CarSNN: An Efficient Spiking Neural Network for Event-Based Autonomous Cars on the Loihi Neuromorphic Research Processor

CarSNN: An Efficient Spiking Neural Network for Event-Based Autonomous Cars on the Loihi Neuromorphic Research Processor

Alberto Viale[1,2,*], Alberto Marchisio[1,*], Maurizio Martina[2], Guido Masera[2], Muhammad Shafique[3]
[1]*Technische Universität Wien, Vienna, Austria* [2]*Politecnico di Torino, Turin, Italy* [3]*New York University, Abu Dhabi, UAE*
Email: {alberto.viale, alberto.marchisio}@tuwien.ac.at, {maurizio.martina, guido.masera}@polito.it, muhammad.shafique@nyu.edu

***Abstract*—Autonomous Driving (AD) related features provide new forms of mobility that are also beneficial for other kind of intelligent and autonomous systems like robots, smart transportation, and smart industries. For these applications, the decisions need to be made fast and in real-time. Moreover, in the quest for electric mobility, this task must follow low power policy, without affecting much the autonomy of the mean of transport or the robot. These two challenges can be tackled using the emerging Spiking Neural Networks (SNNs). When deployed on a specialized neuromorphic hardware, SNNs can achieve high performance with low latency and low power consumption. In this paper, we use an SNN connected to an event-based camera for facing one of the key problems for AD, i.e., the classification between cars and other objects. To consume less power than traditional frame-based cameras, we use a Dynamic Vision Sensor (DVS) [1]. The experiments are made following an offline supervised learning rule, followed by mapping the learnt SNN model on the Intel Loihi Neuromorphic Research Chip [2]. Our best experiment achieves an accuracy on offline implementation of 86%, that drops to 83% when it is ported onto the Loihi Chip. The Neuromorphic Hardware implementation has maximum 0.72 ms of latency for every sample, and consumes only 310 mW. To the best of our knowledge, this work is the first implementation of an event-based car classifier on a Neuromorphic Chip.**

***Index Terms*—Autonomous Driving, AD, Spiking Neural Networks, SNN, Spatio-Temporal Backpropagation, STBP, Intel Loihi, Neuromorphic Computing, Dynamic Vision Sensor, DVS, cars vs. background classification.**

I. Introduction

The interest in Autonomous Driving (AD) has significantly grown in recent years. Therefore, new algorithms and design solutions have to address the challenges offered by this rapidly expanding sector. This paper focuses on practical AD systems by proposing a Spiking Neural Network that is directly implementable on a Neuromorphic Chip using a DVS, which can be easily introduced inside the car control system.

A. Target Research Problem and Research Challenges

If we study the various driving operations for a vehicle, we can understand how large/complex the AD problem space is, and how difficult it is to consider the problem in its entirety. For the purposes of this research, we can separate the AD tasks in the following three principal categories:

- the classification of the environment objects such as pedestrian and cars [3];
- the prediction of the position of these objects [4];
- the prediction of the controls of the car such as the steering angle and the status of the brake pedal and accelerator [5].

These can be viewed as regression and generalization problems, because with some different inputs coming from the sensors, the AD system has to predict a reaction that represents the solution for the task. The main methods to make these decisions in fast and accurate ways are represented by Deep Learning algorithms, that are typically divided into the following types:

- **Deep Neural Networks (DNNs)**, that are the oldest and can be implemented on traditional hardware processors and specialized architectures [6][7]. They are based on the transmission of digital values, but exhibit high power consumption.
- **Spiking Neural Networks (SNNs)**, that closely follow the behavior of neurons and are based on the transmission of spikes. They can be implemented on conventional hardware, but to achieve very low power consumption they are more amenable to the Neuromorphic Chips [8]. This aspect can be also optimized by the implementation on energy-efficient frameworks like SparkXD [9], FSpiNN [10], Q-SpiNN [11] or SpikeDyn [12].

Since every task represents a real-time problem, we want that the entire decision-making system has a good reactivity with a very low latency, in order to minimize the chance to have catastrophic car accidents due to late decisions. Another challenge is related to the robustness of the system that must operate in all conditions, in particular different types of illumination and weather conditions. Moreover, the system design should be optimized for low power consumption, which is an important design criteria for automotive, especially in the battery-driven electric mobility.

In our research, we focus on the *"cars vs. background"* classification problem. To overcome the above-discussed limitations, we identify three main research objectives:

1) the system should use the major robust vision engine, i.e., an event-based camera;
2) the network should be a low-complexity event-based SNN for energy-constrained systems;
3) the developed SNN should fulfill the system constraints to be implemented onto a neuromorphic hardware chip.

Following these research targets, we design, optimize, and implement the SNN on the Intel Loihi Neuromorphic Research Chip [2], and evaluate it on the N-CARS dataset [3]. It is based on Asynchronous Time-based Image Sensor (ATIS) [13], which is an event-based camera.

B. Motivational Case Study

Since, to the best of our knowledge, there were no prior existing works on AD applications implemented onto the Loihi Neuromorphic Chip, to highlight the research problems, we provide a motivational case study by analyzing another application. A well established benchmark in the event-based neuromorphic community is constituted by the *IBM DVS128 gesture dataset* [14], which is a database for gesture classification.

*These authors contributed equally to this work.

For example, the work in [15] uses an SNN trained offline on this dataset to recognize live operator gestures taken by a DVS camera connected to the *Intel Kapoho Bay* Neuromorphic Chip. It achieves about 91% accuracy, having also the possibility to learn online new gestures with few shots, using the On-Chip Learning engine. In the last case it achieves about 80% accuracy results with only 20 shots.

Figure 1 compares the implementations of classifiers on different hardware platforms, in terms of accuracy and latency. The SLAYER implementation on Loihi [16] exhibits the shortest latency (only $4.35\ ms$), with accuracy comparable with an optimized GPU implementation [17]. Therefore, towards real-time use cases, these results motivate us to conduct this research on AD applications on the Loihi chip.

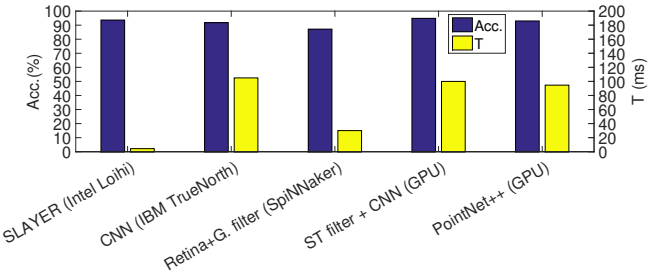

Fig. 1. Accuracy (Acc.) and latency (T) comparison between different implementations for DVSGesture recognition problem [16, 17, 18, 19, 20]. The higher accuracy to latency ratio is achieved by the SLAYER implemented on the Intel Loihi. It also has the lowest power consumption (0.54 mJ), compared to over 19 mJ consumed by the IBM TrueNorth implementation.

C. Our Novel Contributions

In this paper we present **CarSNN**, a novel spiking Convolutional Neural Network (CNN) based classifier method to tackle the classification problem between cars and background images collected by an event-based camera.

Using the attention window strategy, the focus is concentrated only on a part of the original input. To find the best window size and position, we analyze the statistics of input spikes, and focus on the part with more information (**Section III**).

To maintain the temporal correlation between different events we also adopt the accumulation in time of these information (**Section IV**). We use three hierarchical stages for this strategy implementation:

1) During a time window, called *sample time*, the spikes from the event-based camera are collected following the rule for which each channel can have a maximum of one spike per pixel coordinate.
2) The resulting image is given to the input of the network and remain stable for many time steps. We take the class with highest output spikes as the prediction of this single image.
3) To increase the accuracy we can derive more than only one image from the sample stream. To do that, we define a second time window, called *sample length*, that is multiple of *sample time*. Therefore, we have *sample length / sample time* different input images from a single sample stream. Based on the classification made for every single image, the most predicted class represents the prediction for the entire sample stream.

