POLITECNICO DI TORINO
Repository ISTITUZIONALE

LaneSNNs: Spiking Neural Networks for Lane Detection on the Loihi Neuromorphic Processor

(Article begins on next page)

21 May 2024

LaneSNNs: Spiking Neural Networks for Lane Detection on the Loihi Neuromorphic Processor

Alberto Viale[1,2,*], Alberto Marchisio[1,*], Maurizio Martina[2], Guido Masera[2], Muhammad Shafique[3]

[1]*Technische Universität Wien, Austria*   [2]*Politecnico di Torino, 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 represent important elements for the next generation of mobile robots and autonomous vehicles focused on increasingly intelligent, autonomous, and interconnected systems. The applications involving the use of these features must provide, by definition, real-time decisions, and this property is key to avoid catastrophic accidents. Moreover, all the decision processes must require low power consumption, to increase the lifetime and autonomy of battery-driven systems. These challenges can be addressed through efficient implementations of Spiking Neural Networks (SNNs) on Neuromorphic Chips and the use of event-based cameras instead of traditional frame-based cameras.

In this paper, we present a new SNN-based approach, called *LaneSNN*, for detecting the lanes marked on the streets using the event-based camera input. We develop four novel SNN models characterized by low complexity and fast response, and train them using an offline supervised learning rule. Afterward, we implement and map the learned SNNs models onto the Intel Loihi Neuromorphic Research Chip. For the loss function, we develop a novel method based on the linear composition of Weighted binary Cross Entropy (WCE) and Mean Squared Error (MSE) measures. Our experimental results show a maximum Intersection over Union (IoU) measure of about 0.62 and very low power consumption of about 1 W. The best IoU is achieved with an SNN implementation that occupies only 36 neurocores on the Loihi processor while providing a low latency of less than 8 ms to recognize an image, thereby enabling real-time performance. The IoU measures provided by our networks are comparable with the state-of-the-art, but at a much low power consumption of 1 W.

I. INTRODUCTION

In recent years, the design of reliable and efficient Autonomous Driving (AD) systems has become one of the key research directions of the incoming Smart mobility [1]. Therefore, it leads to the development of increasingly advanced algorithms and solutions. This paper proposes a class of Spiking Neural Networks (SNNs) that are directly implementable on one of the most advanced neuromorphic hardware for energy-efficient real-time deployment of advanced AD systems. Furthermore, we leverage the event-based cameras as the vision sensor, due to their appealing properties, such as energy-efficiency, biological plausibility, high dynamic range, and good compatibility with neuromorphic systems.

A. Target Research Problem and Research Challenges

To be able to drive safely, mobile robots and autonomous vehicles must continuously analyze the surrounding environment and must take into account any slightest variation to make the best decision and to prevent catastrophic accidents. Hence, it is essential that the *decision process* takes place in real time. Moreover, it is desirable that the developed AD system maintains low energy consumption[1], especially with its placement into battery-driven electric means of transport.

To better analyze a general AD problem, we can divide the decision process into two parts, which must follow the low-latency and low-power constraints:

1) **Vision**: the external environment is evaluated and captured by one or more sensors;
2) **Computation**: the sensed data is analyzed and the essential information to predict the reaction of the system is provided.

The vision system can be represented by cameras collecting images of the environment. The advanced dynamic vision sensors (DVS) enable event-based cameras that are specialized for detecting illumination changes, which mimic the behavior of the retina. They are very reactive, robust, and low-power devices [2]. Therefore, they represent an efficient choice for advanced AD applications [3].

A modern trend to address the complex AD problems is to deploy *Deep Neural Networks (DNNs)* that achieve high performance, but they are very expensive in terms of power consumption [4]. An alternate trend is to leverage the emerging *Spiking Neural Networks (SNNs)* [5]. Compared to DNNs, these SNNs have higher biological plausibility and exhibit event-based processing, thereby rendering them as a low-latency and energy-efficient choice for AD tasks. To further achieve low power consumption and low latency, the Neuromorphic Chips provide excellent hardware platform options [6], [7]. In this paper, we focus on the *"Lane detection"* problem.

Following these research targets, we design, optimize, and implement SNNs on the Intel Loihi Neuromorphic Research Chip [8], and evaluate them on the DET dataset [9]. Moreover, the vision system is based on a DVS event-based camera [10], [11].

B. Our Novel Contributions

We introduce **LaneSNNs** to detect pixels that represent the lanes on general images collected by an event-based camera. An overview of our novel contributions is shown in Fig. 1. In particular, our key contributions are:

---

*These authors contributed equally to this work.

[1]Note that high-performance GPUs, generally used to face Artificial Intelligence problems, take high power and area, and generate heat (requiring big coolants and package) which makes them infeasible to be placed in the electronic control units (ECUs).

- we follow the *Semantic Segmentation* approach to implement the algorithms (**Sec. II-A**);
- we adopt a *dataset pre-processing unit* to reduce the resolution of input and output images and to guarantee low complexity (**Sec. II-B**);
- we introduce a *novel loss function* that provides a trade-off between the *Weighted Binary Cross Entropy* and the *Mean Squared Error* measures (**Sec. II-C**);
- we implement the SNNs on the *Intel Loihi Neuromorphic Research Chip* [8] (**Sec. IV-C**);

As evaluation, we analyze results in form of different *Pareto Curves* [12] (**Sec. IV-D**) and we compare our results with the state-of-the-art (**Sec. IV-F**).
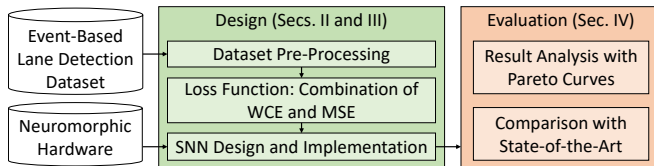

Fig. 1: Overview of our novel contributions.

