The neuron cell corresponding to orientation at index 3 spikes first.

3.3. Inference: neuron and synaptic model

After spike encoding, the output of the Lateral Inhibition layer (i.e., a spike vector) is then fully connected to I&F neurons with M synaptic connections in the third layer. Each neuron is characterized by two internal thresholds, related to their membrane potential, V_mem: a learning threshold, T_learn and a firing threshold, T_fire. The I&F neuron membrane potential is updated as in Equation 3.1.

$$V\_mem = \sum_{i=0}^{i=M-1} s[i] \times w[i] \tag{3.1}$$

The synaptic weights of the I&F neurons in our algorithm have three constraints:

1. The synaptic weights are binary, which means they are either connected (active) or disconnected.
2. All neuron has the exact same amount of active synapse (W in Table 3.1).

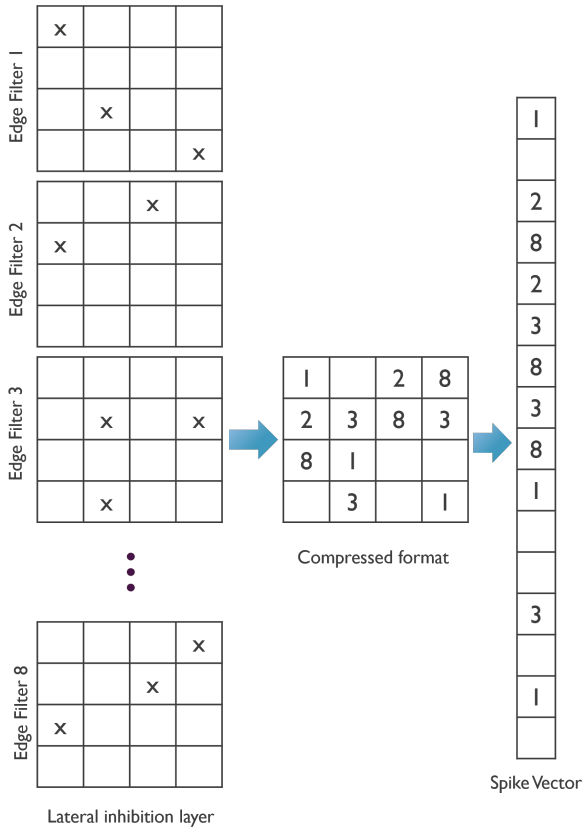


Figure 3.4: The lateral inhibition layer produces binary spikes that can be compressed into a spike vector. Each element of the vector corresponds to the source edge filter that generated the spike.

3. A neuron cannot connect to a unique (X, Y) location of more than one edge filter.

These constraints result in significantly sparse active synaptic weights, which can be compressed in the same format as the spike vector. Throughout this work we choose $W = 64$ thus for a scheme with $M = 8192$ and $W = 64$, a sparsity higher than 99% is present in the neuron's synaptic weights. Fig.3.5 illustrates an example of four weight vectors where each of them has exactly four active synapses ($W = 4$).

When the element of the weight vector matches with the element inside the spike vector, the corresponding neurons integrate the incoming spikes. Therefore, a neuron fires if the number of matching elements exceeds the firing threshold. Each neuron can be trained to recognize a specific feature. Extending this network is possible by adding more I&F neuron layers or increasing the number of I&F neurons in one layer. During inference, all neurons in the last layer are updated for an incoming input.

2		**1**			**3**		**1**	**8**	Spike Vector
2	7			6		7			Weight Vector 1
	8	**1**	3	5					Weight Vector 2
2				5	**3**	2			Weight Vector 3
	1		4		4	8			Weight Vector 4

Figure 3.5: Example of a spike vector and four weight vectors. Each weight vector contains four active synaptic connections. Bold elements in the weight vectors are the one that matches the spike vector.

If at least a neuron reaches it's firing threshold, that indicates the network remembers this pattern from a previous stimulation.

3.4. Learning rule

As mentioned, the binary weights of I&F neurons can be trained. Our training algorithm is a variant of STDP. It is unsupervised and applies locally and independently to each neuron. However, unlike conventional STDP algorithms, the weight change is not gradual, as we use binary weights during the training. Since our vision in designing this system is to perform on-device learning, the previously mentioned constraints on the weight vectors are valid for both training and inference phases and limit the freedom of the learning algorithm. Nevertheless, these features make our algorithm a hardware-friendly option for neuromorphic processing systems.

It is important to mention that the weight sparsity in our algorithm is inherent, meaning that it is not a result of the training process, but rather it is built into the algorithm itself. Therefore, the network's connections are sparse right from the start. This fixed structured sparsity can be utilized to enhance the efficiency of our hardware, as explained later.

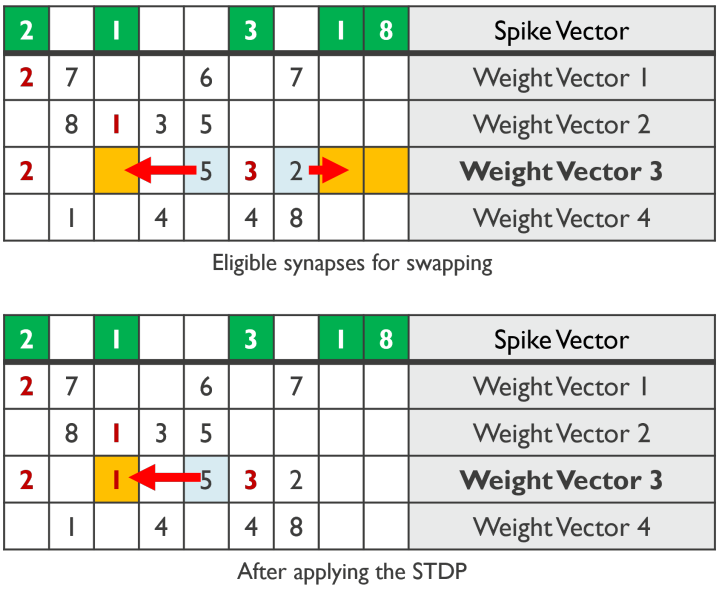


Figure 3.6: Example of the binary STDP algorithm. The learning algorithm applies to neurons whose membrane potential reaches a pre-defined threshold (neuron 3 here).

In a conventional STDP, the weights for synapses without a pre-synaptic firing would gradually reduce (Long Term Depression, LTD) while the weights for synapses with a pre-synaptic activity would gradually increase (Long Term Potentiation, LTP). The gradual change of weights allows a neuron to slowly adapt to new data patterns without forgetting the already learned patterns. It is possible to change the learning rate by increasing/decreasing the weight change factor. However, when using binary weights in our STDP rule, it is impossible to change the weight values gradually. Instead, we must apply an abrupt disconnection from the inactive pre-synaptic neuron for LTD and a full connection to the active (fired) pre-synaptic neuron for LTP.

Figure 3.6 illustrates our STDP rule. The learning process starts with assigning W random ON synaptic weights to each neuron. In the following paragraphs, we explain the STDP rule in detail with the example used in this figure.

Step 1: Inference of spike vector and integration into the membrane potential: In the first step, we integrate the input spike vector to all the neurons by counting the number of matches between the spike vector and the corresponding weight vectors. In Figure 3.6 (top), the membrane potentials of neurons will be 1, 1, 2 and 0 respectively. Assuming the threshold of 2, the membrane potential of neurons 3 will reach the threshold and is eligible to learn the input pattern.

Step 2: All the neuron that passes the threshold will go through a random selection process for learning: It is possible for multiple neurons to reach their learning threshold for one input spike vector (not the case for our example in Figure 3.6). To account for this, we randomly select K neurons from all eligible neurons (K in Table 3.1), and only those neurons can learn the input spike vector.

Therefore, K affects the learning rate. If the total number of neurons who passed the threshold is less than K, all of them will be selected. While K is programmable, we use a $K = 1$ for this implementation, which results in a very efficient learning rule, as only one neuron needs to be updated.

Step 3: Swapping ineffective weights towards the ineffective input spikes: For each selected neuron, we need to select a few active synapses that are not connected to the active input spike vector elements (called ineffective synapses). Similarly, we randomly select the same number of ineffective spikes from the spike vector. Finally, we swap the selected ineffective synapses toward ineffective spikes. In Figure 3.6 (top), neuron 3 has two ineffective synapses and three ineffective spikes. In Figure 3.6 (bottom), we randomly selected one of the eligible synapses and swapped it toward one of the ineffective spikes. After applying the STDP rule, the weight vector of neuron three matches even more with the input vector and will reach a higher membrane potential the next time the same input pattern appears. The number of swapping synapses is dynamic and can be calculated based on $Swap\_Rate\times$ the number of ineffective synapses. $Swap\_Rate$ is a parameter in Table 3.1 which defines the learning speed of each neuron. Swapping the position of active synapses allows us to keep the total number of connections unchanged. Therefore, our STDP rule maintains a constant parameter for the total number of active synapses per neuron (W in Table 3.1).

Step 4: Increasing the learning threshold: Another important factor in STDP is ensuring that a neuron remains selective to a specific pattern it learns and does not react and learn other orthogonal patterns. Therefore, we increase the neuron's threshold after each learning phase with the amount of weight swapping. This mechanism is inspired by the homeostasis observed in biological neurons [84]. As a neuron learns a pattern, it becomes less elastic and unable to learn new patterns. However, increasing the threshold can also make the neuron less active during inference, which is undesirable. To address this issue, we have decoupled the learning and firing threshold. The learning threshold (T_{Learn} in Table 3.1) is initialized with a low value and gradually increases for each neuron. Initially, when a neuron has not been trained, it is not allowed to fire ($T_{Fire} = \infty$ in Table 3.1). After the first learning phase, the firing threshold becomes a factor of the learning threshold. In such a neural network, a young neuron is initially very plastic, with a high tendency to match with a repeating input pattern, while an older neuron, which already learned several times, remains reluctant to learn a new pattern of input spikes.

.

3.5. Supervised clustering for classification

It is possible to use the principles presented here for classification tasks, by means of supervised clustering: we allocate a number of C clusters of I&F neurons in the trainable layer, where C is the number of classes, and during training we use the input label as a cluster selector, such that only the neurons in that cluster are allowed to learn that input pattern, based on the binary STDP rule presented. Each cluster of neurons learns in an unsupervised manner.

For measuring the performance during inference, we propose a very light-weight classifier layer, with a number of neurons equal to the number of classes. Each neuron in the classification layer computes the total activation of firing neurons in a cluster. The output predicted class is given by the index of the maximum neuron activation in the classification layer.

4

Methodology - Hardware architecture and implementation

In this section we present the hardware architecture of the SCNN processor embedding online learning and the design process for each component.

4.1. Proposed hardware architecture

The top level hardware architecture that implements the neural network in Figure 3.1 is depicted in Figure 4.1. The main data-flow between the blocks is represented through the relational arrows in the figure, where each data channel is complemented by two 1-bit handshake signals, `data_valid` and `data_ready`. The implemented hardware architecture is end-to-end compatible with the software implementation.

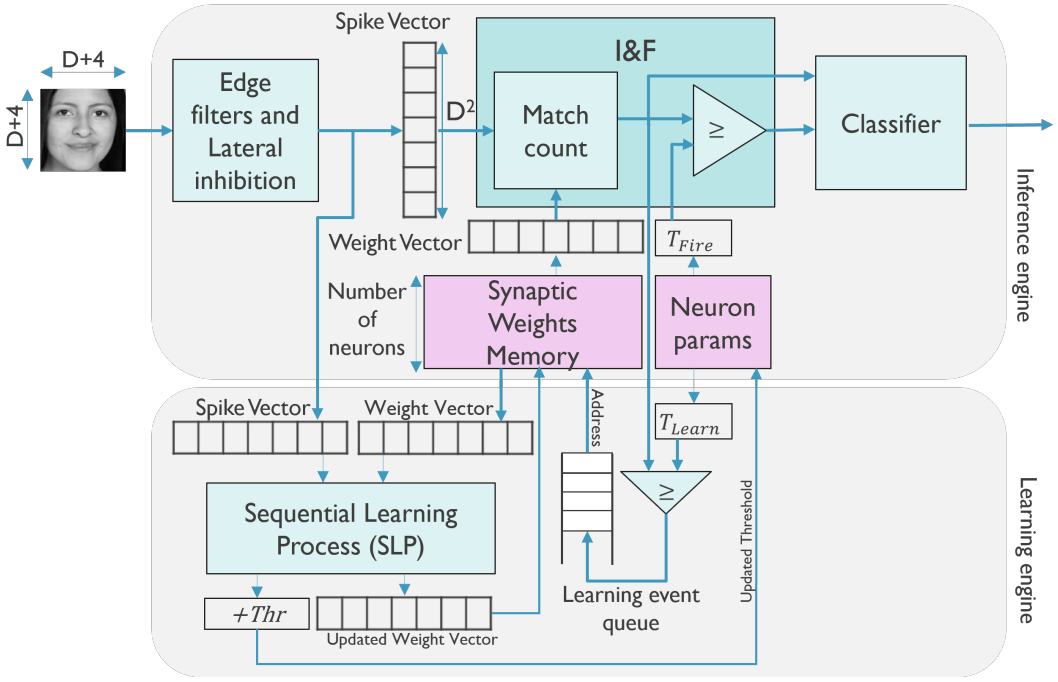


Figure 4.1: Proposed hardware architecture for the SCNN processor.

In brief, the overall hardware architecture has four main computing units, i.e., the Edge filters and Lateral inhibition unit, the I&F unit, a Sequential Learning Process (SLP) unit and a Classifier unit. The main memory blocks in the top level are the Synaptic weights memory and the Neuron States Memory. The latter stores the learning thresholds for each neuron. The firing threshold is computed as a rate of the learning threshold. The firing threshold rate is common to all neurons and do not need to be stored separately for each neuron. Not depicted in the diagram are additional buffers for encoded spikes. The entire training and inference pipeline is orchestrated by an FSM based controller that sends and receives control signals to all units. The main functionality of each block is described:

- ***Edge filters and Lateral inhibition*** unit corresponds to the first two layers of the network architecture in Figure 3.1, i.e., it performs 2D convolution on the input pixels, max-pooling of the four kernel channels and mapping the maximum to a spike vector. The convolution kernels are statically reconfigurable. For simplicity, we denote here the input size by $(D+4)\times(D+4)$, where D is the convolution output size, considering zero-padding.
- ***I&F*** unit updates the membrane potential of each neuron and compares it to a firing threshold. The outputs of the unit are the membrane potential and a logic signal depicting the firing state. It embeds a match unit and a comparator.
- ***SLP*** unit performs online on-device learning and adaptation. During inference, the SLP unit is disabled, unless activated from the Control FSM for online adaptation.
- ***Classifier*** unit is a simple module consisting of 7-bit counters (e.g., for adding membrane potentials) and a comparator that picks the maximum counted value. The classifier works in a pipelined fashion with the I&F unit, outputting the prediction one clock cycle later than the last updated neuron. In the absence of supervised clustering for classification, the classifier unit can be disabled, and the I&F outputs are bypassed to the output interface of the SCNN processor.
- ***Synaptic Weights Memory and Neuron parameters memory:*** The memory blocks shown in Figure 4.1 store the weight vectors, and the learning and thresholds of each neuron. The width of the weight memory is the same as that of the spike vector. The memory used in this study is created using either on-chip SRAM memory or on-chip BRAM/URAM or HBM memory technologies, depending on the number of neurons and bandwidth requirements. Weight memory is either a true dual-port memory or a simple dual port, depending on the memory technology and system performance requirements. One read-only port is used for inference (requires a high bandwidth port) and another independent read/write port can be used for on-device learning.

The two main paths in the processor are the Inference engine and the Learning engine, which can either operate independently or at the same time (i.e., for adaptation).

4.1.1. Enhancement of system performance

Since our vision is to have fast learning and inference, in one trainable layer with many simple I&F neurons, and given that the learning cost is fixed to one neuron per input, where finding the candidate neuron depends on the previous inference operation, the main bottleneck in our solution is the Inference phase, which consists of many memory read accesses and neuron updates. In Figure 4.2 we depict an inference model when M inputs are available per clock cycle and N neurons need to be loaded for each inference operation. We assume a blue box represents one time step, e.g, one clock cycle. For each input, there are N I&F operations that need to be performed.

The number of available M inputs (i.e., spike vectors) is dependant on the input sensor bandwidth, the speed at which inputs are converted to spikes and also the resource limitations for storing the inputs on-chip. Maximizing M is important for high-resolution frames that are segmented into small fields of visions (FoVs), where each field of vision is considered one input which is subsequently fed into the Inference engine. This case will be benchmarked and evaluated in Chapter 5.

When the neuron update logic is very efficient, minimizing the inference latency will be limited mainly by memory access (i.e., memory bottleneck). We note, however, that in a system where the memory bottleneck in reading the trained model could be alleviated or obviated, the input bandwidth can then easily become a bottleneck, if encoding inputs to spikes is slower than reading weights from memory.

To accelerate the inference time, in Figure 4.2, there are two-main approaches, one is to keep the inputs stationary and spread reading the neuron weights over multiple time steps, while the other is to keep weights stationary and apply multiple inputs concurrently to the available weights. A combination of inputs-stationary and weights-stationary is also possible, for best results.

In this work we evaluate an input-stationary approach, and focus on enhancing the system's performance in terms of loading the trained weights in less time steps. Our motivation for evaluating the inputs-stationary approach is that it can be easily extended to more inputs being applied in parallel (i.e., combining inputs-stationary with weights-stationary), once the predominant cost of loading weights model is characterized. To this extent, to enhance the system performance we can utilize multiple I&F processing units (P) that run in parallel. By employing P processing units (as shown in Fig.4.3), we

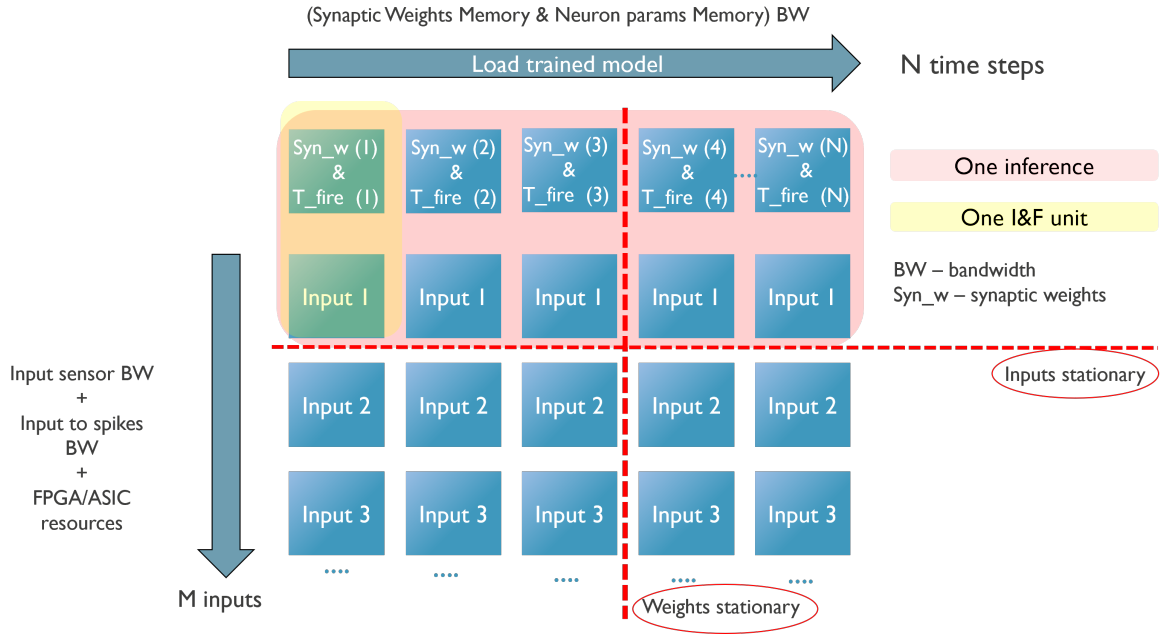


Figure 4.2: Inference array for M inputs and N neurons in the trained model.

can process one input spike vector P times faster. The number of clock cycles required to process the input spike vector depends on the number of neurons, N.

In addition to the logic blocks mentioned in Fig. 4.1, there is also a block that controls the time multiplexing of the system. This block is called the Control Unit, which is a Finite State Machine that provides the addresses of the weight memory to process one weight vector in each clock cycle. Due to the requirements of the learning circuit (as explained in section subsection 4.3.3), the Control Unit always provides a random number as the starting address of the weight memory. However, it then goes sequentially over all the weight vectors, until it finds the first neuron that is a candidate for learning.

4.1.2. Design choices and optimizations for overall chip architecture

In this subsection we mention a set of items that affect the overall chip architecture. The first item concerns a set of parameters which we use in the remainder of this chapter, defined in Table 4.2 and the second item concerns the manner of encoding spikes and synaptic weights.

In the previous chapter, we describe spike codes (i.e., spike vectors) and synaptic weights as being represented on 8 bits (one-hot encoding). To recap, each neuron in the trainable layer has $8 \times N$ binary synapses, which can be stored using a binary vector of equal size (i.e., $8 \times N$ bits). However, since in any binary neural network processor, even if multiply-accumulate is extremely optimized (e.g., multiplication is replaced with XNOR [28]), data movement to and from the synaptic memory, even when localized on-chip, incurs the highest energy consumption and invariably, latency [85], and given that all synaptic weights need to be stored physically for both inference and training, we propose using a more efficient technique, via a 4-bit vector: while, in theory, individual weights and spikes are still 1-bit, in this encoding optimization they are grouped per cluster of orientation-cells, and we only use the index information of the first orientation-selective cell to spike, which we represent on 4-bits (Table 4.1). We note that we use 4 bits to account for the case where no edge filter fires. This optimization reduces the storage of synaptic weights by a factor of two.

The proposed encoding affects the implementation choice of the same methods from chapter 3 in hardware, e.g., not only the I&F neuron membrane update, but also the output of the Edge filters and lateral inhibition and the SLP.

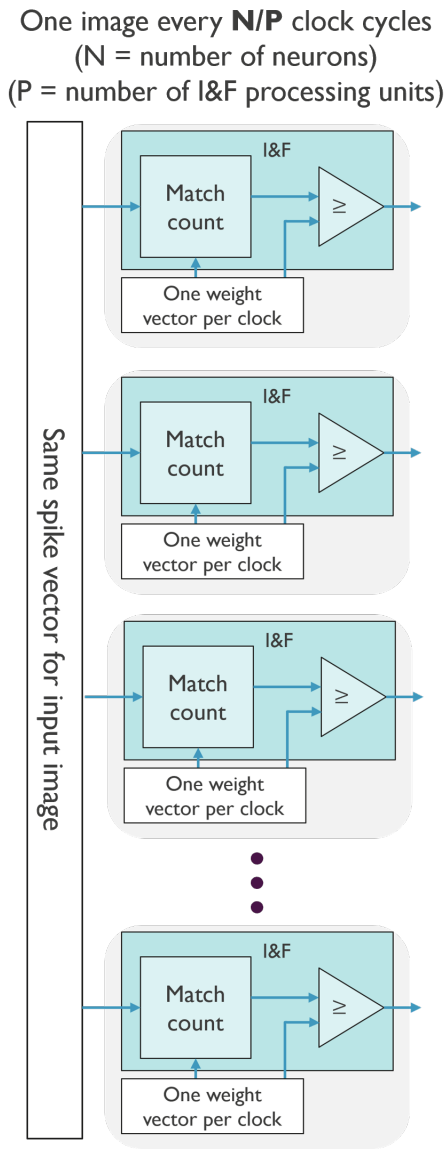


Figure 4.3: To increase the parallelism and, therefore, performance, we can use many I&F processing elements (P) in parallel. Each processing element can process one neuron in every clock cycle. When having N neurons in the system, each input image takes N/P clock cycles.

