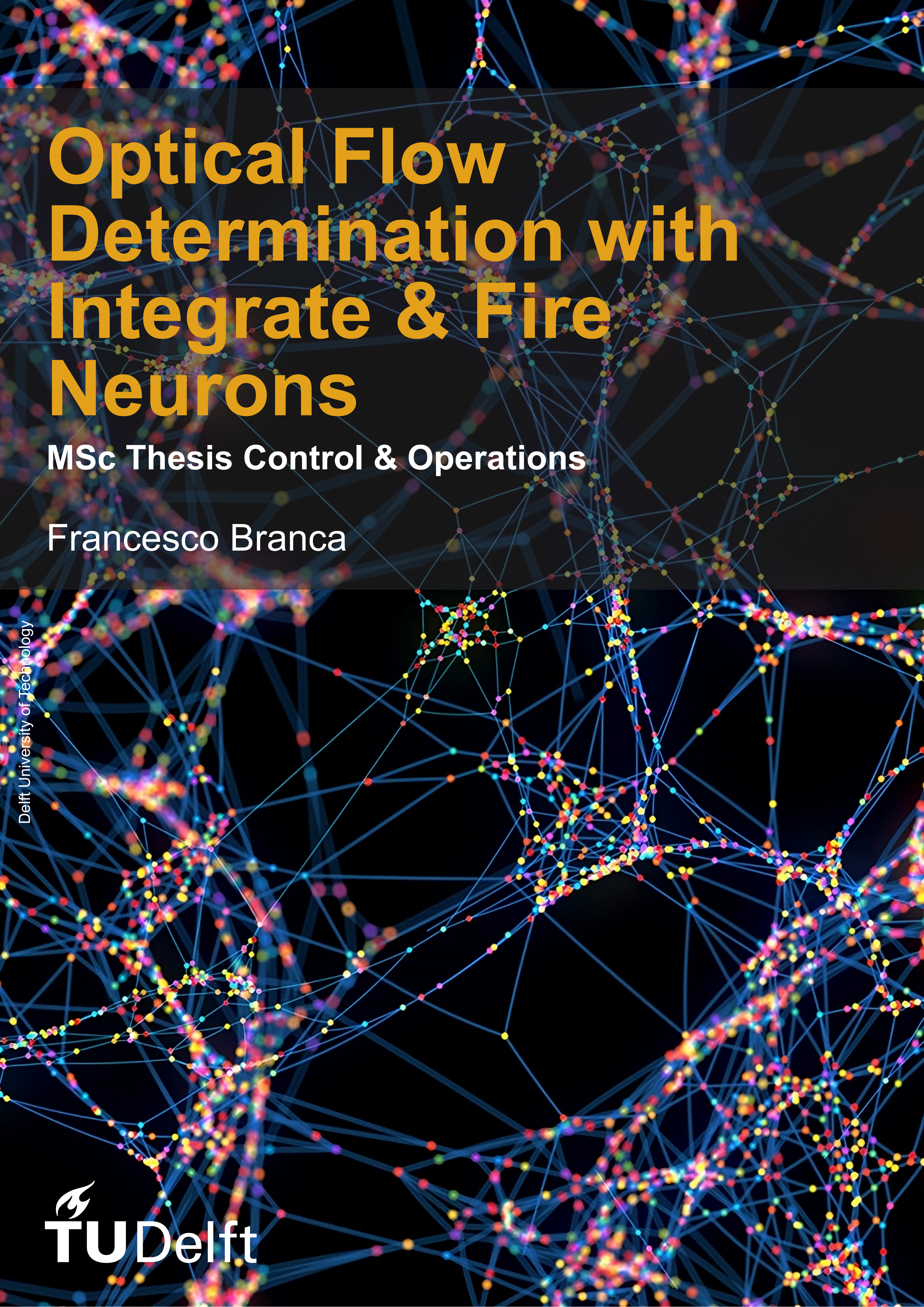


Optical Flow Determination with Integrate & Fire Neurons

MSc Thesis Control & Operations

Francesco Branca

Optical Flow Determination with Integrate & Fire Neurons

MSc Thesis Control & Operations

by

Francesco Branca

Student Name	Student Number
Francesco Branca	4884981

Supervisors:	Dr G. C. H. E. de Croon
	J. J. Hagenaars
Project Duration:	May, 2023 - May, 2024
Faculty:	Faculty of Aerospace Engineering, Delft

Preface

I would like to thank my supervisors for guiding me in this project. Thank you Jesse for being always available to clear all my doubts. Thank you Guido for welcoming me with enthusiasm and for being so inspiring. I am really glad to have worked under your supervision.

Thank you to my family for giving me the opportunity to study here at TU Delft. You motivated me to do better every day and to stay on the right path. Regardless of what the future holds, I know that the strong foundation you've provided will stay with me.

A special thanks to the friends that I made along the way in these six long years in Delft. Thank you to Andrea, Roman, Martino, Aytek, Kevin, Alessandro, Elena, Giacomo, Antonio and Alejandro for accompanying me in my music journey, for the late night jam sessions and for sharing your passion with me. Finally thanks to everyone that I shared unforgettable memories with. Thank you Vincenzo, Marco, Giorgio, Alex, Manos, Maurizio and Jenny.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
ANN	Artificial Neural Network
SNN	Spiking Neural Network
RNN	Recurrent Neural Network
LIF	Leak Integrate & Fire
IF	Integrate & Fire
EBC	Event Based Cameras
SG	Surrogate Gradient
MAV	Micro Air Vehicle
DVS	Dynamic Vision Sensor

List of Tables

List of Figures

Introduction

The field of robotics engineering often draws inspiration in nature, in order to design more efficient and agile systems. Pushing the boundaries of Micro Air Vehicles (MAVs)

The topic is biology-inspired applications for Micro Air Vehicles (MAV), in particular optical flow estimation with Event-Based Cameras (EBCs) and Spiking Neural Networks (SNNs). This research field draws inspiration from flying insects to mimic their navigation techniques and perform more efficient autonomous flight of MAVs. Event-based visual processing has the potential to achieve lower power consumption and reduced latency for MAVs navigation.

The goal of this thesis is to develop an optical flow estimation method using SNNs and implement it on the neuromorphic chip Speck2e from Synsense [1]. The project aims to assess capabilities and limitations of the chip and prove or disprove that the flow ca be learned from device-compatible networks. The official research question is:

Can Spiking Neural Networks be implemented on board of a small neuromorphic device in order to estimate ego motion of a drone?

The difficulty of the project is constructing a network architecture that is both hardware-compatible and able to learn optical flow. Since commonly used Leak-Integrate and Fire (LIF) neuron models are not available on the Speck2e, Integrate-And-Fire models in combination with recurrent connections have to be used instead. This poses the main challenge of the project. The question can be extended with the following sub-questions:

- Can SNNs without Leak-Integrate and Fire Neurons be used to learn optical flow?
- What are the advantages and disadvantages of using the Speck2e to estimate optical flow?

The report is structured in three parts:

1. Part I contains the standalone scientific paper titled *"Optical Flow Determination using Neuromorphic Hardware with Integrate & Fire Neurons"*. This article presents the main contributions of this thesis.
2. Part II contains an in-depth literature study on the relevant topics for the thesis project. The literature study will start with Chapter 1, which introduces conventional frame-based optical flow techniques. Chapter 2 treats EBCs and common applications of this technology. In Chapter 3, the functioning principles, applications, and training challenges of spiking neural networks are explored. In Chapter 4, recent methods of event-based optical flow techniques are showcased. Finally, Chapter 5 will show examples of previous similar on-board applications and expose the neuromorphic device that will be used for the project.
3. Part III includes the preliminary evaluation of Integrate & Fire neurons for optical flow determination. This part starts with Chapter 6 which includes an introduction to the methodology of the project and the tools used. Following, Chapter 7 documents the network design process and the different training strategies implemented. In conclusion, Chapter 8 includes the preliminary hardware implementation.

Part I

Scientific Paper

Optical Flow Determination using Neuromorphic Hardware with Integrate & Fire Neurons

Francesco Branca*
Micro Air Vehicle Laboratory
Delft University of Technology
Delft, Netherlands
F.Branca@student.tudelft.nl

Jesse Hagenaars†
Micro Air Vehicle Laboratory
Delft University of Technology
Delft, Netherlands
J.J.Hagenaars@tudelft.nl

Guido de Croon†
Micro Air Vehicle Laboratory
Delft University of Technology
Delft, Netherlands
G.C.H.E.deCroon@tudelft.nl

* MSc Student, † Supervisor

Abstract—**Spiking neural networks implemented for sensing and control of robots have the potential to achieve lower latency and power consumption by processing information sparsely and asynchronously. They have been used on neuromorphic devices to estimate optical flow for micro air vehicles navigation, however robotic implementations have been limited to hardware setups with sensing and processing as separate systems. This article investigates a new approach for training a spiking neural network for optical flow to be deployed on the speck2e device from Synsense. The method takes into account the restrictions of the speck2e in terms of network architecture, neuron model, and number of synaptic operations and it involves training a recurrent neural network with ReLU activation functions, which is subsequently converted into a spiking network. A system of weight rescaling is applied after conversion, to ensure optimal information flow between the layers. Our study shows that it is possible to estimate optical flow with Integrate-and-Fire neurons. However, currently, the optical flow estimation performance is still hampered by the number of synaptic operations. As a result, the network presented in this work is able to estimate optical flow in a range of [-4, 1] pixel/s.**

Index Terms—**neuromorphic computing, spiking neural networks, integrate-and-fire, computer vision, optical flow, micro air vehicles**

I. Introduction

Neuromorphic computing has emerged as a promising paradigm for biologically inspired robotics applications [1, 2, 3, 4]. In contrast to von-Neumann architectures, the information processing in neuromorphic devices occurs asynchronously and sparsely. The networks hosted on these type of processors are called spiking neural networks (SNNs) [5], also known as the third generation of neural networks. SNNs contain layers of neurons communicating with each other through binary inputs called "spikes" [6]. Similarly to the structure and functionality of the human brain, SNNs are able to perform parallel computations. When implemented on neuromorphic hardware, SNNs result in lower power consumption and latency, which is preferable for efficient real-time systems.

Neuromorphic devices have been used for a variety of applications in computer vision, ranging from object detection [7], tracking [8, 9] and gesture recognition [10, 11]. Regarding time-dependent tasks, one application is determining optical flow for Micro Air Vehicles (MAVs) state estimation [12]. Optical flow is a concept in computer vision that quantifies the apparent motion of points in an image [13]. It was discovered that optical flow serves as an important visual cue for animals. For instance, bees exploit optical flow to land by keeping the image velocity constant [14]. Other studies also hypothesized that it is also used to estimate distances and avoid obstacles [15, 16].

Determining optical flow using SNNs can be done by combining neuromorphic devices with event-based cameras (EBCs) [17]. This type of vision sensors are composed by an array of pixels, where each pixel detects changes in brightness in the image and outputs a binary variable indicating a positive or negative change. Since the static elements in the image are not detected, event-based vision sensors avoid capturing redundant data and result in extremely low power consumption. EBCs consume on average 10 mW of power, while standard cameras power consumption is in the scale of W [17]. This makes them suitable for real-time sensing in robotics systems.

With faster and more energy efficient computations, MAVs can be designed to be lighter, more agile and more similar to real insects. This technology could be helpful in applications such as search and rescue and flying in greenhouses to monitor crop. Paredes-Vallés et al. [18] proposed the first fully neuromorphic vision-to-control pipeline for controlling a MAV. The system runs completely autonomously with the help of Intel's Loihi [19] processor and a DAVIS240C event-based camera.

This article presents a novel approach to estimating optical flow, suited for implementation on lighter and more simple neuromorphic devices. The chip used for experiments is Synsense's speck2e [20, 21, 22], a dynamic vision system-on-chip, integrating sensing and computing in one board. The speck2e is equipped with Integrate-and-Fire neurons. To achieve a performance comparable to the Intel's Loihi, a new method has to be developed to transmit information inside the network, without relying on the leakage system. The prime goal of this research is to design a speck2e-compatible network that is able to estimate direction and magnitude of dense optical flow from an event-based dataset. A Recurrent Neural Network (RNN) with ReLU activation functions is trained in self-supervised learning and converted to spiking for testing. The converted network is then deployed on the speck2e to assess the hardware performance.