Paper Organization: Sec. II presents our target research problem and the general design decisions. **Sec. III** discusses the LaneSNNs design and the anti-overfitting strategies. **Sec. IV** evaluates the experimental results, and the implementations of our LaneSNNs onto the Loihi chip. **Sec. V** concludes the paper.

II. PROBLEM ANALYSIS AND GENERAL DECISIONS

A. Lane Detection Methods

The lane detection problem is one of the key tasks in the AD field. Our goal is to design and develop a device that automatically recognizes which parts of an image collected by a camera represent the lanes marked on the street. In the literature there exist three general classes of methods used to detect and recognize sub-parts of an image [13]:

1) **Object detection** (Fig. 2 (a)): the device recognizes the coordinates of some points which constitute the lanes [14]. After that, to have an output image, these results must be post-processed to obtain the labeled image, thus increasing latency and power consumption.
2) **Semantic segmentation** (Fig. 2 (b)): the device distinguishes only two classes and finds the class of each pixel coming from the input image by looking at it individually. At the output, we can collect an image in which the pixel intensities define its class [15] [16].
3) **Instance segmentation** (Fig. 2 (c)): it is based on the similar concepts as the semantic segmentation, but various lanes can be grouped into different classes [15] [16].

Since we need a real-time response from the detection device to leave more time for the decision-making part of the AD vehicle and we are only interested in the position of the detected lanes, we choose to use the semantic segmentation approach, which can achieve good performance with reduced latency and power consumption.
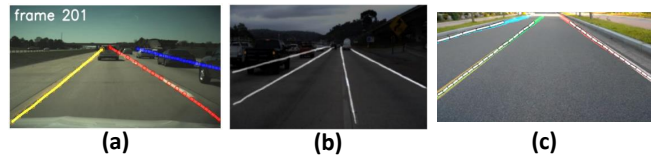

Fig. 2: Example of the (a) Object Detection [14], (b) Semantic Segmentation [17] and (c) Instance Segmentation [18] approaches for the lane detection problem.

B. Dataset Pre-Processing

1) Coding of Input Information into Spikes: The DET dataset is made of labeled grey-scale raw images obtained through the DVS camera. To extract spiking information and directly feed the networks with them, we use the *rate coding* technique. Hence, we compare pixels intensities to random values for converting them into spike trains with Poisson distribution.

2) Reducing the Spatial Resolution: The DET Dataset is recorded by the *CeleX V* DVS camera [11], and it is made by input and label images with high resolution in space (1280×800 pixels per image for both inputs and labels). This property can be very useful during the training of AI models. In fact, it contains sufficient input information to understand how to better generalize the task. On the contrary, the labels with very high resolution induce a considerable imbalance between lane and background classes thus resulting in decreased accuracy when we use a semantic segmentation approach [14].

Moreover, we design SNNs that can be directly implemented on the Intel Loihi Neuromorphic Chip [8]. The used Neuromorphic hardware has some limitations for the collecting of output spike counters related to the output neurons. The maximum number of counters depends on the map of the SNNs and the probes implemented to analyze the performance. Our preliminary analysis indicates that a maximum of 400 spike counters can be implemented. Therefore, we reduce the size of label images to have only 400 pixels. We also limit the resolution of the input image size to 1600 pixels to be more coherent with the resized dimension of the output images.

To prevent the SNNs from a possible *overfitting problem*, before reducing the image size, we perform *data augmentation*. In this particular case, we use 271 random training images and related labels, and perform on them random *vertical translation* (between -100 and 100 pixels) and *rotation* (between $-30°$ and $30°$).

To reduce the size of the dataset images, we use two subsequent steps:

1) *Vertical cropping*: we crop the top 300 pixels rows and the bottom 200 pixels rows for each image, which do not contain relevant information.
2) *Average resizing*: we resize the images from size 1280×300 to 80×20 for the inputs and 40×10 for the labels. This operation is made by the mechanism of area interpolation implemented through the *OpenCV Python library* [19].

For the label images, before the *average resizing* step,

we give the intensity value of 400 to all the lane pixels (*denormalization step*) and then, after performing the resizing operation, each pixel with intensity greater than 0 is labeled as lane, and its value is normalized to 1. This mechanism is necessary because we operate resizing with a large scale, and without the *denormalization/normalization* step we could lose the thinnest lanes.

The steps to reduce the size of the dataset images are summarized in Fig. 3.
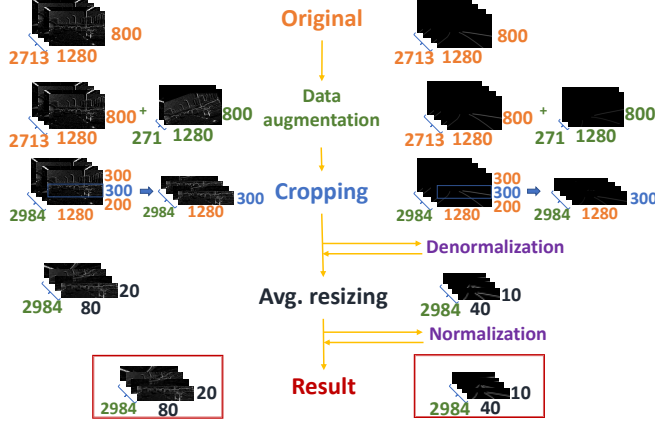

Fig. 3: Steps followed to resize the images of the train set of the DET dataset. On the right, there are the input images. On the left, there are the label images. For the test set, we do not perform *data augmentation*, while all the other steps remain unchanged.