Table 4.1: Spikes and synaptic weights encoding 8-bits to 4-bits correspondence table

Orientation index	Orientation angle	8-bits encoding	4-bits encoding
1	0°	10000000	0000
2	180°	01000000	0001
3	45°	00100000	0010
4	225°	00010000	0011
5	90°	00001000	0100
6	270°	00000100	0101
7	135°	00000010	0110
8	315°	00000001	0111
-	-	00000000	1000

ID	Name	Value	Description
0.	D	default: 10	Number of elements per row/column of the convolution output.
1.	PATCH_DIM	$(D + K_{SZ}/2 * 2)^2$	Number of elements per row/column in an input image. The patch denomination is used to cover the situation where a high-resolution frame is processed in patches via a sliding window.
2.	K_SZ	default: 5	Number of elements per row/column in a kernel
3.	PX_PKT_WITDH	default: 112	Width of input pixels packet, which, for simplicity, in this implementation is a multiple of 8 bits. It is also possible to stream a pixel in two succesive packets with a simple packet decoding and buffering technique.
4.	PX_IN_CNT	PX_PKT_WITDH / 8	Number of pixels streamed per clock cycle.
5.	DATA_POINTS	D^2	The number data points in the convolution output, considering no zero-padding.
6.	SH_REG_STEP	(ceil(real(PATCH_DIM*K_SZ * 8)/ real(PX_PKT_WIDTH)))	Determines in how many clock cycles the shift register is fully loaded, based on the number of pixels received per clock cycle. This number also denotes the shift register loading overhead.
7.	LOAD_IMG_DONE_CNT	(ceil(PATCH_DIM*PATCH_DIM * 8)/ (DATA_WIDTH)))	Determines in how many clock cycles an input image has been fully loaded and all it's convolved outputs are ready. This number also denotes the latency per image of the spike encoder unit.
8.	ENC_BITS	4 or 8	The number of bits used for encoding spikes and synaptic weights. Default is 8 bits, but the optimized version uses 4 bits.
9.	IDX_BITS	CLOG2(DATA_POINTS)	The number of bits used to represent the index of each element in an output convolved patch. The same representation is used for the number of spike codes in a spike list and synapses per neuron.
10.	enc_spikes_p_clk	(PATCH_DIM - K_SZ/2*2) * (SH_REG_STEP * PX_IN_CNT /PATCH_DIM mod K_SZ+1)*ENC_BITS	Number of spikes encoded per clock cycle
11.	EPOCHS_NO	default:1	Number of passes for each input during training.
12.	PKT_SZ	DATA_POINTS * ENC_BITS	General packet size to represent a full spike list or synaptic weights of a neuron's list.

Table 4.2 continued from previous page

ID	Name	Value	Description
13.	SYN_WEIGHTS_PKT_SZ	default: DATA_POINTS * ENC_BITS	Determines the size of a packet of synaptic weights loaded each clock cycle. By default, it is equal to DATA_POINTS*ENC_BITS, however, to accomodate various memory flavours, with varying data widths, a packet can contain weights of more than one neuron, or part of the weights of one neuron.
14.	NO_NEURONS_IN_PKT	SYN_WEIGHTS_PKT_SZ/PKT_SZ	Determines how many neurons' synaptic weights are read in one go. If more than one neuron's full list of synaptic weights are read, than all neurons membrane potential is computed in parallel. If partial weights are read, then the membrane potential is computed in multiple cycles via intermediate accumulators.
15.	W	default: 64	Number of ON synapses in each neuron. It also gives the maximum membrane potential value per neuron, since all other synapses have a weight 0.
16.	W_bits	default: 7	Bits representation for W.
17.	NO_NEURONS	default: 5120	The number of neurons in the trainable layer.
18.	NO_CLASSES	default: 10	Number of classes used for object categorization. This parameter is only used in the Classifier unit.
19.	class_neurons	NO_NEURONS/NO_CLASSES	Neurons per class.
20.	T_learn_0	default: 6	Initial T_learn value. All neurons states are initialized with this value at boot time.
21.	T_fire_rate	default: 0.5	Firing threshold rate of the learning threshold. All neurons share this value.

Table 4.2: Table with shared constant parameters used in this hardware implementation.

4.2. Edge filters and Lateral Inhibition unit

4.2.1. Module implementation objectives and constraints

This module is essential for both online learning and inference, generating spike codes from input images. Moreover, with the objective of coping with streaming data in mind and since ”time-to-first-spike” is of essence, it is important for the spike encoding step to not represent a major latency bottleneck, be it during inference or learning; in other words, we try to minimize the time to the first generated spike in hardware. For this reason, in this implementation we choose to trade-off area and power for improved latency. However, based on future developments and applications needs, this module can be reconfigured for the desired trade-off, e.g., less area and power at a higher latency cost or vice-versa.

As described in the previous chapter, generating spike codes consists of three main steps: 2D convolution with four 5×5 kernels, max-pooling the convolution outputs channel-wise, and mapping the result to a natural index in the range [1,8]. Out of these three steps, performing the 2D convolution is the most expensive operation both in terms of resources, latency and throughput, which is why we focus on optimizing computation resources for maximum achievable throughput. The main avenues for algorithm-hardware co-optimization have been presented in subsection 2.3.1. In this section, we present the combination of those methods used for the current implementation, with it’s specific constraints and objectives.

Firstly, we define the main objectives and their constraints in choosing the architecture of the spike encoder:

1. The spike encoder should have minimum latency in the presence of streaming data. This will be constrained by:
    - the maximum achievable data rate of the used communication protocol to the edge pixels source sensor
    - achievable clock frequency for maximum parallelization, e.g., for an FPGA platform, we observe it is hard to go beyond 100MHz if data stationarity in the convolutional PEs is high (i.e., when we want to convolve too many input pixels with all kernel channels in parallel), however, this limit can be easily surpassed in ASIC platforms
2. In line with the power efficiency objective, the spike encoder should have minimum resource utilization for maximum throughput. This is constrained mainly by:
    - the data precision of the used weights
    - the cost of performing multiplications (also in combination with the used data precision)
    - the number of FF used in a single PE
    - the complexity of the interconnect between FFs and computational units (multipliers and adders) on the one hand, and from input/output to PEs datapath
    - not intrinsic to the spike encoder but related to the whole neural network architecture, there is an extra cost related to the need to buffer the spike codes produces by this unit; this cost is directly related to the output data width and rate of the spike encoder
3. The spike encoder should be reconfigurable in order to accommodate various input widths and rates such that the overall network implementation is not dependent on the input interface. This will allow for easy experimentation in the future and portability of the architecture on various realization platforms.

Consequently, in the following subsections, we describe how these objectives are addressed in the current implementation. To approach the first objective, we make use of combined temporal and spatial unrolling, using a weight-stationary processing fashion, while for the second objective, the main contribution is in replacing all multiplications with shift operations or skipping multiplications with 0.

4.2.2. Implementation method

This module receives as input data packets representing a group of 8-bit grayscale pixels, streamed in a row-wise fashion and outputs a 4-bit spike vector for each (X,Y) position in the input image. Since

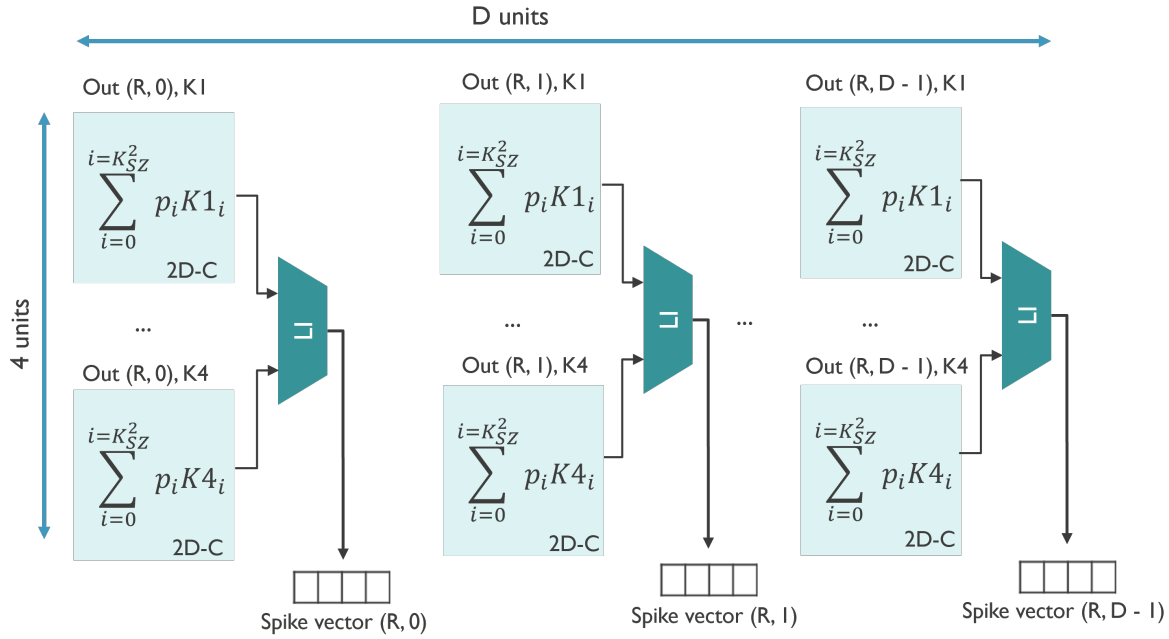


Figure 4.4: Spike encoding implementation: $4 \times D$ units perform the 2D-C (2D-Convolution) of each pixel window with each kernel window in parallel. A LI (Lateral Inhibiton) unit generates the 4-bit spike vector at each position (R,C), where R is the row index and C - the column index. The pixels for the convolution units are read from the shift-register and the convolution weights from a Register File. D spike vectors are generated in parallel

the number of grayscale pixels that can be streamed at a time within a given clock frequency depends on the pixel rate of the interface to the video or image source, the number of received pixels can be reconfigured to accommodate different interfaces. In this implementation we use an input packet `PX_PKT_WITDH = 112` bits (see Table 4.2), which is the equivalent of one row of `PX_IN_CNT = 14` pixels.

The convolution kernels employed here have `K_SZ = 5` and are stored locally in a register file. In order to convolve a streamed input image with a kernel in hardware, at least a number of rows from the input equal to the number of rows from the kernel need to be buffered, to obtain the first output. Thus, we buffer at least 5 image lines by using a shift register with a number of elements equal to 5 $\times$ `PATCH_DIM` elements, where each element has 8 bits.

Once the shift registers are fully loaded, the corresponding registers indexes are multiplied with the corresponding weights in each of the four kernels (see Figure 4.4) via accumulation units. The four channels are computed in parallel, and the maximum achievable number of outputs are computed in one clock cycle, e.g., for this implementation, the spike encoder produces one row of outputs per clock.

Further on, given the set of values that the used weights have, we use a simple weight decoder to achieve the following computational optimizations (Equation 4.1):

- we replace multiplication with left-shift of the input by $log_2(k_i)$, where $i \in [0...K\_SZ^2 - 1]$ and we perform 2's complement conversion in place for negative weights
- we skip multiplications with weights that are 0

$$partial\_conv\_out = \begin{cases} accum_{i=0}^{i-1} + (curr\_px << log_2(k_i)), & \text{if } k_i \in \{1, 2, 4\} \\ accum_{i=0}^{i-1} + (\textbf{not}(px_{in} << log_2(k_i)) + 1), & \text{if } k_i \in \{-1, -2, -4\} \\ accum_{i=0}^{i-1}, & \text{if } k_i = 0 \end{cases} \tag{4.1}$$

Lastly, the convolution outputs are max-pooled channel-wise and depending on the sign of the winning kernel and it's index, a 4 bit code is produced. This is performed by the LI unit in Figure 4.4.

4.2.3. Performance metrics

We summarize here the spike encoder implementation metrics, in terms of overhead, throughput and latency:

- **Overhead:** The encoder has an overhead of `SH_REG_STEP` before producing valid spikes; this is associated with the clock cycles needed for loading the shift register.
- **Throughput:** The encoder produces `enc_spikes_p_clk / ENC_BITS` . The mechanism behind this number is explained in Table 4.2. For this implementation, it corresponds to one row per clock cycle.
- **Latency:** Latency is given by `LOAD_IMG_DONE_CNT` clock cycles. For this implementation, it is equal to 14 clock cycles and it includes overhead.

4.3. I&F neuron implementation

4.3.1. Module implementation objectives and constraints

The I&F neuron checks if spike codes match the corresponding synaptic weight at each index position by accumulating the total number of matches. In line with the URVC concept, the main objectives and constraints of this module are:

1. The I&F neuron should integrate all incoming spikes in the shortest time possible. In sequential hardware terms, the minimum achievable time is one clock cycle, which is equivalent with obtaining a throughput of one output per clock cycle. This is constrained by:
    - maximum achievable bandwidth for reading spikes and synaptic weights. This is related to the available memory resources and differs per off-the-shelf platform (e.g., FPGA, GPU) or can be customized in ASIC designs based on needs. For this implementation, it is necessary that at least two lists of $4 \times D^2$ bits can be read every clock.
    - that the combinational function of the I&F neuron can meet timing in one clock cycle, i.e., the logic gates delay combined with hold and setup time is less than the period of the clock
2. The I&F neuron should use minimum amount of resources such that performance can be enhanced by maximizing the number of possible parallel updates (i.e., less resources per unit lead to more units being used in parallel). This is constrained by:
    - available resources
    - clock frequency
    - routing difficulty or risk of congestion (i.e., interconnecting many units in parallel can be easier in ASIC nodes, where the designer can influence the placement, however it will be limited on platforms such as FPGA's that have a pre-defined 2D programmable routing grid)
    - routing added net delay, which can lead to timing violations

4.3.2. Implementation method

Considering the encoding method presented in chapter 3 and the membrane potential formula in Equation 3.1, the core operation of the IF neuron is essentially a binary multiply-accumulate operation which can be implemented similarly to [28]. However, since we use a hardware-optimized encoding, we proposes a different scheme for performing the multiply, based on a 4-bit XNOR operation.

This module receives spikes and weights packets, and a neuron threshold (i.e, for learning or firing). The spike and weights packets can be of the entire list size `DATA_POINTS x ENC_BITS`, or less, e.g., `SYN_WEIGHTS_PKT_SZ`. The latter situation accommodates various memory flavours with different word widths. If less than an entire list is received, then the match unit produces an intermediate result which is accumulated every clock cycle, to obtain the final membrane potential. The Match unit compares and accumulates all synaptic weights and spikes in a packet during one clock cycle, via a combinational for-loop, which generates an index to decode packets' addresses, and then XNORs the de-muxed

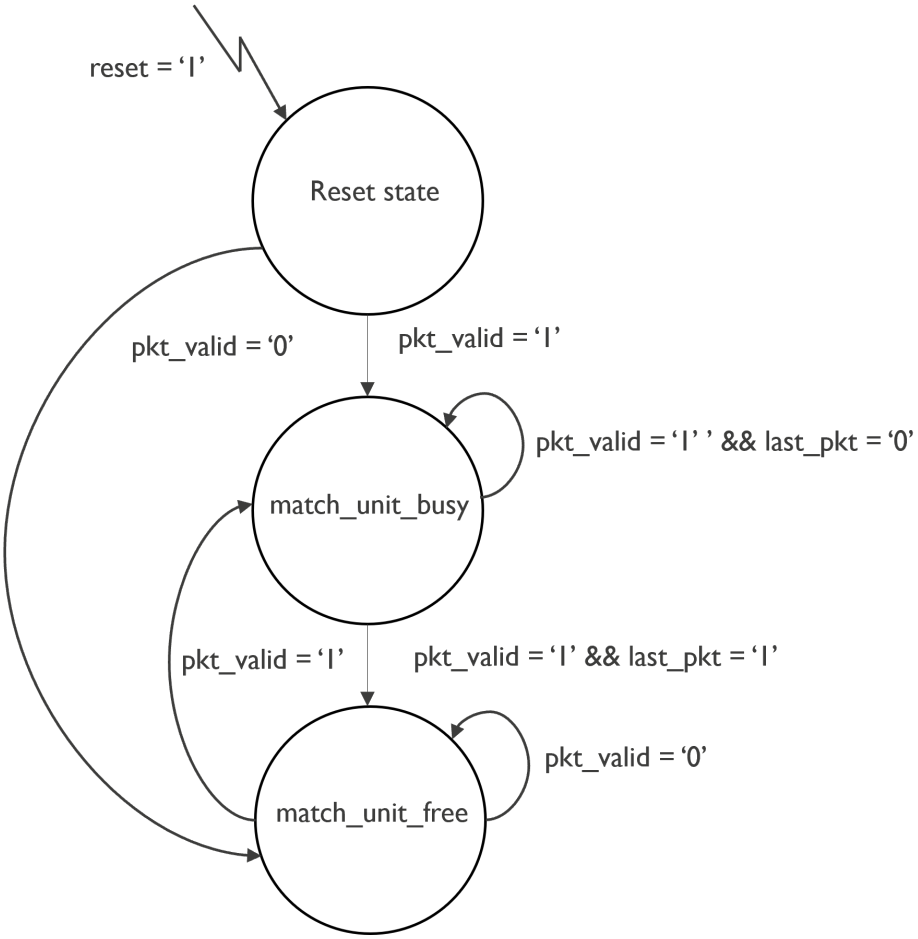


Figure 4.5: Match unit finite state machine

codes. The result is compared with bit-string "1111" to indicate a match, and added to a pop-counter. The unit is externally controlled by a simple FSM (Figure 4.5) which has three states, a `reset` state, a `match_unit_busy` state and `match_unit_free` state. Once a valid packet of spikes and weights is received, the transition to `match_unit_busy` occurs. This state is kept until the last packet has been received. When full lists of weights and spikes are received, `match_unit_busy` takes one clock cycle.

4.3.3. Sequential Learning Process implementation

We implement online learning using the binary stochastic STDP method described in Chapter 3, with the difference that we adapt the hardware implementation to the 4-bits encoding proposed here. This affects how synaptic weights are concentrated on the earliest firing spikes. We enumerate the main steps in the algorithm and the different approaches in hardware versus software:

Step 1: Computing the membrane potential is necessary in advance to verify if the neuron is eligible for learning, on the one hand, and to indicate the number of swaps, $swap\_N$, on the other hand. However, since for this implementation we only allow one neuron to learn/adapt to a new input, we generate a random synaptic memory weight index, after which we continue reading weights sequentially from that address, until we found the first neuron for which the membrane potential reaches the learning threshold.

Step 2: Finding the two lists of ineffective spikes and weights. In software this is a simple bitwise AND between two vectors, where one of them will be logically negated, which is followed by list traversal to determine the positions where the two lists are "1". In software, due to 8-bits encoding, the list has $\mathtt{D} \times 8$ 1-bit elements, while in hardware it will have `D` elements of 4-bit.

Step 3: Generating random indexes to turn weights ON/OFF. In software this can be done via library

functions, such as `randperm` in Matlab [86]. In hardware, generating random numbers or permutations via library functions is only available for simulation but not synthesis. Thus, specialized implementation is necessary. We use a Linear-Feedback-Shift-Register for that.

Step 4: Turn weights ON/OFF: in software, set weight list to binary ”1” or ”0” at the random indexes. In hardware, this will occur at the position of the orientation-cells clusters, and the potentiation/depression is done in a group: at each point, the group of orientation-selective weights are updated at once.

Step 5: Ensuring that only `W` ON synapses are preserved after learning. Since the STDP rule concentrates synapses on the earliest incoming spike lines, we distinguish two situations that can occur:

- a weight is turned ON at an index that was already ON, hence there is no need to turn another weight OFF to preserve W
- a weight is turned ON at an index that was OFF, thus another ineffective weight needs to be turned OFF to preserve W

Step 6: Updating the learning threshold and storing the new synaptic weights and neuron’s state back to the memory.

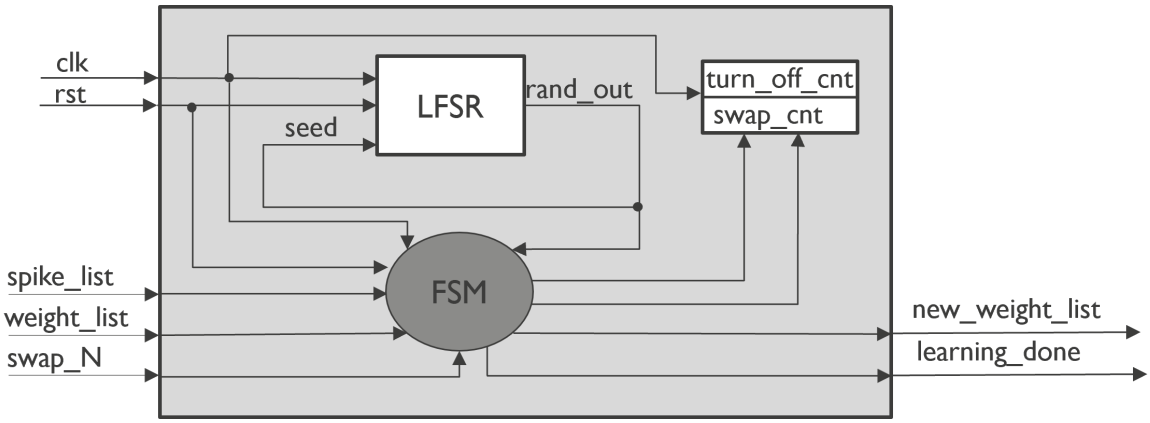


Figure 4.6: Stochastic STDP unit toplevel. All data channels are complemented by `ready` and `valid` handshake signals.

We thus define the main objectives and constraints for the stochastic STDP module:

1. The learning module should have minimum latency such that it can cope with online adaptation for streaming data and incur minimum overhead for inference. This is constrained by:
   - The latency for reading the spikes and weights lists
   - The latency for traversing the lists of spikes and weights to determine which ones are ineffective
   - Latency of generating random indexes
   - The latency for updating weights
2. The learning module should be power-efficient, such that it is suitable for learning at the edge. This is mainly constrained by:
   - The complexity of the operations involved, e.g., if there are multiplications/sorting operations involved.
   - Exploring sparsity in spikes and weights capability
   - The complexity of the random generator

We combine the objectives such that both resources are optimized and latency is reduced. The toplevel module of this unit is depicted in Figure 4.6. A LFSR (see subsection 2.3.2) with $LFSR\_bits = clog_2(DATA\_POINTS)$ bits generates a new random index every clock cycle. This index is evaluated by a finite-state-machine, which performs address decoding on the two lists of spikes and weights and

evaluates whether synapses are updated at that index. The FSM controls when updating a synaptic list is done by updating and evaluation of two counters, one is `swap_cnt`, which is incremented up till `swap_N` everytime a synaptic weight is potentiated, and the other is `turn_off_cnt`, which keeps track of how many synapses need to be depressed to preserve W.

Since LFSR has a fixed number of states it can generate, related to the number of bits of the used shift register, i.e., 2^{LFSR_bits}, some of the random outputs generated by it will not be eligible indexes. Thus, before using the LFSR output as an index, an additional filtering is performed to ensure the index is in the right range. In the contrary situation, we do not proceed with any learning step and we wait for a new index to be generated, hence we skip computation. Moreover, to avoid the costs associated with traversing the full list of weights and spikes to check which ones are ineffective and which ones are effective, for every randomly generated index, we evaluate, in place (in the same clock cycle) whether spikes and weights at that index are eligible for learning (i.e., not equal, but input lines are active) and we either take a series of actions or we skip computations, where sparsity allows it.

In Figure 4.7 we illustrate the hardware implementation of the binary stochastic STDP learning for a use-case with `N = 10` and `swap_N = 3`. Firstly, a 4-bit LFSR generates a new random index every clock cycle, however, since it's range will be (0, 15), but only range (0, 9) is a valid index. Thus, for the first index, 10, further computation is skipped. Next index, 5, is in the valid range, and the FSM evaluates the spike and weight codes at that position. They are equal, thus the new weight at that index remains the same. At the next index, 7, the spike is 0, thus it is not a an eligible ineffective spikes and the new weight stays the same. At the next valid index, the new weight is swapped with the ineffective spike and the swap counter is incremented. At index 3, a new swap occurs, but since the old weight was 0 (turned OFF), the turn_off counter is incremented. After all swap_N weights have been moved, the procedure continues until the turn_off counter is 0 and W is preserved, after which learning_done signal is asserted and a new pair of spikes and weights can be loaded to the unit. For the example in Figure 4.7, the procedures stops after 11 clock cycles. This takes one more clock cycle than the it would have to traverse the lists point-by-point, however, for 4 out of the 11 clock cycles, no computation (apart from index evaluation and state transition) occurs, and for another 2 clock cycles, no updating is performed, thus at a higher latency cost, computation is saved. The last random index is fed as seed for the next LFSR generation.