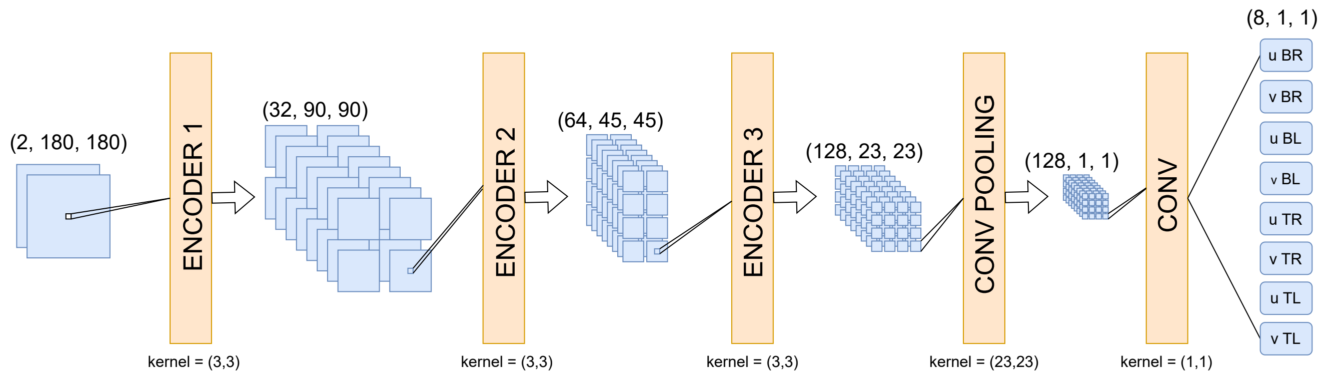


Fig. 1. LIF network diagram from Paredes-Vallés et al. [18].

II. Related Work

A. Neuromorphic Robotics

Neuromorphic devices have been previously used for real-time control of robotics systems. For instance, neuromorphic systems have been sued for control of particular robotic joints to achieve a certain motion [23, 24, 25]. In the field of computer vision, there has been extensive research into developing systems with autonomous navigation using obstacle avoidance and edge tracking [26, 27, 28, 29].

B. Neuromorphic Control of Micro Air Vehicles

Regarding autonomous navigation of MAVs, previous works have implemented neuromorphic computing for sensing and controlling. Hagenaars et al. [30] used SNNs for the first time to control a drone during landing, using optical flow divergence. Following, Paredes-Vallés et al. [18] introduced the first fully neuromorphic pipeline for drone control. A SNN was used to process events from a camera and output low-level control actions to perform autonomous vision-based flight. Both of these works used neuromorphic hardware with fully programmable LIF neuron models. The drone setup included a DAVIS240C event-based camera, a Loihi neuromorphic chip [19] and a a single-board computer UP Squared. The latter component was used to pre-process the events (downsampling and cropping), as well as process the output spikes from the Loihi. A further improvement would be implementing a new device that combines vision and processing in the same board, without requiring an additional component in between.

The architecture proposed in Paredes-Vallés et al. [18] (Figure 1), is denoted here as LIF-3 and it includes three groups of layers (encoders) with recurrent connections and LIF spiking activation functions, one pooling layer and one prediction layer. In total, 4 networks were used, one for each corner of the image and each corner was cropped to a smaller region of interest, in order to reduce the number of input events. Note that defining 4 different regions of interest is not possible with the DVS on the speck2e. To fairly compare the speck2e architectures to LIF-3, the same number of channels per layer is used, however the network is applied to the full 180x180 picture instead of 4 different corners. Note that the output of the LIF-3 is still 8 flow vector components, as shown in Figure 1. In this article, this architecture is used as a starting point for the development of a new network configuration.

C. Previous Speck2e Implementations

Other examples of computer vision tasks tested on the speck2e are face detection from Caccavella et al. [7], binary particle classification [31], CIFAR10, ImageNet and NMNIST classification [32, 33]. This neuromorphic device has not been used yet for time-dependent tasks, requiring recurrent connections for integrating in time.

III. Methodology

A. Event Based Cameras

Event-based cameras react to changes in brightness in the scene by providing as output a stream of independent and sparse inputs. Each event is defined as $\mathbf{e_i} = (x_i, y_i, t_i, p_i)$, where x_i, y_i are the coordinates of the pixel, t_i is the timestamp in microseconds and $p_i \in [-1, 1]$ is the polarity of the event (-1 for decrease and $+1$ for increase in brightness). The asynchronous nature of EBCs is suitable for on board processing on a neuromorphic device. For instance, the speck2e has a processing frequency of 1 MHz, hence a resolution in time of microseconds order. However, when training the SNNs in simulation, the processing frequency has to be decreased in order to speed up the training time, since processing each event individually would take too long. For this reason, the events are binned into event frames, which can be more easily processed by deep learning frameworks [17]. In the following experiments, when training the network, the events are accumulated in time windows of 5 ms.

B. Optical Flow Model

The network architecture proposed estimates dense (per pixel) optical flow in horizontal and vertical direction (u, v) in pixels per millisecond. The network takes the input from the whole DVS array of pixels and predicts the flow for 4 sections of the image, hence it outputs a total of 8 predictions. The corners are denoted as top left (TL), top right (TR), bottom left (BL) and bottom right (BR) (Figure 1). Note that the flow is two-dimensional, hence it is assumed that the EBCs used to record the camera of the dataset is looking at a planar surface.

C. Hardware Constraints

The speck2e circuit is able to support large-scale SNN for various computer vision tasks, such as sign recognition, smart tracking and obstacle detection. The final network configuration shall be able to estimate optical flow from the datasets accurately and the architecture shall be designed to fit the hardware constraints. The speck2e specifications are [34]:

1) Maximum of 9 convolutional layers
2) Maximum number of neurons: 32k
3) Maximum input dimension: 128x128
4) Maximum feature output size: 64x64
5) Maximum feature number: 1024
6) Weight resolution: 8 bit
7) Neuron state resolution: 16 bit
8) Maximum kernel size: 16x16
9) Stride: {1, 2, 4, 8} independent in X/Y
10) Padding: [0...7] independent in X/Y
11) Pooling: 1:1, 1:2, 1:4
12) Fanout: 2
13) Frequency: 1 MHz
14) Synaptic Operations limit per core: 10 millions synops/s
15) Maximum number of channels in readout layer: 15

Importantly, the Speck2e features an Integrate-and-Fire (IF) model. In comparison with a Leak-Integrate-and-Fire (LIF) model, the IF model is simpler and it requires less energy. However, it is also more limited, and at the onset of our study it was unclear if an IF model SNN would be able to estimate optical flow. In the next subsection, we explain the differences between LIF and IF model in more detail.

D. Neuron Model Comparison

This section describes the differences between the LIF model available on the Loihi chip and the IF model from the speck2e. Each layer in a SNN can be visualized as a convolutional layer plus a spiking activation function. The layer contains weights and biases, which define how much the input influences the potential of the neuron and therefore its output. The spiking function outputs a 1 or a 0 at every step, depending on whether the potential exceeded the threshold or not. A schematic of the neurons is provided in Figure 2 for LIF and Figure 3 for IF.

$$s(v[t]) = \begin{cases} 0 & \text{if } v[t] < v_{\text{mem}} \\ 1 & \text{if } v[t] > v_{\text{mem}} \end{cases} \tag{1}$$

$$\begin{aligned} i[t+1] &= w \cdot z[t] + \theta_i \cdot i[t] + b \\ v[t+1] &= v[t] \cdot \theta_v \cdot (1 - s(v[t])) + i[t+1] \end{aligned} \tag{2}$$

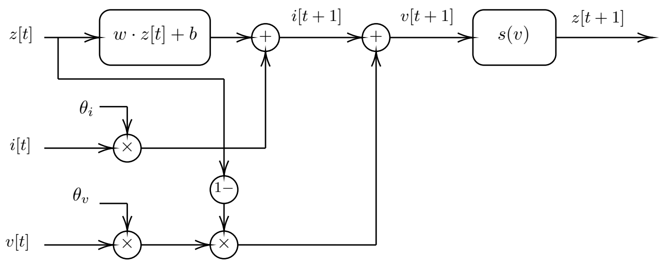


Fig. 2. LIF neuron computational graph.

$z(t)$ is the input spikes vector, $i(t)$ and $v(t)$ are the neurons' current and potential values. Additionally, the potential is multiplied by a reset term $(1 - z)$, so that if a neuron spiked in the previous step ($z = 1$), the potential is reset to zero. The spikes are fed in the convolutional layer, where they are multiplied by the weights and summed with the biases. The output of the convolutional layer represents the change in current due to the spikes, which is added to the previous state current. The new current $i(t+1)$ is summed with the previous potential to compute the new potential $v(t+1)$. Finally, the spiking function $z(v)$ checks if the potential is higher than the threshold and transmits spikes to the following layer.

In the LIF model, θ_i and θ_v stand for the current and potential leak respectively and they are multiplied by the previous state values. The leak represents how much information of the previous state is preserved in the following step. A higher leak value means that the current or potential decreases more slowly and more information is maintained through time. In the IF model the leak parameters are not present and the new incoming current is directly summed to the previous one.

$$\begin{aligned} i[t+1] &= w \cdot z[t] + i[t] + b \\ v[t+1] &= v[t] \cdot (1 - s(v)) + i[t+1] \end{aligned} \tag{3}$$

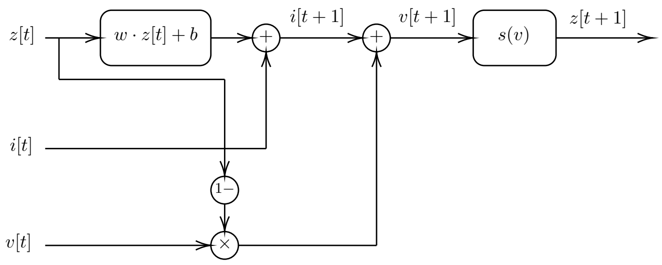


Fig. 3. IF neuron computational graph.

E. Sinabs Neuron Model

Sinabs [33] is a deep learning library based on PyTorch provided by Synsense. It is suited to test SNNs to be implemented on Synsense devices such as the speck2e. The sinabs model of an IF spiking layer is denoted as `IAFSqueeze` and it is characterized by the following parameters:

- `spike_fn` - Function defining the spiking output. If set to `MultiSpike`, a neuron will be able to produce mul-

tiple spikes in given time step. If set to `SingleSpike`, a neuron will produce at most one spike per time step.
- `reset_fn` - Function defining the reset phase of the potential. If set to `MembraneSubtract`, the threshold value is subtracted from the potential after spiking. If set to `MembraneReset`, the potential returns to zero after spiking.

For testing the network in simulation, the `IAFSqueeze` functions are set to `MultiSpike` mode. If the neurons in a certain binned frame are able to send multiple spikes, they can encode more information in that specific time and the flow can be better characterized. Regarding the reset function, the `MembraneReset` option is chosen instead of `MembraneSubtract`, in order to avoid membrane potential accumulation.

F. Recurrent Connections

For time-dependent tasks, IF SNNs would require recurrent connections in order to make predictions on the current state, while considering the previous one. There are two possible recurrent encoder configurations on the speck2e.

Sum Recurrency

In this type of encoder the first layer increases the number of channels from n_1 to n_2. The second layer processes the new input from the first layer, together with the previous state. The output of the second forward layer is summed directly to the new incoming state and then fed in the layer.

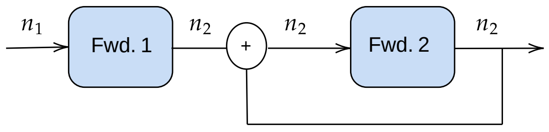


Fig. 4. Sum Recurrency

Concatenation Recurrency

This type of encoder has two forward layers and one recurrent layer. The first forward layer increases the number of channels from n_1 to n_2. The second forward layer processes the output of the first forward layer concatenated with the one of the recurrent layer, hence it has $2 \cdot n_2$ inputs and $2 \cdot n_2$ output channels. Finally, the recurrent convolutional layer decreases the number of channels back from $2 \cdot n_2$ to n_2.

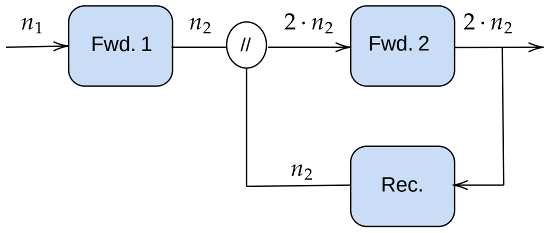


Fig. 5. Concatenation recurrency diagram.

G. Network Conversion & Weight rescaling

The Speck2e has been successfully used for recognition tasks (Section II). However, determining optical flow requires more extensive temporal processing of the events. This is because the network has to integrate the information of the previous states in time to determine the current state. With IF neurons, all the valuable information among layers is encoded in the spikes and the recurrent connections are the only memory mechanism in every layer.