C. Learning Rule and Loss Function

1) Learning Rule: The DET dataset is composed of three labeled parts, namely training, validation, and testing. Therefore, it is convenient to implement a supervised learning rule for the SNNs to achieve higher performance with limited training time, rather than employing an unsupervised learning rule. In particular, we decide to use a *direct supervised learning rule* to reduce the latency of the system. Due to the choice of obtaining the input spike trains through the *rate coding* strategy (Sec. II-B), the spikes are correlated in time and space, since they are based on the same image. Therefore, to achieve high performance, we use the *Spatio-Temporal Back-Propagation (STBP)* [20] learning rule that takes into account both temporal and spatial domains. The core of this learning rule is represented by Eqs. (1) and (2). More details are discussed in [20].

$$\frac{\partial L}{\partial \boldsymbol{b}^n} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{t,\boldsymbol{n}}} \cdot \frac{\partial \boldsymbol{u}^{t,\boldsymbol{n}}}{\partial L \boldsymbol{b}^n} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{t,n}} \quad (1)$$

$$\frac{\partial L}{\partial \boldsymbol{W}^n} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{t,n}} \cdot \frac{\partial \boldsymbol{u}^{t,n}}{\partial x^{t,n}} \cdot \frac{\partial x^{t,n}}{\partial W^n} = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{u}^{t,n}} \cdot \boldsymbol{o}^{t,n-1} \quad (2)$$

They are used to perform the **Gradient Descendent Optimization Algorithm**. With the implementation of the STBP learning rule, the derivative of the spiking nonlinearity is replaced by the derivative of a smooth function, following the **Surrogate Gradient** [21] strategy.

2) Loss Function: In the DET dataset, the lane and background classes are imbalanced also after the dataset pre-processing step, as described in Sec. II-B. Therefore, we employ the *Weighted Binary Cross-Entropy* (WCE) loss function [22] (Eq. (3)) that is a variant of the more common *Binary Cross-Entropy* (BCE) loss function.

$$L_{WCE}(y, \hat{y}) = -(\beta \cdot y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})), \quad (3)$$

where $\hat{y}$ is the predicted probability to have a lane in a determined pixel, and y represents the class value that can be positive ($y = 1$) or negative ($y = 0$).

The WCE function introduces a little improvement for the unbalanced labels, since the positive class (i.e., the presence of the lane) gets weighted by the coefficient β that balances the positive and negative prediction.

However, the STBP learning rule [20] is always studied with the implementation of the *Mean Squared Error* (MSE) loss function (Eq. (4)), which is not widely used for segmentation problems [23].

$$L_{MSE} = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \quad (4)$$

Therefore, for the lane detection task, we develop a novel loss function that can combine the benefits of both MSE and WCE. Such a joint weighted loss function can be formalized by the Eq. (5).

$$L_{MSE\ \&\ WCE} = (1 - p) \cdot L_{MSE} + p \cdot L_{WCE}, \quad (5)$$

where p denotes the parameter to weight the contribution of WCE and $(1 - p)$ denotes the contribution of MSE, such that $0 \geq p \geq 1$.

III. LaneSNNs Design and Strategies to avoid Overfitting

Based on the above discussion, we present the design of our LaneSNN networks in the following along with specific design decisions. To better summarize the most important steps of our design, we present our *design methodology* in Fig. 4.

A. Input and Output

To be coherent with the decisions made on the DET dataset discussed in Sec. II-B, since we generate only one spike-train per pixel from each raw gray-scale image, we develop our networks with only one input channel.

The output size must be consistent with the size of the label images of the modified DET dataset. Therefore, the last layer should have 400 output neurons, one for each pixel. Their firing rates represent the probability for the related pixel to be a lane in the resulting image.

B. Network Architecture

In literature, there are many examples of algorithms based on NNs for facing general semantic segmentation problems. They can be classified into two different classes according to their implementation:

- *End-to-End*: these algorithms use only the NN without any pre and post-processing steps to face the lane detection
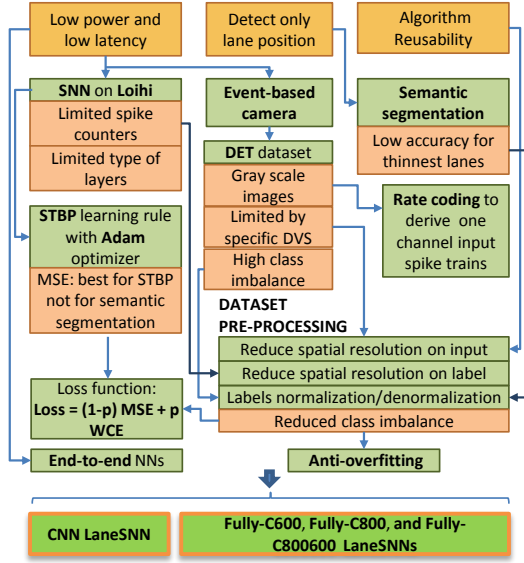
Fig. 4: Design methodology of *LaneSNNs* models. On the top, there are the main three desired properties (orange boxes). In the middle, the different *design steps* and *decisions* (green boxes) are made to follow the properties and overcome the *research challenges* (red boxes). The result is the design of four LaneSNN models at the bottom.

problem. Usually, in these cases the network is divided into two subsequent parts, i.e., reducing (*downsampling*) and increasing (*upsampling*) the image size during the elaboration [24] [25].