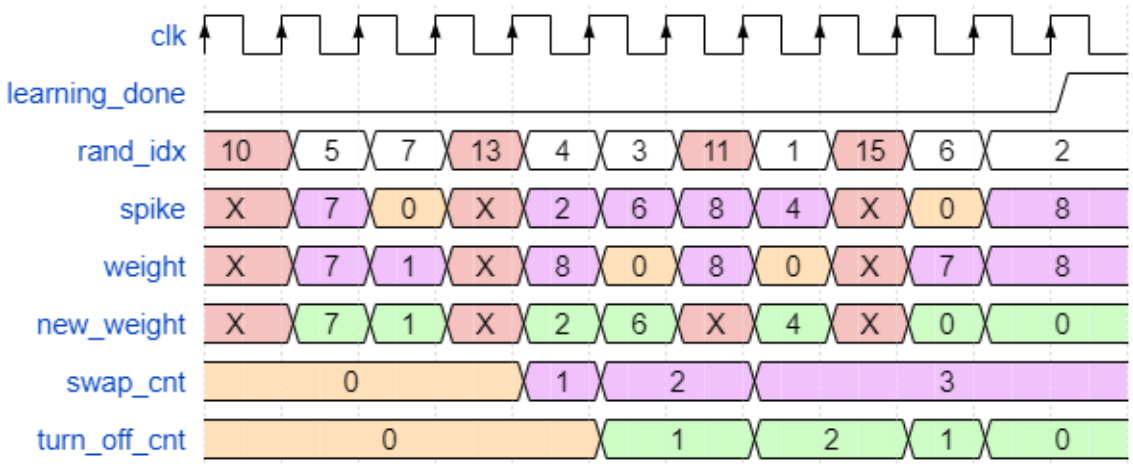


Figure 4.7: Example random learning for N = 10, and swap_N = 3

To summarize the methods used in this module, we explore sparsity in the weights and spikes lists to skip computation and we use an inexpensive LFSR to generate a random index. We note that LFSR also has a disadvantage here, since it incurs extra overhead (but no major computations during the overhead) when it's output is not a valid index. The overhead can be completely reduced by normalizing the output to the desired range, however, this method will incur extra power since it needs multiplications and divisions operation every clock cycle. However, this would not be in line with the hardware-friendly methods we use throughout the architecture and given that our implementation optimizes all multiplication with shift operations and no division is used (apart from the booting phase, when constant parameters are computed for the entire chip architecture based on the N-of-M scheme

used, the number of neurons or the input bandwidth. For this implementation, we trade-off latency for improved energy consumption, however, as it will be shown in the next chapter, for the average situation, latency is much less than N.

4.3.4. Performance metrics

The stochastic STDP takes a variable amount of cycles per each input learned, which depends on the membrane potential of the learning neuron and it's learning threshold. However, in the worst case, the learning latency takes 2^{LFSR_bits}.

5
Experimental setup

In this chapter, we describe the framework used in evaluation and benchmarking of the proposed algorithm. We explore, evaluate the accuracy performance and validate the algorithm in software using the MATLAB R2020b [86] platform. The end-to-end SCNN (i.e., from input grayscale pixels to the output predicted class) is then implemented in VHDL RTL, and synthesised for both ASIC and FPGA. The RTL design is also validated in behavioural simulation using Cadence Xcelium Logic Simulator [87]. For FPGA, performance is evaluated post-layout in Vivado Design Suite [88] for a target VU37P-HBM FPGA platform [89], at 100 MHz. For ASIC, the RTL design is synthesised at 500 MHz for a typical corner at 0.8V and 25° C for a GF-22nm FDSOI node using the Cadence Genus [90] tool and time-based power measurements are generated with Cadence JOULES [91], which reports power with an accuracy within 15% of signoff.

5.1. Tasks and datasets

The algorithm performance is evaluated on both new and conventional image classification tasks, to showcase the unique features of our algorithm.

The first task is binary classification between faces and any other objects. For faces, we use the UTKFace dataset [10] and for the ”no face” class we use the CIFAR10 dataset [11], which we present to the network only during inference.

The second task is to demonstrate online adaptation starting from a pre-trained network. We build this starting from the previous task.

The third task is digit classification using the well-established MNIST dataset [12] to demonstrate and benchmark the performance of our algorithm using a common basis recognized in the research world. To enforce the dataset-agnosticism of the proposed algorithm, we also report the accuracy obtained on the Fashion-MNIST dataset.

The fourth task/benchmark is also built upon the first task and evaluates fast inference while coping with streaming data in UHD frames.

Table 5.1: Experiments parameters

Name	Description	Value
`trainset_sz`	number of inputs fed to the network during training	varied
`testset_sz`	number of inputs fed to the network during inference	10000
`N_neurons`	Number of learning neurons denoting the network capacity	varied
`im_res`	Working resolution of input images after downscaling	varied
`N` and `M`	Maximum $N = im\_res^2$ input spikes active out of $M = N \times 8$	varied
`W`	Number of active synaptic weights	64
`T_learn0`	initial learning threshold of neurons that have not learned	6
`T_fire_rate`	firing threshold used for inference	0.5
`swap_rate`	Ratio of connections that are moved to active input during learning	1
`K`	Number of neurons allowed to learn a pattern during training	1
`N_classes`	Number of unsupervised clusters for classification tasks	varied

For all experiments, the parameters in Table 5.1 are used. In line with dataset-agnosticism, we keep learning specific parameters fixed for all datasets. We also employ the same convolutional kernel weights for the initial edge detection across all tasks and datasets.

The publicly available datasets (Figure 5.1) used for measuring the performance of the proposed learning algorithm and neural network are:

- **UTK Large Scale Face Dataset [10]**: This dataset contains over 20000 distinct faces in the wild, of people with ages from 0 to 116 years old and of various gender and ethnicity. The dataset has a cropped and centered version consisting of 9778 samples, which we subsequently use in our experiments. The initial resolution of the cropped images is 200x200 RGB pixels. The images are stored in the order of age, gender, race and date and time of the image collection. For our tests, we randomize the samples such that our network captures a range of features variations.
- **CIFAR10 dataset [11]**: This dataset contains 60000 labeled images of 32x32 RGB pixels from 10 classes (i.e., airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck). The dataset in split into 50000 images for training and 10000 images for testing. We only employ the testset in our experiments.
- **MNIST handwritten digit database [12]:** This labeled dataset contains 60000 training images and 10000 test images from 10 classes of digits (i.e., 0 to 9). The images are grayscale and have a resolution of 28x28 pixels.
- **Fashion-MNIST [13]**: This dataset has the same configuration as MNIST, with the difference that it depicts fashion items for each class.

Figure 5.1: Example of samples from the used datasets: a.) UTKFace [10], b.) CIFAR10 [11], c.) MNIST [12], d.) Fashion-MNIST [13]

5.2. Algorithm benchmarking in software

5.2.1. Task 1: Unsupervised binary classification for face/no face

For this experiments, we split the faces dataset in a 5000 set for testing and 4778 set for training. During learning, we train a population of neurons in an unsupervised manner with only faces. During evaluation, we present a testset of 10000 samples consisting of 50% non-faces images from the CIFAR10 dataset and 50% images from the faces dataset. The faces presented during inference are different from the ones used for training. Since we perform binary classification, only one class is needed for this experiment, while the other class should not stimulate the population of neurons.

We measure accuracy as:

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{5.1}$$

where:

- True Positive (TP) occurs when at least one neuron fires for the correct input data, i.e., a face
- False Positive (FP) occurs when at least one neuron fires for the incorrect input data, i.e., a no-face
- True Negative (TN) occurs when no neuron fires for an incorrect input data, i.e., no-face
- False Negative (FN) occurs when no neuron fires for the correct input data, i.e., a face

Following subsections present a set of experiments ran for this dataset and task.

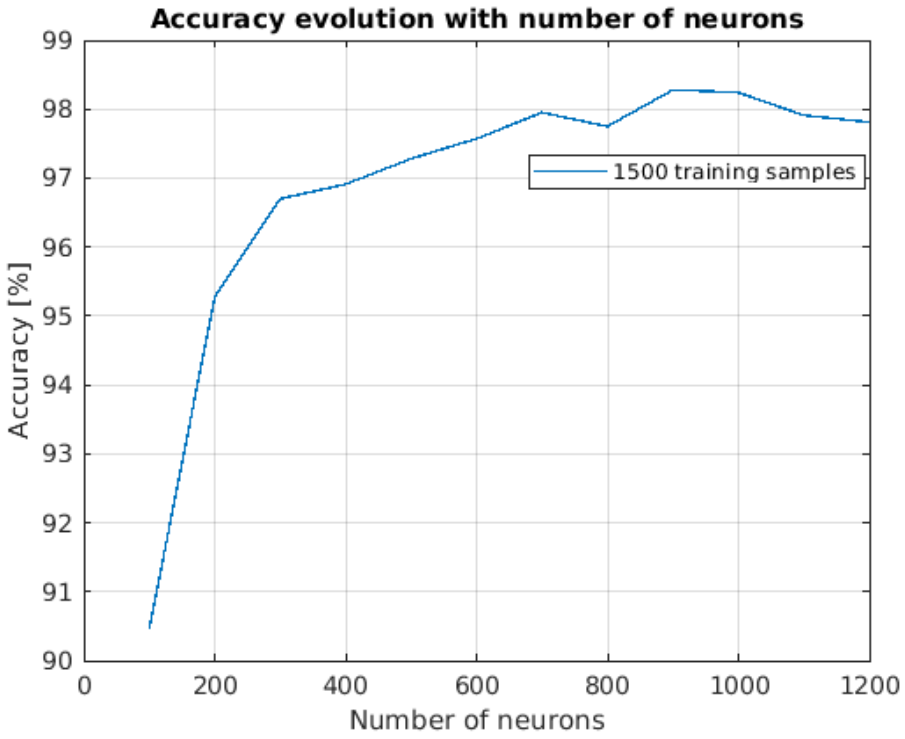


Figure 5.2: Plot of accuracy evolution with the number of neurons in the training layer, for a fixed trainset size.

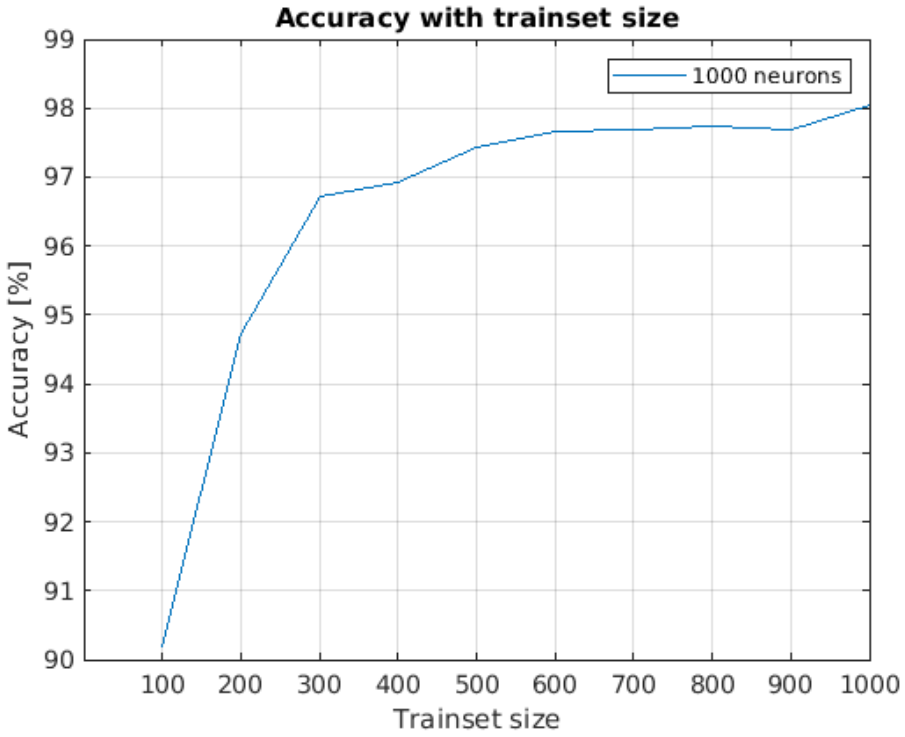


Figure 5.3: Plot of accuracy evolution with the number of training inputs, for a fixed number of 1000 neurons.

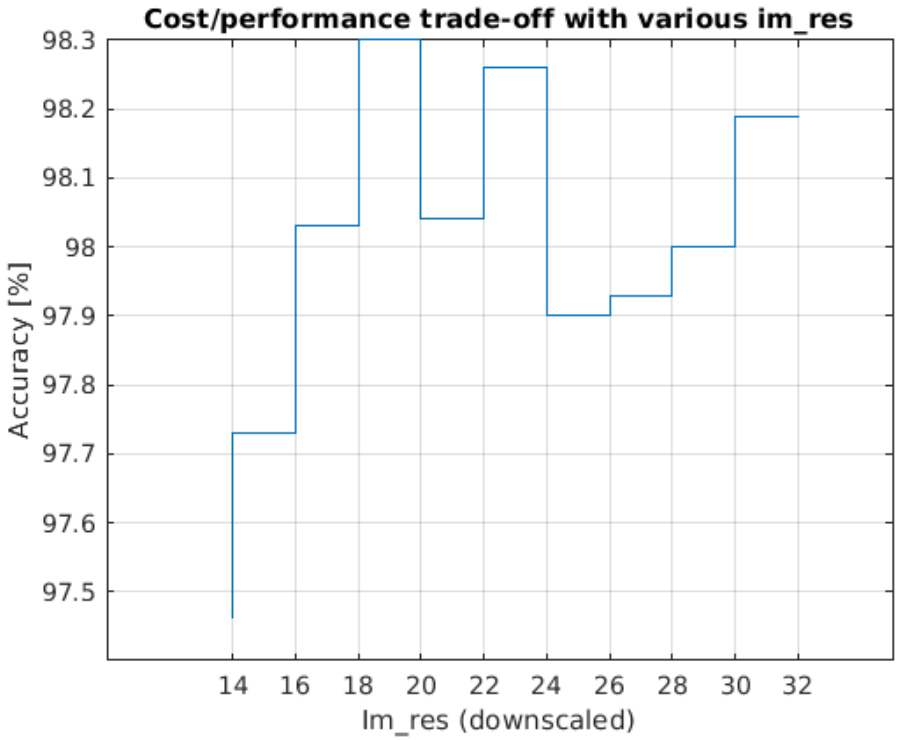


Figure 5.4: Cost/performance trade-off with various input image resolution. The cost refers to computational burden, that scales up with the input image resolution.

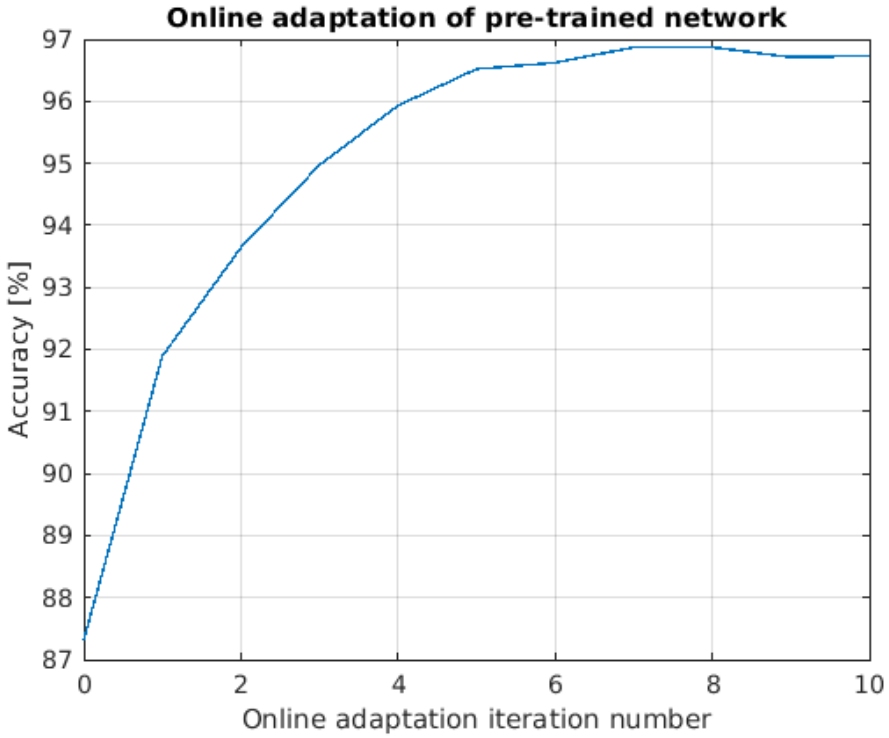


Figure 5.5: Plot of accuracy evolution with each adaptation iteration. Adaptation starts with a pre-trained network. The accuracy of the pre-trained network is for iteration 0 on the abscissa

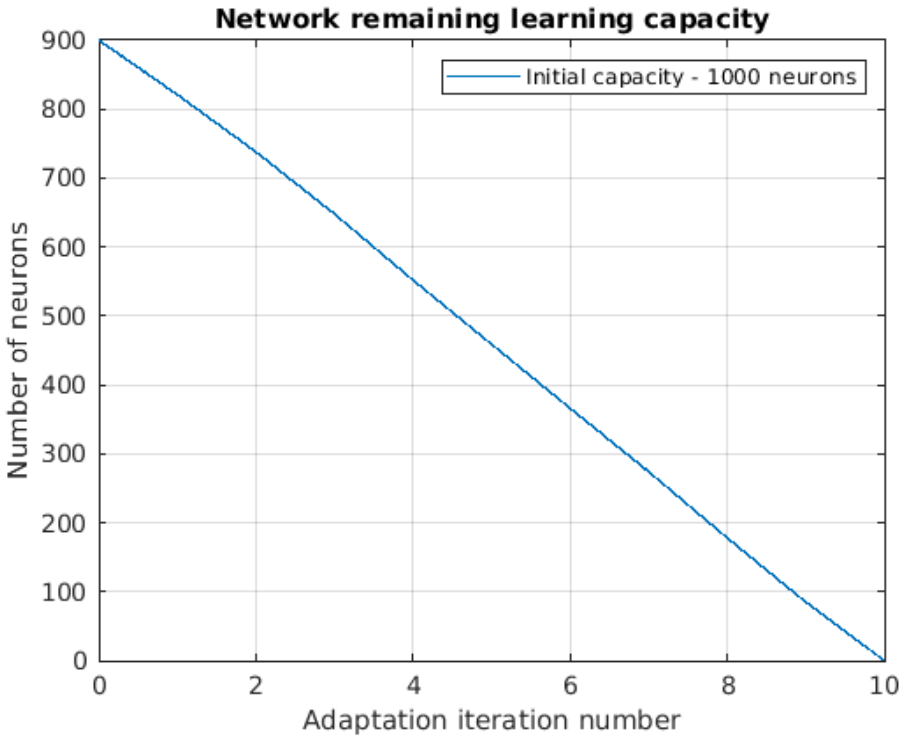


Figure 5.6: Plot of network capacity to learn new features after each adaptation iteration

Experiment 1: Evolution of accuracy with number of neurons

In this experiment, we evaluate how the accuracy of the network evolves when the trainset size is fixed, but the number of neurons in the training layer increases. After each test, we measure accuracy on the same testset. Here, we vary the number of neurons from 100 to 1000, with a step of 100, and we keep the trainset size fixed to 1500 samples. The `im_res` is kept to 32. The result is plotted in Figure 5.2 and shows that accuracy increases with the number of neurons available in the network. The plot also shows that after approximately 1000 neurons, the accuracy does not improve with more neurons.

Experiment 2: Evolution of accuracy with number of training samples

This experiment is similar to the previous one, with the difference that the number of neurons is kept fixed, while the trainset size is varied from 100 to 1000, with a step of 100. The result is plotted in Figure 5.3. The plot shows that the accuracy increases with the number of training samples.

Experiment 3: Evolution of accuracy with input down-sampling

In this experiment, 1000 neurons are trained using 1000 faces, for inputs being down-scaled to various `im_res`. The result is depicted in Figure 5.4. For each `im_res` on the abscissa, the same 1000 samples are used for training and the same testset for accuracy evaluation. The result is depicted in Figure 5.4 and it shows that the accuracy negligibly increases from `im_res` between 14 and 18, but then oscillates until 32.

5.2.2. Task 2: Online adaptation starting from a pre-trained network

For this task we employ the same use case as in the previous task: for binary classification between faces and no-faces, we show online adaptation starting from a pre-trained network.

Experiment 1: Evolution of accuracy with online adaptation

For this experiment, we pre-train a network with a maximum learning capacity of 1000 neurons with only faces. To be able to see the network adaptation ability, we pre-train until a reasonably-high accuracy is reached. After 100 samples, the network reaches an accuracy of over 87%. After this step we present, for 10 iterations a set of 200 samples, where the first 100 samples are faces and the next 100 samples are non-faces. If at least one neuron fires for each sample, learning is triggered, and the network adapts to the new input. If the network fires for a non-face, the wrong class will be learned. After each iteration, we measure accuracy again, according to Equation 5.1 on the same test set. Each time a neuron's synaptic weights are updated, it's firing threshold is also updated accordingly. The results can be seen in Figure 5.5. The plot shows that the the accuracy increases the first 8 adaptation iterations (equivalent to extra 800 faces samples being presented), after which it saturates.

Conversely, to showcase that new features are learned by neurons that have not learned anything before, we plot the remaining number of neurons that have not learned anything (i.e., their initial learning threshold is unchanged), after each iteration, in Figure 5.6. The plot shows that after pre-training on the 100 samples (i.e., at iteration 0), the network capacity is 900 neurons, and it decreases linearly with each training samples that the network adapts to.

5.2.3. Task 3: Multi-class classification using supervised clustering

For this task, we employ the MNIST dataset. We use the split of 60000 samples for training and 10000 samples for testing. All experiments are evaluated on the same testset, while for training the number of used samples varies per experiment. For all experiments, we use a `im_res` of 14, however, comparable accuracy is reached for `im_res` of 28.