Moreover, with the above-discussed steps, we obtain a compression of the information, which is extremely important for low power applications.

All the developed networks followed the constraints of an existing Neuromorphic Chip, the Intel Loihi, to face the cars classification problem with event-based camera streams (**Section V**).

To give an overview of every technical part used to perform the task, in **Section II** we:

- focus on what an *event-based vision sensor* is and what are the advantages over frame-based cameras (**Section II-A**);
- present the *Neuromorphic-cars* (*N-CARS*) dataset, that can be used to train an SNN for the current classification problem (**Section II-B**);
- expose what a *Spiking Neural Network* is, its advantages and how it can be modeled and trained (**Section II-C**);
- summarize the main features and advantages of the *Spatio-Temporal BackPropagation* Supervised Learning rule used in this paper (**Section II-D**);
- explain the behavior of the *Intel Loihi Neuromorphic Chip* used to find the major network constraints (**Section II-E**).

For reproducible research, the source code for training and deploying our *CarSNN* models has been released at https://github.com/albertopolito/CarSNN.

II. Background and Related Work

A. Event-Based Cameras

In recent years, the event-based cameras [1], which are bio-inspired sensors for the acquisition of visual information, were proposed and designed to overcome the performance of the classic frame-based cameras. They recognize the same matrix of pixels, but they collect the information in a different way.

- A **frame-based camera** records the video as set of images and every image is collected with a constant delay from the neighbor in time, without any compression.
- An **event-based camera** records the video as a set of events. If and only if a pixel changes its brightness, the camera triggers an event with these information:
  - x, y: the coordinates of the pixel;
  - t: the timestamp of when the event occurred;
  - p: the polarity of the variation of the brightness, which is ON or 1 if the pixel is brighter, and OFF or 0 if the brightness is reduced.

For their structure, event-based cameras are extremely useful when coupled with the SNNs, because the spikes generated by the sensor can directly feed the SNNs' inputs.
Event-based cameras have others advantages:

- *High resolution in time*: it can record two different events delayed by few microseconds. Therefore, it does not suffer from oversampling, undersampling, and motion blur.
- *Adaptive data rate and less memory usage*: there is no need to store the redundant information, but only the changes, to obtain an efficient storage of the information.
- *High dynamic range* (up to $120dB$): it can record scenarios with a great change of brightness without losing any information.

The major drawback of this camera is its lower resolution in space than a frame-based camera.

B. N-CARS Dataset

The main event-based datasets derive from a simulation of an event-based camera on frame-based recording images [21][22]. Hence, these benchmarks lose the great time bandwidth of event-based cameras [1]. Introduced to overcome the limited numbers

of event-based data recorded by an event-based camera from the real word, the N-CARS dataset [3] is a recording of 80 $minutes$ with an ATIS camera [13]. This sensor has a resolution in space of 304×240 and it is mounted behind the windshield of a car. For recognition purposes, the outcoming events are transformed into grey-scale images. These are processed with a state-of-the-art object detector [23][24], to automatically extract bounding boxes around the two classes:

- **cars**: 12336 samples;
- **background**: 11693 samples.

The maximum bounding boxes size is of 120 × 100 pixels.

The dataset is split into 7940 car and 7482 background training samples; 4396 car and 4211 background testing samples. Each example lasts 100 milliseconds. The dataset files are grouped by class and are made as 1 $channel$ stream with two possible event values -1 and 1. An example of the accumulated grey-scale images is shown in Figure 2.

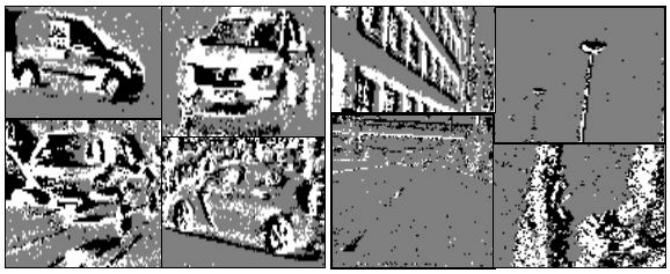

Fig. 2. Example of grey-scale accumulated images of the N-CARS dataset [3]. The four pictures on the left represent car samples, while the images on the right represent background samples.

C. Spiking Neural Networks (SNNs)

As previously discussed, the **Spiking Neural Networks** (**SNNs**) introduce a revolution in the Artificial Intelligence and Machine Learning field [25], since they are known as the third generation of neural networks. An SNN is the result of the research of what the *neurons* effectively do into the brain. It is based on the most biological probable behavior, where the information is encoded into *spikes* and spread through the neurons by *axons* and *synapses*. The behavior of the brain can be simulated with many kinds of models which can be more or less complex. For example, a complex model exhibits great performance, but on the other hand it is difficult to implement, due to high latency and high power consumption.

The SNNs offered many advantages w.r.t. the older (non-spiking) DNNs [26]:

- *low power consumption*, due to the adaptation of the consumption with the intensity of the inputs;
- *straightforward interface of event based sensors*, for example with DVS cameras as inputs to the system;
- *low computation latency*, due to the asynchronous computation of the spikes and the speed of their spread.

The most simple spiking neuron model is the **Integrate and Fire** (**IF**) model [27]. It is based on the idea that every neuron can be represented by a resistance-conductance (RC) equivalent circuit.

A little more complicated, but also more biological plausible model is represented by a modified version of the IF model, which is called **Leaky-Integrate and Fire** (**LIF**) model [27].

Figure 3 illustrates the model behavior of the interaction between two Neurons, called Pre-synaptic and Post-synaptic neurons. The *pre-synaptic meuron* connects its *axon* to the *dentrites* (*soma* in the figure) of the *post-synaptic neuron* through a *synapse*. The *synapse* is represented by a low-pass filter, while the *dentrite* or *soma* is represented as a reservoir of charge, i.e., a capacitance. When the reservoir state, called **Post-synaptic Membrane Potential** (PSP) overcomes the threshold (θ) the *neuron* fires a spike through its *axons*, and the reservoir state (PSP) resets to a value that is always less than the threshold θ (it can be 0 or a positive value). After that time, if there are others input spikes, the potential can increase again. This behavior can be modeled by the following differential Equation 1:

$$I(t) = \frac{u(t)}{R} + C\frac{dv}{dt} \tag{1}$$

The **Membrane Time Constant** τ_m is derived with Equation 2:

$$\tau_m \frac{du}{dt} = -u(t) + RI(t), \tag{2}$$

where $u(t)$ is the neuronal membrane potential at time t and $I(t)$ denotes the pre-synaptic input, which is determined by the weighted pre-neuronal activities.

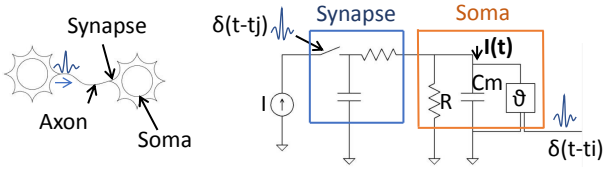

Fig. 3. Circuital representation of the **LIF** model for Post-synaptic Neuron that receives a spike sent by the Pre-synaptic Neuron [27].