- *More than one step*: the NN represents only a part of the entire detection algorithm, and it helps other more complex conventional algorithms [26] [17] when these standalone NNs achieve low performance [27].

To reduce the latency and power consumption of the entire system, we choose to implement *End-to-End* algorithms. For the Loihi implementation, we choose to design the networks through the NxTF library [28]. It can describe only convolutional, fully-connected and average pooling layers.

We develop a spiking CNN inspired from the analysis made by the works in [27] [29]. In the first work [27] a small fully-connected network is introduced at the end of the NN as the *upsampling* part. The second work [29] emphasizes the importance of convolutional layers over others types for the *downsampling* structure.

Therefore, we design our first network, called **CNN LaneSNN** (see Tab. I), with five convolutional layers by which the input sample image size (80×20) is reduced to 20×5 pixels for each of the 16 channels. Then, the image enters into the upsampling part made of 400 output neurons connected to the following convolutional layer. We adopt a *dropout layer*, as discussed later in Sec. III.

TABLE I: Structure of *CNN LaneSNN*.

Layer type	In ch.	Out ch.	Kernel size	Padding	Stride	% Dropout
Convolution	1	4	3	1	1	—
Convolution	4	4	3	1	1	—
Convolution	4	8	3	1	2	—
Convolution	8	8	3	1	1	—
Convolution	8	16	3	1	2	—
Dropout	16	16	—	—	—	10
Dense	1600	400	—	—	—	—

To further decrease the power consumption, we develop simpler structures that use only one or two hidden fully-connected layers. Moreover, low-complexity fully-connected networks are likely to consume low power and are easily implementable onto different Neuromorphic Chips.

Therefore, we develop two fully connected networks with 800 and 600 neurons for the *hidden layer* (**Fully-C800 LaneSNN**, Tab. II, and **Fully-C600 LaneSNN**, Tab. III, respectively), and a structure with two fully connected *hidden layers* called **Fully-C800600 LaneSNN**. (Tab. IV).

TABLE II: Structure of *Fully-C800 LaneSNN*.

Layer	Number of neurons
Input	1600 (image of size 80×20)
Hidden	800
Output	400 (image of size 40×10)

TABLE III: Structure of *Fully-C600 LaneSNN*.

Layer	Number of neurons
Input	1600 (image of size 80×20)
Hidden	600
Output	400 (image of size 40×10)

TABLE IV: Structure of *Fully-C800600 LaneSNN*.

Layer	Number of neurons
Input	1600 (image of size 80×20)
Hidden	800
Hidden	600
Output	400 (image of size 40×10)

C. Anti-Overfitting Strategies

Due to the label imbalance of the DET dataset [9] discussed in Sec. II-B, the networks hardly generalize the task, since they are affected by overfitting problems. This problem is mostly noticed in the layers of the networks that present many connections. For this reason, in the *CNN LaneSNN* we insert a dropout layer between the last convolutional layer and the fully-connected layer. Since the percentage of dropout is not so high, it does not hamper the training of the structure. Its value of 10% is chosen after some preliminary experiments.

On the other developed networks we do not apply dropout strategies, because, due to the small complexity of the networks, this kind of operation can drastically reduce the achieved results.

We use the *Gaussian noise insertion* technique before all the layers of all the four networks. We define its entity with the *relative standard deviation* σ_r. All the inserted Gaussian noise have σ_r equal to 0.1 for each layer of each developed network.

Finally, we apply the *decoupled weight decay regularization* on every layer of every network (Eq. (6) [30]).

$$w_{t+1} = (1 - \lambda) \cdot w_t - lr \cdot \nabla f_t (w_t), \qquad (6)$$

where:
- w_{t+1} and w_t are respectively the new and the old synaptic weights on which we apply the optimizer;
- λ defines the rate of the weight decay per step;
- $\nabla f_t (w_t)$ is the tth batch gradient;
- lr is the learning rate.

Tab. V summarizes all the implemented strategies.

TABLE V: Implemented anti-overfitting strategies.

Anti-overfitting strategy	Networks	Where/when	Entity
Data augmentation	All	Train dataset	+271 images
Dropout	CNN	Before output layer	10%
Gaussian noise	All	Input of all layers	$\sigma_r = 0.1$
Weight decay	All	Optimization step	different values of λ

IV. EVALUATION OF LANESNNS

As discussed in Sec. II, we perform the training of the network with the STBP learning rule. It uses Eqs. (1) and (2) [20] to evaluate the gradients. These computations are too complex to be executed onto the on-chip learning engine of the Intel Loihi Neuromorphic Chip. Therefore, our *LaneSNNs* are trained offline and then we implement the networks achieving best results onto the neuromorphic hardware (Sec. IV-C).

A. Accuracy Definition

As discussed in Sec. III-A, the output of our LaneSNNs represents the probability for each pixel to be a lane.

Compared to having a direct prediction of the class value, the probability prediction is a more flexible method, which allows to tune and even calibrate the threshold for how to interpret the predicted probabilities.