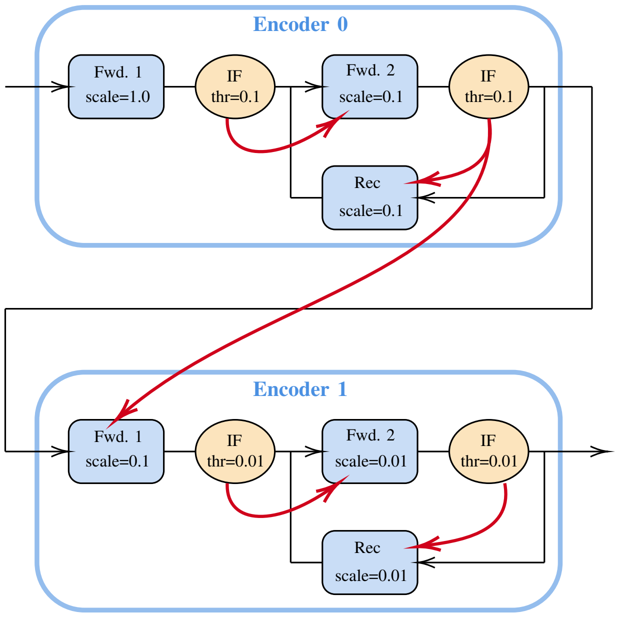


Fig. 6. Weight rescaling diagram.

A common strategy to train SNNs is to first train an ANN for ReLU activation functions and then replace them with spiking activation functions, as explained in Cao et al. [35]. This method works well for non time-dependent tasks such as image classification and object detection [10, 7]. In sinabs, when substituting ReLU functions with `IAFSqueeze`, the thresholds are set to $\pm$ 1.0 by default. However, if the synaptic weights have a different range of values, this thresholds might not be suitable and the layer might result in too high or too low activity. Rueckauer et al. [36] proposes a solution to this problem, which consists in normalizing the activity in every layer by scaling the weights to be in the appropriate range. However, this method was only applied to an object detection task, hence without recurrent connections and time dependency. In this article we propose the same approach as in Rueckauer et al.[36], but for a recurrent spiking network. The main difference is that, while for recognition tasks the weights are scaled to be in the right rang but still lower than the thresholds, in our method the weights are required to be higher than the thresholds.

Without weight rescaling, the information in every layer tends to accumulate membrane potential, instead of transmitting spikes to the following layers. If the information between layers is not promptly transmitted as it arrives, the recurrent connections are not going to be activated and therefore the network will lose the previous states. Setting the threshold too high with respect to the scale of the synaptic inputs will result in losing relevant information over time, while setting it too low will result in excessive information flow.

By observing the range of the output of the ReLU functions in every layer, a common threshold can be defined for every encoder. The threshold should be low enough to allow some spikes to pass through at every binned frame of 5 ms. Note that finding the optimal threshold value to allow the minimum number of spikes to pass, while still encoding the relevant information, is a complex optimization problem. There is not one ideal scaling factor for the weights, as this will depend on many parameters such as the spike rate in the dataset, the network depth, the spike threshold and many more. For this reason the thresholds will be selected empirically.

In the first layer of the first encoder, the ReLU outputs are in the range 10^{-1}, thus the upper and lower thresholds are set to $\pm$ 0.1 for all layers of the first encoder. The weights of the following layer are then re-scaled by 0.1. This pattern of rescaling the weights by the threshold value of the previous layer is repeated for all layers, as shown in Figure 6. In the second encoder the synaptic output range is in the order of 10^{-2}, thus the threshold is set to $\pm$ 0.01.

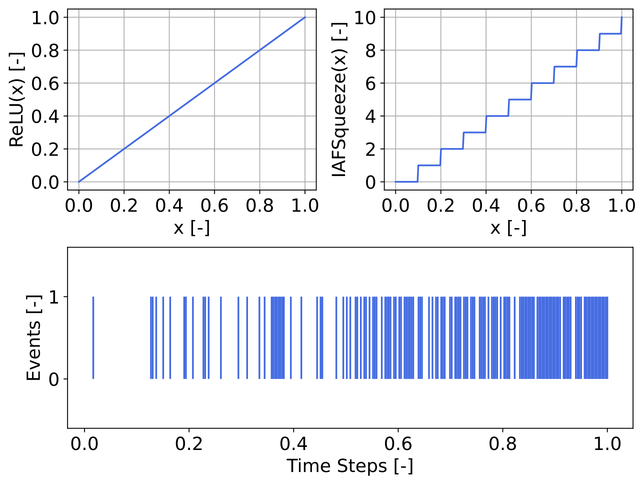


Fig. 7. Activation functions and resulting spike rate output.

Figure 7 shows the output of the ReLU function (continuous) compared to the output of the `IAFSqueeze` function (discrete) and the resulting spike rate. The idea of ANN-to-SNN conversion is using only the spikes to transmit information through layers and therefore reducing the resolution or sampling the information. Equation 4 shows that the output of the IF function z_{IF} is equal to the integer part of the ReLU output z_{ReLU} divided by the threshold v_{mem}. For instance, if the input of a neuron equal to 0.421 and the threshold of the neuron is set to 0.1, 4 spikes will be transmitted to all the neighboring neurons.

$$z_{\text{IF}} = \left\lfloor \frac{z_{\text{ReLU}}}{v_{\text{mem}}} \right\rfloor \tag{4}$$

H. Self-Supervised Loss Function

The network is trained in self-supervised learning. The loss function for is composed by two terms: *contrast maximization* and *flow smoothing*.

1) Contrast Maximization: this function quantifies the flow loss given a set of events. It was proposed by Gallego et al. [37] and it solely relies on the events stream, without any additional data required. Consider a set of positive and negative events in a certain spatio-temporal neighborhood. The events triggered by the same moving edges are expected to follow the same trajectories. The translational displacement of the pixels can be described by Equation 5, where (u, v) are the flow components.

$$\begin{pmatrix} x_i' \\ y_i' \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} + (t - t_i') \begin{pmatrix} u(x_i, y_i) \\ v(x_i, y_i) \end{pmatrix} \tag{5}$$

Calculating the correct flow can be interpreted as finding the best fitting trajectory that passes through the events generated by the same moving edge. Consider the scenario displayed in Figure 8. During training the network performs 5 forward passes before computing the loss. For each of the 5 frames, the events are transposed in time using the last flow vectors estimation. Essentially, the 5 frames are used to reconstruct the events at a reference time. The 5 reconstructions are then summed together and, if the flow estimation is correct, the pixels should be aligning without any blur. In reality, the first optical flow estimates will not be correct and will therefore result in a blurred reconstructed image (Figure 8). In order to measure this blur and minimize it, the density of events per pixels is calculated. If the flow estimation is accurate, the reconstructed events will be aligning on the same pixel.

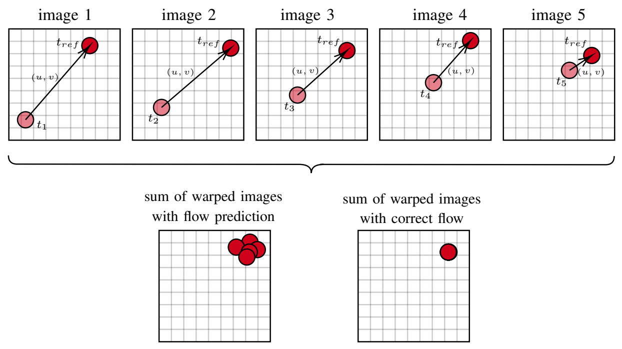


image 1
image 2
image 3
image 4
image 5
t_{ref}
(u, v)
t_1
t_2
t_3
t_4
t_5
sum of warped images with flow prediction
sum of warped images with correct flow

Fig. 8. Contrast maximization computation scheme.

To compute the contrast maximization loss function, first the events are separated by polarity. Following, the average timestamp image is generated at each pixel for each polarity, as in Zhu et al. [38].

$$T_{p'}(\mathbf{x};\mathbf{u}|t_{\text{ref}}) = \frac{\sum_j \kappa(x-x'_j)\kappa(y-y'_j)t_j}{\sum_j \kappa(x-x'_j)\kappa(y-y'_j)+\epsilon} \tag{6}$$
$$\kappa(a) = \max(0, 1-|a|)$$

To make the loss function convex, in Paredes-Vallés et al. [39] the contrast maximization function was scaled with the number of pixels with at least one warped event (Equation 7).

$$\ell_{\text{contrast}}(t_{\text{ref}}) = \frac{\sum_{\mathbf{x};\mathbf{u}} T_+(\mathbf{x};\mathbf{u}|t_{\text{ref}})^2 + T_-(\mathbf{x};\mathbf{u}|t_{\text{ref}})^2}{\sum_{\mathbf{x}}[n(\mathbf{x}')>0]+\epsilon} \tag{7}$$

2) Flow Smoothing: this term encourages smoothness in the estimated optical flow field. It is based on the assumption that neighboring pixels should have similar flow values. The smoothness function is meant to regularize the output flow and it is applied in the temporal domain to subsequent per-corner optical flow estimates.

$$\ell_{\text{smooth}} = \sum_{x,y}\sum_{i,j\in\mathcal{N}(x,y)} \rho(u(x,y)-u(i,j))+\rho(v(x,y)-v(i,j)) \tag{8}$$

The total loss function used during training to learn optical flow is Equation 9, where λ is a scalar balancing the two functions.

$$L_{\text{total}} = \ell_{\text{contrast}} + \lambda\ell_{\text{smooth}} \tag{9}$$

I. Synops Loss Function

As specified in Section III-C, the limit on the number of synaptic operations is 10 millions synops/s. By synaptic operation, it is meant all the spikes sent from the neurons of one layer to the neurons of another layer. In order to regularize the activity of the layers during training and prevent excessive spiking, Equation 10 is introduced as an additional term in the loss function.

$$\text{loss} = \sum_{k=1}^{N} \frac{(\text{total output})_k}{(\text{threshold})_k(\#\ \text{parameters})_k} \tag{10}$$

The total output of every layer at every step is the total number of spikes that a layer produces. During training the network is using ReLU functions and the output of every neuron is a floating point. However the range of weights values for every layer changes and, if it is not rescaled, the deeper layers with lower weight values will have less importance in the cost function. Moreover, layers with more parameters need to have a higher spike rate to send more information, thus the layer output is also scaled by the number of parameters. The weight of the synops loss term on the total loss function has to be tuned, in order to achieve a balance that allows to learn optical flow properly and with the minimum number of operations.

IV. Simulation Results

A. Training and Testing Datasets

The network is trained on two main datasets. The first one is addressed as CyberZoo dataset[18] and it includes 40 minutes of event data, which are split into 25 minutes for training and 15 for testing. This dataset includes translational and rotational motion in multiple directions and at different speeds. The CyberZoo dataset will be used to quantify the network's accuracy with respect to ground truth data and assess its performance.

The second dataset is provided by the University of Zurich and it is denoted as Davis dataset [40]. This dataset includes two main video sequences of 60 seconds each. It contains translational and rotational motion, however due to its limited size, it is not as complex and diversified as the CyberZoo dataset. It was observed that networks trained on the Cyberzoo dataset have a higher number of synops with respect to networks trained on the Davis dataset. This is most likely due to the higher complexity of the Cyberzoo sequences, which require more spikes to identify the motion. Therefore, for the hardware experiments, a sequence from the Davis dataset is used as input in the speck2e, instead of the events coming from the DVS. This is done primarily to avoid collecting another dataset and to validate the network on more simple motions.

B. Network Configurations

A total of 5 networks is tested on the datasets and compared to the peformance of the LIF network used in Paredes-Vallés et al. [18].

1) RNN-2-S: recurrent neural network with 2 encoders, sum recurrency and ReLU activation functions.
2) IF-2-S: spiking equivalent of RNN-2-S.
3) RNN-2-C: recurrent neural network with 2 encoders, concatenation recurrency and ReLU activation functions.
4) IF-2-C: spiking equivalent of RNN-2-C.

Note that the number of encoders is decreased from 3 to 2, because of the limit on number of convolutional cores on the speck2e.

C. Results on CyberZoo dataset

The network architectures are tested on the CyberZoo dataset and compared to the LIF network. The comparison metrics are Signal-to-Noise Ratio (SNR), Average Endpoint Error (AEE) and The Ratio of Squared Average Timestamps (RSAT). Table I and Figure 9 shows the results for the different network configurations compared to the LIF-3 [18]. All the metrics are computed on a single sequence from the dataset.

Signal-to-Noise Ratio

To calculate the SNR, a sequence of 5.0 s from the testing dataset is considered. The network outputs 8 optical flow predictions, a horizontal and a vertical component for each of the 4 corners of the image. To distinguish the power of the signal from the noise, the predictions are transposed to the frequency domain using Fast Fourier Transform (FFT).