Here, in case of no incoming spikes from the synapses of the neuron, the PSP decreases over time by a fraction called $voltage\ decay$. This model also introduces the concept of $refractory\ period$, that is a short time, after the input of a spike from the synapses, in which the neuron is unable to consider others spikes at the input, which are then discarded. Every synapse of the neuron has a *weight* that multiplies the incoming spikes before it can affect the PSP [28]. This is the key feature of the generalization and regression mechanisms of the SNNs.

To realize a given task accurately, the SNNs' weights need to be properly adjusted through a **Learning Process**. During this step, the SNN is fed by the train inputs of the dataset. With some method (**Learning Method**), that can be different according to the desired application, the weights are adjusted, in order to hit the target.

The SNN learning methods today are grouped in three main classes [28]:

1) *Direct supervised*: the SNN is stimulated with different patterns of spike trains and the synaptic weights are adjusted to achieve the desired output spike trains. The most common algorithms are based on back-propagation mechanism [29, 30].
2) *Indirect supervised*: a DNN is trained and then converted into an equivalent SNN [31, 32].
3) *Unsupervised*: the SNN is stimulated with a pattern of spike trains, but no human-produced labels are given. The SNN by itself searches the correlation properties between every sample. Unsupervised SNN learning can involve tasks such as cluster analysis [33] and anomaly detection [34].

Focusing on the *direct supervised* method and in particular on the SNN back-propagation, there are two main problems:

- $Spike\ as\ activation\ function$: since the SNN is based on spikes, i.e., impulses, the derivative of an impulse does not exist. The possible solution can be its approximation (e.g.,

a *surrogate gradient*) [35], but its implementation detaches from the biological model. However, with this solution many different learning rules can be applied and the SNN can also achieves high performance [29, 30].

- $Weight\ transport\ problem$ [28]: the SNN needs to have two paths, one for forward and one for backward. In this situation, the weights for these paths are correlated, being one the transposition of the other. This coherence is hard to maintain. One solution is to have random weights on the backward path, but this can be used only for simple problems [36].

D. Spatio-Temporal Back-Propagation Learning Method

The most common SNN learning method based on Back-propagation is the Spatio-Temporal Back-Propagation (**STBP**) [29]. The most used supervised learning rules control the interaction between neurons (**Spatial Domain** or **SD**) in order to find new weights. On the other hand, the unsupervised methods monitor the trend over time of the neurons' PSP (**Time Domain** or **TD**), to do their tasks. The STBP uses both information to train the SNNs, thus using the back-propagation on both dimensions. The STBP algorithm starts from the LIF neuron model (Equation 2) and resolves this TD differential problem to obtain Equation 3:

$$u(t) = u(t_{i-1})e^{\frac{t_{i-1}-t}{\tau}} + RI(t) \tag{3}$$

In this way, both the TD and the SD components are present in the STBP method. $I(t)$ represents the *spatial accumulation* and $u(t_{i-1})$ represents the *leaky temporal memory*. Then, since the back-propagation algorithm takes many advantages from the iterative representation of the gradient descent, the authors of [29] developed iterative LIF-based SNNs, in which the iterations occur in both the SD and TD as follows (Equation 4):

$$\begin{aligned} x_i^{t+1,n} &= \sum_{j=1}^{l(n-1)} w_{ij}^n o_j^{t+1,n-1} \\ u_i^{t+1,n} &= u_i^{t,n} f\left(o_i^{t,n}\right) + x_i^{t+1,n} + b_i^n \\ o_i^{t+1,n} &= g\left(u_i^{t+1,n}\right), \end{aligned} \tag{4}$$

where:

$$f\left(o_i^{t,n}\right) \approx \begin{cases} \tau, & o_i^{t,n} = 0 \\ 0, & o_i^{t,n} = 1 \end{cases} \quad for\ little\ \tau \tag{5}$$

$$g(x) = \begin{cases} 1, & x \geq V_{th} \\ 0, & x < V_{th} \end{cases} \tag{6}$$

The notations of the above formulas (Equations 4, 5 and 6) are the following:

- the index t represents the current time step;
- n and $l(n)$ denote the n^{th} layer and its number of neurons, respectively;
- w_{ij} is the synaptic weight between the j^{th} pre-synaptic neuron and the i^{th} post-synaptic neuron;
- o_j is the neuronal output of the j^{th} neuron;
- x_i is the pre-synaptic input of the i^{th} neuron;
- b_i is the bias of the i^{th} neuron.

The learning rule defines a **Loss Function** (Equation 7) and a *Gradient Descendent Optimization Method* that consists of minimizing the loss function under a given time window T, using its derivative.

$$L = \frac{1}{2S}\sum_{s=1}^{S}\left\| y_s - \frac{1}{T}\sum_{t=1}^{T} o_s^{t,N} \right\|_2^2 \tag{7}$$

where y_s is the label and o_s is the output of the network for the s^{th} sample.

Then, four different cases to perform the calculation of Equation 8 are distinguished:

$$\delta_i^{t,n} = \frac{\partial L}{\partial o_i^{t,n}} \tag{8}$$

1) $t = T\ and\ n = N\ (output\ layer)$:

$$\frac{\partial L}{\partial u_i^{T,N}} = \delta_i^{T,N} \frac{\partial o_i^{T,N}}{\partial u_i^{T,N}} \tag{9}$$

2) $t = T\ and\ n < N\ (inner\ layer)$:

$$\frac{\partial L}{\partial u_i^{T,n}} = \delta_i^{T,n} \frac{\partial g}{\partial u_i^{T,n}} \tag{10}$$

3) $t < T\ and\ N = n\ (output\ layer)$:

$$\frac{\partial L}{\partial u_i^{t,N}} = \delta_i^{t+1,N} \frac{\partial g}{\partial u_i^{t+1,N}} f\left(o_i^{t,n}\right) \tag{11}$$

4) $t < T\ and\ N < n\ (inner\ layer)$:

$$\frac{\partial L}{\partial u_i^{t,n}} = \delta_i^{t,n} \frac{\partial g}{\partial u_i^{t,n}} + \delta_i^{t+1,n} \frac{\partial g}{\partial u_i^{t+1,n}} f\left(o_i^{t,n}\right) \tag{12}$$

Afterwards, these differential equations (Equations 13 and 14) can be defined:

$$\frac{\partial L}{\partial \boldsymbol{b}^n} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{t,n}} \frac{\partial \boldsymbol{u}^{t,n}}{\partial L \boldsymbol{b}^n} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{t,n}} \tag{13}$$

$$\frac{\partial L}{\partial \boldsymbol{W}^n} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{t,n}} \frac{\partial \boldsymbol{u}^{t,n}}{\partial x^{t,n}} \frac{\partial x^{t,n}}{\partial W^n} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{t,n}} \boldsymbol{o}^{t,n-1} \tag{14}$$

such that they can be used to perform the **Gradient Descendent Optimization Algorithm** to achieve high performance.

Another key point of this rule is the approximation of the derivative of Dirac functions. The process for which each occurrence of the derivative of the spiking nonlinearity is replaced by the derivative of a smooth function is called **Surrogate Gradient** [35].

E. The Loihi Neuromorphic Research Chip

Towards high energy-efficiency, it is convenient to implement SNNs on a specialized hardware, called **Neuromorphic Chip**, to guarantee high efficiency both in terms of working time and power consumption of the application. More specifically, neuromorphic hardware platforms simulate the processes that happens in the brain with one neural model, for example the LIF, using an asynchronous mechanism, as shown in Figure 4, in which every part represents one neuron attribute, for example *Axon*, *Synapse* and *Dendrite*. In some cases, on the Chip there is also a Learning part that can be used for *Online Learning* or *Continual Lifelong Learning* [37].