To derive the best threshold value for the predicted probabilities, we study the graphs that correlate the *Precision* and *Recall* values (**PR curves** [31]) evaluated by Eq. (7).

$$\text{Precision} = \frac{TP}{TP+FP}, \quad \text{Recall} = \frac{TP}{TP+FN} \quad (7)$$

- *Precision* is the number of lane pixel predictions matched with the label (*True Positive* or *TP*), divided by the number of pixels predicted as lane (*True Positive* and *False Positive* or *FP*).
- *Recall* is the number of lane pixel predictions matched with the label (*TP*), divided by the number of lane pixels in the label (*TP* and *False Negative* or *FN*).

Then we define the **F-measure** (Eq. (8)) to find the best threshold to balance the two parameters.

$$\text{F-measure} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (8)$$

To distinguish between lanes and background classes, this parameter is computed for all the possible thresholds applied to the output probabilities, and the maximum F-measure, which corresponds to the best threshold, is selected. Moreover, to objectively compare the performance of our networks, we use the **Intersection over Union** (**IoU**) value (Eq. (9) [32]).

$$IoU = \frac{|\text{Predicted lanes} \cap \text{True lanes}|}{|\text{Predicted lanes} \cup \text{True lanes}|} \quad (9)$$

We calculate the best thresholds for every N predicted images and we define the overall best threshold as the numerical mean of them (Eq. (10)).

$$\overline{best\ th} = \frac{\sum\limits_{i=1}^{N} best\ th_i}{N} \quad (10)$$

Afterwards, we apply the $\overline{best\ th}$ on the results and compute the IoU value distinctly for each image (Eq. (11)).

$$IoU = \frac{\sum\limits_{i=1}^{N} IoU_i(\overline{best\ th})}{N} \quad (11)$$

B. LaneSNNs Experimental Setup

Our *LaneSNNs*, described using the PyTorch library [33], are trained on the DET dataset [9], after performing the pre-processing operations discussed in Sec. II-B. We run the experiments on a workstation having CentOS Linux release 7.9.2009 as the operating system and equipped with an Intel Core i9-9900X CPU and multiple Nvidia RTX 2080-Ti GPUs. An overview of the tool flow for conducting the experiments is shown in Fig. 5.
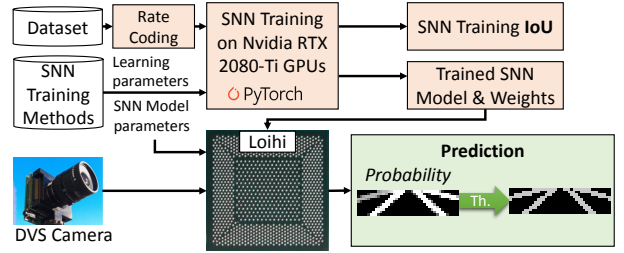

Fig. 5: Setup and tool-flow for conducting our experiments.

As discussed in Sec. II-C, we use the STBP [20] learning rule and the loss function summarized by Eq. (5) for computing the distance between prediction and labels during training. With these modifications, the complete training set has 241837 and 951763 pixels representing the lane and the background classes, respectively. This corresponds to a negative (background) over positive (lanes) ratio equal to 3.93. Therefore, to contrast the imbalance, we set the coefficient β of Eq. (5) to 4.0 for every experiments. On the other hand, the value p, presented in the same Eq. (5) and used to set the percentage of loss derived by the WCE over the MSE loss functions, varies from 0.0 to 0.5 for each experiment. This is because, after some experiments, we notice that the insertion of a contribution of the MSE loss function makes the SNNs converge faster. Focusing on other specific learning rule hyper-parameters we set:

- **Optimizer**: we use *Adam* [34], since it is efficient when coupled with the STBP. On that, we apply the *decoupled weight decay* strategy [30] as discussed in Sec. III-C. We vary λ of Eq. (6) from 0.0 to $5e^{-4}$ with steps of $1e^{-4}$.
- **Learning rate** (**lr**): we use the fixed learning rate approach varying in the range from $1e^{-5}$ to 1^e-3. These values are found after preliminary analyses and guarantee the convergence of the method in a few epochs.

The adopted learning rule is directly based on the SNNs with LIF neuron models. The formalization of the membrane potential update ($u_i^t + 1, n$) is defined in Eq. (12), where:

- $u_i^{t,n}$ is the membrane potential before the update;
- $o_i^{t,n}$ represents the presence (1) or the absence (0) of a spike generated on the output axon;

- $\sum_{j=1}^{l(n-1)} w_{ij}^n o_j^{t+1,n-1}$ represents the incoming synaptic weighted spikes;
- b_i^n is a bias term.

$$u_i^{t+1,n} = u_i^{t,n}\tau(1 - o_i^{t,n}) + \sum_{j=1}^{l(n-1)} w_{ij}^n o_j^{t+1,n-1} + b_i^n \quad (12)$$

The main tunable parameters of a LIF neuron are:

- **membrane threshold** (V_{th}): it is the same for all neurons, and its value changes from 0.2 to 1.0;
- **membrane reset potential** (V_{reset}): for all the experiments, it is the same for each neuron and it is always set to $0\ V$;
- **membrane time constant** (τ): for all the experiments, it is set to $0.2\ ms$.

Moreover, the STBP learning rule uses the *surrogate gradient* approach to approximate the derivative of the spiking nonlinearity with simple functions. For this purpose, we adopt the rectangular pulse function (Eq. (13)).

$$h_1(u) = \frac{1}{a_1}\,\text{sign}\left(|u - V_{th}| < \frac{a_1}{2}\right) \quad (13)$$

This assumption is coherent with the work in [20], since different types of approximations do not involve a great variation of the accuracy, and the rectangular pulse function represents an efficient formula developed for this purpose.