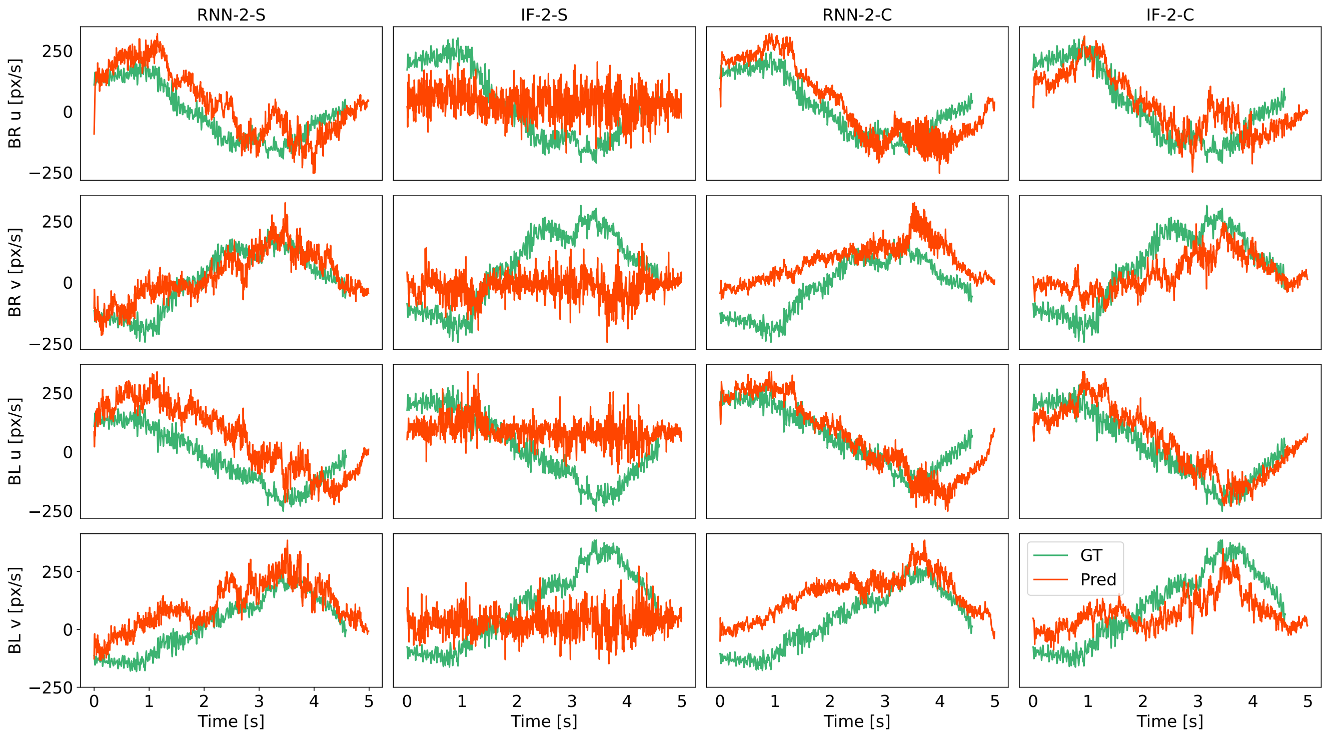


Fig. 9. Ground truth compared to network prediction for RNN-2-S, IF-2-S, RNN-2-C, IF-2-C

In this way, the main signal frequency can be identified by observing the peaks. The frequency range of the main signal is defined by observing the FFT of the ground truth signal. Using Equation 11 the average SNR can be calculated for each flow vector signal and compared to the ground truth.

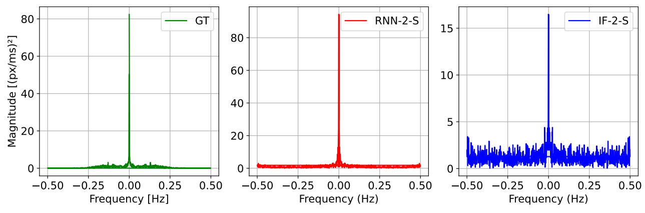


Fig. 10. FFT of ground truth signal compared to RNN-2-S and IF-2-S.

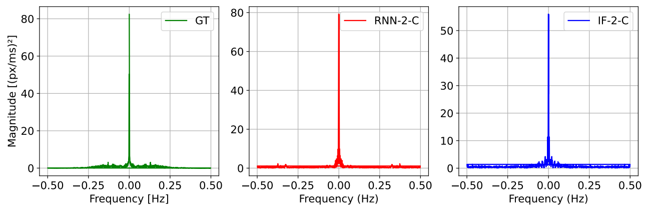


Fig. 11. FFT of ground truth signal compared to RNN-2-C and IF-2-C.

Figure 10 and 11 show the FFT of the ground truth signal compared to the network predictions of the ReLU and IF network. By zooming in the graph, it is found that the ground truth has most of its power in the frequencies $\pm 2.5 \cdot 10^{-3}$ Hz, hence this range is chosen to define the main signal. The FFT shows that the IF prediction is weaker in that range and has more power in other frequencies, hence the SNR is expected to be lower.

$$\mathrm{SNR} = 10 \cdot \log_{10}\left(\frac{P_{\text{signal}}}{P_{\text{noise}}}\right) \tag{11}$$

Average Endpoint Error & Standard Deviation

The Average Endpoint Error (AEE) is a metric used to evaluate the accuracy of optical flow algorithms. It is calculated by taking the each prediction of the network per corner of the image (D) and calculating the squared difference with ground truth. Then the respective errors of u and v are summed and the average of all corners is taken. This value represents the endpoint error at a certain time step. For all the error values the average and standard deviation can be calculated. The AEE is the average of all endpoint errors at every time step (N)

$$e_i = \frac{1}{D}\sum_{j}^{D}\left[(u_j - \hat{u}_j)^2 + (v_j - \hat{v}_j)^2\right] \tag{12}$$

$$\mathrm{AEE} = \frac{1}{N}\sum_{i}^{N} e_i \quad \mathrm{STD} = \sqrt{\frac{1}{N}\sum_{i}^{N}(e_i - \mathrm{AEE})^2} \tag{13}$$

Ratio of Squared Average Timestamps

RSAT is the ratio of the squared sum of the per-pixel and per-polarity average timestamp of the image of warped events and that of the image of (non-warped) events. Essentially, the RSAT measures the sharpness of the reconstructed image and it is an indication of how well the flow is estimated. The lower the value of this metric, the better the optical flow estimate. Note that this metric is sensitive to the number of input events.

TABLE I
SNR, AEE AND RSAT FOR THE DIFFERENT NETWORKS ON A CYBERZOO DATASET SEQUENCE.

Configuration	SNR (dB) ↑	AEE ↓	STD ↓	RSAT ↓
LIF-3 [18]	9.09	0.199	0.0531	0.917
RNN-2-S	9.11	0.183	0.0585	0.913
IF-2-S	-0.877	0.163	0.0637	0.987
RNN-2-C	12.2	0.211	0.0567	0.896
IF-2-C	6.98	0.171	0.0668	0.965

D. Network Configuration Trade-off

In this section, the results of the different network configurations on the Cyberzoo dataset are discussed.

Recurrency Type

With the same number of encoders, RNN-2-S has a slightly lower AEE than RNN-2-C and a more or less similar STD, although this does not necessarily indicate a better performance, since the RSAT is lower for concatenation networks. By observing the predictions compared to ground truth in Figure 9 it can be seen that both networks are able to follow the motion fairly accurately. However, in terms of noisiness, RNN-2-C and IF-2-C have a better SNR. In Figure 9 it can be seen that RNN-2-S predictions are more subject to additional fluctuations that are not present in the ground truth signal. Moreover, when converted, IF-2-S does not work properly with the weight rescaling method and most of the main signal seems to be lost. Concatenation seems to work better in this case, most likely because of its additional convolutional layer in the recurrent connection. This layer processes the previous state and it serves as a filter for non-relevant information. With sum recurrency, the previous state is directly summed with the new incoming state and then processed, therefore the convolutional layer does not distinguish between past and present information.

Network Conversion

Converting RNN to IF inevitably lowers the SNR and introduces additional noisy frequencies. When the network is converted, the resolution of the information between layers is reduced and therefore output signal becomes less defined. With sum recurrency (SNR = -0.877 dB), the conversion has way more impact on the SNR than with concatenation (SNR = 6.98 dB). Moreover, when converting, the AEE drops while the RSAT and STD increase. As specified earlier, a lower AEE does not indicate better performance. Indeed the STD increase proves that the errors are more dispersed.

Neuron Models

From the previous discussion, it was established that IF-2-C appears to be a more suitable option. Compared to LIF-3, IF-2-C performance worse in most of the metrics. This is because LIF-3 not only has more hyperparameters than IF-2-C, but also a higher level of complexity of the neuron model. LIF neurons are better at capturing temporal dynamics, since not all the information has to be sent through the spikes and some of the irrelevant one is filtered out by the leak system. Despite the limitations, the IF-2-C is able to achieve a comparable performance with respect to LIF-3.

E. Results on Davis Dataset

The Davis dataset does not include ground truth optical flow, thus the performance of the network is evaluated by visual inspection. The size of the dataset sequences is reduced from 128 pixels to 90, in order to further reduce the number of synaptic operations for hardware implementation. Figure 12 shows the output of IF-2-C on a horizontal and a vertical motion sequence. The pictures on top represent the color-coded dense optical flow. Every pixel has a different color depending on the flow at that point (check Figure 19 for color map).

F. Synaptic Operations Analysis

Besides using the synops loss term, some of the network hyperparameters can be altered to influence the number of synops.

Stride

To minimize the synops, the stride of the first encoder can be changed to 4, as in the second encoder. By halving the size of the images in the first encoder, less information is forwarded, hence the layers produce less synops (Figure 20 and 21).

Number of Channels

Increasing the number of channels can make the network's activity more sparse. Choosing a configuration with more neurons means having more neurons doing less work, therefore less operations per neuron overall (Figure 22 and 23).

Early Stop

During training, the network establishes the synaptic weights to optimize for minimal loss. However, in this case the minimum loss is often achieved in the first few epochs. By letting the network train more, additional connections are formed, which make the predictions more accurate. These redundant weights are useful for making the network more precise, however they increase the number of operations significantly. If the training is stopped earlier and the model is still able to estimate the flow accurately, a more operations efficient result can be obtained (Figure 24 and 25).

Considering the influence of the hyperparameters, a new network configuration is trained for implementation on the speck2e (Table IV).

G. Synops Loss Term

Recurrent connections on the speck2e process at a frequency of 1 MHz, thus the number of synops could change significantly compared to IF-2-C in simulation. Nevertheless, the total number of spikes produced by all the layers can still be reduced by including the synops loss term explained in Section III-I.

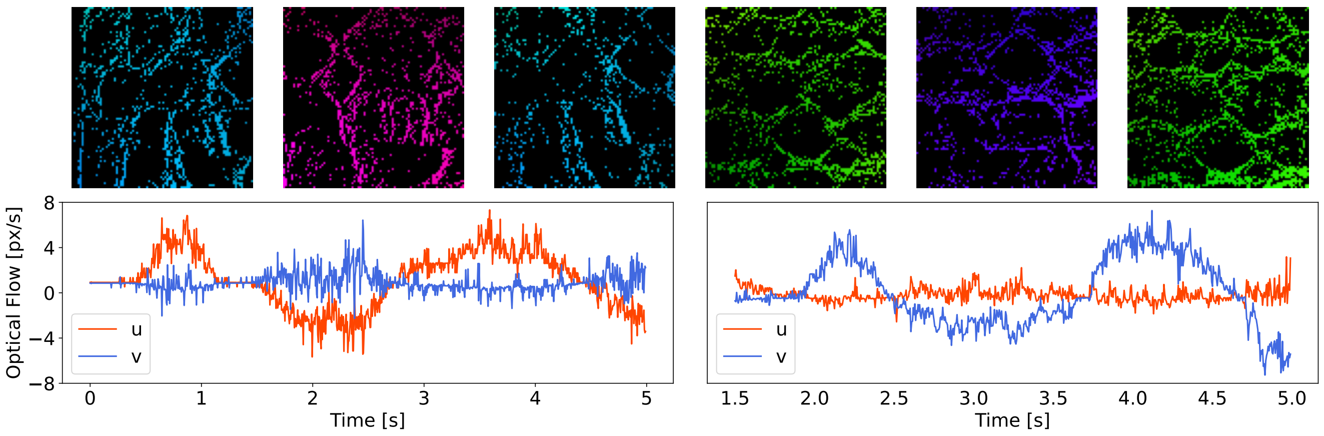


Fig. 12. IF-2-C optical flow vectors for the Davis dataset sequences.