There exist several neuromorphic chips developed by premier industries and academia, like $IBM\ Truenorth$ [38], $SpiNNaker$ [39], $Intel\ Loihi$ [2]. The Loihi chip, which is used in this work, adopts the **CUrrent BAsed (CUBA) LIF** to model the neurons' behavior. I.e., all the neurons are a reservoir of charge (**Dendrite**), and when this overcomes the voltage threshold, there is a current spike on the output axons. This mechanism is very similar to the behavior that happens on the LIF model and it can be visualized with the help of the following Equation 15.

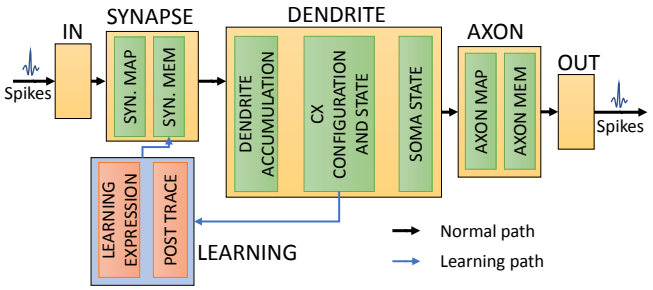

Fig. 4. Simplified Neurocore abstraction mechanism on the Loihi Neuromorphic chip [2].

$$\dot{v}_i(t) = -\frac{1}{\tau_v} v_i(t) + u_i(t) - \theta_i \sigma_i(t), \tag{15}$$

where:

- τ_v is the leakage contribution;
- u_i is the input current from the synapses;
- v_i is the dendrite potential;
- θ_i is the voltage threshold;
- σ_i represents the generation of an output spike on axons.

The Loihi chip is composed of *neurocores* that represent groups of neurons, whose behavior is simulated by the *compartments*. Every neurocore exchanges information (spikes) to the others by an asynchronous Network-on-Chip (NoC) in the form of packetized messages. To spread the spikes in an asynchronous way, a *mesh operation* is used for each time step, which can be summarized in four points:

1) each neurocore independently iterates its compartments and, if a compartment is in spike firing state, this information is send with a message onto the NoC;
2) the messages are sent to all destination neurocores;
3) when a neurocore ends its internal distribution, it sends a barrier signal to the neighbors;
4) when all the neurocores receive such signal, the time step is incremented.

Every chip can contain 128 neurocores, but to implement wider and deeper SNNs, many chips can work together without any increase in the latency of message exchange. The programmer has to follow some constraints in the implementation:

- every neurocore can have a maximum of 1024 compartments;
- the max fan-in of every neurocore is 4096 pre-synapses;
- the max fan-out of every neurocore is 4096 post-synapses;
- the total synaptic fan-in state mapped to any neurocore must not exceed 128 KB.

The *On-chip Learning* engine can operate to implement *Online Learning* strategies or unsupervised learning with local information. On the other hand, for the back-propagation methods, the given SNN can be trained offline and then transported to the Loihi with the $NxSDK\ API$ commands. Its API gives many facilitation to the programmer. For example, the main SNN temporal parameters (synaptic, axon and refractory delay) can be configured and adapted to have a polysynchronous dynamic. Moreover, a noise injector can be activated to limit overfitting when the learning engine is used.

III. Problem Analysis and General Design Decisions

In the classification problem that we face, we can use a supervised learning method and train the network based on the desired behavior. Every sample is represented by a stream of events, where a stream represents the same object to classify. In the same sample, the present spikes are correlated in time and space with the past and future spikes [3]. To achieve good performance, we have to take into account this temporal correlation and use a learning method capable to exploit this property. As claimed in [29], the STBP is one of the best offline learning methods, and achieves very high classification accuracy in tasks that involve event-based camera streams. It also uses both TD and SD to calculate the gradients and train the SNN. Therefore, we employ this learning method in our experiments.

This is also a real-time problem, as the system should be very reactive and perform the correct prediction in few milliseconds. Since we want a very reactive prediction, we can use only a subset of input information, and therefore implement the **Attention Window** strategy. To find the area which focuses the attention on input data, we analyze and evaluate the event occurrences, both in train and test sets of the N-CARS dataset [3]. Due to the relatively large dimension of this dataset, this study resembles with a good approximation the real problem and does not affect the generalization property of our system.

The evaluation of the event occurrences in different attention windows is shown in Figure 5. Most of the information is contained in the area of size 50×50 in the bottom-left corner, both in train and test set. Hence, as reported in Table I, we can use this as the $first\ attention\ window$. The $second\ attention\ window$ has a doubled size (i.e., 100×100) and also starts from the bottom-left corner.

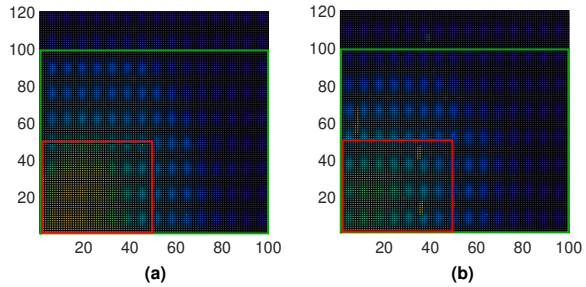

Fig. 5. Event occurrences on (a) test and (b) train sets of the N-CARS dataset.

TABLE I
DELIMITED POINTS FOR **ATTENTION WINDOWS**.

Attention window	P. 0 (x,y)	P. 1 (x,y)	P. 2 (x,y)	P. 3 (x,y)
First attention window	(0,0)	(0,50)	(50,50)	(50,0)
Second attention window	(0,0)	(0,100)	(100,100)	(100,0)

Considering its practical implementation on an existing **Neuromorphic Hardware**, the **Intel Loihi Research Chip**, the network is designed following all the constraints of this chip, summarized in Table II.

TABLE II
MAIN CONSTRAINTS FOR DEVELOPING THE SNN IMPLEMENTED ON THE **INTEL LOIHI NEUROMORPHIC RESEARCH CHIP**.

Property	Constrain
Maximum Compartments per Core	1024 Compartments
Maximum fan-in of a Core	4096 Pre-Synapses
Maximum fan-out of a Core	4096 Post-Synapses
Synaptic fan-in state size	128 KB

A summary of the general decisions taken after analyzing the problem is shown in Table III.

TABLE III
GENERAL DECISIONS TAKEN AFTER ANALYZING THE PROBLEM.

Properties of the problem	Decision
Knowledge of the correct output	Use supervised learning rule
Time and space correlation	Take into account TD and SD
Real-time	Use simplest SNN
High performance of vision sensor	Use event-based camera
Good profiling of real problem	Use N-CARS dataset
Many information in limited area	Use attention windows
Low power consumption	Use Neuromorphic Chip

IV. CarSNN: Our Proposed SNN for Event-based Cars vs. Background Classification

Our methodology to design the SNN model for the *"cars vs. background"* classification, which we call **CarSNN**, is composed as a three-step process, as shown in Figure 6. After the definition of the SNN model architectures considering different attention windows in Section IV-A, the methods for finding the parameters for SNN training and feeding input data are discussed in Sections IV-B and IV-C, respectively.