Therefore, according to Eq. (13), we can adjust the parameter $\frac{a_1}{2}$ representing the pulse width. It is set to the same value of the V_{th}, as made in [20].

All the experiments run for 200 epochs with batch size equal to 4. The batch size value is set based on a preliminary analysis and it represents a trade-off between the achieved accuracy and the training time.

Based on the discussions of Sec. III-A, for each pixel of a gray-scale input image, we create a single spike-train. Moreover, since every single spike train is made of 30 time steps, it can contain up to 30 spikes. The spike trains per image are not calculated offline before training but are generated at run-time. Hence, they are different for each training epoch, to increase the robustness of the training process. Since this information is given as input without applying any accumulation strategy, the *LaneSNNs* analyze each input image for 30 time steps.

C. LaneSNNs Implemented on Loihi

To implement our trained LaneSNNs onto the Intel Loihi Neuromorphic Chip, we have to set its model parameters such as *Compartment Voltage threshold* ($V_{th\ mant}$), *Compartment Current Decay* (δ_i), *Compartment Voltage Decay* (δ_v), *Compartment Bias* ($bias$), *Synaptic Weights* ($weight$) and *Weight Exponent* ($wgtExp$). This is made according to the neuron models similarities exploited in [3].

We multiply weights and V_{th} by a factor (k) calculated from the weight magnitudes of all the SNN synapses as in Eq. (14):

$$k = \frac{2^4 - 1}{\max_{all\ synapses}(|\ weight_i\ |)} \quad (14)$$

With the multiplication of the weights by k we use all the dynamic range for the maximum value, thus minimizing $wgtExp$. All the setup parameters are summarized in Tab. VI.

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

TABLE VI: Translation of parameters to the Loihi Chip for the *LaneSNNs* (For the Loihi the $weight$ bits also include the $wgtExp$ bits).

We implement all the trained LaneSNNs developed in the previous Sec. IV-B onto the Loihi Neuromorphic Chip, considering all the possible values of V_{th}, λ and lr, characterized by the best parameter p. The implementation is conducted using the Intel Nx SDK API version 1.0.0 running onto the Nahuku32 partition. This code is developed using the *NxTF* Layers and in particular *NxConv2D* and *NxDense* utilities [28]. The *LaneSNNs* are tested on the testing set of the DET dataset [9]. For feeding the input images, we use the same method applied for the offline training discussed in Sec. IV-B. Hence, we create a spike train for each pixel of each input image on-the-fly. Each spike train lasts for 30 time steps. We insert a blank time of 10 time steps between two consecutive samples.

To perform the computation of the IoU measure, we find the best threshold directly from the reconstructed images at the output of the Loihi chip. Then, we perform the steps discussed in Sec. IV-A.

D. Pareto Optimal Solutions

To efficiently evaluate the *LaneSNNs* implemented onto the Intel Loihi, we can analyze multiple Pareto-optimal models. Therefore, we use the **Pareto Optimal frontier curve** to find the best trade-off solutions between the achieved **IoU** and:

1) **latency**, i.e., the mean time duration required for the classification of all the pixels of a single image (Fig. 6.1);
2) **power consumption** of the entire Intel Loihi chip (Fig. 6.2);
3) **network complexity**, number of neurocores occupied (Fig. 6.3).

From the first graph (Fig. 6.1) we can see that within the Pareto-optimal curve, the maximum time to detect the lanes on a stream of 30 time steps is limited to **7.27 ms** and it is achieved by the *Fully-C800* network (label ①). It can be reduced to $6.02\ ms$ with a reduction on IoU of about 6% (label ③). In the second graph (Fig. 6.2), we can notice that the lowest Pareto-optimal power consumption is achieved by the simplest *Fully-C600* network (labels ①-⑤), but the highest IoU is reached by the *Fully-C800* network (label ⑥). Overall, the power consumption of Intel Loihi chip does not vary much around $1W$.

Fig. 6.3 shows that, among the Pareto-optimal solutions, the *Fully-C600* network is the simplest due to the lower
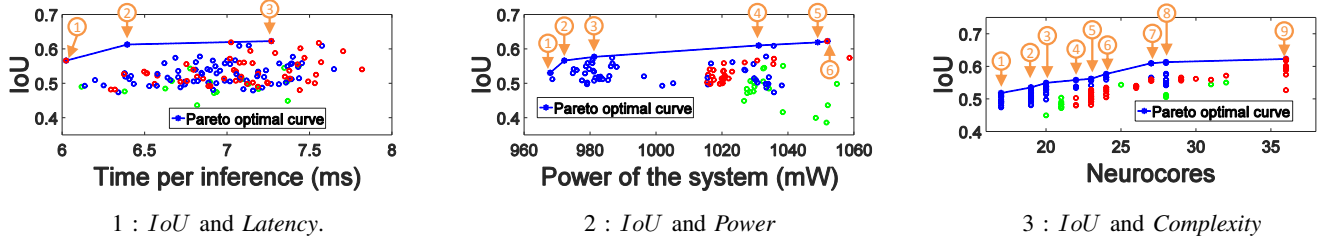
1 : *IoU* and *Latency*.　　2 : *IoU* and *Power*　　3 : *IoU* and *Complexity*

Fig. 6: Pareto-optimal solutions for Fully-C800, Fully-C600, Fully-C800600 *LaneSNNs*.

number of occupied neurocores (labels ①-⑤). Moreover, the best *IoU* is achieved by the *Fully-C800* network, but its complexity is significantly greater than the minimum value.