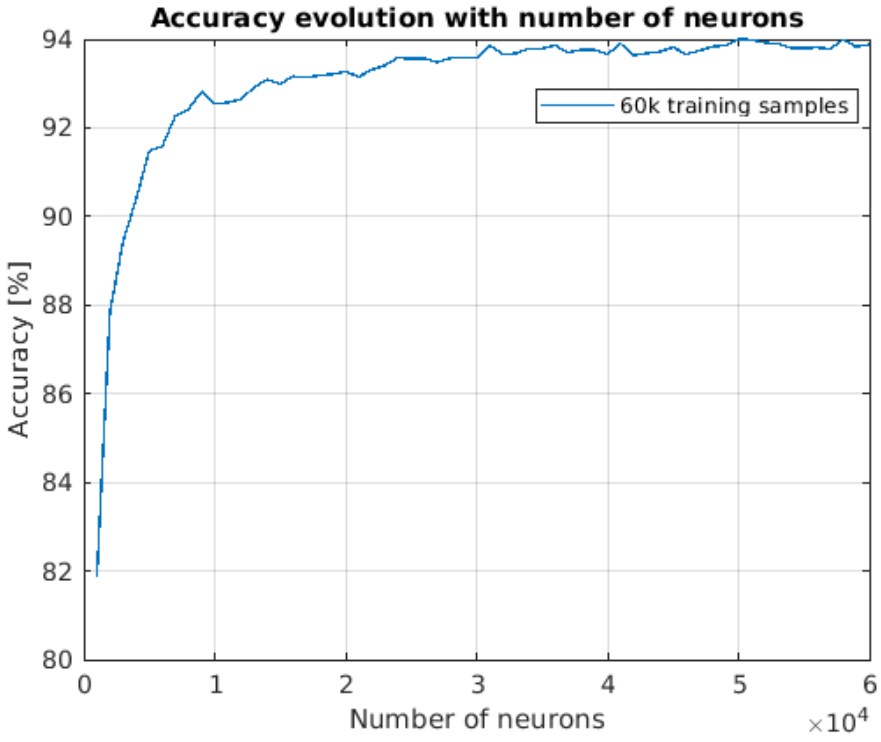


Figure 5.7: Plot of accuracy evolution on the MNIST dataset with increasing the number of neurons

Experiment 1: Evolution of accuracy with number of neurons

This experiment is similar to the one in subsubsection 5.2.1.1. Here, the trainset size is kept fixed to 60000, while the number of neurons are varied between 1000 and 60000 with a step of 1000. The

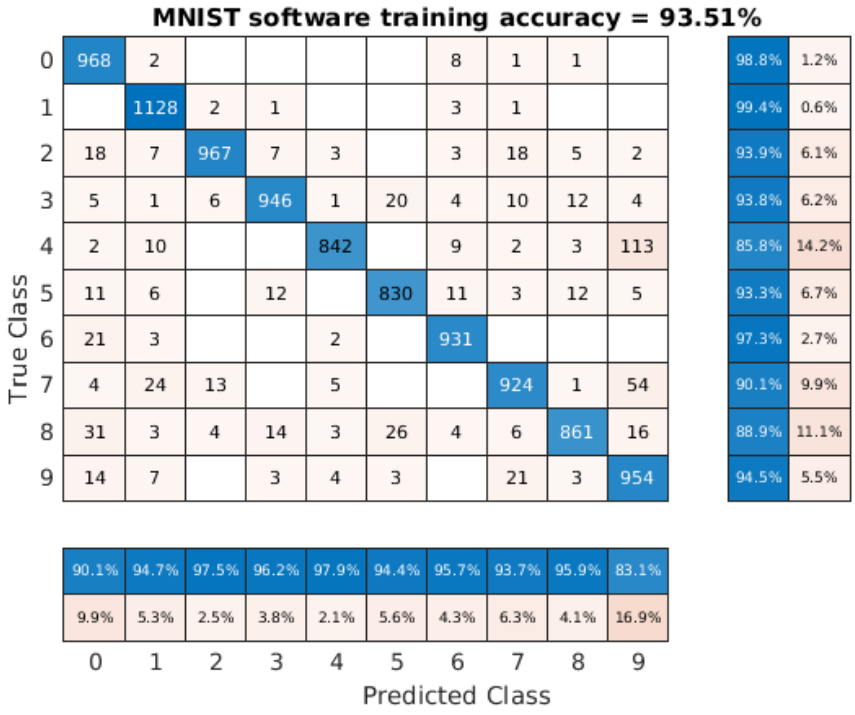


Figure 5.8: MNIST confusion matrix for 30000 neurons and 60000 training data

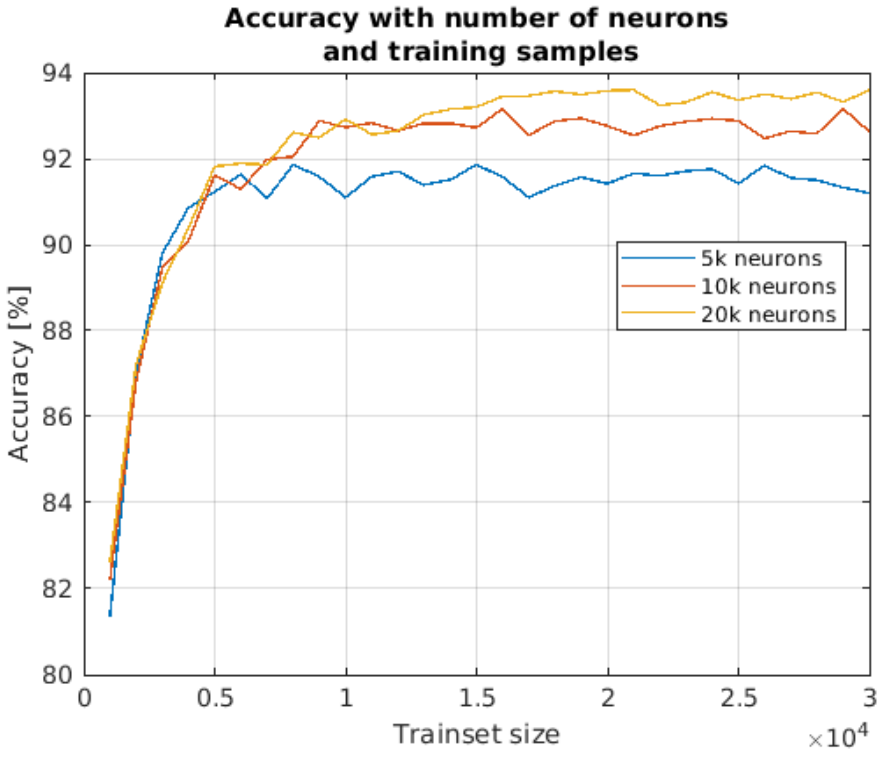


Figure 5.9: Plot of accuracy evolution on the MNIST dataset with increasing the trainset size

result is plotted in Figure 5.7 and it shows that accuracy steeply increases with the number of neurons until approximately 10000 neurons, and continues growing at a slower rate until around 30000 neurons, after which it closely oscillates under 94%.

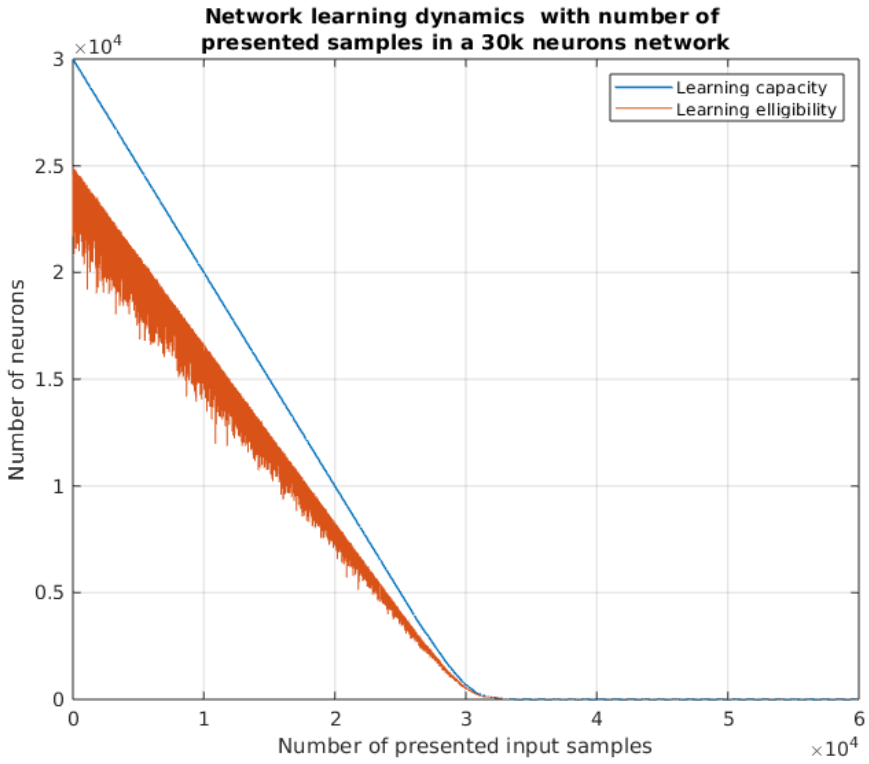


Figure 5.10: Evolution of network learning dynamics with number of training samples

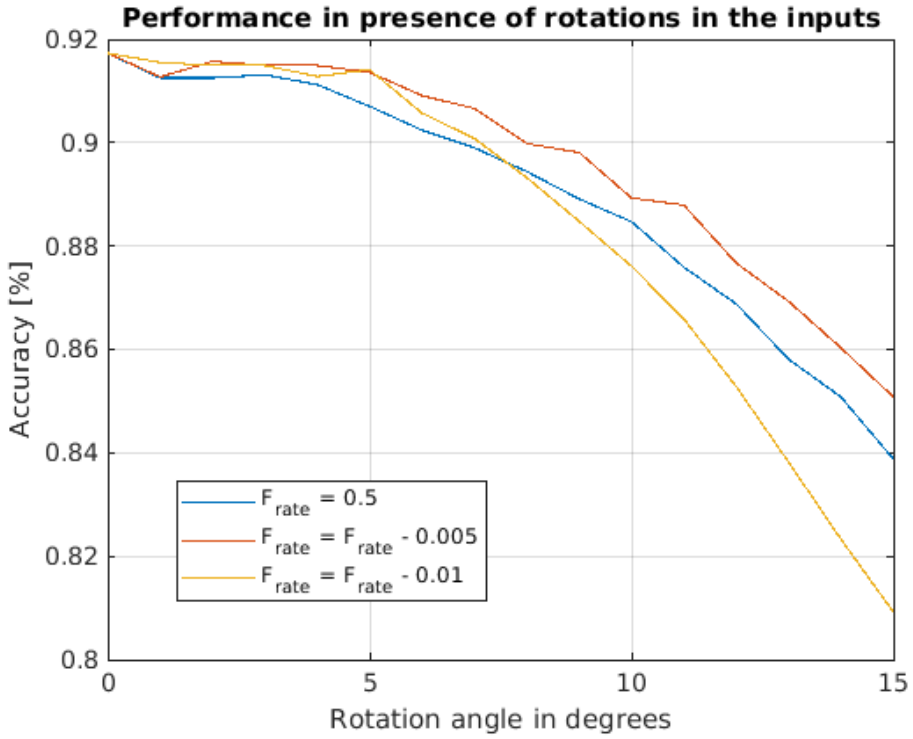


Figure 5.11: Network robustness to rotation variations in the test set

In Figure 5.8 we represent the confusion matrix for the case with N_neurons = 30000 and trainset_sz

= 60000. In the matrix, we can see that the highest accuracy, 99.4% is achieved for digit 1, while the lowest accuracy is for digit 4, at 85.8%, which is confounded highly with digit 9. Next lowest accuracy is for digit 8, which also is confounded highly with digit 9.

Experiment 2: Evolution of accuracy with number of training samples

In this experiment, the number of neurons are kept fixed to 5000, 10000 and 20000 respectively, while the trainset size is varied between 1000 and 30000, with a step of 1000 samples, for each of the three cases. The result is plotted in Figure 5.9. The plot shows, firstly, that for the same number of trainset data, the accuracy increases with more neurons. The plot also shows that the accuracy tends to saturate after a number of training samples equal to the number of neurons have been presented. The main result is that the accuracy increases with more training data, up till a saturation point is reached.

Experiment 3: Evolution of network learning dynamics with number of training samples

In relation to the previous experiment, this exercise shows how the network learning dynamic changes with the online learning of more samples. The result is depicted in Figure 5.10 and it shows that, for a network with initial capacity of 30000 neurons, the learning capacity decreases with each input samples, reaching 0 after a number of samples equal to the initial network capacity have been presented. Under the blue line, the learning eligibility is plotted, which shows, for each sample, how many neurons are passing their learning threshold. This number is less than the entire network capacity at each learning point.

Experiment 4: Robustness to rotation variations in input

To evaluate the network's capacity to perform well in dynamic environments, we evaluate the accuracy of a trained network on rotated versions of the testset. We rotate the testset between 1 and 15 degrees. The result is depicted in Figure 5.11. The blue curve shows the accuracy evolution when the firing threshold rate is the same as in previous experiments (i.e., 0.5). However, since, intuitively, the more rotated the input is, the matching between synaptic weights and the input spikes decreases, we observe the effect a slight adaptation in the firing rate, at each rotation, has. For instance, we test what happens when the firing rate is decreased with 0.005 and 0.1, at each step on the abscissa. We notice that a small decrement for the adaptation incurs a smaller accuracy loss.

Validation of algorithm on Fashion-MNIST
We apply the same network on the Fashion-MNIST dataset. Using 30k neurons, and 30k training samples, we obtain 77.65% accuracy on Fashion-MNIST.

5.2.4. Task 4: Benchmarking for UHD frame processing

For this experiment, we explore the benefits of the proposed algorithm in the situation of pattern recognition in UHD (Ultra-High-Definition) frames. The goal is to evaluate how many correct full faces our algorithm predicts in a frame with many small faces. This is inspired by the fact that humans are able to spot only two known faces in a crowd, at a time [92]. One quarter of a UHD processed frame is depicted in Figure 5.12. Some faces have thicker circles around due to multiple neighbouring positions triggering the population of neurons, due to the algorithm robustness to shift-invariance.

The main concept behind this task is to process a high resolution frame using a sliding window (similar to 2D convolution) with a dimension of $im\_res \times im\_res$. We name each such window one patch. Each patch is then processed by the SCNN network we proposed: a spike vector is generated from each patch and then the inference engine is triggered.

The UHD frame processed here is a collage of faces and non-faces, using the same datasets as for Task 1, and a layer of 400 neurons. If we process an UHD frame in segments of 32×32 and a stride of 1, then $(2160 - 32) \times (3840 - 32) = 8103424$ positions need to be computed (i.e., this is the total number of inference operations).

In this experiment, the UHD frame has 10% faces in the collage, placed at random positions. The rest are non-faces. A total of 8100 small patterns are present in the collage. Since we process the

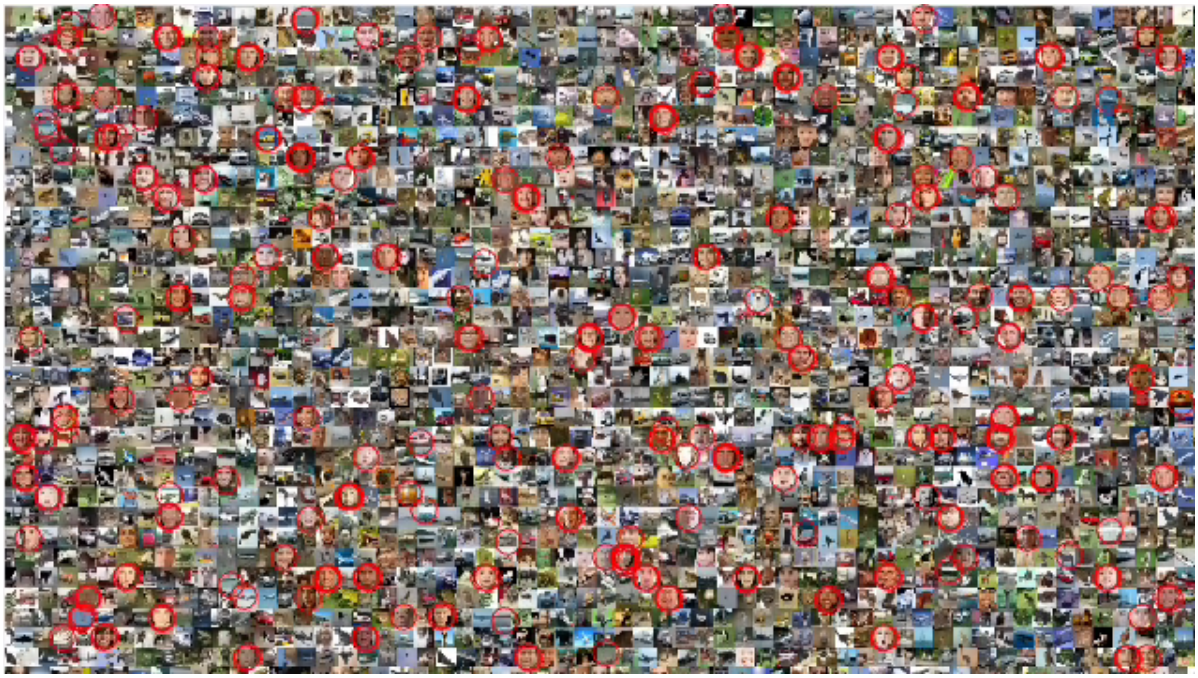


Figure 5.12: One quarter of one processed UHD frame. Some faces have thicker circles around due to multiple neighbouring positions triggering the population of neurons, due to the algorithm robustness to shift-invariance

UHD frame using a sliding window, sometimes the field of vision can include both part of a face and a non-face. We thus do not account for those as wrong predictions.

For this experiment, 720 out of 798 faces were correctly spotted, thus more than 90% of the faces.

5.3. Hardware implementation benchmarking

In this section, we evaluate and benchmark the proposed method in hardware. The main framework is based on setting up a full learning and inference task on MNIST corresponding end-to-end to the MATLAB implementation. The neurons are updated in a time-multiplexed manner (i.e., for each spike list, one neuron is updated per clock cycle, such that results per unit per clock cycle can be reported. In order to minimize the energy consumption for the memory block, we use a `im_res` of 14 throughout the hardware performance measurements and we report metrics for `N_neurons` = 5120. The results can however be easily extended to a different configuration of patch size and total number of learning neurons. For each module in Figure 4.1, we report the following metrics:

- FPGA resource utilization
- ASIC area and cell count
- latency and throughput in ASIC and FPGA (i.e., these differ per clock frequency)
- energy consumption in ASIC
- online training accuracy on-chip for the MNIST dataset

We extend these results to report the achievable throughout in FPS for UHD video processing, along with it's impact on hardware resources.

5.3.1. Benchmarking on FPGA

The RTL code is synthesised and implemented for the VU37P-HBM target FPGA platform [93]. The choice of memory uses BRAM. The reported BRAM results refer to 36Kb blocks. One CLB consists of 8 LUTs and 16 FFs. The clock frequency at which the design is implemented is 100Mhz.

Resource utilization in FPGA

The breakdown of the resource utilization per each unit, reported in Vivado after the post-implementation of the top level module which encapsulates the end-to-end SCNN architecture is compiled in Table 5.2.

Table 5.2: FPGA resource utilization at 100Mhz, on the VU37P-HBM target platform

Module name	**LUT**	**CLB**	**BRAM**	**FFs**
	1303680	162960	2016	2607360
Edge filters and lateral inhibition	6010	969	0	744
I&F unit	449	152	0	7
Sequential learning process (SLP)	1518	447	0	845
Synaptic Weights Mem.	0	0	56	0
Neurons params	0	0	2	0
Classifier	76	20	0	41
Control FSM and global parameters	117	106	0	914
Total	8170	1694	58	2551
% of available resources	0.62%	1.03%	2.87%	0.09%

Latency and throughput in FPGA

In a data streaming situation, the overhead of the spike encoder is hidden after the first encoded spike list, as the rate at which spikes are encoded compared to the rate at which neurons are updated can be, in most cases, faster, and thus an intermediate spike buffer can be used. For simplicity, in our experiments, we use a `{data_ready, data_valid}` handshake to denote when a new spike packet can be received. The classifier is pipelined with the I&F unit, thus it will give the prediction one clock cycle later than the last neuron update. Without parallelization of the number of I&F units during inference (i.e., P in subsection 4.1.1) is 1), the prediction latency is $(\texttt{LOAD\_IMG\_DONE\_CNT} + \texttt{N\_neurons} + 1) \times \texttt{clk\_period}$. Thus, for a 10 ns clock period, and taking into account the spike encoder overhead, the latency is $(14 + 5120 + 1) * 10 = 51350\texttt{ns}$ and the throughput is $10^9 \backslash 51350 = 19474.19$ inferences per second. Without the spike encoder overhead, the throughput will be 19527.43 inferences per second (IPS) (Table 5.3).

Table 5.3: Inference latency and throughput of the digit classification task on the FPGA

Metric	**Value**
Latency	51350 ns
Throughput	19474.19 IPS

MNIST accuracy in hardware versus software

For this experiment, we set a full training pipeline with 5120 neurons and 5120 training points, followed by a full inference on the 10000 images testset. We have behaviourally simulated EON-1 toplevel in Cadence Xcelium [87]. We processed the accuracy based on the output labels from the hardware inference in Matlab. We have obtained an accuracy of 91.96% in hardware and 91.75% in software. The slight difference between the two comes from the fact that different random samples were used for training.

5.3.2. Design space exploration for system scalability and performance enhancement on FPGA

In this experiment, we perform Design Space Exploraton (DSE) to evaluate the scalability of the proposed network on two axes:

- the first axis is in terms of input patch resolution, i.e., im_res

- the second axis is in terms of number of neurons in the learning layer and the parallelization degree possible when adopting various memory types: on the FPGA chip (i.e., Distributed RAM, BRAM, URAM) or adjacent to the FPGA chip (HBM).

Experiment 1: Resource utilization scalability with various input resolutions

We evaluate how the FPGA LUT utilization scales up with various patch resolutions, im_res, for two central units in the proposed SCNN: the Edge Filters and Lateral Inhibition unit and the I&F unit. We plot the synthesis results for resolutions between 14 and 32 in Figure 5.13 and Figure 5.14. The figures show that the utilization scales almost linearly with the increased resolution.

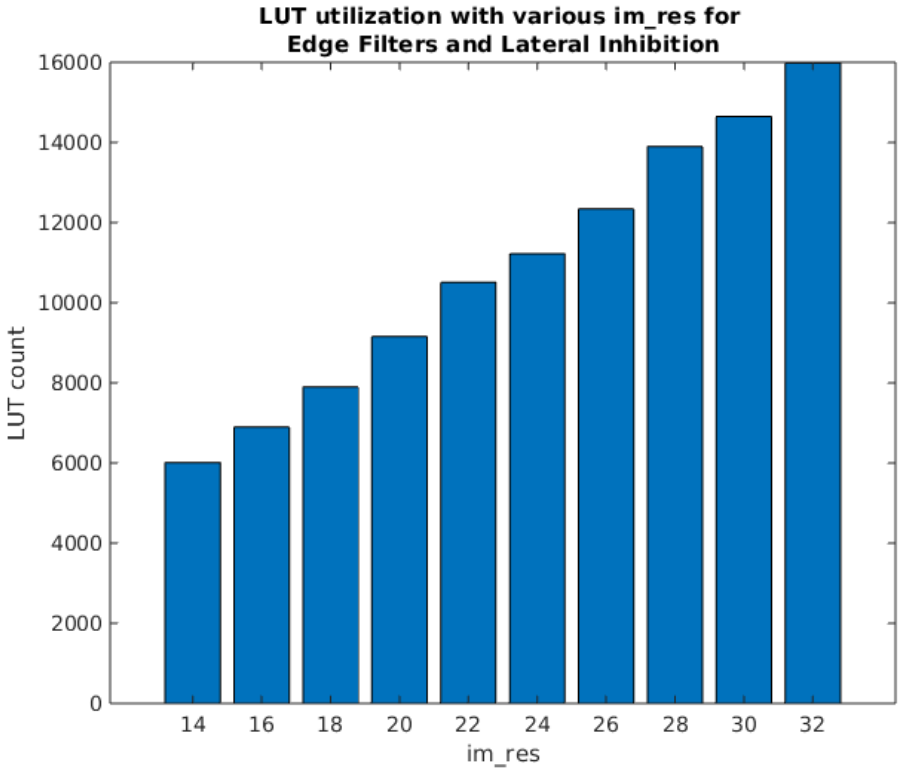


Figure 5.13: LUT utilization scalability with input resolutions for the Edge Filters and Lateral Inhibition Unit

Based on synthesis results, a maximum of $P = 400$ I&F units can be used in parallel, for a resolution of $im\_res = 32$. This result gives the degree of system performance enhancement described in subsection 4.1.1.

Experiment 2: Learning layer scalability

For this experiment we explored what is the maximum achievable number of synaptic weights vectors for each memory type on the FPGA, and also the maximum number of weights vectors, P_W, that can be accessed every clock cycle for each memory. Since we define in subsection 4.1.1 P as the number of I&F units that can be used in parallel, and $P\_W < P$, achieving maximum system performance for the target FPGA is possible if P is maximized. Thus, we define $P\_S = P/P\_W$ the number of spike vectors that need to be accessed in parallel to maximize P.

The memory types available on-chip on the target FPGA are: Distributed RAM, BRAM, URAM, including the FPGA-chip adjacent HBM memory [1]. The on-chip memories reported weights bandwidth is based on True-Dual-Port mode and two reads per clock cycle. The HBM memory is accessed via

[1] HBM is placed on the same interposer as the FPGA die and has a theoretical bandwidth of 460GB/s [93]

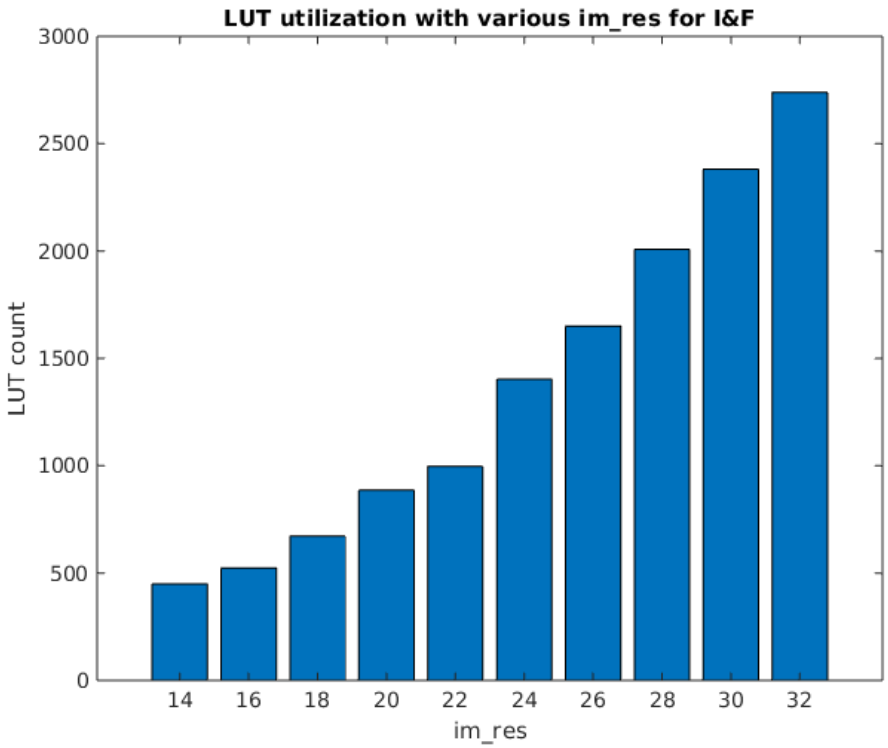


Figure 5.14: LUT utilization scalability with input resolutions for the I&F Unit

AXI3 protocol, and thus only one read and one write is possible per request. We compile the DSE results in Table 5.4.