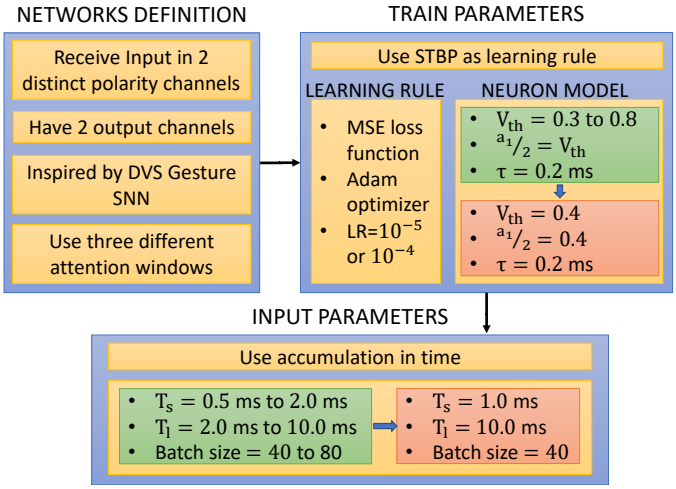

Fig. 6. Three-step process followed to design our *CarSNN* with the training and feeding input parameters.

A. CarSNN Model Design

To achieve good classification results, our *CarSNN* receives the input events in two distinct polarity channels, one for positive and one for negative events. Towards the generalization of the problem, we consider this as a multi-classification problem (i.e., not as a simple binary classification problem). Therefore, the output layer of the *CarSNN* consists of two neurons that correspond to the two possible classes, one for cars and the other for background objects.

Since the architecture proposed in [15] achieved high classification accuracy and low latency on the IBM DVS128 gesture dataset [14], we modify this model to correctly function for the N-CARS dataset. Compared to the model of [15], our *CarSNN* has different output channels, kernel size and padding on the first convolutional layer, and different sizes of the last two dense layer.

Based on the attention window analysis, we develop three different SNNs for the three different sizes of input images:

1) Size 128×128 (Table IV): the model is very similar to the SNN proposed in [15]. Since this size overcomes the N-CARS dataset image size, which is 120×100, the exceeded pixels do not produce spikes and are padding by zeros (no event). This image size is equal to the resolution of one of the most used DVS camera [13]. Therefore, this network can be easily implemented with it.
2) Size 50×50 (Table V): this uses the first attention window as described in Section III.
3) Size 100×100 (Table VI): this uses the second attention window as described in Section III.

TABLE IV
SNN MODEL FOR FULL-SIZE IMAGES (INPUT SIZE 128×128).

Layer type	In ch.	Out ch.	Kernel size	Padding	Stride
Av. pooling	2	2	4	–	–
Convolution	2	32	3	1	1
Av. pooling	32	32	2	–	–
Convolution	32	32	3	1	1
Av. pooling	32	32	2	–	–
Dense	2048	1024	–	–	–
Dense	1024	2	–	–	–

TABLE V
SNN MODEL FOR FIRST ATTENTION WINDOW (INPUT SIZE 50×50).

Layer type	In ch.	Out ch.	Kernel size	Padding	Stride
Av. pooling	2	2	4	–	–
Convolution	2	32	3	1	1
Av. pooling	32	32	2	–	–
Convolution	32	32	3	1	1
Av. pooling	32	32	2	–	–
Dense	512	144	–	–	–
Dense	144	2	–	–	–

TABLE VI
SNN MODEL FOR SECOND ATTENTION WINDOW (INPUT SIZE 100×100).

Layer type	In ch.	Out ch.	Kernel size	Padding	Stride
Av. pooling	2	2	4	–	–
Convolution	2	32	3	1	1
Av. pooling	32	32	2	–	–
Convolution	32	32	3	1	1
Av. pooling	32	32	2	–	–
Dense	1568	512	–	–	–
Dense	512	2	–	–	–

B. Parameters for Training

Using a supervised learning rule based on backpropagation, like the STBP, it is possible to tune several hyper-parameters.

We focus our attention on:

- **loss function**: we adopt the **Mean Squared Error** (**MSE**) loss criterion, since it achieves the highest performance in [29];
- **optimizer**: we use **Adam** [40], because it seems the best for the STBP;
- **learning rate** (**lr**): after some preliminary tests, we find the best value is around $1e^{-5}$ and $1e^{-4}$, where with the latter value the training is faster and the SNN achieves good accuracy results in fewer epochs.

Since the adopted learning rule is directly implemented on the SNNs with LIF neurons, other specific parameters can be adjusted. The kernel of the LIF neuron model can be described by Equation 4 (discussed in section II-D). We focus on the formalization of the membrane potential update ($u_i^{t+1,n}$) and highlight the membrane potential decay factor τ (Equation 16).

$$u_i^{t+1,n} = u_i^{t,n}\tau(1 - o_i^{t,n}) + \sum_{j=1}^{l(n-1)} w_{ij}^n o_j^{t+1,n-1} + b_i^n \qquad (16)$$

Another fundamental parameter of a LIF neuron is its **threshold** (V_{th}). If the membrane potential overcomes this value an output spike is generated and the potential is reset to a specific

value. For each experiment, all the neurons have the same V_{th} and 0 as reset value.

The third parameter that needs to be set ($\frac{a_1}{2}$) is related to the approximation of the derivative of spiking nonlinearity. We use the rectangular pulse function defined in Equation 17:

$$h_1(u) = \frac{1}{a_1} \text{sign}\left(|u - V_{th}| < \frac{a_1}{2}\right) \tag{17}$$

In the following, we perform some experiments to set the previously-discussed parameters, with a particular focus on V_{th}. We made these decisions:

- V_{th}: we change this value from 0.3 to 0.8 and evaluate which curve achieves the best accuracy;
- $\frac{a_1}{2}$: it assumes the same value of the threshold, as this assumption is made in [29];
- τ: this value must be small to have good approximation of the neuron model and in particular of $f(o_i^{t,n})$ (see Equation 5). We set it to be equal to 0.2 ms.

To speed up this process and have good performance, we introduce an accumulation mechanism. We accumulate the spikes at a constant time-rate called sample time (T_{sample}); for these first experiments this value is set to 10 ms. Every T_{sample} time we construct a new input image that feeds the SNN. The events that compose the image are summed by the following simple rule, based on which each pixel can have a maximum of one spike per channel. Each derived image is maintained stable to the input of the proposed SNN by a time window of 15 time steps. Therefore, this accumulation mode can compress the input information. The accuracy that we evaluate is referred to every single sample (i.e., accumulated image) on a training of 300 epochs. Table VII and Figure 7 report the results of these experiments, where we use the SNN with the full size image (Table IV).

TABLE VII
EXPERIMENTS TO FIND THE BEST VALUE OF V_{th}.

input size	V_{th}	$\frac{a_1}{2}$	τ	T_{sample}	batch size	lr	accuracy
			ms	ms			%
128 × 128	0.3	0.3	0.2	10	20	$1e^{-5}$	83.0
128 × 128	**0.4**	0.4	0.2	10	20	$1e^{-5}$	**84.0**
128 × 128	0.5	0.5	0.2	10	20	$1e^{-5}$	82.4
128 × 128	0.6	0.6	0.2	10	20	$1e^{-5}$	81.9
128 × 128	0.8	0.8	0.2	10	20	$1e^{-5}$	82.6

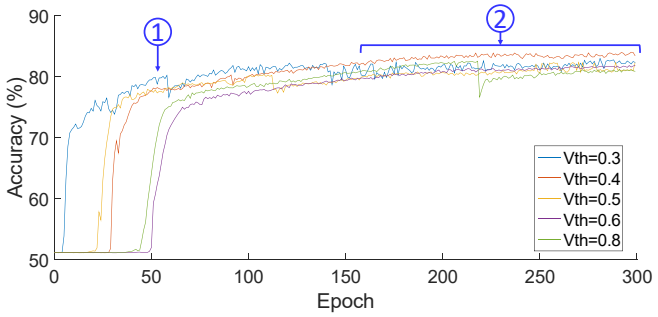

Fig. 7. Percentage of accuracy for the experiment made to evaluate the best value for V_{th}.