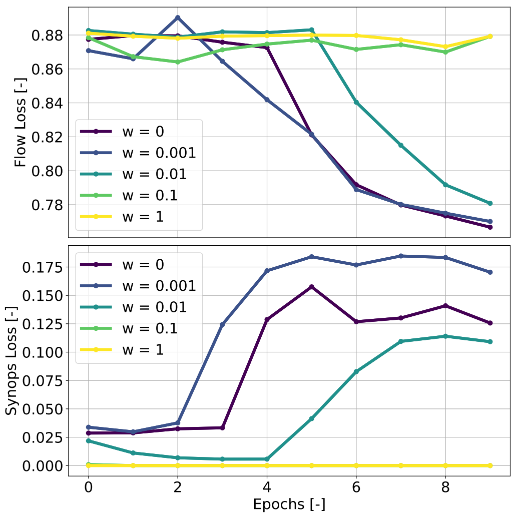


Fig. 13. Flow and synops loss during training with different weighting.

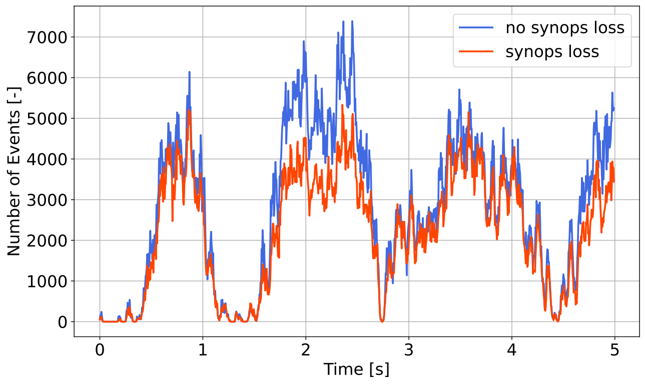


Fig. 14. Total number of synops for network trained with synops loss compared to network trained without synops loss.

Figure 13 shows the flow loss and synops loss over 10 epochs. By looking at the synops loss, it is clear that 0.01 is an optimal value for the weight, as it allows the flow loss to converge to a slightly higher value, while keeping the synops lower. If the weight is increased to 0.1, the network minimizes the synops to zero and it is prevented from learning flow. If it is decreased to 0.001, its influence is almost unnoticeable and the synops increase to more than necessary. Figure 14 shows the influence of the loss function on the total number of spikes of all the layers in the network.

V. Hardware Implementation

A. Hardware Setup Overview

The speck2e hosts the 2 encoders and the pooling layer, while the prediction layer post-processes the spikes outside the chip and translate them into optical flow vectors. The spikes are binned in time windows of 5 ms before being fed into the prediction layer. For the preliminary tests, the input is not from the DVS, but instead a sequence from the Davis dataset is used. A diagram of the hardware setup is shown in Figure 15.

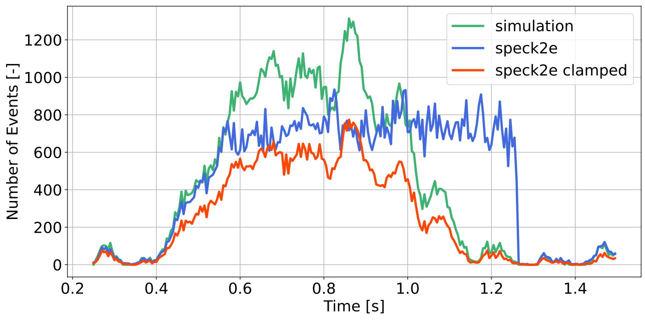


Fig. 16. Spike activity comparison

B. Spike Activity Comparison

Figure 16 shows the total number of spikes for every 5 ms frame in the first layer of the first encoder. While in simulation the layer follows a specific trend dictated by the input spikes,

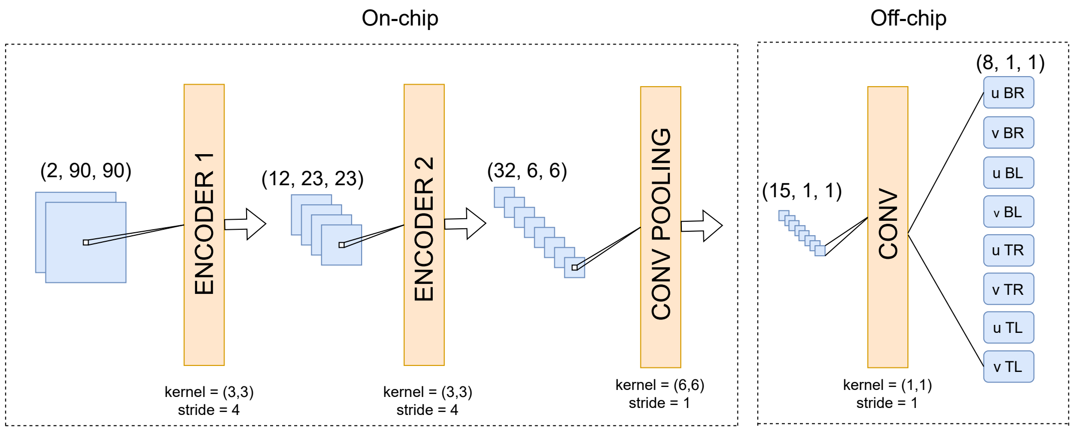


Fig. 15. Overview of the network implementation on the speck2e

on the speck2e the layer appears to be reaching a saturation point, after which it cannot produce more spikes. When the input spikes increase, the output spikes of the layer increase up to a certain point and then they start accumulating. When the inputs decrease again around 1000 ms, the accumulated outputs are released sequentially. Accumulation obviously results in information loss, because the timestamp of the spikes is delayed. Although it is still possible to retrieve the optical flow prediction by entering a predefined sequence (Figure 18), these settings are not ideal for real-time vision and control. If the network circuit received inputs from the DVS directly and the spikes between layers started to accumulate, this would result in a backlog, since the DVS would not be able to transmit more spikes to the network.

C. Weight Clamping

Although different methods to reduce the spikes were implemented in Section IV-F, the convolutional cores still cannot output as many spikes as necessary to predict the flow properly. This gap in performance between simulation and hardware is most likely due the weight values being larger than the thresholds.

To prove this, the network is tested with the same input sequence, but the weights are clamped to a maximum equal to the threshold and a minimum equal to the negative of the threshold. Figure 17 shows the effect of clamping on the weight distribution. In the first layer of the first encoder the quantized thresholds are approximately $\pm$ 30. Constraining the weights to the threshold values results in a reduction of information, because for every synaptic input, a neuron can spike at most once. This restricts the range of possible neuronal activity patterns, hence it reduces the accuracy of the network. However, the spike activity does not get stuck at the saturation point anymore (Figure 16 red line). With these settings, the information flows more smoothly throughout the network without accumulating.

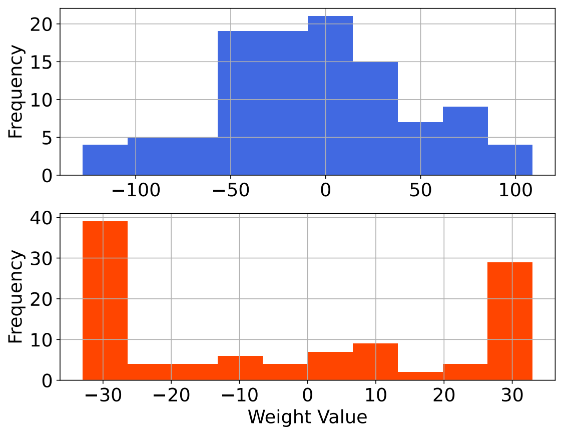


Fig. 17. Effect of clamping on weights distribution in first layer of first encoder.

D. Optical Flow Prediction

The spikes of the pooling layer are binned into frames and fed in the last off-chip layer. The resulting optical flow is shown in Figure 18.

IF-2-C on speck2e
When the weights are not clamped, the network prediction manages to follow the simulation result. However, especially at the peaks, the prediction from the speck2e deteriorates. This is because, for the network to be accurate at higher magnitudes of optical flow, more spikes need to be transmitted between the layers, which is not possible if the layers reach the saturation point.

IF-2-C on speck2e with clamped weights
By clamping the weights, the layers are able to send spikes without reaching the limit. However, the network parameters are way more different from the trained solution, hence the prediction loses accuracy. This proves that the limitations of the speck2e are encountered when using weights values with higher values than thresholds, as this results in a too high demand for spikes processing.

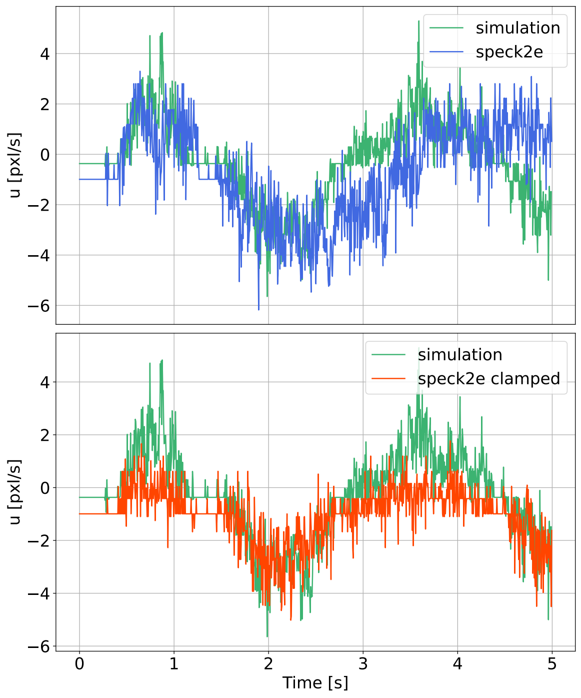


Fig. 18. Optical flow prediction for normal network and network with clamped weights on the speck2e.

It is interesting to notice that for negative flow values, the speck2e output is more accurate than for positive ones. This is likely because during training, the network associated fewer spikes with negative flow and more spikes with positive flow. To predict higher values of positive flow, the layers will need to spike more than allowed by the device limitations. However, for negative flow values, less spikes are required and a better flow estimate is obtained. Using clamped weights, the range of possible optical flow values is roughly [-4,1] pixel/s. To further improve the prediction, one could re-train the network by constraining the activity to obtain a range of possible optical flow that is symmetric in positive and negative direction.

VI. Results Discussion & Conclusion

In this paper, we presented a novel approach for optical flow determination using neuromorphic computing. The speck2e device from Synsense was used in the experiments and the network configuration was inspired by Paredes-Vallés et al. [18]. This device could result in lighter and more power efficient MAVs, as it includes sensing and computing into one board. It was shown that it is possible to estimate optical flow from event-based datasets in simulation with a RNN trained and converted to SNN. A mechanism of weight rescaling was applied on the network parameters to avoid information loss between the layers after the conversion. Although the simulation results seem promising, they are based on the assumption that neurons on board of the speck2e can have synaptic weight values higher than the thresholds and output a spike rate proportional to such weight. Uploading and testing a network with this characteristics is possible, however the resulting performance of the network on hardware is quite different than in simulation. It was noticed that the convolutional cores reach a limit on the number of synaptic operations, despite the regularization term used during training. This saturation point is reached if the weights are higher than the thresholds, since the potential reaches its limit much faster. The result is spikes accumulation which results a loss of information. It is possible to alter the parameters to make the network run without spikes accumulation. This is achieved by clamping the weight values to the threshold limits. Note that this severely reduces the range of possible weight values, hence it makes the network more limited and the prediction less exact.

This paper served to explore the limitations and challenges of training a speck2e-compatible IF network to estimate optical flow. With the help of this analysis it might be possible to further improve the performance of the network, by re-training it under the constraints of the device. A good improvement would be re-training the network to have its range of possible optical flow values to be symmetric in positive and negative direction. One limitation that is still constraining the speck2e is the limit on synaptic operations per second. This represent a complex challenge, considering that all the information has to be sent through the spikes between layers.

References

[1] Carlo Michaelis, Andrew B. Lehr, and Christian Tetzlaff. “Robust Trajectory Generation for Robotic Control on the Neuromorphic Research Chip Loihi”. In: *Frontiers in Neurorobotics* 14 (2020). ISSN: 1662-5218. DOI: 10.3389/fnbot.2020.589532. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2020.589532.

[2] Raphaela Kreiser, Alpha Renner, and Yulia Sandamirskaya. “Error-driven learning for self-calibration in a neuromorphic path integration system”. In: Aug. 2019.

[3] Raphaela Kreiser et al. “Error estimation and correction in a spiking neural network for map formation in neuromorphic hardware”. In: May 2020, pp. 6134–6140. DOI: 10.1109/ICRA40945.2020.9197498.

[4] Catherine D Schuman et al. “A survey of neuromorphic computing and neural networks in hardware”. In: *arXiv preprint arXiv:1705.06963* (2017).