Table 5.4: Scaling up the learning layer

Memory type	P_W [1]	P_S [2]	Max. neurons
Distributed RAM	200	2	3200
BRAM	64	6	16K
URAM	32	12	64K
HBM	1	400	16M

[1] Weights vectors bandwidth/parallelization factor
[2] Spike vectors parallelization factor

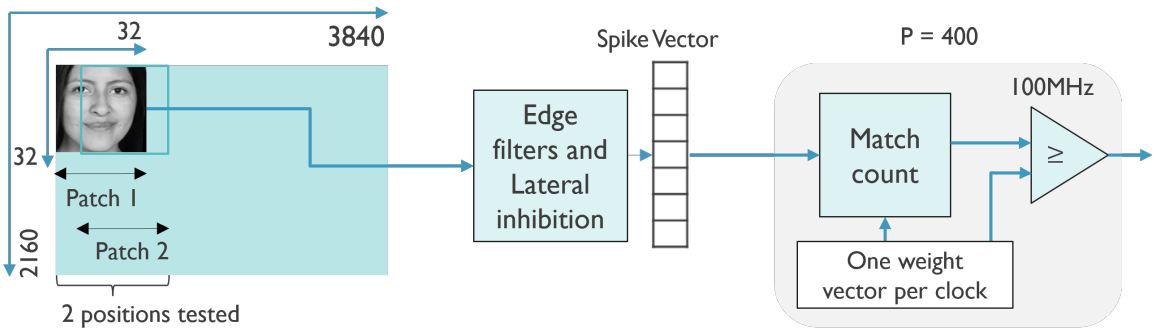


Figure 5.15: Processing an UHD frame using a shifting window. P = 400 I&F units available in parallel.

Experiment 3: Processing UHD frames in real-time on the FPGA

We report the processing time per UHD frame [2] and achievable FPS rate on the target FPGA platform, by segmenting each frame in small Fields of Vision of D^2 size. This concept is depicted in Figure 5.15.

We explore a combination of input-stationary and weights-stationary approach, to match the maximum achievable number of I&F units in parallel, i.e., $P = 400$.

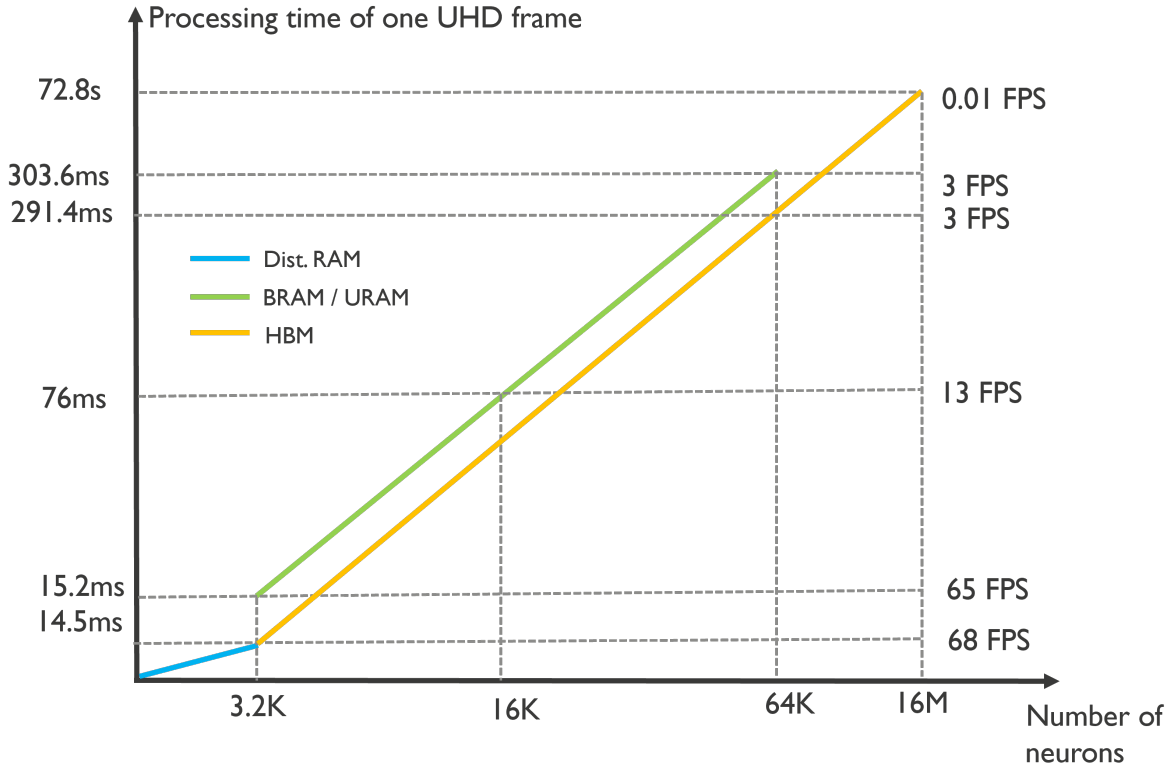


Figure 5.16: Processing time of one UHD frame based on different memory types and number of neurons, if $P = P\_S \times P\_W = 400$ is maximized

For the situation in Figure 5.12, in subsection 5.2.4, we can achieve 12 FPS, providing that 8M positions are tested. However, the UHD frame in Figure 5.12 is an extreme case of high-resolution frame containing many extremely-small elements. In practical situations, input frames can be highly downscaled before being fed into pattern recognition engines. The faces that we used in the experiment are also downscaled from 200×200 pixels to 32×32 pixels. Moreover, our algorithm is resilient to input downscaling, as has been shown in subsubsection 5.2.1.3. Thus, if a UHD frame is dowsncaled by a factor of at least $\sqrt{5}$, this reduces the number of positions that need to be tested by 5, and we can easily achieve inference at 60 FPS.

If we follow the same downscaling factor as for the faces experiment (i.e., $200/32 = 6.25$), this will result in $(2160/6.25 - 32) \times (3840/6.25 - 32) = 182166$ positions that need to be tested per frame. We evaluate how the processing time per one UHD frame evolves with different numbers of neurons in the learning layer, in combination with various memory types. We plot the results in Figure 5.16. The plot shows how the acvhievable throughput decreases with more neurons in the learning layer.

[2]The reported number is for downscaling the UHD frame by a factor of 6.25

5.3.3. Benchmarking on ASIC

For ASIC, the RTL design is synthesised at 500 MHz for a typical corner at 0.8V and 25° C for a GF-22nm FDSOI node using the Cadence Genus [90] tool and time-based power measurements are generated with Cadence JOULES [91].

Area and cell count in ASIC

Area and cell count are reported by the Cadence Genus synthesis tool and are compiled in Table 5.5. The Synaptic Memory is implemented with pseudo-dual port SRAM technology.

Table 5.5: ASIC GF22 area and cell count

Module name	**Cell count**	**Cell area (μm^2)**	**Net area (μm^2)**	**Total area (μm^2)**
Edge filters and Lateral Inhibition	9152	8377.24	2768.14	11145.38
I&F unit	631	456.53	115.37	571.91
Sequential learning process (SLP)	11087	6251.84	3074.276	9326.12
Synaptic Weights Mem.	0	535000.64	0	535000.64
Neurons params [1]	0	0	0	0
Classifier	227	144.1	55.34	199.44
Control FSM and global parameters	1841	1635.04	638.01	2273.05
Total	22938	551465.42	6651.15	558116.57

[1] The neuron parameters are included in the Synaptic Weights Memory cost

Energy consumption in ASIC

To measure energy consumption for each unit in the proposed SCNN, we use the equation in Equation 5.2:

$$E_{unit} = (T_{end} - T_{start}) \times P_{total}/no\_runs, \tag{5.2}$$

where E_{unit} is the total energy for one unit, T_{start} and T_{end} are the simulation starting time and end-time, P_{total} is the average power reported by Joules during the measured time frame, and no_runs is the number of instances ran back-to-back during that time frame. We detail our measurement framework:

Table 5.6: ASIC energy consumption per block

Module name	**Energy (pJ)**
Edge filters and Lateral Inhibition	
Per patch	337.77
Per encoded input	3.37
I&F unit	
Per neuron	0.324
Sequential learning process (SLP)	
Per swapped weight	18
Synaptic Weights Mem.	
Read 400b	72
Write 400b	88
Neurons params	
Read 7b	1.26
Write 7b	1.54
Classifier	
Per inference	47.54

- to measure the I&F unit cost for one neuron, we divided the averaged energy consumption of 15 inferences back-to-back to the total number of neurons, i.e., 5120.

- for the Edge Filters and Lateral Inhibiton unit, 512 images are encoded back-to-back and the average result is reported
- for SLP, 5000 inputs are learned back-to-back in which one swap per neuron is allowed, and the average is reported

We measure energy consumption per inference[3] as in Equation 5.3: firstly, the I&F unit is used `N_neurons` times, and during that time one spike list and `N_neurons` weights lists are read, together with `N_neurons` neuron states. Thus, for this case, the total energy consumption measured with Joules, during one inference, is $376.822nJ$. If an accuracy error of 15% from the Joules tool reporting is considered, then the worst case power consumption is $433.345nJ$. In Figure 5.17 we report the performance/-cost trade-off when scaling the learning layer from 1000 neurons to 30000 neurons for the MNIST digit recognition task.

$$\mathrm{E_{inference}} = \mathrm{N\_neurons} \times (\mathrm{E_{I\&F\_unit}} + \mathrm{E_{read\_7b}}) + (\mathrm{N\_neurons} + 1) \times \mathrm{E_{read\_400b}} \tag{5.3}$$

Online learning energy consumption is computed as the total energy cost for performing one inference (including the spike encoder and classifier), in order to find the neuron that reaches it's learning threshold, followed by the energy cost of one neuron learning: the cost of performing the worst case number of swaps during learning (i.e., $W - T\_learn\_0 = 64 - 6 = 58$), the cost of reading one spike list and one weight list and also writing back one weight list and updating one neuron state. The formula is given in Equation 5.4. Thus, the total energy consumption for one neuron learning is $1.27nJ$, and if the inference cost, the spike encoder cost and the classifier cost is added, the energy consumption is $378.48nJ$ per learned patch, or $435.256nJ$, if we consider an error margin of 15%. Tabel Table 5.7 compiles the inference and learning energy cost.

$$\mathrm{E_{learning}} = \mathrm{E_{inference}} + 58 \times \mathrm{E_{one\_swap}} + 2 \times \mathrm{E_{read\_400b}} + \mathrm{E_{write\_400b}} + \mathrm{E_{write\_7b}} \tag{5.4}$$

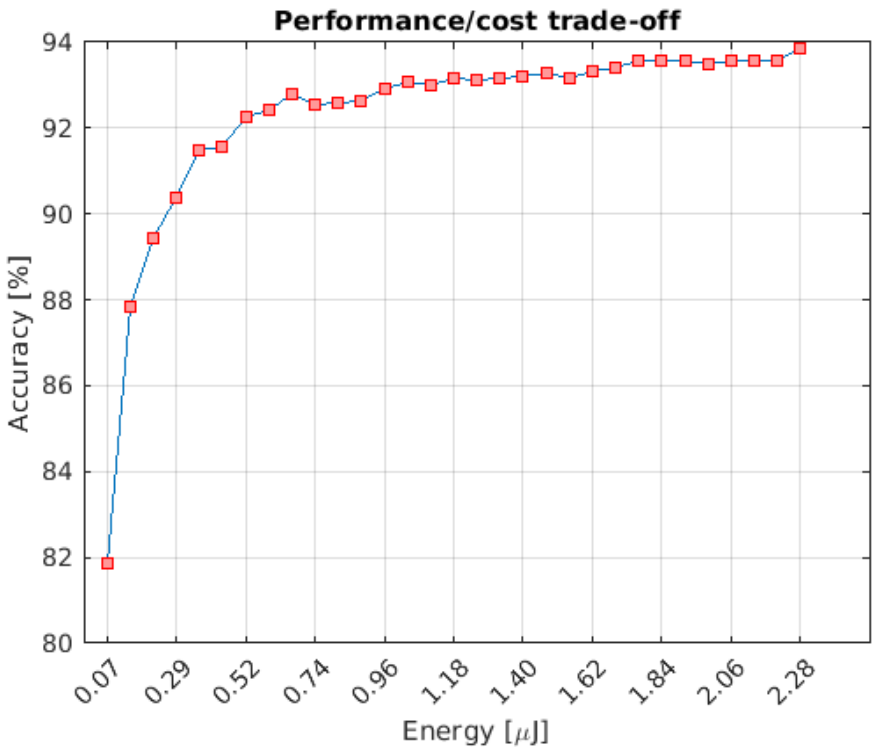


Figure 5.17: Performance/cost trade-off when scaling the learning layer from 1000 neurons to 30000 neurons for the MNIST digit classification task

[3]The classifier and spike encoder can be added to the total inference cost.

Table 5.7: Energy consumption for inference and learning in GF22

Metric	Energy
Inference	
W/o spike encoder and classifier	376.82nJ
per SOP [1]	0.09pJ
With spike encoder and classifier	377.2
per SOP [1]	0.09pJ
Learning	
Per neuron	1.27nJ
Per SOP [2]	1.5pJ
Including inference	378.4nJ

[1] SOP during inference is computed as total inference energy divided by number of synapses ($5120 \times 800 = 2M$)
SOP during learning is computed as total learning energy for one neuron divided by number of synapses per neuron (800)

Table 5.8: Latency and throughput of the digit classification task on the ASIC node

Metric	Value
Inference	
Latency	*10270 ns*
Throughput	*95020*
Learning	
Latency	*10526 ns*
Throughput	*95020*

6
Results discussion and benchmarking

In this chapter we summarize and discuss the results obtained during the experiments presented in chapter 5 and we benchmark them against similar works.

In order to have a fair evaluation of the algorithm behaviour, we established common learning parameters for all tasks and datasets. The learning parameters of interest are:

- $swap\_rate = 1$, which enforces a fast performance convergence, due to one-shot learning, i.e., all the eligible connections are moved to active input lines during learning. As a result of one-shot and fast learning, only one training epoch is necessary. However, a slower learning rate is possible by adjusting this parameter to a smaller ratio.
- $T\_learn0 = 6$ is the initial learning threshold. We chose these value empirically. Since all synaptic weights are initially random, this value should be high enough to denote a sufficient degree of coincidence between the input and a prototypical neuron model, such that learning is triggered, but small enough to ensure that the salient features are learned.
- T_fire_rate is kept to half the learning threshold throughout the experiments. This value has also been experimentally obtained after exploring a range of rate values, and it should reflect a sufficient degree of selectivity to salient features such that it indicates a meaningful prototypical shape. To avoid the situation in which a neuron has not learned anything, i.e., it's $T\_learn = T\_learn0$, and then the firing threshold would be extremely low and easily achieved, we initialize the firing threshold to a value that converges to ∞, meaning we ensure no firing occurs if not input has been learned. Since, due to the features of our algorithm, at no time can there be more than W activated lines, we set the initial $T\_fire = W + 1$.
- $K = 1$ represents the number of neurons that are allowed to learn an input. During inference, many neurons can fire for the same input. However, to avoid that the network over-generalizes the same few inputs and to decrease the learning cost, we explored the network performance when allowing only one neuron, randomly chosen from the firing pool, to learn.

6.1. Performance of the proposed online learning rule

With these fixed learning parameters, we evaluated the algorithm firstly on a simpler, binary classification task, and then on a more complex task, digit recognition, based on the MNIST dataset. On both these tasks, we first show how, provided a fixed trainset size, the classification accuracy increases with the number of neurons in the learning layer, i.e., the higher the network capacity, the better the network can generalize, until it reaches a saturation point. We demonstrated this in Figure 5.2 and Figure 5.7. Conversely, we demonstrate the network's ability to learn and improve continuously with more training data in Figure 5.3 and Figure 5.9. The results show that once the network reaches it's capacity, it stops learning. This outcome is expected in all neural networks, but it is worth noting that in this work, the learning occurs continuously on-device.

Due to the fixed $swap\_rate$ used here, it is difficult for neurons to learn new patterns once they are locked to a specific input. This can be seen in Figure 5.10, where once the number of presented samples is higher than the network capacity, the network learning capacity reaches 0. Presumably, this type of algorithm property could indicate that the neurons in our network are overfitting the learned inputs, acting like template-matching of specific inputs. However, we show that even with a small amount of training points (i.e., 5000 out of the full trainset of 60000), we reach satisfactory accuracy of over 90% on the testset. This is achieved as a result of stochasticity in our learning rule, which also prevents overfitting and allows for satisfactory generalization over the entire testset of previosuly

unseen inputs. Nevertheless, for future work, our algorithm is flexible to implementing forgetting of old patterns, by adjusting the learning parameters accordingly, for instance the $swap\_rate$ and how T_learn is updated, or by opting for LIF neurons.

On the binary classification task, we reach over 97% accuracy starting with 400 neurons, up till 98% accuracy with 900 neurons. While this accuracy is much higher than for the other tasks, it is expected due to the simpler nature of the task, that of detecting faces and ignoring any other object. Future work, however, can consider specific identification of different faces. A potential future solution can also be to extend the current network with a follow-up layer that identifies specific faces, after the previous layer indicates that a face is present in a natural image.

For the MNIST dataset, we reach over 90% with 4000 neurons (i.e., 400 neurons allocated per class), and up to 94% accuracy when using 42000 neurons (i.e., 4200 neurons allocated per class). This accuracy is lower than some state-of-the art local learning rules with STDP [94], [95], since we use very low-complexity learning, with fully binary activations and weights, however, it still is comparable with most recent work on local learning with SNNs for the Edge [46], [43]. We elaborate on this comparison in section section 6.3. We also report the confusion matrix which shows the accuracy reached for each digit in Figure 5.8, when 30000 neurons are used to reach 93.51% accuracy. It is noteworthy to mention that, for MNIST, we noticed more than 5% of the accuracy is lost on the correct prediction being the second one. This means that, with a better and more complex classifier, our network has the potential to achieve much higher performance.

We have also reported the accuracy obtained on the Fashion-MNIST dataset, with 30000 neurons, namely 77.65%, which enforces the dataset-agnosticism property of our algorithm. This result is higher than in other works, i.e., in [46], the authors propose an online learning rule that achieves 93.54% accuracy on MNIST, but only 59.46% on Fashion-MNIST. Thus, our network is highly dataset-agnostic compared to other solutions.

In what concerns the network robustness to variations in the testset input data, we showed in Figure 5.11 that the network keeps robust accuracy up till 5 degrees rotations, and shows a 1 to 2 % loss for rotation degrees between 5 and 10.

Another result we report concerns the impact a smaller input size (i.e., image resolution) has on the network performance. We showcase this result in Figure 5.4, which shows a small loss in accuracy for down-scaling the inputs. This result is relevant particularly for extreme Edge resource-constrained devices, since the computational cost of our network scales with the input size.

6.2. Online adaptation with the proposed rule

On-device learning could be a critical aspect of EdgeAI systems when there is a shortage of labeled data to train the network before deployment. However, fully supervised on-device learning experiments cannot simulate such scenarios. To demonstrate the scenario where the labeled data is too little, but the network can adapt to new inputs in a unsupervised manner, we carried out an experiment, on the binary classification task, using a small amount of face data to initially pre-train the network. The pre-trained network is then used to self-supervise subsequent learning steps. After pre-training, the network receives a random selection of input data (with and without faces). If the input data is classified as a face (at least one neuron firing), the self-supervised learning process allows the network to learn the input data. Figure 5.5 shows how the accuracy of the self-supervised learning process increases by providing more face data. However, since the input data may be misclassified as a face, the self-supervised learning results in lower accuracy than the fully-supervised learning result in Figure 5.3. Therefore, the quality of the pre-trained network significantly affects the final accuracy. This effect can be exacerbated for more complex classification tasks, such as MNIST, where a mis-prediction will trigger learning an input by the wrong cluster. However, with this simple example, we demonstrate that a self-supervised learning mechanism, when combined with on-device learning, can improve the network's accuracy during inference time.

6.3. Hardware results and benchmarking against similar work

6.3.1. General hardware results

For hardware benchmarking and evaluation, we use a training layer with 5120 neurons. This is a trade-off we choose between algorithm performance and computational cost, in order to better evaluate against similar solutions. This stems from the fact that our network uses a layer of many simple IF neurons, and a light-weight training rule, and thus updating all fan-out neurons can lead in a higher inference time for our solution, while most works use more complex modelling for the neuron and synaptic updates rule, which allows them to obtain similar or better accuracy with much less neurons and synapses. However, in Figure 5.17, we show the trade-off between computational cost during inference and accuracy, which facilitates a more loyal comparison between our work and other solutions, with respect to the cost needed to achieve the same accuracy.

We note that, in our design, the inference energy cost is the bottleneck, due to the need to update all fan-out neurons. In our work, neuron memory (i.e., synaptic weights and neuron parameters) reads consume over 99% of memory during inference. However, given that the neuron model can be implemented with binary MAC, future work can consider alleviating the memory access overhead by employing Computation-in-memory technologies, which can enable significant energy improvement, according to results previously reported for BNN architectures [96]. Similarly, in terms of area, neuron memory occupies more than 95% of the area. Future work can consider optimizing the synaptic weights area and memory access energy by using a more efficient encoding scheme: i.e., since we only keep $W = 64$ synaptic connections on, we can use for instance a synaptic encoding based on the address at which the synaptic connection is on and the orientation. For instance, for the $im\_res = 32$ case, there are 784 output points, each representing a code between 0 and 8. Since we only need to know at which address there is a valid orientation, we need 10 bits to represent the address and 3 bits to represent the orientation., thus 13 bits per code. We have only $W = 64$ valid codes, thus $64 \times 13 = 832$bits. Compared to the current encoding scheme, using 4bits per code, such a scheme would result in $4 \times 784/832 = 3.76$ times reduction in neuron area and memory accesses, given a input $im\_res = 32$. Another possible future avenue for reducing the inference energy consumption is exploring whether a few random neuron updates per cluster could accurately indicate recognition.

We further compare our hardware results to similar solutions that employ the MNIST dataset for benchmarking, that are implemented both in ASIC and FPGA. We note that the goal of our algorithm is not to solve the MNIST task, which has already been solved by classical DNNs with more than 99.84% accuracy [97]. However, over the past years, for the emerging field of neuromorphic computing, the MNIST dataset has constituted a common ground for benchmarking the hardware efficiency of various learning solutions and SNNs networks. Moreover, as we have already demonstrated, our algorithm is suitable for various datasets.

Without further ado, the key result of these comparisons is the learning overhead for inference. Since most works do not report whether the learning cost includes the inference cost, we assume it does, and thus we compute the overhead as *(learning energy - inference energy)/inference energy*: compared to all similar solutions, our work achieves the best learning overhead, of under 1%, which is at least 10x better than other works. In that sense, to our knowledge, our work achieves the state-of-the-art in SNN processors with online learning on-device that incurs minimum overhead for inference.

6.3.2. Benchmarking against similar FPGA solutions

Table 6.1 compares EON-1 to other similar online learning with SNNs solutions implemented on FPGA platforms and benchmarked on MNIST. Metrics of interest are the reported accuracy for online learning, resource utilization and throughput. Overall, EON-1 has the least resource utilization, compared to the works that report this metric. The LUT utilization of the neuron core in [98] is similar to our work, however, updating the membrane potential in their proposal is performed over several clock cycles, as compared to one clock cycle in our work. We compute the learning overhead based on power consumption reported by these works for learning and inference. The work in [99] has much higher accuracy compared to our work, however, their learning uses surrogate gradient, and although a much smaller number of neurons are needed, the resource utilization is more than $2\times$ higher than ours, and

the throughput achieved during inference is $16\times$ lower. Overall, our solution achieved the highest throughput.

6.3.3. Benchmarking against similar ASIC solutions

Table 6.1 compares EON-1 to other similar online learning with SNNs solutions implemented in ASIC nodes and benchmarked on MNIST. Metrics of interest are the obtained accuracy for online learning, the core area, energy consumption and achievable throughput. The total core area we report includes the cost of spike encoding and classification on-device. Since similar works are benchmarked using various ASIC nodes, we use the method in [100] to normalize area and energy to our node. However, we note that due to complex differences between various technology nodes, e.g., variety of short-channel effects, the scaling in [100] is not an accurate predictor [101], however, for the purpose of this project, they help in providing an overall comparison estimation. Our design does not include I/O power and it is measured post-synthesis using time-based power analysis within 15% signoff. By far, our design is not the most energy or area efficient, however, it achieves comparable cost with state-of-the-art, provided that we have the best learning overhead for inference, and we use stochastic learning with fully binary weights and activations.