From Table VII, we notice that the best accuracy is achieved when V_{th} is equal to **0.4**. Moreover from Figure 7, we can notice that, while a V_{th} equal to 0.3 leads to a relatively high accuracy after a few epochs (see pointer ① in Figure 7), the training curve with V_{th} equal to 0.4 have less instability than for the other experiments (see pointer ②). These two reasons lead us to choose **0.4** for the V_{th} parameter.

C. Parameters for Feeding the Input Data

As discussed in the previous section, the input spikes are given to the SNN with an accumulation strategy, to speedup the training. From the experiments conducted in Table VII, despite this limitation, we notice that the accuracy is quite high. Therefore, we keep this property that gives us some advantages:

- decrease power consumption;
- increase the reactivity of the system, because input data are compressed.

Moreover, We also give an upper bound to the latency of the system of 10 ms. Hence, for the train, we take only 10 ms from the dataset sample stream with a random initial point. This is defined as the maximum acceptable sample length (T_l). With this constraint, two different approaches can be adopted:

1) accumulate the spikes every T_l time ($T_{sample} = T_l$) and do the prediction on a unique input image for the entire input stream, as we did in the previous experiments;
2) accumulate the spikes in order to have more than one input image for every input stream ($T_{sample} < T_l$), then see what is the class with majority prediction.

We conduct some analyses to find the best sample time and the variation of the accuracy with two different batch sizes (BS on Table VIII). In these experiments we use the second approach for the image accumulation and we set the parameters as follows: $V_{th} = 0.4$, $\frac{a_1}{2} = 0.4$, $\tau = 0.2$ ms.

The training lasts for 200 epochs and to speed up this process, as discussed in section IV-B, we use a learning rate equal to $1e^{-4}$ and a minimum batch size of 40. We also use three different metrics to evaluate the accuracy:

- one shot accuracy on test data ($acc._s$): it is the accuracy found on all the samples taken at T_s of the test dataset;
- accuracy on test data ($acc._{test}$): it is the accuracy for all the sample stream of the test dataset, computed based on the majority prediction of the part of the stream with sample length equal to T_l;
- accuracy on train data ($acc._{train}$): it is the counterpart of the accuracy on test data, but calculated on train streams of the dataset.

TABLE VIII
EXPERIMENTS TO FIND THE BEST VALUE FOR T_s, T_l AND BATCH SIZE.

Input size	T_s	T_l	BS	lr	$acc._s$	$acc._{test}$	$acc._{train}$
	ms	ms			%	%	%
128 × 128	1.0	2.0	80	$1e^{-4}$	80	79	83
128 × 128	1.0	4.0	80	$1e^{-4}$	80	80	86
128 × 128	1.0	6.0	80	$1e^{-4}$	51	51	51
128 × 128	1.0	8.0	80	$1e^{-4}$	80	79	89
128 × 128	1.0	2.0	40	$1e^{-4}$	80	77	86
128 × 128	1.0	4.0	40	$1e^{-4}$	80	83	88
128 × 128	1.0	6.0	40	$1e^{-4}$	72	70	90
128 × 128	**1.0**	**8.0**	**40**	$1e^{-4}$	**81**	**86**	91
128 × 128	1.0	10.0	40	$1e^{-4}$	80	86	**94**
128 × 128	2.0	10.0	40	$1e^{-4}$	51	51	51
100 × 100	0.5	10.0	40	$1e^{-4}$	75	80	84
100 × 100	**1.0**	**10.0**	**40**	$1e^{-4}$	**81**	**85**	**92**
100 × 100	2.0	10.0	40	$1e^{-4}$	51	51	51
50 × 50	0.5	10.0	40	$1e^{-4}$	67	71	79
50 × 50	1.0	10.0	40	$1e^{-4}$	71	75	81
50 × 50	**2.0**	**10.0**	**40**	$1e^{-4}$	**74**	**77**	**83**

The results in Table VIII provide us the necessary feedback for setting the value of T_s. If it is small (i.e., 0.5 ms) there are more points for the same stream sample. However, it is very difficult to train the SNNs, because the accumulation has not effect and

the temporal correlation is lost. On the other hand, the accuracy is low when we use high T_s (i.e., 2 ms). The best trade-off is obtained when T_s is equal to 1 ms.

Moreover, the batch size influences the training process. Indeed, to have high accuracy, the value of BS should be limited to 40.

In the first experiments of Table VIII, we consider only the variation of T_l and BS. With constant BS and same value of $acc._s$, the $acc._{test}$, as expected, increases or remains stable with the increasing of T_l. This behavior is due to having more sub-predictions to compute the final result when T_l is large. The changes in the $acc._s$ are justified by the non-deterministic training process.

V. Evaluation of our CarSNN Implemented onto the Loihi Neuromorphic Chip

The STBP learning method is based on the backpropagation, without using the local information. Moreover, the gradients are computed with Equations 13 and 14, which are not directly implementable into the on-chip learning section of the Intel Loihi Neuromorphic hardware. For these reasons, our *CarSNN* is trained offline and then the resulting parameters are mapped onto the neuromorphic chip. An overview of the tool-flow for conducting the experiments is shown in Figure 8.

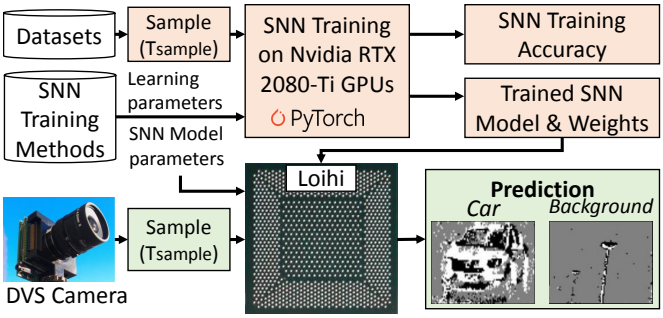

Fig. 8. Setup and tool-flow for conducting our experiments.

A. Experimental Setup and Accuracy Results for CarSNN Offline Training

Coherently with the analysis due in previous sections, in order to train and validate the prediction system we use the N-CARS dataset. We take into account this dataset also for the two fundamental reasons that it collects event-based camera streams and it is the largest labeled event-based dataset acquired in real-world conditions [3].

We describe the SNNs using the PyTorch library [41]. In these codes, we model the SNNs' functional behavior with the implementation of the Equation 16 that contains the mechanism to update the membrane potential.

We run the experiments on a workstation having CentOS Linux release 7.9.2009 as operating system and equipped with an Intel Core i9-9900X CPU and Nvidia RTX 2080-Ti GPUs.

The setting of the hyper-parameters follows the analyses made in sections III and IV, and are summarized in Table IX.

TABLE IX
Parameters of the experiments.

Epochs	T_s	T_l	BS	lr	V_{th}	$\frac{a_1}{2}$	τ
	ms	ms					ms
200	1.0	10.0	40	$1e^{-3}$ *to* $1e^{-6}$	0.4	0.4	0.2

The dataset streams are randomly shuffled and the sample of T_l is taken starting from a random initial point. We set the BS to 40, that gives best accuracy in the previous experiments (according to Table VIII), and maintains a reasonable the training time duration. We set the same values of $T_s = 1\ ms$ and $T_l = 10\ ms$ for the three experiments, to have a fair comparison between them. These two values leverage the trade-off found from the results in Table VIII. The parameters for the SNN model are the same used in Section IV-C. The learning rate (lr) decreases by 0.5 every 20 epochs, starting for the value $1e^{-3}$. With this approach, the accuracy slightly increases, compared to having a fixed lr.