E. Best Results for Each LaneSNN

The best results in terms of *IoU* for each type of *LaneSNN* for both offline and online implementations are summarized in Tab. VII.

TABLE VII: Best *IoU* measures achieved by the different *LaneSNNs* for offline (*GPU*) and online (*Loihi*) implementations.

IoU CNN		IoU Fully-C600		IoU Fully-C800		IoU Fully-C800600	
GPU	*Loihi*	*GPU*	*Loihi*	*GPU*	*Loihi*	*GPU*	*Loihi*
0.598	0.208	0.637	0.527	0.633	0.542	**0.652**	0.416
0.551	0.349	0.632	**0.623**	0.629	0.613	0.590	0.550

The *CNN LaneSNN* has the lowest *IoU* values for both online and offline implementations. Its offline implementation achieves an acceptable value of *IoU*, but this result dramatically decreases for the online implementation, due to the weights approximation errors propagating layer by layer during the translation from offline to online.

The highest offline result is achieved by the *Fully-C800600* network, which has two fully-connected hidden layers. However, it achieves lower *IoU* for the online implementation than the same achieved by the simpler fully-connected networks *Fully-C600* and *Fully-C800*. The best online result is achieved by the *Fully-C600* network, which is the simplest SNN. Moreover, its online and offline *IoU* measures, even if they are slightly greater, are comparable to the results obtained with the *Fully-C800* network.

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

In our work, as discussed in Sec. II, we prefer less complex SNNs. This means that they can be effectively implemented onto the Intel Loihi Neuromorphic Chip and achieve competitive results for real-time embedded systems, with low power consumption and low latency. This is also favored by the use of event-based cameras as vision sensors of the AD system. On the other hand, in literature, there are many algorithms that involve non-spiking NNs to face the problem of lane detection.

For a fair comparison, we consider the results achieved by state-of-the-art networks on the same dataset (DET dataset [9]). Therefore all these NNs also use an event-based camera as the vision sensor for the AD system.

The results of the state-of-the-art methods, which are FCN [29], DeepLabv3 [35], RefineNet [36], LaneNet [37],

TABLE VIII: Comparison of *IoU* achieved by different algorithms [25] for the lane detection problem faced by the semantic segmentation approach.

Classifier	$IoU_{offline}$	IoU_{online}	Number of parameters
FCN	0.585	–	132.27 M
DeepLabv3	0.585	–	39.05 M
RefineNet	0.614	–	99.02 M
LaneNet	0.647	–	0.53 M
SCNN	0.673	–	25.16 M
LDNet	0.767	–	5.71 M
CNN LaneSNN (**ours**)	0.598	0.349	1.39 M
Fully-C600 LaneSNN (**ours**)	0.637	0.623	1.20 M
Fully-C800 LaneSNN (**ours**)	0.633	0.613	1.60 M
Fully-C800600 LaneSNN (**ours**)	0.652	0.550	2.00 M

SCNN [15] and LDNet [25], are compared to our *LaneSNNs* in Tab. VIII.

We can notice that our *CNN LaneSNN* achieves higher performance than the FCN and DeepLabv3 algorithms despite they use more complex networks to make their predictions. The FCN uses AlexNet [38] made of five convolutional layers, three pooling layers, and three fully-connected layers. DeepLabv3 [39] is made by Atrous Spatial Pyramid Pooling (ASPP) layers. It probes an incoming convolutional feature layer with filters at multiple sampling rates. This method is not yet developed onto the Loihi with only NxTF facilities [28] that do not implement the ASPP layers. Moreover, it has lower performance than RefineNet (based on the use of long residual connections), LaneNet (using upsampling layers and conventional algorithms at the end of the process), SCNN (based on slice-by-slice convolutions), and LDNet (using ASPP, many convolution stacks, and upsampling layers). These structures cannot be developed onto the Loihi with the NxTF facilities [28] that implement pooling, fully connected, and traditional convolution layers.

All the other fully-connected *LaneSNNs* have *IoU* comparable with LaneNet and overcome the performance of more complex algorithms. Moreover, the *Fully-C800600 LaneSNN* achieves a similar result as the one obtained by the SCNN algorithm, while using less than $10\times$ number of parameters. On the other hand, the highest *IoU* value has been measured by the LDNet, at the price of very high complexity and furthermore, it cannot be implemented onto the neuromorphic hardware.

The previous considerations do not take into account that all *LaneSNNs* are tested on the modified DET dataset (discussed in Sec. II-B) and not on the original, as it is for all the other presented algorithms. However, the DET dataset pre-processing step allows all the *LaneSNNs* to be directly implementable on the Intel Loihi Neuromorphic Chip achieving competitive performance also online.

V. CONCLUSION

In this paper, we presented *LaneSNNs*, a novel class of simple SNN models based on the *semantic segmentation* approach, to find the position of the lanes on the driving road, thanks to event streams coming from an event-based camera. To the best of our knowledge, they represent the first SNNs implemented onto the Intel Loihi Neuromorphic Chip able to face the lane detection problem. Since they are implemented as an *end-to-end* system, they do not use a post-processing stage of grouping and clustering. For training, we use a *direct supervised* learning rule and we develop a *novel loss function* that is the linear composition of WCE and MSE.