In terms of energy consumption, similar work also report the SOP (synaptic operation) energy during inference and learning. Since SOP cost depends on the synaptic rate in a proposed design, which can vary greatly from one work to another, the SOP energy is not a good estimate of the energy efficiency of a proposed custom SNN processor following a top-down approach, however it is useful for bottom-up neuromorphic hardware architectures which can be used for a higher variety of tasks. Moreover, authors do not use a standard method for providing this method. However, since our design fits within the top-down design approach for neuromorphic processors, reporting SOP here can offer a good indication of the computational complexity of a proposed method.

The work in [44] is the most similar to ours, in terms of using a stochastic and binary weights online learning rule. They also update only one neuron per learned input. In terms of area and energy efficiency, they outperform our solution, however, this comes at the cost of less on-device accuracy and higher learning cost and overhead for inference. A similar situation occurs compared to [40]: the energy consumption during inference is much less than ours, based on a rank order encoding scheme, however, the reported accuracy is much less. We note that we could reach a similar accuracy with 1500 neurons, which would cut our energy consumption during inference by a factor of 3.4x.

Overall, the SOP energy we achieve is more efficient than all similar works, except for the work in [44], i.e., they have 7x and 3x energy efficiency for SOP during learning and inference, respectively. We note, however, that for reaching a similar accuracy, we could use a network of 2000 neurons, which results in a similar energy per SOP during inference. Our learning cost is however fixed, independent of the learning layer size.

6.4. Benchmarking proposed solution for UHD frame processing

To demonstrate the benefits of our solution, we proposed a test case where we evaluate fast inference while coping with streaming data in UHD videos. Starting from the binary classification experiment, we created a collage of many small faces and Images from the CIFAR-10 dataset, randomly placed in a UHD frame. We first evaluated the performance of our recognition engine, showing that our algorithm correctly spots 720 out of 798 the faces, while our EON-1 processor takes less than 84ms to process the full frame of small faces, without any downscaling. By contrast, humans can only spot 2 faces in a crowd, at a time [92].

We then evaluated the achievable frame rate for practical UHD applications, where with a down-scaling factor of at least 6.25 we can easily process UHD videos in real-time, at 68 FPS. This result demonstrates the unique features of our solution, showing that we can achieve fast inference while coping with streaming data in real-time UHD videos, at the Edge. For future work, we plan to extend this benchmark to a real-time pattern recognition demonstrator.

Table 6.1: Comparison of EON-1 with other FPGA solutions benchmarked on MNIST

	TCSI'21 [69]	**TCSII'21 [98]**	**Neuro'17 [70]**	**ICTA'23 [99]**	**EON-1**
Accuracy	85.28 %	90.58%	89.1%	95.49 %	91.96%
Learning					
Online	yes	yes	yes	NR	yes
On-chip	yes	no	yes	yes	yes
Rule	STDP var.	STDP var.	STDP	SG [1]	S-BSP [2]
Neurons					
Model	LIF	LIF	LIF	LIF	IF
Number	300	2304	1591	320	5120
Synapses	176800	NR	638208	NR	4M
Precision	16bit float	2bit	16bit fixed	8bit	1-bit
Encoding	rate	rate	rate	rate	TTFS
on-chip	no	yes	no	no	yes
Chipset (*)	Virtex-7	ZCU102	Virtex-6	VC707	VU37P HBM
Clock Frequency	100MHz	200MHz	120MHz	115MHz	100MHz
LUT				22779	
Spike encoder	-	113	-	-	6010
Neuron core	NR	455	69781	NR	449
Learning	NR	-	1817	NR	1518
Classifier	NR	1641	-	-	76
FF				15072	
Spike encoder	-	84	-	-	744
Neuron core	NR	85	50771	NR	7
Learning	NR	-	134	NR	845
Classifier	NR	278	-	-	41
CLB					
Spike encoder	-	NR	NR	NR	969
Neuron core	NR	NR	NR	NR	152
Learning	NR	NR	NR	NR	447
Classifier	NR	NR	NR	NR	20
BRAM					
Spike encoder	-	1	-	NR	0
Neuron core	NR	-	-	NR	58
Syn. mem.	NR	431.5	68	NR	0
Learning	NR	-	136	NR	0
Classifier	NR	20	-		
Energy					
Learning	26.32mJ	-	1.33J	0.67mJ	-
Inference	5.04mJ	17mJ	1.12J	0.21mJ	-
Throughput (FPS)					
Learning	61	-	0.05	NR	19.1K
Inference	285	46.44	0.11	1183	19.4K
Learning overhead for inference	5.22	-	0.18	2.19	**<0.01**

NR - Not Reported [1] Surrogate gradient [2] Stochastic Binary STDP

Table 6.2: Comparison of EON-1 with other ASIC solutions benchmarked on MNIST

	TCSI'22 [43]	**TBCAS"19 [44]**	**TBCAS'19 [40]**	**TCSII'23 [46]**	**ISCAS'20 [45]**	**EON-1**
Accuracy	93%	87.4%	84.5%	93.54 %	92.8% - 95.3% [2]	91.96%
Learning						
Online	yes	yes	yes	yes	yes	yes
On-chip	yes	yes	yes	yes	yes	yes
Rule	STDP var.	spike count	SDSP	add-STDP [1]	DRTP [3]	S-BSP [4]
Neurons						
Model	LIF	IF	LIF/Izk.	LIF	NR	IF
Number	384	400	256	2048	128	5120
Synapses	176800	230400	64000	2M	NR	4M
Precision	9-bit fixed	1-bit	4-bit	8-bit	8-bit	1-bit
Encoding	temporal	rate	rate & rank [7]	temporal	TTFS	TTFS
on-chip	yes	no	no	NR	yes	yes
ASIC Node	28nm	65nm	28nm	28nm	28nm	22nm
Clock Frequency	333MHz	384MHz	75MHz	500MHz	150MHz	500MHz
Voltage	0.9V	1.2V	0.55V	0.81V	0.6V	0.8V
Area (mm^2)	1	0.39	0.086	6.22	0.26	0.558
Norm. area [5](mm^2)	0.617	0.044	0.053	3.83	0.16	0.558
Energy						
Learning	660nJ	2630nJ	105nJ	NR	NR	378.4nJ
SOP	NR	1.42pJ	NR	4.99pJ/SOP	NR	1.5pJ
Inference	500nJ	310nJ	15nJ–404nJ	NR	313nJ	377.2nJ
SOP	NR	0.26pJ	12.7pJ	1.28pJ/SOP	NR	0.09pJ
Norm. energy [6]						
Learning	409.7nJ	395.62nJ	174nJ	NR	NR	378.4nJ
SOP	NR	0.21pJ	NR	3.82pJ/SOP	NR	1.5pJ
Inference	310.4nJ	46.6nJ	24.93nJ–671.58nJ	NR	437.2nJ	377.2nJ
SOP	NR	0.03pJ	21.11pJ	0.98pJ/SOP	NR	0.09pJ
Throughput (FPS)						
Learning	211.77k	NR	NR	22.8GSOPS	NR	95K
Inference	277.78k	NR	NR	81.92GSOPS	8.5K	97.6K
Learning overhead for inference [9]	0.32	7.5	7 & 0.25 [7]	2.89	0.16 [8]	**<0.01**

NR - Not Reported [1] Additive STDP [2] Reported accuracy for one epoch is 92.8%, and after 100 epochs it reaches 95.3% [45]. [3] Direct Random Target Projection [45] - a modified back-propagation version, suitable for online and local learning [4] Stochastic Binary STDP [5] Normalized area = Area × (22/Node)^2
[6] Normalized energy = Energy × (22/Node) × (0.8/voltage)^2 [7] The neuromorphic processor in [40] supports both rate coding and temporal, rank-order coding for input spikes. Reported inference energy and learning overhead is for rank and rate coding, respectively. Reported learning energy excludes inference cost.
[8] Learning overhead is obtained from reported power.
[9] Energy overhead computed as *(learning energy - inference energy)/inference energy*
TCSI'22 [43], TBCAS"19 [44] report area/energy results post-layout and they do not include I/O area/energy cost; TBCAS'19 [40], TCSII'23 [46], ISCAS'20 [45] report area/energy results post-tapeout.

7
Conclusions and future work

7.1. Overview and summary of results

Motivated by the desire to bring intelligent processing at the Edge, particularly to perform online learning and adaptation with minimum incurred overhead for inference, in this work we have proposed EON-1, an Edge ONline Learning SCNN (Spiking Convolutional Neural Network) processor with 1-bit synaptic weights, 1-spike per neuron and 1-neuron updated per input, which we have benchmarked for both ASIC and FPGA platforms. To this aim, taking inspiration from URVC in the biological visual system, we have explored the benefits of using a SNN of IF neurons with stochastic plasticity, binary spikes and synaptic weights, coupled with a Time-to-First-Spike encoding scheme and lateral-inhibition among orientation-selective cells, for Edge AI applications. Our key contribution is proposing a binary and stochastic SDTP rule which, benchmarked in an ASIC node, achieves less than 1% energy overhead for inference. To our knowledge, our solution incurs the least energy overhead for inference, compared to state-of-the-art solutions, showing a better efficiency by at least a factor of 10x. We also report 94% and 77.65% accuracy on the MNIST and Fashion-MNIST classification tasks, and we achieve 0.09pJ/SOP and 1.5pJ/SOP during inference and learning, respectively. We extend our results to demonstrate how we can achieve 60 FPS UHD video processing. Following, we discuss and conclude on the main results achieved by our work.

7.1.1. Online learning rule performance

Through our experiments, we have explored the boundaries of learning locally, in a single trainable layer, with extreme spike-efficiency. To that extent, we have evaluated our proposed method on a series of tasks and datasets, demonstrating 94% accuracy on MNIST, 77.65% on Fashion-MNIST, and 98% accuracy on a binary classification task between faces and no-faces. We first showed that our network is able to improve continuously with more training data being presented online, and that the network performance increases with more neurons in the training layer. We have also demonstrated that with limited training data, and through one-shot learning, our network reaches satisfactory accuracy. As a result of stochasticity in the proposed learning rule, our solution is resilient to overfitting, however, no forgetting of old-patterns is implemented at this point; the neurons, however, are able to detect slight changes in the input patterns (e.g., we demonstrate this through presenting rotated versions of the trainset during the MNIST classification task) and future work can consider implementing tracking these changes.

Based on a face/no face binary classification example, we have demonstrated that a self-supervised learning mechanism, when combined with on-device learning, can improve the network's accuracy during inference time, thus showing the online adaptation capability of our network. We note, however, that inaccurate predictions can lead to false positives being learned by neurons (i.e., the inference prediction is subsequently used as a label), which can result in a decrease in accuracy for more complex tasks (e.g., multi-class classification tasks).

In terms of scalability, we have shown that our network is scalable in terms of learning layer size and input resolution. To that extent, we have provided an analysis of what the benefits and disadvantages of scaling are: for scaling the learning layer size, we have shown that the network performance increases with the network capacity. However, this comes at the cost of higher energy and latency during inference. In what concerns the input resolution, our algorithm is suitable for extreme down-scaling, benefiting the overall computational cost of the network, however, this comes at the cost of a couple of percentiles accuracy loss.

7.1.2. Hardware results and benchmarking

To evaluate the hardware efficiency benefits for Edge AI applications, we have benchmarked our solution in a GF22 ASIC node, and a VU37 HBM FPGA platform. The ASIC harwdare results show that our processor consumes 0.09pJ/SOP and 1.5pJ/SOP, during inference and learning respectively. Based on 5120 neurons and 4M synapses, our processor occupies 0.558 mm^2, and consumes 377.2nJ during inference and only 1.2nJ for learning.

Based on end-to-end network implementation, our solution achieves the best throughput and resource utilization compared to similar works benchmarked on FPGA platforms. Compared to similar works benchmarked in ASIC, although we have the highest number of neurons and synapses, and we use only 1-bit synaptic weights, we achieve comparable accuracy and energy efficiency to state-of-the-art, while maintaining the best learning energy overhead for inference, which is at least 10x better than other works.

To demonstrate the unique features of our solutions, we have shown that our network is suitable for fast inference while coping with streaming data in UHD video, being able of achieving standard 60FPS processing. Since the number of neurons in the network is correlated to the number of different classes and patterns that can be learned/recognized by our engine, we provided an in-depth analysis of the trade-off between achievable UHD frames throughput and network size on a high-end FPGA platform, VU37 HBM. In less than 84ms, our engine spots 720 out of 798 faces in a UHD collage frame of 8100 small patterns.

7.2. Future work

In this work, we have explored the boundaries of learning locally with a single layer of simple IF neurons, with binary synaptic weights and activations, while enforcing one-shot learning and only 64 active synaptic weights per neuron. While our current algorithm version demonstrates dataset-agnosticism, robustness to variations in the input, fast and energy efficient learning and inference and can be extended to practical applications such as real-time inference on UHD streams, there are multiple avenues for future improvement.

Firstly, our current algorithm version locks neurons to a specific input, due to fast, one-shot learning. For future work, we want to explore using a smaller swap rate, such that one neuron can become selective to more inputs that are highly-correlated. A slower learning rate could also result in our network behaving like an associative memory, e.g., our network could better recall patterns that are degraded or only partially resemble learned inputs [102]. Moreover, since we currently concentrate a fixed number of weights to random active input lines, future work can consider exploring performance when swapping weights only to the most salient features, which we could obtain by sorting the convolution values of the edge filters.

Another aspect of our current algorithm is that we do not implement any forgetting of old patterns. For future work, we consider exploring decaying the learning threshold such that forgetting is facilitated and neurons can be reused to learn new patterns or track changing versions of the old patterns (e.g., detect an aging face or other slight variations with time of learned inputs).

In our current design implementation, synaptic memory access is responsible for more than 99% of the energy consumption during inference and more than 95% of the total processor area. We consider three main avenues that could alleviate the dominating energy consumption incurred by memory accesses and improve area for future work:

- Since the main operation during our network's inference is a binary multiply-and-accumulate, the first avenue is employing emerging technologies such as computation-in-memory (CIM). Previous work has shown that compared to conventional row-by-row SRAM access, CIM can achieve more than $30\times$ energy improvement for inference of BNNs [103].
- the second avenue is to optimize the synaptic weights encoding scheme: since we only keep $W = 64$ synaptic connections active, we can use for instance a synaptic encoding based on the address at which the synaptic connection is active, followed by the orientation at that address. For instance, for the $im\_res = 32$ case, there are 784 output points, each representing a code between 0 and 8. Since we only need to know at which address there is a valid orientation, we need 10 bits to represent the address and 3 bits to represent the orientation, thus 13 bits per code.

We have only $W = 64$ valid codes, thus $64 \times 13 = 832$bits. Compared to the current encoding scheme, using 4bits per code, such a scheme would result in $4 \times 784/832 = 3.76$ times reduction in neuron area and memory accesses, given an input $im\_res = 32$. This encoding scheme would however need extra decoding logic of the neuron synaptic weights.

- the third future avenue for reducing the inference energy consumption is exploring whether a few random neuron updates per cluster could accurately indicate recognition, i.e., we can explore the trade-off between less energy consumption (and invariably, less latency) and the prediction confidence.

In terms of improving the network accuracy on typical classification benchmarks (e.g., MNIST, Fashion-MNIST), future work can explore either improving the classifier or exploring in depth the situations where the correct prediction is precisely the second. Another possible avenue is to explore the benefits of extending our solutions to multiple learning layers.

Lastly, future work considers building an FPGA-based demonstrator for the real-time UHD video inference.

Bibliography

[1] H. Paugam-Moisy and S. M. Bohte, "Computing with spiking neuron networks." *Handbook of natural computing*, vol. 1, pp. 1–47, 2012.

[2] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 15, no. 2, p. 1–35, Apr. 2019. [Online]. Available: http://dx.doi.org/10.1145/3304103

[3] K. Yamazaki, V.-K. Vo-Ho, D. Bulsara, and N. Le, "Spiking neural networks and their applications: A review," *Brain Sciences*, vol. 12, no. 7, p. 863, 2022.

[4] S. J. Thorpe, A. Delorme, and R. VanRullen, "Spike-based strategies for rapid processing," *Neural Networks*, vol. 14, pp. 715–726, 2001.

[5] S. Saha. (2018) A comprehensive guide to convolutional neural networks. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[6] Z. Keita. (2023) An introduction to convolutional neural networks (cnns). [Online]. Available: https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns

[7] T. Masquelier and S. J. Thorpe, "Learning to recognize objects using waves of spikes and spike timing-dependent plasticity," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–8.

[8] A. Delorme, J. Gautrais, R. van Rullen, and S. Thorpe, "Spikenet: A simulator for modeling large networks of integrate and fire neurons," *Neurocomputing*, vol. 26-27, pp. 989–996, 1999.

[9] A. Delorme, L. Perrinet, and S. Thorpe, "Networks of integrate-and-fire neurons using rank order coding b: Spike timing dependent plasticity and emergence of orientation selectivity," *Neurocomputing*, vol. 38, pp. 539–545, 06 2001.

[10] (2017) Utkface - large scale face dataset. [Online]. Available: https://susanqq.github.io/UTKFace/

[11] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[12] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[13] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[14] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," 2022.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[16] ImageNet. (2023) Imagenet. [Online]. Available: https://www.image-net.org/about.php

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[19] N. Alarcon. (2020) Openai presents gpt-3, a 175 billion parameters language mode. [Online]. Available: https://developer.nvidia.com/blog/openai-presents-gpt-3-a-175-billion-parameters-language-model

[20] J. Langston. (2020) Microsoft announces new supercomputer, lays out vision for future ai work. [Online]. Available: https://news.microsoft.com/source/features/innovation/openai-azure-supercomputer

[21] L. F. e. a. N. Maslej, "The ai index 2023 annual report," Institute for Human-Centered AI,Stanford University, Stanford, CA, Tech. Rep., 2023.

[22] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of bloom, a 176b parameter language model," 2022.

[23] J. Li, J. Chen, H. Bai, H. Wang, S. Hao, Y. Ding, B. Peng, J. Zhang, L. Li, and W. Huang, "An overview of organs-on-chips based on deep learning," *Research*, vol. 2022, 2022.

[24] K. Ueyoshi, I. A. Papistas, P. Houshmand, G. M. Sarda, V. Jain, M. Shi, Q. Zheng, S. Giraldo, P. Vrancx, J. Doevenspeck, D. Bhattacharjee, S. Cosemans, A. Mallik, P. Debacker, D. Verkest, and M. Verhelst, "Diana: An end-to-end energy-efficient digital and analog hybrid neural network soc," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

[25] N. P. Jouppi and e. a. Cliff Young, "In-datacenter performance analysis of a tensor processing unit," 2017.

[26] H. Liao, J. Tu, J. Xia, H. Liu, X. Zhou, H. Yuan, and Y. Hu, "Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 789–801.

[27] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, 2021.

[28] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," 2016.

[29] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kb memory," 2022.

[30] Y. Yang, G. Li, and R. Marculescu, "Efficient on-device training via gradient filtering," 2023.

[31] e. a. Jan M. Rabaey, Marian Verhelst, "Ai at the edge - a roadmap. technical report," IMEC, KU Leuven, Ghent University, VUB, EPFL, ETH Zurich and UC Berkeley, Tech. Rep., 2019.

[32] J. Shalf, "The future of computing beyond moore's law," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2166, p. 20190061, 2020.

[33] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, 2020.

[34] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. A. Behram, J. Huang, C. Bai, M. Gschwind, A. Gupta, M. Ott, A. Melnikov, S. Candido, D. Brooks, G. Chauhan, B. Lee, H.-H. S. Lee, B. Akyildiz, M. Balandat, J. Spisak, R. Jain, M. Rabbat, and K. Hazelwood, "Sustainable ai: Environmental implications, challenges and opportunities," 2022.

[35] O. R. N. Laboratory. (2023) Oak ridge national laboratory - frontier. [Online]. Available: https://www.olcf.ornl.gov/frontier/

[36] A. Madhavan. (2023) Brain-inspired computing can help us create faster, more energy-efficient devices — if we win the race. [Online]. Available: https://www.nist.gov/blogs/taking-measure/brain-inspired-computing-can-help-us-create-faster-more-energy-efficient#:~:text=The%20human%20brain%20is%20an,just%2020%20watts%20of%20power

[37] K. Roy, A. Jaiswal, and P. Panda, “Towards spike-based machine intelligence with neuromorphic computing,” *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[38] C. Frenkel, D. Bol, and G. Indiveri, “Bottom-up and top-down approaches for the design of neuromorphic processing systems: Tradeoffs and synergies between natural and artificial intelligence,” *Proceedings of the IEEE*, vol. 111, no. 6, pp. 623–652, 2023.

[39] J. Stuijt, M. Sifalakis, A. Yousefzadeh, and F. Corradi, “µbrain: An event-driven and fully synthesizable architecture for spiking neural networks,” *Frontiers in Neuroscience*, vol. 15, 2021.

[40] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, “A 0.086-mm^2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 1, pp. 145–158, 2019.

[41] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, “Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013.

[42] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, “Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[43] C. Sun, H. Sun, J. Xu, J. Han, X. Wang, X. Wang, Q. Chen, Y. Fu, and L. Li, “An energy efficient stdp-based snn architecture with on-chip learning,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 12, pp. 5147–5158, 2022.

[44] H. Tang, H. Kim, H. Kim, and J. Park, “Spike counts based low complexity snn architecture with binary synapse,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1664–1677, 2019.

[45] C. Frenkel, J.-D. Legat, and D. Bol, “A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas,” in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[46] Y. Zhong, Z. Wang, X. Cui, J. Cao, and Y. Wang, “An efficient neuromorphic implementation of temporal coding-based on-chip stdp learning,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 11, pp. 4241–4245, 2023.

[47] Z. Mai, R. Li, J. Jeong, D. Quispe, H. Kim, and S. Sanner, “Online continual learning in image classification: An empirical survey,” 2021.

[48] W. Maass, “Networks of spiking neurons: the third generation of neural network models,” *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[49] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov, “Spiking neural networks and online learning: An overview and perspectives,” *Neural Networks*, vol. 121, pp. 88–100, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608019302655

[50] S. J. Thorpe and J. Gautrais, “Rank order coding,” *Computational Neuroscience: Trends in Research*, pp. 113–118, 1998.

[51] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama, “Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems,” *Frontiers in Neuroscience*, vol. 15, 2021.

[52] E. Izhikevich, “Simple model of spiking neurons,” *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[53] C. Frenkel, J.-D. Legat, and D. Bol, “Morphic: A 65-nm 738k-synapse/mm^2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 999–1010, 2019.

[54] A. Yousefzadeh, E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, “On practical issues for stochastic stdp hardware with 1-bit synaptic weights,” *Frontiers in Neuroscience*, vol. 12, 2018. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00665

[55] M. Fabre-Thorpe, “The characteristics and limits of rapid visual categorization,” *Frontiers in Psychology*, vol. 2, 2011.

[56] I. Tomomi and H. Samar, “On and off signaling pathways in the retina and the visual system,” *Frontiers in Ophthalmology*, vol. 2, 2022.

[57] S. Thorpe and R. V. Rullen, “Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex,” *Neural computation*, vol. 13, no. 6, pp. 1255–1283, 2001.

[58] S. Thorpe, D. Fize, and C. Marlot, “Speed of processing in the human visual system,” *nature*, vol. 381, no. 6582, pp. 520–522, 1996.

[59] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, “Hfirst: A temporal approach to object recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 10, pp. 2028–2040, 2015.

[60] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, “Robust object recognition with cortex-like mechanisms,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp. 411–426, 2007.

[61] T. Masquelier and S. J. Thorpe, “Unsupervised learning of visual features through spike timing dependent plasticity,” *PLoS computational biology*, vol. 3, no. 2, p. e31, 2007.

[62] R. Rullen, J. Gautrais, A. Delorme, and S. Thorpe, “Face processing using one spike per neuron,” *Biosystems*, vol. 48, pp. 229–239, 1998.

[63] S. Thorpe, “Ultra-rapid scene categorization with a wave of spikes,” in *Biologically Motivated Computer Vision*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 1–15.

[64] P. Pietrzak, S. Szczęsny, D. Huderek, and □. Przyborowski, “Overview of spiking neural network learning approaches and their computational complexities,” *Sensors*, vol. 23, no. 6, 2023.