To ease the model mapping onto the Loihi Neuromorphic Chip, only the weights are updated during training, while the bias is forced to 0. The train lasts for 200 epochs and every sample taken at T_s time is evaluated for 20 time steps. With these hardware and software settings, the training for one single epoch on all the dataset samples is measured to be about 300 seconds. For the inference, the mean latency for all samples, given at the time T_s, is about 0.8 ms. Table X shows the results in terms of the same accuracy policies as defined in Section IV-C.

TABLE X
Results of the offline training experiments.

Input size	$acc._s$	$acc._{test}$	$acc._{train}$
	%	%	%
128×128	80.1	85.7	93.6
100×100	80.5	86.3	95.0
50×50	72.6	78.7	85.3

The accuracy values for the attention window of size 100×100 are comparable to the results for the full image size (128×128), and indeed exhibit slightly higher $acc._{test}$ and $acc._{train}$. It can be explained because the cropped part of the sample is not important for the correct classification and might lead to an SNN misfunctioning. On the other hand, the input values consisting of a small part of the original image (50×50) lead to a significant accuracy decrease.

Moreover, from the results in Table X, we can notice an overfitting, due to the gap between $acc._{test}$ and $acc._{train}$, which can be considered to be the upper bound of the accuracy for our developed *CarSNN* models.

B. CarSNN Implemented on Loihi

To implement our network on the Intel Loihi Neuromorphic Chip we have to exploit some similarity between its model and our offline model used for the previous experiments. Equation 18 reports how the Compartment Voltage ($CompV$), which represents the membrane voltage of a neuron, is evaluated by the neuromorphic hardware [2].

$$CompV_{t+1} = CompV_t \frac{2^{12} - \delta_v}{2^{12}} + CompI_{t+1} + bias \qquad (18)$$

The Compartment Current ($CompI$) is formulated by Equation 19, where the sum expression represents the accumulation of the weighted incoming spikes from j^{th} pre-synaptic neuron.

$$CompI_{t+1} = CompI_t \frac{2^{12} - \delta_i}{2^{12}} + 2^{6+wgtExp} \sum_j w_j s_{j_{t+1}} \qquad (19)$$

In Equations 18 and 19, we can set the following parameters:

- δ_i: Compartment Current Decay;
- δ_v: Compartment Voltage Decay;
- $bias$: bias component on $CompV$;
- $wgtExp$: value used to implement very different weights between different SNN layers.

Comparing the formulation of our offline model (i.e., Equation 16) and the Equation 18, we notice its similarity to

Equations from 20 to 23.

$$CompV_t = u_t \tag{20}$$

$$CompI_t = \sum_j w_j o_{j_{t+1}} \quad if\ \delta_i = 2^{12} \tag{21}$$

$$\frac{2^{12} - \delta_v}{2^{12}} = \tau \tag{22}$$

$$bias = b \tag{23}$$

We implement only the *CarSNN* described in Table IV, which achieves good offline accuracy results (as indicated in Table X) and it represents the most complex developed network, based on latency, power consumption and number of neurons.

The Loihi Neuromorphic Hardware uses only 8 bits for the storage of weights. The maximum range of our weights is $(-7, 6)$. Since these values are very different between layers and the $wgtExp$ is limited we:

1) multiply weights and V_{th} by 25 (this value do not consider the default multiplication for 2^6 of weights and V_{th} made on the Loihi);
2) use all the 8 bits to store our values.

According to Equations 20-23, the other neuromorphic hardware parameters can be adjusted.

All the setup parameters are summarized in Table XI.

TABLE XI
TRANSLATION OF PARAMETERS TO THE LOIHI CHIP.

Offline implementation			Loihi implementation		
Parameter	Value	Precision	Parameter	Value	Precision
V_{th}	0.4	Floating point 64 bits	$V_{th\ mant}$	10	Fixed point 12 bits
$weight$	$\times 1$	Floating point 64 bits	$weight$	$\times 25$	Fixed point 8 bits
τ	0.2	Floating point 64 bits	δ_v	3276	Fixed point 12 bits
b	0	Floating point 64 bits	$bias$	0	Fixed point 8 bits
–	–	Floating point 64 bits	δ_i	0	Fixed point 12 bits

We define our *CarSNN* using the Intel Nx SDK API version 0.9.5 and run it on the Nahuku32 partition, in particular we use the NxTF Layers, such as NxConv2D, NxAveragePooling2D and NxDense utilities. This kind of implementation is useful to automatically improve the performance of the SNN in a simple manner. The *CarSNN* is tested on the N-CARS dataset. Every sample at T_s is replicated for 10 timestep and between samples we insert a blank time of 7 timestep. The number of timesteps per inference is 17. This decision is necessary to follow the real-time constraint of a maximum inference latency of 1 ms.

In the results reported in Table XII, the mean latency, referred to the time used to evaluate every sample at T_s, is calculated through the multiplication between the mean total execution time (in timesteps) and the number of timesteps per inference.

On the other hand, the maximum latency is referred to the maximum "spiking time" for every timestep, considering the time in which the Loihi Chip is used and makes the classification decision. This value can be used to evaluate whether the latency constraint is met. It does not include the time overhead used to exchange results between the chip and the host system, that can be suppressed by directly using output ports.

TABLE XII
RESULTS OF THE *CarSNN* IMPLEMENTED ONTO THE LOIHI CHIP.

$acc._s$	$acc._{test}$	Neurons	Synapses	Neurocores	Mean latency	Max latency
%	%	number	number	number	μs	μs
72.16	82.99	54,274	5,122,048	151	899.6	≈ 700

From Table XII and the Figure 9, the following observations can be made:

- The $acc._{test}$ for the implementation onto the Loihi chip is 2.6% lower than the offline application.

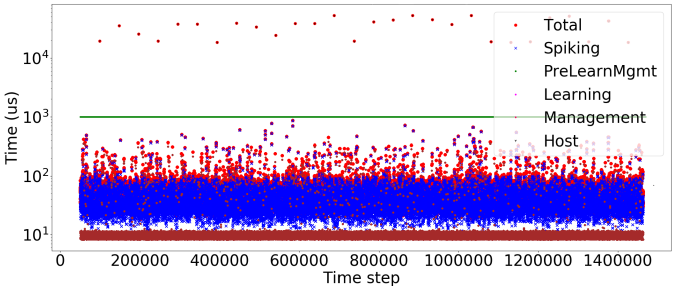

Fig. 9. Execution time for every timestep. The green line represents the limit for the spiking time and it is set to 1 ms (T_s).

- The maximum latency does not exceed T_s (1 ms).

Table XIII describes the power and energy consumption of the application implemented on the Neurompрphic Chip. In particular:

- *LakeMounts Power*: it is the consumption of the embedded processors [2] used to menage neurons and exchange messages with the host system.
- *Neuro-cores Power*: it represents the consumption for the neurons.
- *System Power*: it is the consumption of the entire system, where a large part of it is represented by the static power used for the inactive chip of the used partition. It uses only 2 chips out of 32.
- *Energy per inference*: it is the mean energy consumed to classify one sample.

TABLE XIII
POWER AND ENERGY CONSUMPTION OF THE *CarSNN* IMPLEMENTED ONTO THE LOIHI CHIP.

LakeMounts Power	Neuro-cores Power	System Power	Energy per Inference
mW	mW	mW	μJ
40.8	314.5	1375.4	319.7

Hence, Table XIII reports the power and energy consumption of the application implemented onto the Loihi Chip, that is several orders of magnitude lower than the same measured on GPUs.