[5] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models”. In: *Neural Networks* 10.9 (1997), pp. 1659–1671. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(97)00011-7. URL: https://www.sciencedirect.com/science/article/pii/S0893608097000117.

[6] André Grüning and Sander M Bohte. “Spiking neural networks: Principles and challenges.” In: *ESANN*. Bruges. 2014.

[7] Caterina Caccavella et al. *Low-power event-based face detection with asynchronous neuromorphic hardware*. 2023. arXiv: 2312.14261 [cs.NE].

[8] David Drazen et al. “Toward real-time particle tracking using an event-based dynamic vision sensor”. In: *Experiments in Fluids* 51 (Nov. 2011), pp. 1465–1469. DOI: 10.1007/s00348-011-1207-y.

[9] Zhenjiang Ni et al. “Asynchronous event-based high speed vision for microparticle tracking”. In: *Journal of microscopy* 245 (Nov. 2011), pp. 236–44. DOI: 10.1111/j.1365-2818.2011.03565.x.

[10] Arnon Amir et al. “A Low Power, Fully Event-Based Gesture Recognition System”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[11] Enea Ceolini et al. “Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing”. In: *Frontiers in Neuroscience* 14 (2020). ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00637. URL: https://www.frontiersin.org/articles/10.3389/fnins.2020.00637.

[12] Haiyang Chao, Yu Gu, and Marcello Napolitano. “A Survey of Optical Flow Techniques for Robotics Navigation Applications”. In: *Journal of Intelligent I& Robotic Systems* 73 (May 2013). DOI: 10.1007/s10846-013-9923-6.

[13] S. S. Beauchemin and J. L. Barron. “The Computation of Optical Flow”. In: 27.3 (Sept. 1995), pp. 433–466. ISSN: 0360-0300. DOI: 10.1145/212094.212141. URL: https://doi.org/10.1145/212094.212141.

[14] Javaan Chahl, Mandyam Srinivasan, and Shaowu Zhang. “Landing Strategies in Honeybees and Applications to Uninhabited Airborne Vehicles”. In: *I. J. Robotic Res.* 23 (Feb. 2004), pp. 101–110. DOI: 10.1177/0278364904041320.

[15] Harald E. Esch and John E. Burns. “Distance Estimation by Foraging Honeybees”. In: *Journal of Experimental Biology* 199.1 (Jan. 1996), pp. 155–162. ISSN: 0022-0949. DOI: 10.1242/jeb.199.1.155. eprint: https://journals.biologists.com/jeb/article-pdf/199/1/155/3107314/jexbio_199_1_155.pdf. URL: https://doi.org/10.1242/jeb.199.1.155.

[16] Mandyam Srinivasan. “Honeybees as a Model for the Study of Visually Guided Flight, Navigation, and Biologically Inspired Robotics”. In: *Physiological reviews* 91 (Apr. 2011), pp. 413–60. DOI: 10.1152/physrev.00005.2010.

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

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

[19] *Loihi 2: A New Generation of Neuromorphic Computing*. URL: https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html.

[20] *Speck: Event-Driven Neuromorphic SoC — Synsense*. URL: https://www.synsense.ai/products/speck-2/.

[21] *Neuromorphic Intelligence I& Application Solutions — Synsense*. URL: https://www.synsense.ai/.

[22] Ole Richter et al. *Speck: A Smart event-based Vision Sensor with a low latency 327K Neuron Convolutional Neuronal Network Processing Pipeline*. 2023. arXiv: 2304.06793 [cs.NE].

[23] Fernando Perez-Peña, Alejandro Linares-Barranco, and Elisabetta Chicca. “An approach to motor control for spike-based neuromorphic robotics”. In: *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*. 2014. DOI: 10.1109/BioCAS.2014.6981779.

[24] J.C. Gallacher and J.M. Fiore. “Continuous time recurrent neural networks: a paradigm for evolvable analog controller circuits”. In: *Proceedings of the IEEE 2000 National Aerospace and Electronics Conference. NAECON 2000. Engineering Tomorrow (Cat. No.00CH37093)*. 2000, pp. 299–304. DOI: 10.1109/NAECON.2000.894924.

[25] D. Roggen et al. “Hardware spiking neural network with run-time reconfigurable connectivity in an autonomous robot”. In: *NASA/DoD Conference on Evolvable Hardware, 2003. Proceedings.* 2003, pp. 189–198. DOI: 10.1109/EH.2003.1217666.

[26] Di Hu et al. “Digital implementation of a spiking neural network (SNN) capable of spike-timing-dependent plasticity (STDP) learning”. In: *14th IEEE International Conference on Nanotechnology*. 2014, pp. 873–876. DOI: 10.1109/NANO.2014.6968000.

[27] Patrick Rocke et al. “Reconfigurable Hardware Evolution Platform for a Spiking Neural Network Robotics Controller”. In: *Reconfigurable Computing: Architectures, Tools and Applications*. Ed. by Pedro C. Diniz et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 373–378. ISBN: 978-3-540-71431-6.

[28] Giacomo Indiveri and Paul Verschure. “Autonomous vehicle guidance using analog VLSI neuromorphic sensors”. In: *Artificial Neural Networks — ICANN’97*. Ed. by Wulfram Gerstner et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 811–816. ISBN: 978-3-540-69620-9.

[29] Scott Koziol, Stephen Brink, and Jennifer Hasler. “A Neuromorphic Approach to Path Planning Using a Reconfigurable Neuron Array IC”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.12 (2014), pp. 2724–2737. DOI: 10.1109/TVLSI.2013.2297056.

[30] Jesse J. Hagenaars et al. “Evolved Neuromorphic Control for High Speed Divergence-based Landings of MAVs”. In: *CoRR* abs/2003.03118 (2020). arXiv: 2003.03118. URL: https://arxiv.org/abs/2003.03118.

[31] Steven Abreu et al. “Flow Cytometry With Event-Based Vision and Spiking Neuromorphic Hardware”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2023, pp. 4139–4147.

[32] Kangrui Du et al. *Temporal Flexibility in Spiking Neural Networks: A Novel Training Method for Enhanced Generalization Across Time Steps*. 2024. URL: https://openreview.net/forum?id=RmQAKu1wCe.

[33] *Sinabs (Sinabs Is Not A Brain Simulator)*. URL: https://sinabs.readthedocs.io/en/1.2.8/index.html.

[34] *Samna*. URL: https://pypi.org/project/samna/.

[35] Yongqiang Cao, Yang Chen, and Deepak Khosla. "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition". In: *International Journal of Computer Vision* 113 (May 2015), pp. 54–66. DOI: 10.1007/s11263-014-0788-3.

[36] Bodo Rueckauer et al. "Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification". In: *Frontiers in Neuroscience* 11 (2017). ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2017.00682.

[37] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. "A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3867–3876. DOI: 10.1109/CVPR.2018.00407.

[38] Alex Zihao Zhu et al. "Unsupervised Event-based Learning of Optical Flow, Depth, and Egomotion". In: *CoRR* abs/1812.08156 (2018). arXiv: 1812.08156. URL: http://arxiv.org/abs/1812.08156.

[39] Federico Paredes-Vallés, Jesse J. Hagenaars, and Guido de Croon. "Self-Supervised Learning of Event-Based Optical Flow with Spiking Neural Networks". In: *CoRR* abs/2106.01862 (2021). arXiv: 2106.01862. URL: https://arxiv.org/abs/2106.01862.

[40] Elias Mueggler et al. "The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM". In: *CoRR* abs/1610.08336 (2016). arXiv: 1610.08336. URL: http://arxiv.org/abs/1610.08336.

Appendix A
Optical Flow Colormap

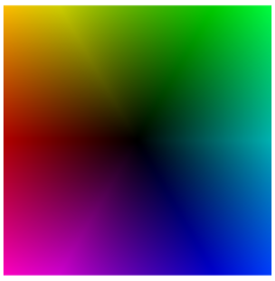


Fig. 19. Optical flow color map.

Appendix B
Network Configuration Details

This section includes the network configuration details, meaning the channels number, size, kernel, stride and padding for each convolutional layer.

TABLE II
Network Architecture for RNN-2-S on CyberZoo dataset.

	Layer	In Channels	Out Channels	In Size	Out Size	Kernel	Stride	Padding
Encoder 0	Fwd.	2	4	180	90	3	2	1
	Rec.	4	4	90	90	3	1	0
Encoder 1	Fwd.	4	8	90	23	3	4	1
	Rec.	8	8	23	23	3	1	0
	Pooling	8	8	23	1	23	1	0
	Prediction	8	8	1	1	1	1	0

TABLE III
Network Architecture for RNN-2-C on CyberZoo dataset.

	Layer	In Channels	Out Channels	In Size	Out Size	Kernel	Stride	Padding
Encoder 0	Fwd. 1	2	4	180	90	3	2	1
	Fwd. 2	8	8	90	90	3	1	1
	Rec.	8	4	90	90	3	1	0
Encoder 1	Fwd. 1	8	16	90	23	3	4	1
	Fwd. 2	32	32	23	23	3	1	1
	Rec.	32	16	23	23	3	1	0
	Pooling	32	8	23	1	23	1	0
	Prediction	8	8	1	1	1	1	0

TABLE IV
Network Architecture for RNN-2-C on speck.

	Layer	In Channels	Out Channels	In Size	Out Size	Kernel	Stride	Padding
Encoder 0	Fwd. 1	2	6	90	23	3	4	1
	Fwd. 2	12	12	23	23	3	1	1
	Rec.	12	6	23	23	3	1	0
Encoder 1	Fwd. 1	12	16	23	6	3	4	1
	Fwd. 2	32	32	6	6	3	1	1
	Rec.	32	16	6	6	3	1	0
	Pooling	32	15	6	1	6	1	0
	Prediction	15	8	1	1	1	1	0

Appendix C
Effect of Hyperparameters on Synops

This section includes the plots of number of synaptic operations and maximum number of operations per neuron over time, for the first convolutional layer for different network configurations. In each pair of plots, a single hyperparameter was altered to observe the effect on the synaptic operations.

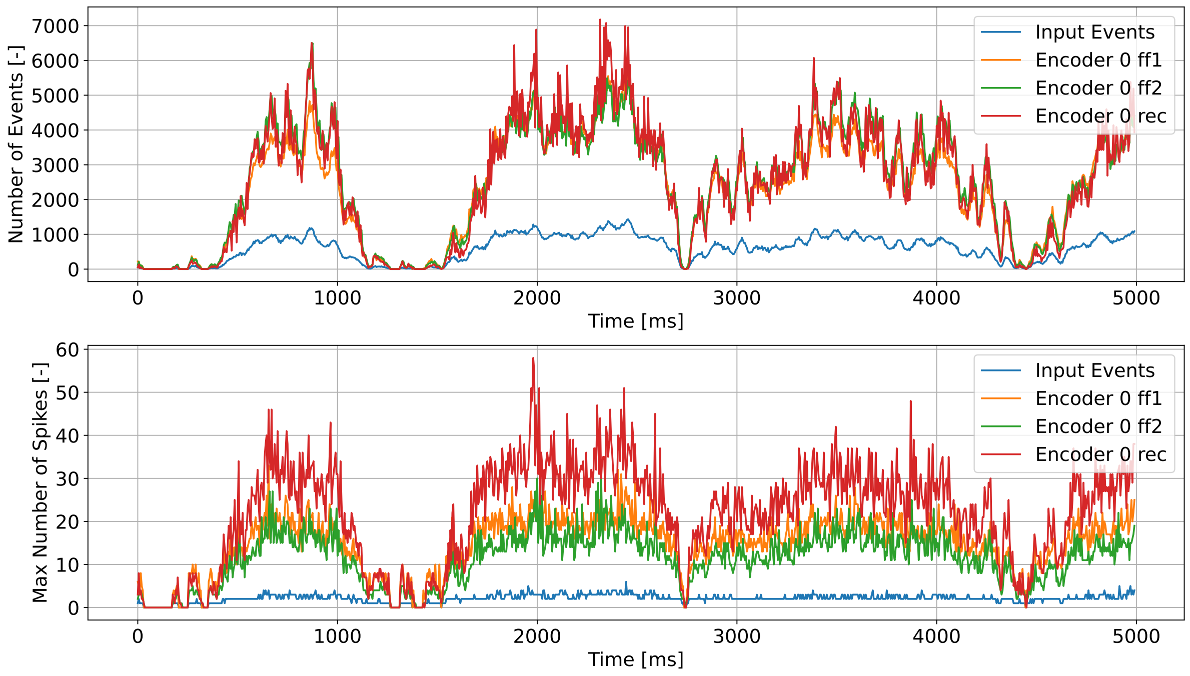


Fig. 20. Spike activity in RNN-2-C with stride 2 in first encoder.