[65] H. Wu, Y. Zhang, W. Weng, Y. Zhang, Z. Xiong, Z.-J. Zha, X. Sun, and F. Wu, “Training spiking neural networks with accumulated spiking flow,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 10 320–10 328.

[66] S. M. Bohte, J. N. Kok, and J. A. La Poutré, “Spikeprop: backpropagation for networks of spiking neurons.” in *ESANN*, vol. 48. Bruges, 2000, pp. 419–424.

[67] A. Tavanaei and A. Maida, “Bp-stdp: Approximating backpropagation using spike timing dependent plasticity,” *Neurocomputing*, vol. 330, pp. 39–47, 2019.

[68] F. Liu, W. Zhao, Y. Chen, Z. Wang, T. Yang, and L. Jiang, “Sstdp: Supervised spike timing dependent plasticity for efficient spiking neural network training,” *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2021.756876

[69] S. Li, Z. Zhang, R. Mao, J. Xiao, L. Chang, and J. Zhou, “A fast and energy-efficient snn processor with adaptive clock/event-driven computation scheme and online learning,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 4, pp. 1543–1552, 2021.

[70] Q. Wang, Y. Li, B. Shao, S. Dey, and P. Li, “Energy efficient parallel neuromorphic architectures with approximate arithmetic on fpga,” *Neurocomputing*, vol. 221, pp. 146–158, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231216311213

[71] A. S. Cassidy, J. Georgiou, and A. G. Andreou, “Design of silicon brains in the nano-cmos era: Spiking neurons, learning synapses and neural architecture optimization,” *Neural Networks*, vol. 45, pp. 4–26, 2013, neuromorphic Engineering: From Neural Systems to Brain-Like Engineered Systems.

[72] Y. Zhong, X. Cui, Y. Kuang, K. Liu, Y. Wang, and R. Huang, “A spike-event-based neuromorphic processor with enhanced on-chip stdp learning in 28nm cmos,” in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[73] J. D. Nunes, M. Carvalho, D. Carneiro, and J. S. Cardoso, “Spiking neural networks: A survey,” *IEEE Access*, vol. 10, pp. 60 738–60 764, 2022.

[74] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

[75] S. J. Thorpe and M. Imbert, “Biological constraints on connectionist modelling,” *Connectionism in perspective*, pp. 63–92, 1989.

[76] T. J. Gawne, T. W. Kjaer, and B. J. Richmond, “Latency: another potential code for feature binding in striate cortex,” *Journal of neurophysiology*, vol. 76, no. 2, pp. 1356–1360, 1996.

[77] T. Gollisch and M. Meister, “Rapid neural coding in the retina with relative spike latencies,” *Science*, vol. 319, no. 5866, pp. 1108–1111, 2008.

[78] S. Thorpe, *”Timing, spikes, and the brain” in Time and Science (In 3 Volumes): Volume 2: Life Sciences*, ser. G - Reference,Information and Interdisciplinary Subjects Series. World Scientific Publishing (UK)Limited, 2023.

[79] S. Thorpe, A. Yousefzadeh, T. Masquelier, and J. Martin, “Unsupervised learning of repeating patterns using a novel stdp based algorithm,” *Journal of Vision*, vol. 17, no. 10, pp. 1079–1079, 2017.

[80] A. Tavanaei, T. Masquelier, and A. S. Maida, “Acquisition of visual features through probabilistic spike-timing-dependent plasticity,” in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016. [Online]. Available: http://dx.doi.org/10.1109/IJCNN.2016.7727213

[81] H.-K. Kwan and T. Okullo-Oballa, “2-d systolic arrays for realization of 2-d convolution,” *IEEE Transactions on Circuits and Systems*, vol. 37, no. 2, pp. 267–233, 1990.

[82] P. Alfke. (1996) Efficient shift registers, lfsrcounters, and long pseudorandom sequence generators. [Online]. Available: https://docs.xilinx.com/v/u/en-US/xapp052

[83] B. A. Olshausen and D. J. Field, “Emergence of simple-cell receptive field properties by learning a sparse code for natural images,” *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[84] J. Yin and Q. Yuan, “Structural homeostasis in the nervous system: a balancing act for wiring plasticity and stability,” *Frontiers in cellular neuroscience*, vol. 8, p. 439, 2015.

[85] B. Moons, D. Bankman, L. Yang, B. Murmann, and M. Verhelst, “Binareye: An always-on energy-accuracy-scalable binary cnn processor with all memory on chip in 28nm cmos,” 2018.

[86] T. M. Inc. (2022) Matlab (r2022b). [Online]. Available: https://nl.mathworks.com/products/new_products/release2020b.html

[87] Cadence. Cadence xcelium logic simulator. [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/simulation-and-testbench-verification/xcelium-simulator.html

[88] Xilinx. (2020) Vivado design suite user guide. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/sw_manuals/xilinx2020_1/ug973-vivado-release-notes-install-license.pdf

[89] AMD. Virtex ultrascale+ hbm vcu128 fpga evaluation kit. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/vcu128.htmlf

[90] Cadence. Genus synthesis solutions. [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html

[91] ——. Joules rtl power solution. [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/power-analysis/joules-rtl-power-solution.html

[92] V. Thoma, "Face-specific capacity limits under perceptual load do not depend on holistic processing," *Psychonomic bulletin & review*, vol. 21, pp. 1473–1480, 2014.

[93] AMD. (2022) Vcu128 evaluation board user guide (ug1302). [Online]. Available: https://docs.xilinx.com/r/en-US/ug1302-vcu128-eval-bd

[94] P. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, 2015.

[95] J. M. Brader, W. Senn, and S. Fusi, "Learning real-world stimuli in a neural network with spike-driven synaptic dynamics," *Neural computation*, vol. 19, no. 11, pp. 2881–2912, 2007.

[96] T. Wang and W. Shan, "An energy-efficient in-memory bnn architecture with time-domain analog and digital mixed-signal processing," in *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2019, pp. 1–6.

[97] A. Byerly, T. Kalganova, and I. Dear, "No routing needed between capsules," *Neurocomputing*, vol. 463, pp. 545–553, 2021.

[98] H. Zheng, Y. Guo, X. Yang, S. Xiao, and Z. Yu, "Balancing the cost and performance trade-offs in snn processors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 9, pp. 3172–3176, 2021.

[99] K. Huang, W. Liu, Y. Liu, S. Xiao, and Z. Yu, "Towards efficient on-chip learning for spiking neural networks accelerator with surrogate gradient," in *2023 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)*, 2023, pp. 89–90.

[100] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital integrated circuits*. Prentice hall Englewood Cliffs, 2002, vol. 2.

[101] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180nm to 7nm," *Integration*, vol. 58, pp. 74–81, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926017300755

[102] G. for Geeks. (2023) Associative memory. [Online]. Available: https://www.geeksforgeeks.org/associative-memory/

[103] X. Sun, R. Liu, X. Peng, and S. Yu, "Computing-in-memory with sram and rram for binary neural networks," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, 2018, pp. 1–4.

A
Appendix - Paper draft

EON-1: A Brain-Inspired Processor for Near-Sensor Extreme Edge Online Feature Extraction

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

4th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

5th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

6th Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—Edge AI applications require online learning and adaptation on resource-constrained embedded devices to deal with fast sensor-generated streams of data in changing environments. This can address the challenges of learning from limited training points and alleviate the hindering factors of offline training. However, maintaining low-latency and power-efficient inference is crucial for Edge AI computing systems, and thus online learning and adaptation on the device with minimal incurred overhead for inference is crucial. In this paper, we explore energy-efficient learning and adaptation on-device for Edge AI applications using Spiking Neural Networks (SNNs), which follow the principles of brain-inspired computing, such as high-parallelism, neuron co-located memory and processing, and event-driven processing. We propose EON-1, a brain-inspired processor for near-sensor extreme edge online feature extraction, which can enable online feature extraction and incremental learning on resource-constrained devices. We evaluate EON-1's performance on benchmark datasets and show that it achieves high accuracy while maintaining low power consumption and latency.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

With the emergence of Edge AI ~~(Artificial Intelligence)~~ applications, enabling on-device learning and adaptation on resource- and latency-constrained embedded devices is becoming increasingly appealing, as it has the potential to tackle a wide range of challenges: firstly, it can deal with on-the-fly adaptation to fast sensor-generated streams of data under changing environments. Secondly, it could address learning from limited amounts of training points and thirdly, it can alleviate a variety of hindering factors associated with offline training in the Cloud, such as incurred energy consumption of sensor data transfers and extra memory storage for the training samples, but also data privacy and security concerns. Concurrently, maintaining low-latency and power-efficient inference is paramount for Edge AI computing systems, and thus learning/adapting on the device with minimal incurred overhead for inference is crucial.

Identify applicable funding agency here. If none, delete this.

Quantized and binary neural networks (QNNs/BNNs) and their neuromorphic recurrent sibblings spiking neural networks (SNNs) are currently the main genres of neural models for accelerated low-latency low-power inference. When it comes to online on-device learning/adaptation of such models there is no mainstream approach established due to the inherent difficulties of making back-propagation scalable or efficient for this purpose [1], [2]. Thus the topic is subject of active research with a small handful of custom processors that embed some sort of limited online learning or adaptation capability already proposed in the literature [4], [3], [5], [6], [29]. The key differentiators among them is typically energy, area efficiency, speed of learning, and supported learning rule.

In this paper we explore the merits of the process of Ultra-Rapid Visual Categorization (URVC) [7] that takes place in the mammalian visual cortex, for online hardware-assisted on-device learning and adaptation. In URVC, fast analysis and classification of images is attributed to efficient encoding and transmittion of information from the retinal ganglion cells to the orientation-selective cells in the visual cortex in the shortest time possible (i.e., with the first emitted spike) via the optic nerve [8], [9]. Past work has defended the efficiency of URVC with binary STDP [10] in networks of integrate-and-fire (IF) neurons for tasks such as face identification [11] and natural scene recognition [12], [13], [14].

Building upon these past ideas, we introduce EON-1, an energy and latency efficient edge-AI neural accelerator for spiking and binary convolutional neural networks, that uses 1-spike per neuron and 1-bit synaptic weights. EON-1 embeds a hardware-assisted very fast on-device online learning algorithm (amortizable for one-shot or few-shot) founded on binary stochastic STDP, which when benchmarked in an ASIC node, achieves less than 1% energy overhead for on device learning (relative to the inference). To our knowledge, our

solution incurs the least energy overhead for learning on device, compared to state-of-the-art solutions, showing a better efficiency by at least a factor of 10x. We also report 94% and 77.65% accuracy on the MNIST and Fashion-MNIST classification tasks, and we achieve 0.09pJ/SOP and 1.5pJ/SOP energy efficiency during inference and learning, respectively. We extend our solution to demonstrate a practical use-case of performing inference in real-time UHD videos while coping with streaming data and we showcase 60 FPS UHD video processing.

In the remaining of this paper Section II presents the main background and describes our methodology, Section III presents the experimental setup and results, while Section IV presents a short discussion and conclusion of our work.

II. Background and Methods

A. Overview of ultra-rapid feature extraction in the visual cortex

In the biological visual system, it has been observed that ultra-rapid high-level feature extraction occurs in the orientation-selective first layer of the visual cortex based on the *first spike* emitted by ganglion cells in the retina [15]. In order to generate the first spike that carries sufficient information for high-level recognition, the ganglion cells in the retina act as intensity-to-latency converters [16], such that they emit spikes in the order of the strength of the visual stimulation.

This behavior can be modelled in a convolutional network (CNN) with orientation selective edge filters in the first layer [17], followed by a temporal 1-winner-take-all (1-WTA) circuit of applying lateral inhibition [10], that allows only the dominant edge filters to propagation information downstream (max-pooling channel-wise).

B. Learning algorithm and refinements~~methodology~~

The hardware-optimized algorithm developed for this work is partially inspired by [18], [13] and [14]. Instead of using complex neuron models, our vision in this work is to use millions of simplified neurons with a hardware-friendly learning algorithm. In this section, we will explain the inference and learning processes in the proposed algorithm. Although we explain inference first, it's important to note that just like the biological brain, learning and inference will run simultaneously in the proposed hardware platform (Section II-C) and are not separated.

1) Low-latency 1-spike–based inference: The network structure is depicted in Fig.1. It starts with processing input images with (pre-defined) edge filters in the first layer, followed by a lateral inhibition layer. Lateral inhibition ensures that for every (X, Y) location, only one spike is fired from all the edge filtering orientations (i.e. one edge filter is selected as in channel-wise max-pooling). The result is a highly sparse binary spiking output that can be encoded in a spike vector representation, and compressed as shown in Fig.2.

The output of the lateral inhibition layer is then connected to a layer of IF neurons. Each IF neuron is fully connected to the output of the lateral inhibition layer through a weight

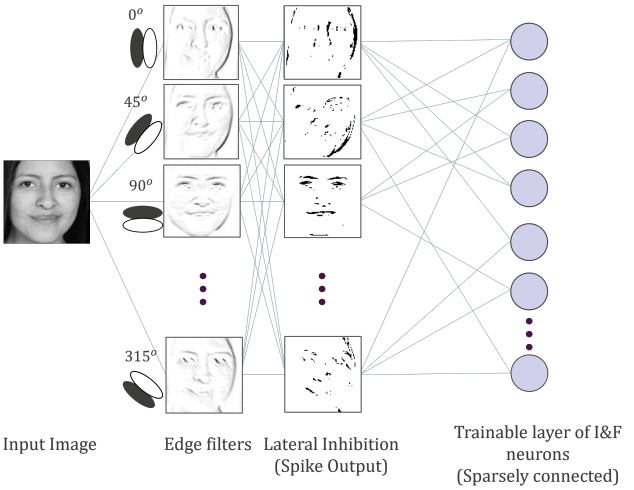


Fig. 1: The neural network structure for this work. It includes a layer of edge-filtering convolution, a lateral inhibition layer, and a layer of fully connected neurons equipped with binary STDP training.

TABLE I: Parameters for IF Neuron layers and synaptic plasticity

Name	Description
N_{IF}	Number of IF neurons in learning layer
W	Number of active synaptic connections per neuron
$T_{Learn}[0]$	Initial learning threshold
T_{Fire}	Firing threshold. Can be fixed, or variable (fraction of T_{Learn} but ∞ at the beginning).
$Swap\_Rate$	Fraction of ineffective active synapses per neuron that get inactive (swapped) during learning to permit activation of effective synapses (on which spikes are received).
K	Maximum number of neurons which are allowed to learn (an input spike vector)

vector. The synaptic weights of each of the IF neurons in our algorithm have three constraints:

- The synaptic weights are binary, which means they either connect (activate) or disconnect (deactivate) synapses.
- All neurons have the exact same amount of active synapses (W parameter in Table I).
- For every neuron the synaptic connection to only one edge filter is active at each (X, Y) location in the input image.

These constraints result in very sparse synaptic weights vectors (Fig.1), which can be encoded in the same (sparse) vector format as the spike vector (see example in Fig.3 for $W = 4$ active synapses per neuron).

When elements of the weight vector of a downstream IF neuron align in index position (pixel position) and value (edge-filter) with corresponding elements of a spike vector, the respective IF neurons integrate the incoming spikes from the edge-filters layer. Thereafter, a neuron fires if the number of matching elements exceeds the firing threshold T_{Fire}. Each neuron in this process is trained to recognize a specific abstract

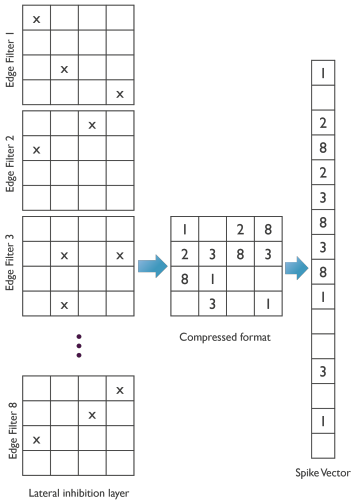


Fig. 2: The lateral inhibition layer produces binary spikes that can be compressed into a spike vector. Each element of the vector designates the source edge filter-id that generated the spike for the respective pixel position.

2		1			3		1	8	Spike Vector
2	7			6		7			Weight Vector Neuron 1
	8	**1**	3	5					Weight Vector Neuron 2
2				5	**3**	2			Weight Vector Neuron 3
	1		4		4	8			Weight Vector Neuron 4

Fig. 3: Example of a spike vector and four weight vectors. Each weight vector contains four active synaptic connections. Bold elements in the weight vectors are the one that matches the spike vector.

pattern as a composition of base features. It is possible to extend the network depth with more IF neuron layers, or the IF layer width by increasing the number of IF neurons. However, given the theoretical equivalence [20]–[22] between arbitrary deep versus wide neural networks (with up to two hidden layers), for reasons of hardware parallelization, latency and learning simplicity with a local rule (discussed in the following section), we have favored in this work a single very wide output layer.

2) Fast learning (one/few-shots): The binary synaptic weights of IF neurons downstream from the lateral inhibition layer can be trained by swapping the active synaptic connections during the learning process. The training rule that we use is a variant of STDP, namely stochastic binary STDP [19], and is triggered independently for each neuron. By contrast to gradient-based learning with a global signal (like in back-propagation), here the choice of a local rule like STDP and its particularly simple and lean variant of binary STDP, (a) is functionally justified by the shallow nature of the network architecture (i.e. a global signal is not required to traverse a

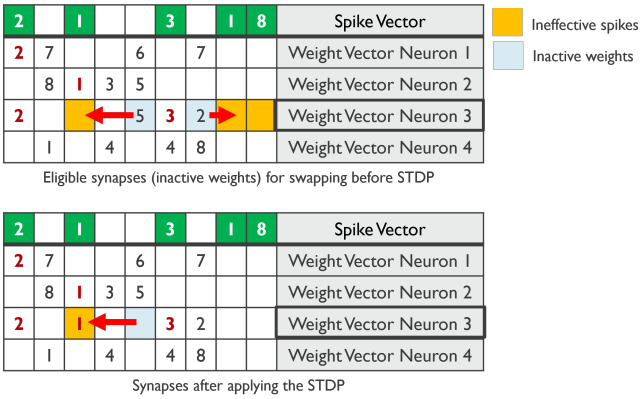


Fig. 4: We use the example in Fig.3 to explain the update processes of the binary STDP algorithm. The learning algorithm applies to neurons whose membrane potential reaches a pre-defined T_{Learn} (neuron 3 here). [Top] Neuron 3 has two ineffective synapses (turquoise) that it can swap with inactive ones to align with three ineffective spikes in the spike vector (yellow). [Bottom] One of them is randomly selected and swapped with the ineffective spike.

deep model structure); and (b) is architecturally well suited for efficient hardware implementation given its parallelization potential and sparse independent application on each neuron: each IF neuron can learn independently of all other, when the learning threshold (T_{Learn} in Table I) is exceeded. After each update this threshold increases gradually, to reduce the neuron plasticity and increase its selectivity for a specific pattern. Optionally if forgetting needs to be incorporated in the algorithm for out-dated patterns and non-stationary input distributions the learning threshold may slowly decay.

Moreover, in "conventional" STDP where the real-valued weights for synapses without a pre-synaptic firing would gradually reduce (Long Term Depression, LTD) and the weights for synapses with a pre-synaptic activity would gradually increase (Long Term Potentiation, LTP), a neuron can slowly adapt to new data patterns without forgetting the already learned patterns. By contrast here, with the use of binary weights in our STDP rule, it is impossible to adapt the weight values gradually. Nevertheless, by increasing/decreasing the maximum number of neurons K that are allowed to update their weights during learning and the max number of synapses per neuron that can be swapped during an update step ($Swap\_Rate$), we can control the learning rate (or equivalently forgetting rate) even under such an extremely low weight precision regime.

The previously mentioned constraints on the weight vectors result in features that make our algorithm a hardware-friendly option for neuromorphic processing systems. Specifically, the weight sparsity is inherent in our algorithm due to the constraints we apply (parameter W), rather than a result of the training process. The network's connections are sparse right from the start and remain sparse. This constraint-enforced sparsity is utilized to enhance the efficiency in our hardware,

and constrain *a-priori* the resources needed.

Algorithm 1 Stochastic STDP(`s[M-1:0],w[M-1:0], T_learn, swap_rate`)

Input data: An M-bit spike vector, s, an M-bit synaptic weights vector, w, its corresponding learning threshold, T_learn, and swapping rate, $swap\_rate$.
Init: `V_mem = 0`
for `i = 0 to M - 1` **do**
  `V_mem = V_mem + s[i] ∧ w[i]`
if `V_mem ≥ T_learn` **then**
  `swap_N = swap_rate × (W - V_mem)`
  for `s = 0 to swap_N - 1` **do**
    `ineff_s = find((s ∧(¬ w)) ≠ 0)`
    `inact_w = find((¬ s ∧ w) ≠ 0`
    `rand_ON_idx = randperm(ineff_s, 1)`
    `rand_OFF_idx = randperm(inact_w, 1)`
    `w[rand_ON_idx] = 1'b1`
    `w[rand_OFF_idx] = 1'b0`
  `T_learn = T_learn + swap_N`
return `w, T_learn` =0

Finally learning progresses according to the steps described in the algorithm in 1 and exemplified in 1):

i. **Perform inference**: Generate spike vector, match against matrix of weight vectors of IF neurons, update membrane potentials, and prepare to evaluate T_{Fire} and T_{Learn}. In Fig.4 (top), the membrane potentials of neurons 1-4 will then be 1, 1, 2 and 0 respectively.
ii. **Select (up to) K neurons to learn**: For all neurons that exceeded T_{Learn}, select randomly up to max K for updating their weights based on the input pattern. Assuming $T_{Learn} = 2$, in Fig.4 (top) neuron 3 will be eligible to update its weights.
iii. **Swap active synapses** (bit-flip weights): For each of the K selected neurons, de-activate (reset) up to $V \leq W$ ineffective synapses, and for each of them activate (set) an inactive synapse to align with one of the unmatched entries of the spike vector. The swap processes of ineffective synapses is based on $Swap\_Rate$ [1]. In Fig.4, neuron 3 chooses one of the two ineffective synapses (turquoise) to swap it with an inactive one that it aligns with one of the three unmatched spikes in the spike vector (yellow). So next time a variant of the same input pattern appears, neuron 3 will have higher probability of firing. Note that the max number of active synapses (related to sparsity) per neuron in this swapping process remains fixed.
iv. **Increase** T_{Learn}; by an amount that equals the number of synapses swapped. This mechanism is inspired by the homeostasis observed in biological neurons [27] for selectivity towards frequently seen patterns. The firing threshold T_{Fire}, may be a function of the learning threshold (thus being adaptive) or can also be fixed and independent.

[1]Either with probability $Swap\_Rate$ for all ineffective synapses, or by randomly selecting $Swap\_Rate$ of all ineffective synapses

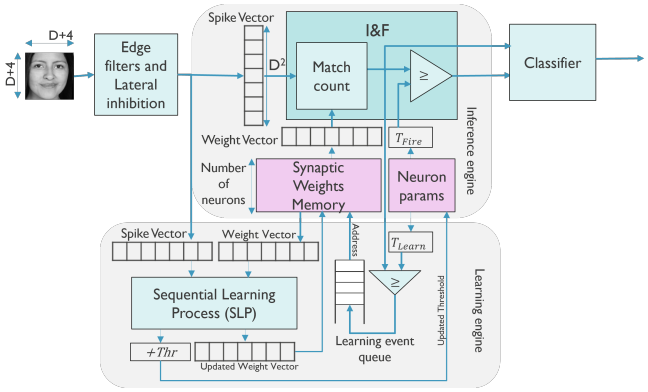


Fig. 5: Base block-architecture of the main components of the hardware implementation of the EON-1 neural accelerator, equipped an inference engine and a learning engine with our STDP-based learning algorithm. This architecture can be flexibly and trivially scaled-out by vectorizing either of: the IF units, the edge-filter units, and/or the sequential-learning processes (depending on design-space requirements)

C. *Hardware architecture* ~~*methodology*~~

Fig.5 illustrates the hardware architecture of the EON-1 neural accelerator that implements the algorithm discussed in II-B. Notice that the inference and learning parts are intertwined highlighting the fact that on-device learnng is not an add-on but rather and embedded feature of the accelerator. At the same time as we discuss next, the two parts operate asynchronously and fully in-parallel, and without imposing any inter-dependencies or bottlenecks to each other.