C. Comparison with the State-of-the-Art

To the best of our knowledge, *CarSNN* is the the first Spiking Convolutional Neural Network (CNN) designed to perform event-based *"cars vs. background"* classification on neuromorphic hardware. This is also the first method that uses statistic analysis of events occurrences to indicate different attention windows on it. In this paper, we use a simple yet efficient mechanism for event accumulation in time, to maintain the time correlation between spikes. In the related works, to achieve good performance, the time correlation is maintained with different methods:

- Histograms of Averaged Time Surfaces (HATS) [3]: it uses local memory to compute the average of Time Surfaces, which represents the recent temporal activity within a local spatial neighborhood.
- Hierarchy Of Time Surfaces (HOTS) [42]: it uses the computation of Time Surfaces in a hierarchical way between the layers.
- Gabor-filter [43]: it considers the spatial correlation between different events and assigns them to the channels based on this information.

In HATS [3], all approaches are evaluated by a simple linear Support Vector Machine (SVM) classifier on the N-CARS dataset. The results of this simple classifier method are compared with our *CarSNN* in Table XIV. The Gabor-filter method adopts a two-

layer SNN before the SVM. As discussed in Section IV-B, since the upper bound of T_l is 10 ms for the real-time constraint, the comparison is made taking into account this limitation.

TABLE XIV
COMPARISON OF RESULTS FOR $T_l = 10\,ms$.

Classifier (Accumulation approach)	$acc._{test}$
Linear SVM (HOTS)	≈ 0.54
Linear SVM (Gabor-SNN)	≈ 0.66
Linear SVM (HATS)	≈ 0.81
CarSNN (128×128 attention window)	0.86
CarSNN (100×100 attention window)	0.86
CarSNN (50×50 attention window)	0.79

As highlighted in Table XIV, our *CarSNN* achieves better accuracy with a limited T_l than the Linear SVMs implemented after the use of different and more complicated accumulation approaches.

VI. CONCLUSION

In this work, we present *CarSNN*, a novel SNN model for the *"cars vs. background"* classification of event-based streams implemented on neuromorphic hardware. With a three-step process, the network model, training parameters, and input parameters are defined. An attention window mechanism is proposed to accumulate the events focusing the attention on the region in which the majority of the events occur. Two versions of our *CarSNN* with different attention windows achieve 86% accuracy (drops to 83% when implemented onto the Loihi Chip), with only 0.72 ms latency, in the worst case, which is 5% higher than the previous state-of-the-art approaches with an upper bound of 10 ms latency. Moreover, considering also the power consumption of only 315 mW for its implementation on the Loihi Neuromorphic Chip, our *CarSNN* establishes as a prominent method for embedded real-time classification, and opens new avenues toward resource-constraint efficient AD applications on neuromorphic hardware.

ACKNOWLEDGMENTS

This work has been partially supported by the Doctoral College Resilient Embedded Systems, which is run jointly by the TU Wien's Faculty of Informatics and the UAS Technikum Wien. This work was also jointly supported by the NYUAD Center for Interacting Urban Networks (CITIES), funded by Tamkeen under the NYUAD Research Institute Award CG001 and by the Swiss Re Institute under the Quantum Cities™ initiative, and Center for CyberSecurity (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104.

REFERENCES

[1] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 x 128 120db 30mw asynchronous vision sensor that responds to relative intensity change," *ISSCC*, 2006.
[2] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, 2018.
[3] A. Sironi *et al.*, "Hats: Histograms of averaged time surfaces for robust event-based object classification," *CVPR*, 2018.
[4] P. de Tournemire *et al.*, "A large scale event-based detection dataset for automotive," *arXiv:2001.08499*, 2020.
[5] Y. Hu *et al.*, "Ddd20 end-to-end event camera driving dataset: Fusing frames and events with deep learning for improved steering prediction," in *ITSC*, 2020.
[6] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, 2017.
[7] M. Capra *et al.*, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, 2020.
[8] C. D. Schuman *et al.*, "A survey of neuromorphic computing and neural networks in hardware," *ArXiv:1705.06963*, 2017.
[9] R. V. W. Putra, M. A. Hanif, and M. Shafique, "Sparkxd: A framework for resilient and energy-efficient spiking neural network inference using approximate dram," in *DAC*, 2021.
[10] R. V. W. Putra and M. Shafique, "Fspinn: An optimization framework for memory-efficient and energy-efficient spiking neural networks," *IEEE TCAD*, 2020.
[11] R. V. W. Putra and M. Shafique, "Q-spinn: A framework for quantizing spiking neural networks," in *IJCNN*, 2021.
[12] R. V. W. Putra and M. Shafique, "Spikedyn: A framework for energy-efficient spiking neural networks with continual and unsupervised learning capabilities in dynamic environments," in *DAC*, 2021.
[13] C. Posch, D. Matolin, and R. Wohlgenannt, "A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds," *IEEE JSSC*, 2011.
[14] A. Amir *et al.*, "A low power, fully event-based gesture recognition system," in *CVPR*, 2017.
[15] K. Stewart, G. Orchard, S. B. Shrestha, and E. Neftci, "Online few-shot gesture learning on a neuromorphic processor," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2020.
[16] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *NeurIPS*, 2018.
[17] R. Ghosh *et al.*, "Spatiotemporal filtering for event-based action recognition," *arxiv:1903.07067*, 2019.
[18] A. Amir *et al.*, "A low power, fully event-based gesture recognition system," in *CVPR*, 2017.
[19] Q. Liu and S. Furber, "Real-time recognition of dynamic hand postures on a neuromorphic system," *IJECE*, 2015.
[20] Q. Wang, Y. Zhang, J. Yuan, and Y. Lu, "Space-time event clouds for gesture recognition: From rgb cameras to event cameras," in *WACV*, 2019.
[21] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, 2015.
[22] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output," *Proceedings of the IEEE*, 2014.
[23] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *CVPR*, 2017.
[24] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE TPAMI*, 2015.
[25] N. K. Kasabov, *Time-Space, Spiking Neural Networks and Brain- Inspired Artificial Intelligence*. Springer-Verlag Berlin Heidelberg, 2019.
[26] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, 2018.
[27] N. Phong *et al.*, "Silicon synapse designs for vlsi neuromorphic platform," *NORCHIP*, 2014.
[28] A. V.Gavrilov and K. O.Panchenko, "Methods of learning for spiking neural networks. a survey," *APEIE*, 2016.
[29] W. Yujie *et al.*, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in Neuroscience*, 2018.
[30] P. Gu, R. Xiao, G. Pan, and H. Tang, "Stca: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks," in *IJCAI*.
[31] R. Massa, A. Marchisio, M. Martina, and M. Shafique, "An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor," in *IJCNN*, 2020.
[32] Y. Xu, H. Tang, J. Xing, and H. Li, "Spike trains encoding and threshold rescaling method for deep spiking neural networks," in *SSCI*, 2017.
[33] P. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, 2015.
[34] J. Pereira and M. Silveira, "Learning representations from healthcare time series data for unsupervised anomaly detection," in *BigComp*, 2019.
[35] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, 2019.
[36] S. Bohte, J. Kok, and J. Poutré, "Spikeprop: backpropagation for networks of spiking neurons.," in *ESANN*, 2000.
[37] G. I. Parisi *et al.*, "Continual lifelong learning with neural networks: A review," *arXiv:1802.07569v4*, 2019.
[38] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, 2014.
[39] X. Jin *et al.*, "Modeling spiking neural networks on spinnaker," *Computing in Science Engineering*, 2010.
[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.
[41] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.
[42] X. Lagorce *et al.*, "Hots: A hierarchy of event-based time-surfaces for pattern recognition," *IEEE TPAMI*, 2017.
[43] A. C. Bovik, M. Clark, and W. S. Geisler, "Multichannel texture analysis using localized spatial filters," *IEEE TPAMI*, 1990.