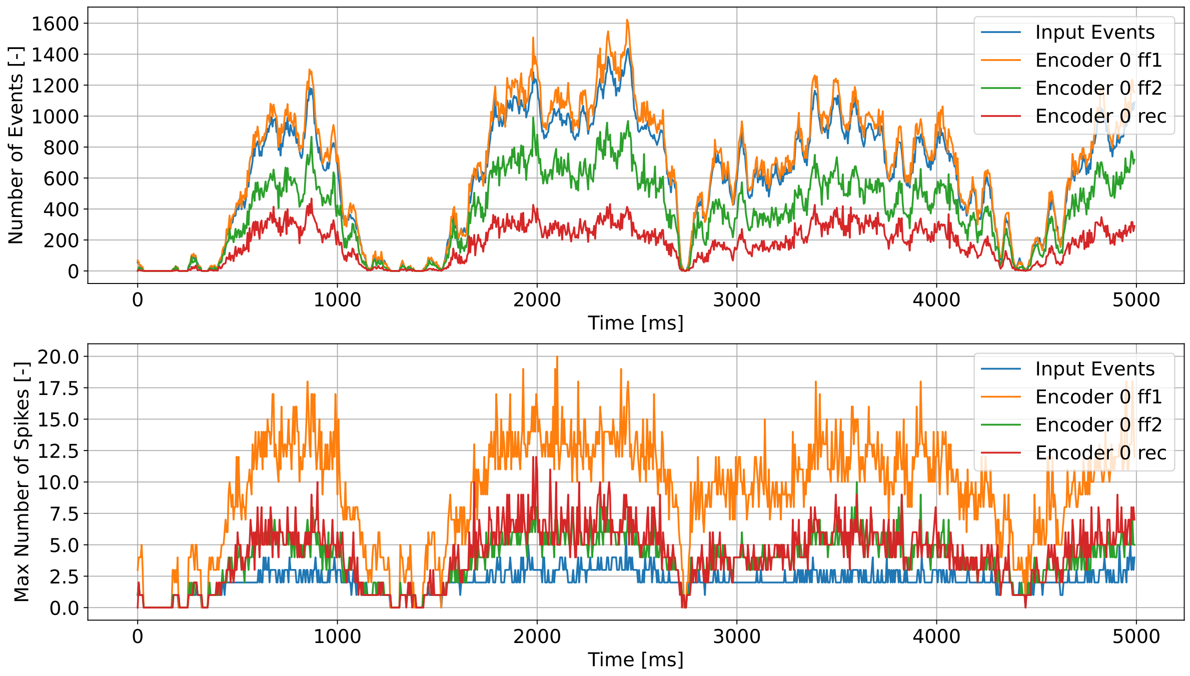


Fig. 21. Spike activity RNN-2-C with stride 4 in first encoder.

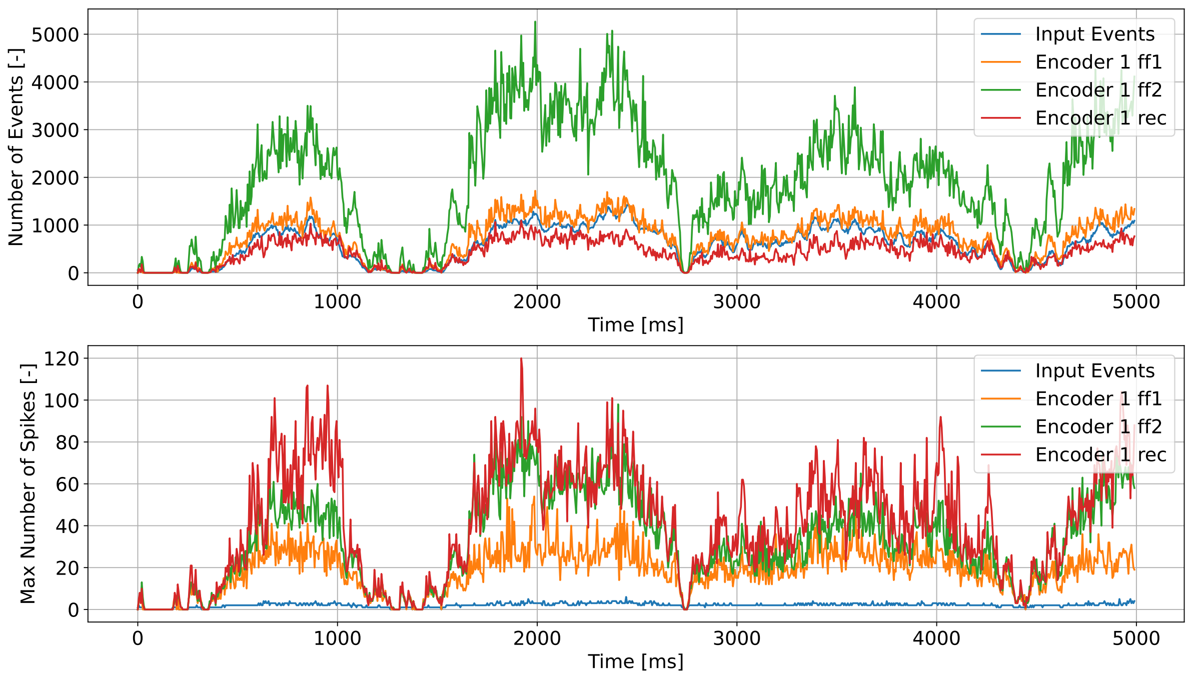


Fig. 22. RNN-2-B with increased number of channels.

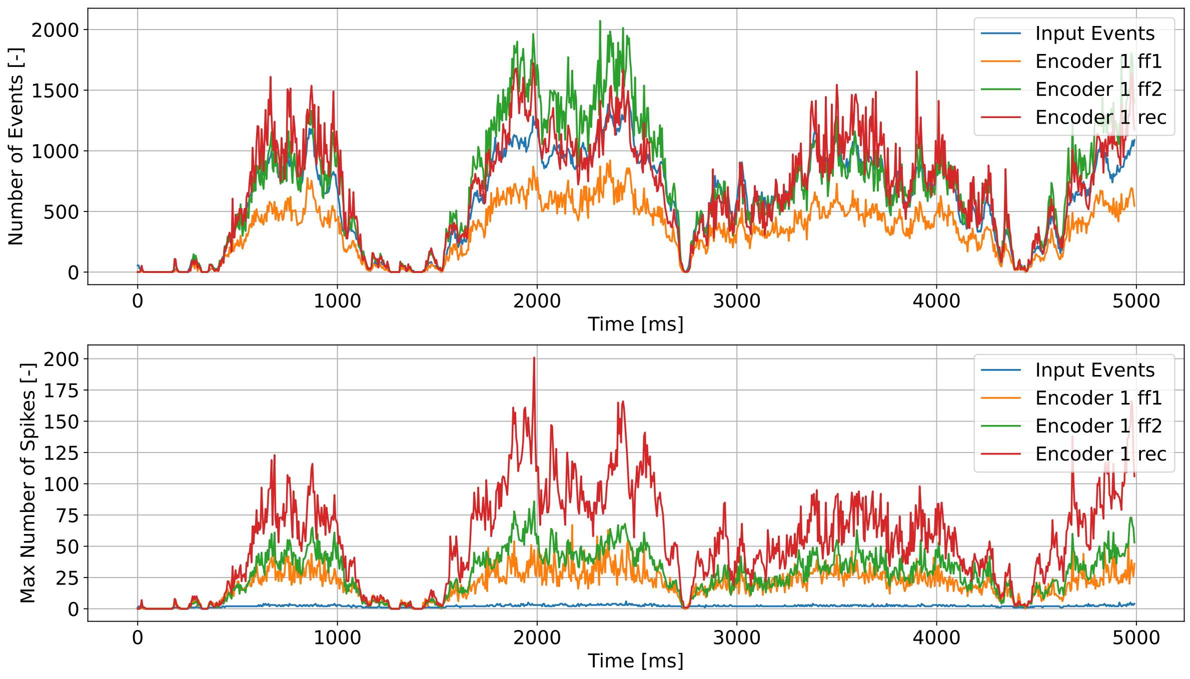
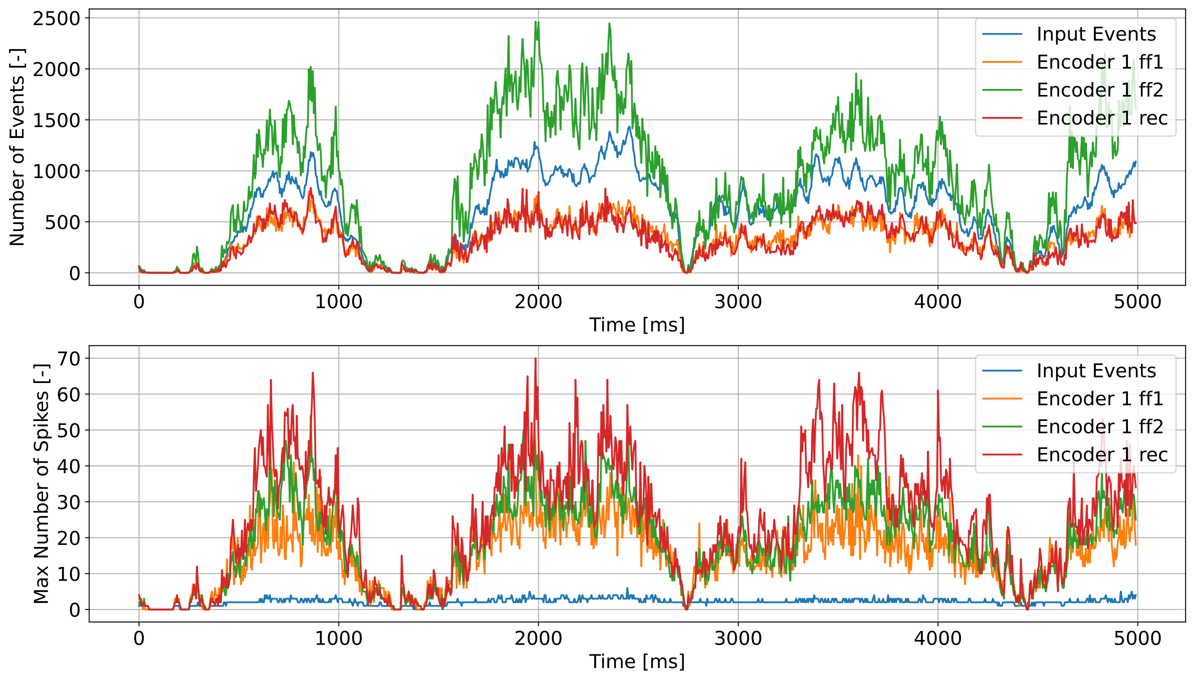


Fig. 23. RNN-2-C with decreased number of channels.



H

Fig. 24. RNN-2-C after 80 epochs.

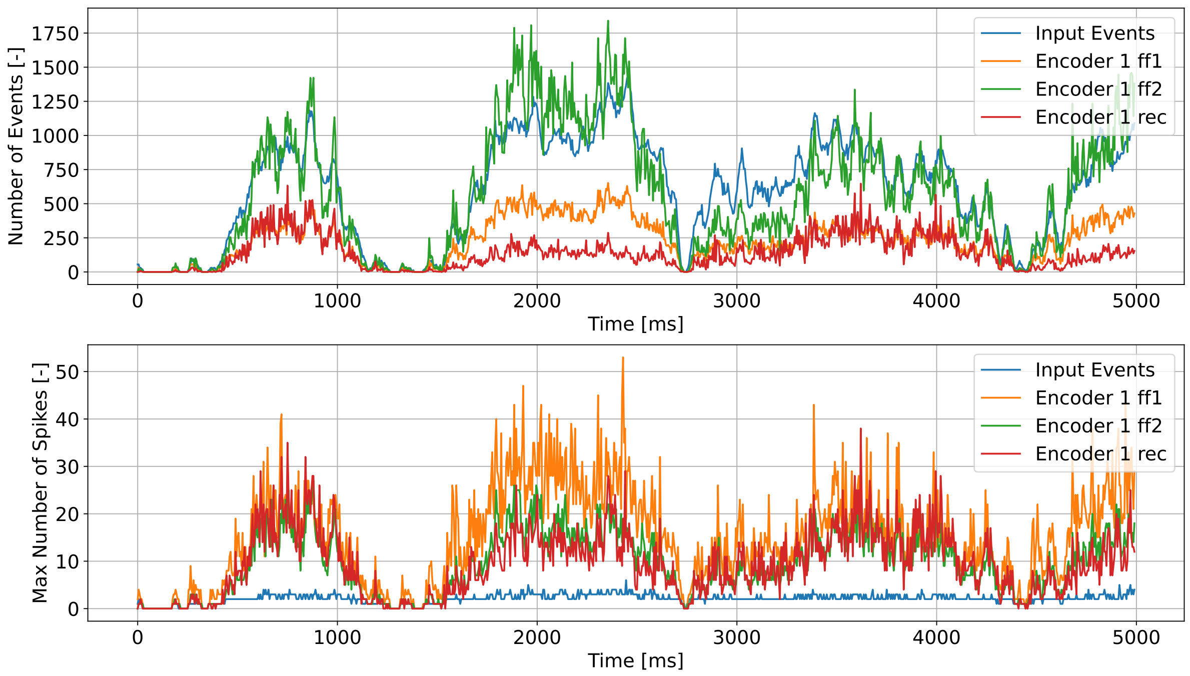


Fig. 25. RNN-2-C after 10 epochs.