We design four structures with different complexity degrees and we call them *CNN*, *Fully-C600*, *Fully-C800* and *Fully-C800600*. The first is made of five convolution layers and a final fully-connected layer. The second and third are made by a fully-connected hidden layer made of 600 and 800 neurons respectively. The fourth is made of two fully connected hidden layers.

We train the *LaneSNNs* with different parameters and then we implement the resulting SNNs onto the Intel Loihi Neuromorphic Chip. The best offline result for the offline implementation is achieved by the *Fully-C800600 LaneSNN* with IoU equal to 0.652, while the best online result is achieved by the *Fully-C600* network with IoU equal to 0.623. These values are comparable with the same achieved by other state-of-art algorithms such as LaneNet [37] and RefineNet [36]. Thanks to its implementation onto the Loihi Neuromorphic Research Chip, its maximum latency is less than $8\ ms$ and its power consumption is about $1\ W$ during the classification of a single image, thereby making it superior to all the state-of-the-art techniques in terms of performance and power efficiency.

LaneSNNs enable high-performance yet energy-efficient lane detection for AD systems while leveraging the advanced neuromorphic processors.

ACKNOWLEDGMENTS

REFERENCES

[1] S. Šurdonja, T. Giuffrè, and D. Aleksandra, "Smart mobility solutions – necessary precondition for a well-functioning smart city," *Transportation Research Procedia*, 2020.

[2] A. Marchisio *et al.*, "R-SNN: an analysis and design methodology for robustifying spiking neural networks against adversarial attacks through noise filters for dynamic vision sensors," in *IROS*, 2021.

[3] A. Viale *et al.*, "Carsnn: An efficient spiking neural network for event-based autonomous cars on the loihi neuromorphic research processor," in *IJCNN*, 2021.

[4] 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.

[5] R. V. W. Putra and M. Shafique, "tinySNN: Towards memory- and energy-efficient spiking neural networks," *arXiv:2206.08656*, 2022.

[6] C. D. Schuman *et al.*, "A survey of neuromorphic computing and neural networks in hardware," *ArXiv:1705.06963*, 2017.

[7] 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.

[8] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, 2018.

[9] W. Cheng *et al.*, "Det: A high-resolution dvs dataset for lane extraction," in *CVPR Workshops*, 2019.

[10] G. Gallego *et al.*, "Event-based vision: A survey," *IEEE TPAMI*, 2020.

[11] S. Chen and M. Guo, "Live demonstration: Celex-v: A 1m pixel multi-mode event-based sensor," in *CVPR Workshops*, 2019.

[12] R. Kanbur, "Pareto's revenge," *JSED*, 2005.

[13] J. Tang, S. Li, and P. Liu, "A review of lane detection methods based on deep learning," *Pattern Recognition*, 2021.

[14] S. Chougule *et al.*, "Reliable multilane detection and classification by utilizing cnn as a regression network," in *ECCV Workshops (5)*, 2018.

[15] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, "Spatial as deep: Spatial cnn for traffic scene understanding," *arXiv:1712.06080*, 2017.

[16] S. Lee *et al.*, "Vpgnet: Vanishing point guided network for lane and road marking detection and recognition," *arXiv:1710.06288*, 2017.

[17] S. Lu *et al.*, "A fast and robust lane detection method based on semantic segmentation and optical flow estimation," *Sensors*, 2021.

[18] M. Oh *et al.*, "An urban autodriving algorithm based on a sensor-weighted integration field with deep learning," *Electronics*, 2020.

[19] Itseez, *The OpenCV Reference Manual*, 2.4.9.0 ed., April 2014.

[20] W. Yujie *et al.*, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Front. Neuroscience*, 2018.

[21] 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 SP*, 2019.

[22] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," *IEEE Access*, 2020.

[23] S. Jadon, "A survey of loss functions for semantic segmentation," *CIBCB*, 2020.

[24] P. Ondruska *et al.*, "End-to-end tracking and semantic segmentation using recurrent neural networks," *arXiv:1604.05091*, 2016.

[25] F. Munir, S. Azam, and M. Jeon, "Ldnet: End-to-end lane detection approach using a dynamic vision sensor," *arXiv:2009.08020*, 2020.

[26] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, 1981.

[27] J. Kim and M. Lee, "Robust lane detection based on convolutional neural network and random sample consensus," in *ICONIP*, 2014.

[28] B. Rueckauer *et al.*, "Nxtf: An api and compiler for deep spiking neural networks on intel loihi," *arXiv:2101.04261*, 2021.

[29] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *arXiv:1411.4038*, 2015.

[30] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv:1711.05101*, 2019.

[31] J. Cook and V. Ramadas, "When to consult precision-recall curves," *The Stata Journal*, 2020.

[32] R. Real and J. Vargas, "The probabilistic basis of jaccard's index of similarity," *Systematic Biology - SYST BIOL*, 1996.

[33] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2017.

[35] L.-C. Chen *et al.*, "Rethinking atrous convolution for semantic image segmentation," *arXiv:1706.05587*, 2017.

[36] G. Lin *et al.*, "Refinenet: Multi-path refinement networks for high-resolution semantic segmentation," in *CVPR*, 2017.

[37] D. Neven *et al.*, "Towards end-to-end lane detection: an instance segmentation approach," *arXiv:1802.05591*, 2018.

[38] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *NeurIPS*, 2012.

[39] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv:1606.00915*, 2017.