1) Inference engine: The key building blocks of the inference engine (top part of Figure 5) implementing the algorithm functionality described in Section II-B1 are the following:

- **Edge-Filters and Lateral-Inhibition** block(s) implement the convolutional layer(s) of the algorithm (edge filtering) with lateral inhibition (Figure 1), and generate the "spike output" in the compressed representation of the spike vector (i.e. indexes of the top-scoring edge filters for each pixel position (X, Y) – see Figure 2). Every row of the input image is processed by applying all edge detectors simultaneously, in a single clock cycle. Therefore, for an input image of size $D \times D$, a spike vector (of equal size) will be ready in D clock cycles. For 8 edge filters (in this work) we need $4b$ resolution in the spike vactor to represent the 9 possibilities (zero represents the *no-firing* case).
- **Weight Memory and neuron parameters memory** blocks in Figure 5 store the weight vectors and parameters/state of the IF neurons in the trainable layer. In this work the parameters (Table I) are shared among all neurons and thus need not to be stored per neuron. The width of the weight memory is the same as that of the spike

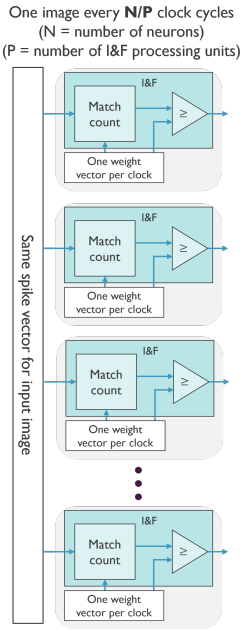


Fig. 6: To increase parallelism, and therefore latency for inference with very large output layers, we can spawn several IF processing elements in a P-wide vector pipeline. Each processing element contains only a match count a comparator and local memory for one weight vector, and can process one neuron per clock cycle. With N neurons in the system, each input image takes N/P clock cycles.

vector. We have used on-chip dual-port SRAM memory or off-chip pseudo-dual-port HBM memory, depending on the number of neurons and bandwidth requirements. One read-only port is used for inference (requires a high bandwidth port) and another independent read/write port is used for on-device learning.

- **Match Counter and Firing-threshold Comparator**. The match counter counts the number of matching index positions between spike and weight vectors, and the result is compared with the firing threshold. The counting happens in a single clock cycle, based on a pipeline with a number stages that is determined by the size of the two vectors. We refer to these two blocks together as a neuron unit since they essentially implement the IF neuron functionality.

Multiple IF neuron units can process a spike vector in parallel for vectorized processing as shown in Fig.6. Assuming that each IF unit has access to local memory, the number of clock cycles required to process the input spike vector in the general case for a layer of N IF neurons and a parallel circuit of P IF units is N/P.

2) Learning engine: The lower half of the hardware architecture shown in Fig.5 contains the building blocks for implementing an *event-driven* on-device learning functionality. This architecture decouples the inference from the learning temporally and yet allows them to take place concurrently or simultaneously. Typically, learning takes place slower than inference, but this can be partly controlled by the maximum number of neurons K (Table I) allowed to learn at a time.

- **Learning-threshold Comparator and Learning Event Queue**. During inference, in addition to comparing an IF neuron state (output of the match count block) against the firing threshold, it is also compared against the learning threshold. If this is exceeded, the address of the corresponding weight vector is pushed to the learning event queue. The active capacity of this queue limits the maximum number of neurons that can undergo learning triggered by one input sample (image) and is thus set by the parameter K mentioned earlier.
- **Sequential Learning Process (SLP)** is an FSM that consumes the learning event queue. At each clock cycle, SLP examines one element of a weight vector. If it encounters an inactive synapse, it disconnects and replaces it with an ineffective spike location. The order in which the weight vector elements are inspected is randomly generated using an LFSR unit [**?**]. The sequential learning process stops after a sufficient number of swaps has been reached. Since learning is a slower process than inference sequential processing is a more area-efficient solution. The exact time it takes for each weight vector to be updated depends on the weight vector dimension (D^2) and the number of swap operations. For a worst-case scenario where a weight vector update takes always D^2 clock-cycles, the learning process for N neuron takes $K \times D^2$. To ensure that the learning process never lags behind inference, $K \times D^2$ should remain smaller than N. Multiple SLPs can be instantiated in parallel, if necessary (similar to IF neuron units).

Note that since only up to $K << N$ neurons will be scheduled in for learning at every input inference, a question of bias strategy arises as to which neurons should be prioritised. Plausible strategies can be to select the top-K least-recently fired neurons (i.e. least recently seen pattern), or highest membrane state (most likely pattern). However, these are costly solutions that require sorting and imply latency cost. Selecting the first-K is by far the fastest and cheapest solution, yet erratic (likely only a small subset of neurons only participating in learning) if the neurons are always sequenced in the same order. To warrant thus that all IF neurons have the same fair chance of being scheduled-in for learning, during inference a random number generator produces a different start address every time for sequencing the IF neurons, thus providing a cheap arbitration.

III. Results

A. Experimental setup

We evaluated two instantiations of EON-1, one on FPGA, using a Xilinx VU37P HBM chip, and one using gate-level ASIC simulation [2] for GF 22nm FDSOI technology node.

[2]Cadence Genus [**?**] and Cadence JOULES [**?**]

The results are based on two tasks. A multiclass classification task based on MNIST (which despite its simplicity) allowed us to compare against other state of the art both for the FPGA and ASIC instantiations, in terms of hardware-related metrics. Then a binary class face detection task based on the FACES [?] and CIFAR-10 [?] datasets enabled us to evaluate the suitability of EON-1 for real-time adaptation and processing of streaming high-definition (UHD) data. We present results that validate both the algorithm effectiveness/performance for on-line adaptation and the hardware efficiency (primary goal) of our system.

For consistent algorithm behaviour across all tests, we kept the same learning parameters throughout, and set for very fast, one-shot, learning. All accuracies were measured using the test set with deactivated on-device learning.

- $Swap\_rate = 1$, i.e., all eligible connections are moved to active input lines during learning. Thus, only one training epoch is sufficient to memorize a pattern (one-shot). Please note that the resulting weight vector won't necessarily match the spike vector completely since there are possibly more active input lines than W.
- $TLearn[0] = 6$ is the initial learning threshold. We chose this value empirically to allow the triggering of learning early-on but not too often since the learning subsystem is sequential and can become a bottleneck at the beginning.
- $T_{Fire} = T_{Learn}/2$. Recall that $T_{Fire}[0] = \infty$ to force the inference circuit to be silent before any learning has taken place. After the first learning event for an IF neuron, its T_{Fire} is set to this value, activating it for inference. This value has also been experimentally obtained to give an optimal ratio of time that the inference circuit is active compared to the learning circuit (taking into account that the latter is slower).
- $K = 1$ to allow only one neuron, randomly chosen among all the eligible neurons, to adapt its weights every time learning is triggered. For one-shot learning, it is important to avoid wasting model memory when multiple neurons learn and lock in the same pattern, while $K > 1$ allows better generalization (lower selectivity) with slower few-shot learning ($Swap\_rate < 1$).

B. Hardware measurements and comparison with SoA – multiclass classification task

Tables II and III list various characteristics and performance metrics achieved by EON-1 and compared to other FPGA and ASIC based solutions in the recent literature; that claim support on-device leanrning/adaptation. These results are based on the MNIST task. To obtain the hardware measurements we have divided each experiment in (a) an initial "training" or "adaptation" phase during which 5120 samples are streamed through and while both inference and learning engines are active, followed (b) by an "inference-only" phase, where the learning engine is disabled, during which an additional 10000 samples are streamed through. The image resolution is 14×14. The IF neuron layer comprises of 5120 neurons and the classifier circuit is clustering model that groups subsets of IF neurons to clusters for each of the 10 digits. The classifier is pipelined with the IF units, thus it will give the prediction one clock cycle later than the last neuron processing. The spike-encoding is likewise pipelined with the IF neuron processing and therefore the overhead of the spike encoding is hidden after the first encoded spike vector, and averaged out (as insignificant) after all data-samples are processed.

TABLE II: Comparison of EON-1 with other FPGA solutions benchmarked on MNIST

	TCSI'21 [23]	**TCSII'21** [24]	**Neuro'17** [25]	**ICTA'23** [26]	**EON-1** this work
Accuracy	85.28%	90.58%	89.1%	95.49%	91.96%
Neur. model	LIF	LIF	LIF	LIF	IF
# Neuron	300	2304	1591	320	5120
# Synapses	176800	NR	638208	NR	4M
Weight Prec.	16b float	2b	16b fixed	8b	1b
Encoding	rate	rate	rate	rate	rank-1
Learning spec					
Online	yes	yes	yes	NR	yes
On-chip	yes	no	yes	yes	yes
Rule	STDP	STDP	STDP	BP	binSTDP
Hardware spec					
FPGA Chip	Virtex-7	ZCU102	Virtex-6	VC707	VU37P
Clock Freq.	100MHz	200MHz	120MHz	115MHz	100MHz
Throughput (fps)					
Learning	61	NR	0.05	NR	19.1K
Inference	285	46.44	0.11	1183	19.4K
$\frac{Learn.}{Infer.}$ ovh	5.22	NA	0.18	2.19	<**0.01**

NR: Not Reported, BP: Back Propagation,
binSTDP: Stochastic Binary STDP, HBM: High Bandwidth Memory

In the measurements we did not use parallelization of the IF neuron processing, and therefore the latency reported is measured as $(SpikeVecGen + N_{IF} + 1) \times clk\_period$. Thus, e.g. for a 10 ns clock period (FPGA), and taking into account the spike encoder overhead, the latency is $(14 + 5120 + 1) \times 10 = 51350ns$ and the throughput is $10^9/51350 = 19474.19$ inferences (frames[3]) per second.

For the energy consumption (ASIC) per inference/learning we used the equations

$$\begin{aligned} E_{inf} =& N_{IF} \times (E_{IFexec} + E_{rdmem(7b)}) \\ &+ (N_{IF} + 1) \times E_{rdmem(400b)} \\ E_{learn} =& E_{inf} + 58E_{1\_swapop} \\ &+ 2 \times E_{rdmem(400b)} + E_{wrmem(400b)} \\ &+ E_{wrmem(7b)} \end{aligned} \quad (1)$$

where $E_{rd/wrmem(7b)}$ refers to the energy to read/write 7bit neuron state from memory and $E_{rd/wrmem(400b)}$ is the energy to read/write a 400bit spike vector or weight vector from memory. Online learning energy consumption is computed as the total energy cost for performing one inference (including the spike encoder and classifier), in order to find the neuron that reaches it's learning threshold, followed by the energy cost of the neuron learning updating its weights. Based on

[3]1 inference is 1 input digit frame for these measurements

TABLE III: Comparison of EON-1 with other ASIC solutions benchmarked on MNIST

	TCSI'22 [29]	**TBCAS'19** [4]	**TBCAS'19** [3]	**TCSII'23** [5]	ISCAS'20 [28]	**EON-1**
Accuracy	93%	87.4%	84.5%	93.54 %	92.8% - 95.3%[1]	91.96%
Neuron Model	LIF	IF	LIF/Izk.	LIF	NR	IF
# Neuron	384	400	256	2048	128	5120
# Synapses	176800	230400	64000	2M	NR	4M
Weight Precision	9b fixed	1b	4b	8b	8b	1b
Encoding	temporal	rate	rate[4]	temporal	TTFS	rank-1
Learning spec						
Online	yes	yes	yes	yes	yes	yes
On-chip	yes	yes	yes	yes	yes	yes
Rule	STDP var.	spike cnt	SDSP	addSTDP	DRTP	binSTDP
Hardware spec						
ASIC Node	28nm	65nm	28nm	28nm	28nm	22nm
Clock Frequency	333MHz	384MHz	75MHz	500MHz	150MHz	500MHz
Voltage	0.9V	1.2V	0.55V	0.81V	0.6V	0.8V
Area (mm^2)	1	0.39	0.086	6.22	0.26	0.558
Norm. area[2](mm^2)	0.617	0.044	0.053	3.83	0.16	0.558
Energy						
Learning	660nJ	2630nJ	105nJ	NR	313nJ	378.4nJ
per SOP	NR	1.42pJ	NR	4.99pJ	NR	1.5pJ
Inference	500nJ	310nJ	15nJ–404nJ	NR	NR	377.2nJ
per SOP	NR	0.26pJ	12.7pJ	1.28pJ	NR	0.09pJ
Norm. Energy[3]						
Learning	409.7nJ	395.62nJ	174nJ	NR	437.2nJ	378.4nJ
per SOP	NR	0.21pJ	NR	3.82pJ	NR	1.5pJ
Inference	310.4nJ	46.6nJ	24.93nJ–671.58nJ	NR	NR	377.2nJ
per SOP	NR	0.03pJ	21.11pJ	0.98pJ	NR	0.09pJ
Throughput (fps)						
Learning	211.77k	NR	NR	22.8GSOPS	NR	95K
Inference	277.78k	NR	NR	81.92GSOPS	8.5K	97.6K
$\frac{Learn.}{Infer.}$ overhead[5]	0.32	7.5	7 & 0.25 [7]	2.89	0.16	<**0.01**

NR: Not Reported, BP: Back Propagation, addSTDP: Additive STDP, binSTDP: Stochastic Binary STDP, DTRP: Direct Random Target Projection

[4], [29] report area/energy results post-layout and they do not include I/O area/energy cost; [3], [5], [28] report area/energy results post-tapeout.

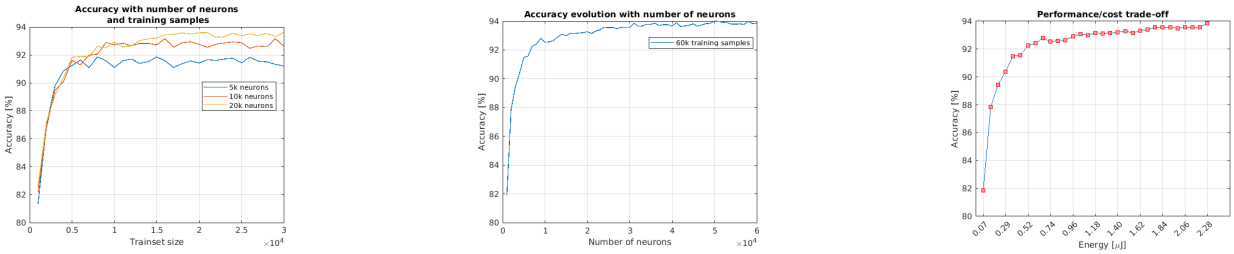
[1] Reported accuracy for one epoch is 92.8%, and after 100 epochs it reaches 95.3%.

[2] Area $\times$ $(22/\text{Node})^2$

[3] Energy $\times$ $(22/\text{Node})$ $\times$ $(0.8/\text{voltage})^2$

[4] [3] supports both rate coding and temporal, rank-order coding for input spikes. Reported inference energy and learning overhead is for rank and rate coding, respectively. Reported learning energy excludes inference cost.

[5] Learning overhead is obtained from reported power.



(a) (b) (c)

Fig. 7: On-line on-device learning evolution on MNIST digit classification task. (a) Accuracy as a function of data samples presentation for 3 IF layer sizes. (b) Accuracy as a function of the IF layer size for 60K samples. (c) Accuracy versus energy while scaling the IF layer from 1K neurons to 30K neurons.

these energy costs we compute the learning overhead as $(E_{learn} - E_{inf})/E_{inf}$. Finally, as the total energy consumption measured with Joules, have an error tolerance of 15% from tool reporting we report the worst case numbers.

What stands out from these comparisons is that EON-1, while in the same league with the SoA it is (a) the most efficient solution in terms of overhead introduced for supporting online on-device learning; (b) it has one of the highest throughputs; and (c) it is the only one that supports very fast (one-shot) learning(!); both in FPGA and ASIC.

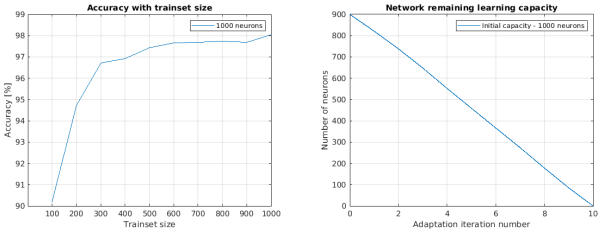
Last, Fig. 7 reveals how the algorithm performs as we as expose it to more training data samples (Fig. 7a) or add more resources to it in terms of IF neurons, namely increasing its capacity (Fig. 7b), and how that impacts the energy cost (Fig.7c).

C. Streaming data processing – binary classification

One of the main motivations of this exploration has been to see if on an small edge device we can facilitate fast learning and processing of continuous streaming data from a sensor. In this experiment we aimed to test just that. We combined face images from the UTK Face Dataset [**?**] with non-face images from the CIFAR-10 [**?**] dataset to create high-resolution/size (UHD) collage images such as the one shown in Fig.9a. The task here is to detect faces in the UHD image. Moreover, this image cannot be processed in one go as a single patch residing in memory, but rather a rolling window needs to parse the image with a stride and load a patch-at-a-time in memory. So the challenge here is to see if the speed of processing of these images with EON-1 is acceptable for real-time video playback.

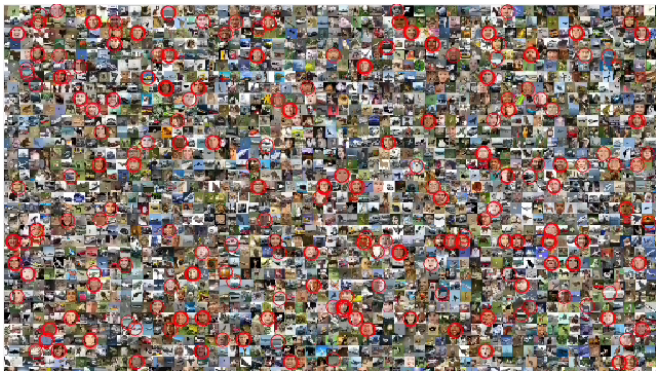
The results are reported in Fig. **??** for the FPGA-based instantiation of EON-1. At full UHD resolution we were able to process 12 UHD images per second. This is about half the baseline framerate for real-time motion video (25fps) and about 1/6 of UHD video (60fps). Operating the circuit at the ASIC clock frequency (500MHz) would make real-time UHD video processing indeed possible(!).

Finally in Fig.8 we report how the on-line learning affects the performance of the algorithm (accuracy) for this task over time, as a function of the data points seen in the training phase (Fig. 8a) and how the memory capacity is drained-up as we present the same data points over and over across multiple iterations. Notice that although the capacity reaches 0 after a few iterations, the accuracy never gets to 100%, which means that even with "aggressive" one-shot learning regime, the algorithm does learn the input patterns exactly (i.e. does not overfitting on the training data).



(a) (b)

Fig. 8: On-line on-device learning evolution on face detection task. (a) Accuracy as a function of data samples presentation (IF layer 1000neu) (b) IF layer memory capacity as a function of iterations of dataset presentations. More and more neurons end up learning the same patterns, until after about 10 iterations the capacity is reduced to zero. Allowing the T_{Learn} to decay would allow the network to continue learning while forgetting older patterns.



(a) Collage of many small faces and Images from the CIFAR-10 dataset, randomly placed in one UHD frame. The collage contains 790 faces.

resolution	latency	fps
2160x3840 (UHD)	84ms	12
345×614	15ms	68

(b) Face detection accuracy achieved with pre-trained network: 90.2%. One entire UHD frame can be processed in full resolution with our FPGA implementation in just under 85ms. With a down-scaling factor of 6.25 we were able to follow-up real-time UHD video playback (>60fps).

Fig. 9: Face detection in UHD video.

REFERENCES

[1] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kb memory," arXiv:2206.15472[cs.CV], 2022

[2] Y. Yang, G. Li, and R. Marculescu, "Efficient on-device training via gradient filtering,", arXiv:2301.00330v2[cs.CV], 2023

[3] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm2 12.7-pj/sop 64k-synapse 256- neuron online-learning digital spiking neuromorphic processor in 28-nm cmos," IEEE Transac- tions on Biomedical Circuits and Systems, vol. 13, no. 1, pp. 145–158, 2019

[4] H. Tang, H. Kim, H. Kim, and J. Park, "Spike counts based low complexity snn architecture with binary synapse," IEEE Transactions on Biomedical Circuits and Systems, vol. 13, no. 6, pp. 1664–1677, 2019

[5] Y. Zhong, Z. Wang, X. Cui, J. Cao, and Y. Wang, "An efficient neuromorphic implementation of temporal coding-based on-chip stdp learning," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 70, no. 11, pp. 4241–4245, 2023

[6] C. Frenkel, J.-D. Legat, and D. Bol, "Morphic: A 65-nm 738k-synapse/mm2 quad-core binary- weight digital neuromorphic processor with stochastic spike-driven online learning," IEEE Trans- actions on Biomedical Circuits and Systems, vol. 13, no. 5, pp. 999–1010, 2019.

[7] M. Fabre-Thorpe, "The characteristics and limits of rapid visual cate-

gorization," Frontiers in Psy- chology, vol. 2, 2011
[8] Tim Gollisch and Markus Meister. "Rapid Neural Coding in the Retina with Relative Spike Laten- cies". In: Science 319.5866 (2008), pp. 1108–1111
[9] I. Tomomi and H. Samar, "On and off signaling pathways in the retina and the visual system," Frontiers in Ophthalmology, vol. 2, 2022
[10] T. Masquelier and S. J. Thorpe, "Learning to recognize objects using waves of spikes and Spike Timing-Dependent Plasticity," The 2010 International Joint Conference on Neural Networks (IJCNN), pp. 1-8, Barcelona, 2010
[11] R. Rullen, J. Gautrais, A. Delorme, and S. Thorpe, "Face processing using one spike per neuron," Biosystems, vol. 48, pp. 229–239, 1998
[12] S. Thorpe, "Ultra-rapid scene categorization with a wave of spikes," in Biologically Motivated Computer Vision. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–15, 2002
[13] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," PLoS computational biology, vol. 3, no. 2, p. e31, 2007
[14] A. Delorme, J. Gautrais, R. van Rullen, and S. Thorpe, "Spikenet: A simulator for modeling large networks of integrate and fire neurons," Neurocomputing, vol. 26-27, pp. 989–996, 1999
[15] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," nature, vol. 381, no. 6582, pp. 520–522, 1996
[16] . J. Gawne, T. W. Kjaer, and B. J. Richmond, "Latency: another potential code for feature binding in striate cortex," Journal of neurophysiology, vol. 76, no. 2, pp. 1356–1360, 1996
[17] S. J. Thorpe, A. Delorme, and R. VanRullen, "Spike-based strategies for rapid processing," Neural Networks, vol. 14, pp. 715–726, 2001
[18] S. Thorpe, et al. "Method, digital electronic circuit and system for unsupervised detection of repeating patterns in a series of events." U.S. Patent No. 11,853,862. 26 Dec. 2023
[19] A. Yousefzadeh, E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, "On practical issues for stochastic stdp hardware with 1-bit synaptic weights," Frontiers in Neuroscience, vol. 12, 2018
[20] G. Cybenko. Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst., 2(4):303–314, 1989
[21] K. Hornik. Approximation capabilities of multilayer feedforward networks. Neural Netw., 4(2): 251–257, 1991
[22] A. Pinkus. Approximation theory of the MLP model in neural networks. Acta Numer., 8:143–195, 1999
[23] S. Li, Z. Zhang, R. Mao, J. Xiao, L. Chang, and J. Zhou, "A fast and energy-efficient snn processor with adaptive clock/event-driven computation scheme and online learning," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 4, pp. 1543–1552, 2021
[24] . Zheng, Y. Guo, X. Yang, S. Xiao, and Z. Yu, "Balancing the cost and performance trade-offs in snn processors," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 68, no. 9, pp. 3172–3176, 2021
[25] Q. Wang, Y. Li, B. Shao, S. Dey, and P. Li, "Energy efficient parallel neuromorphic architectures with approximate arithmetic on fpga," Neurocomputing, vol. 221, pp. 146–158, 2017.
[26] K. Huang, W. Liu, Y. Liu, S. Xiao, and Z. Yu, "Towards efficient on-chip learning for spiking neural networks accelerator with surrogate gradient," in 2023 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA), 2023
[27] Y. Jun, Y. Quan, "Structural homeostasis in the nervous system: a balancing act for wiring plasticity and stability", in Frontiers in cellular neuroscience, vol. 8, pp. 439, 2015.
[28] C. Frenkel, J.-D. Legat, and D. Bol, "A 28-nm convolutional neuromorphic processor enabling online learning with spike-based retinas," in 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1–5
[29] C. Sun, H. Sun, J. Xu, J. Han, X. Wang, X. Wang, Q. Chen, Y. Fu, and L. Li, "An energy efficient stdp-based snn architecture with on-chip learning," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 69, no. 12, pp. 5147–5158, 2022