Part II

Literature Study

1 Frame-Based Optical Flow

In this chapter, Section 1.1 introduces the concept of optical flow, Section 1.2 discusses the conventional methods for optical flow determination and Section 1.3 treats the deep learning methods.

1.1. General

Optical flow is a concept in computer vision that quantifies the apparent motion of points in an image. It can be described as the vector field defining the motion of the individual pixels in the image. Optical flow draws inspiration from biology and especially from insects and birds. It was discovered that bees exploit optical flow to land by keeping the image velocity constant [2]. Other studies also hypothesized that bees use image motion to estimate distances and avoid obstacles [3, 4].

In the field of robotics, optical flow is used to project the three-dimensional motion of objects, onto the two-dimensional image plane of the visual sensor [5]. Applications of this concept are mainly in the field of navigation and obstacle avoidance for both ground and aerial robots.

1.2. Conventional Methods

Conventional optical flow estimation techniques can be classified in three main categories: intensity-based differential methods, frequency-based methods and correlation-based methods [5].

1.2.1. Intensity-Based Differential Methods

Differential methods use the spatio-temporal derivatives of the intensity function $I(x, y, t)$ of each pixel. Note that the intensity value is also influenced by lighting conditions and not only by the movement of the visual sensor. However, most of the computation algorithms for optical flow assume that changes due to light are negligible compared to changes due to motion [6]. After a time step dt every pixel is displaced in the 2D plane by dx and dy. The velocity vectors describing the displacement are the optical flow components u and v, as shown in Equation 1.1 [5].

$$I(x+dx, y+dy, t+dt) = I(x, y) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \tag{1.1}$$

Because Equation 1.1 does not provide enough information to calculate the flow components at every pixel, algorithms such as the Lucas-Kanade [7] assume that the local flow within a certain region is approximately constant, thus the pixels have the same u and v and the change in intensity is approximately zero (Equation 1.2). In this way, the problem can be solved using least-squared regression (Equation 1.3).

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{1.2}$$

$$\begin{bmatrix} I_x(1,1) & I_y(1,1) \\ I_x(k,l) & I_y(k,l) \\ \vdots & \vdots \\ I_x(n,n) & I_y(n,n) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_t(1,1) \\ I_t(k,l) \\ \vdots \\ I_t(n,n) \end{bmatrix} \tag{1.3}$$

Similarly, the Horn-Shunck algorithm [8] assumes that neighboring points have similar velocities and the velocity field varies smoothly everywhere in the image. To measure the smoothness of the optical flow, the Laplacians of u and v are computed and the squared sum is minimized. Despite being old, the differential methods for optical flow are still being used for navigation tasks such as velocity and height estimation [9, 10, 11, 12, 13].

1.2.2. Frequency-Based Methods

Frequency-based methods use the frequency domain representation of images to estimate motion. By examining the phase shifts and frequency components between corresponding points of consecutive frames, the optical flow vectors can be reconstructed. These methods have proved to be better at extracting optical flow features from random dot patterns, since the resulting energy may be more readily extracted in the frequency domain [5]. Examples of these methods are explained Adelson and Bergen [14], Hegeers [15] and Fleet and Jepson [16]. Despite the efficiency, frequency-based methods are less commonly used in on board applications because of the high computational demand.

1.2.3. Correlation-Based Methods

Correlation-based techniques are based on the idea that corresponding points in consecutive frames of an image sequence will exhibit a high degree of similarity. By comparing a small window in one frame to a region in the following frame using cross-correlation, the optical flow vectors can be derived. The correct displacement will be the one that maximizes the similarities between the two regions. Correlation-based techniques have been used for real-time applications by Kendoul, Fantoni and Nonami [17].

1.3. Deep Learning Methods

In recent years, advancements in deep learning have made Convolutional Neural Networks (CNNs) a popular choice for learning optical flow. Dosovitskiy et al. [18] proposed a CNN trained in supervised manner on ground truth flow data. The networks was able to predict the flow from 2 images with sufficient accuracy. Ranjan et al. [19] introduced a coarse-to-fine approach. At every level of resolution, one image is warped by the current flow estimate to compute the updated flow. A different neural network is trained for every level. Because of the difficulty of acquiring labeled data, unsupervised learning methods have been developed [20, 21, 22]. These involve using prior knowledge and including a loss function to capture the flow features from the data. For instance, the loss functions are brightness constancy loss, gradient constancy loss and spatial smoothness loss. Additional deep learning methods involving SNNs will be later exposed in Chapter 4.

2
Event-Based Cameras

This section will treat Event Based Cameras (EBC), a state-of-the-art biologically-inspired technology in the field of computer vision. Section 2.1 will explain the working principle, Section 2.2 will outline the advantages of these sensors and the different applications in the world of robotics will be treated in Section 2.3.

2.1. Working Principle

Standard vision sensors are limited by their acquisition method. Normal cameras output a sequence of shots at discrete points in time, therefore they are constrained by the frame rate. Additionally, if the scene is not changing significantly, most of the acquired information is redundant and it causes latency and high power consumption. EBCs are dynamic sensors that only respond to brightness changes in the image. Every pixel works individually and outputs a stream of variables called ”events”, representing a positive or negative change in brightness in the image. EBCs working principle is much closer to biological vision systems, where the neurons' detect the changes in the scene and react individually and asynchronously [23]. This acquisition method has the potential to reduce the latency and power consumption significantly.

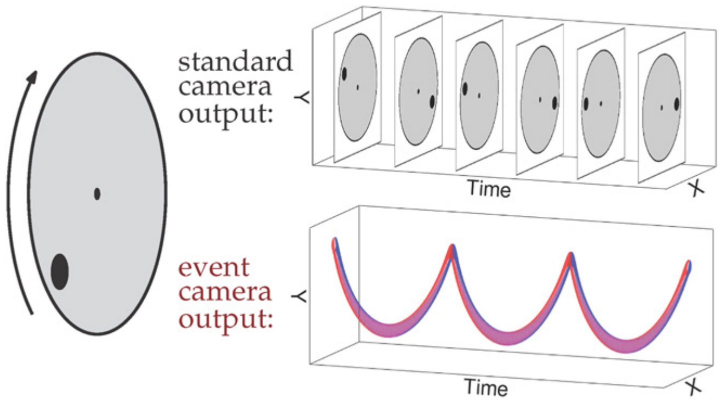


Figure 2.1: EBC compared to standard cameras [24].

When combining EBCs with SNNs, the events enter the network asynchronously and they are represented by the x and y coordinates in the image, the time and the polarity p_k (positive or negative change in brightness). Every time a pixel records an event, it memorizes the log intensity of the brightness change $\Delta L\left(\mathbf{x}_k, t_k\right)$. The pixel will only output another event if the next change in brightness is higher than a threshold C with respect to the saved value. Equation 2.2 shows the condition to be satisfied for an event to occur.

$$\Delta L\left(\mathbf{x}_k, t_k\right) \doteq L\left(\mathbf{x}_k, t_k\right) - L\left(\mathbf{x}_k, t_k - \Delta t_k\right) \tag{2.1}$$

$$\Delta L\left(\mathbf{x}_k, t_k\right) = p_k C \tag{2.2}$$

The number of events produced by EBC is proportional to how fast the image is changing. If the brightness changes due to light variations are not significant, the events can be used to describe the motion of the objects in the scene and therefore the relative motion of the camera itself. The pixel sensors have a bandwidth limit on how many events can be generated, hence some of the high frequency variations is filtered out.

2.2. Advantages

EBC show numerous advantages compared to standard cameras [25]:

- *Power consumption*. When a pixel detects no changes in brightness, it does not produce outputs. As a consequence, the output is minimized and only the relevant information in the image is detected. This leads to a reduction in power consumption.
- *Lower latency*. Figure 2.1 shows the differences between a standard camera and an EBC. For standard cameras, an entire frame is processed at a certain limited frequency. Meanwhile, the individual pixel sensors in the EBC constantly output events asynchronously and independently. As a result, the events are processed as they occur with minimal latency.
- *High temporal resolution*. The event-monitoring system in EBCs is an analog circuit and it reacts extremely fast, even to small changes. In this way, EBC are capable of capturing fast movements accurately without motion blur issues and with high temporal resolution.
- *High dynamic range*. EBCs have extremely high dynamic range, meaning that they can capture small changes in brightness in daylight and moonlight. However, this also means that they are quite sensitive to flickering lights from fluorescent or LED lights. Flickering light produces more unwanted events that need to be filtered out [26].

2.3. Applications

EBCs have been used in combination with SNNs for different applications such as image recognition and classification, image segmentation, object detection and tracking and optical flow estimation (the latter is better explained in Chapter 4).

Because EBCs only work for moving scenery, the main challenge for classification consists in capturing the key features of a moving object. For this reason, EBCs are mainly used to recognize objects from their movement or gestures [27, 28]. Another popular application is the event-based version of the MNIST dataset or N-MNIST [29]. This is mainly used as a benchmark problem to validate SNNs architectures and/or neuromorphic hardware [30, 31].

Image segmentation is especially challenging for EBCs, because events carry very little information about the objects. To overcome this issue, additional information is provided, for instance a known object shape or motion. The difference between the expected features and the obtained ones is estimated and used as a loss function [32, 33].

Regarding object detection and tracking, the main challenge is to identify correspondences between events at different times and in different areas of the image, belonging to the same object. Some examples have a stationary camera, capturing moving objects [34, 35]. Moreover, including additional information about the object can reduce the computations [36].

3

Spiking Neural Networks

This chapter elaborates on SNN and their characteristics. Section 3.1 explains the bio-inspired working principle. Section 3.2 treats the different neuron models. In Section 3.3, the different hyperparameters of SNNs and the training challenges are exposed. Finally, Section 3.4 discusses on using SNNs in combinations with recurrent connections.

3.1. Working Principle

The neurons' mechanism in SNNs is inspired to the working principle of biological neurons. Real neurons consist of multiple parts: dendrites (where the neuron receives information), cell body (where the information travels) and axon (where the neuron transmits information). The connections between the axons and dendrites of different neurons are called synapses and they allow signals to be transmitted all over the nervous system. When a neuron accumulates a certain amount of charge received from other neurons, it generates an action potential that is transmitted through the synaptic connection and influences the charge of its neighboring neurons. This phenomenon is called firing and its occurrence depends on the membrane potential threshold of the neuron.

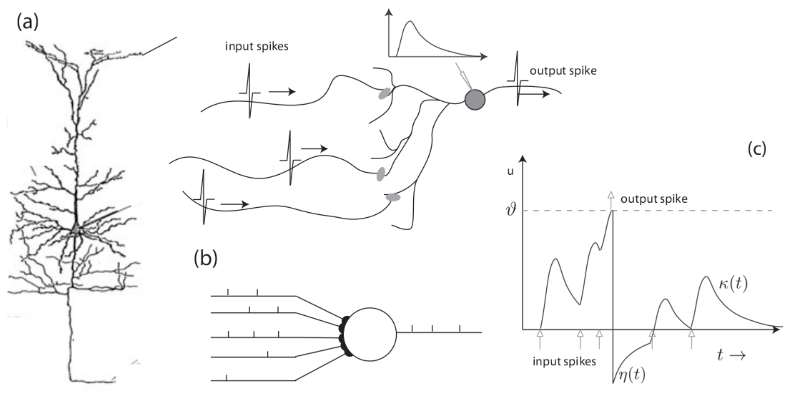


Figure 3.1: (a) Biological neuron. (b) Neuron schematic with input and output spikes. (c) Membrane potential over time. [37]

In SNNs, neurons are treated as dynamical systems that receive discrete electrical pulses called "spikes". The spikes alter the membrane potential of the neuron, therefore modifying its state in time. Each neuron spikes to its neighbours when the membrane potential exceeds a certain threshold. The information inside the network is processed asynchronously and sparsely, thus each neuron in the every layer receives inputs independently [37].

The main advantage of SNNs over ANNs lies in the way the inputs are processed. In SNNs, the incoming spikes are processed asynchronously, as each neuron works independently. Moreover, when used in combination with an EBCs, the information is mostly sparse, since only changes in brightness in the image are fed in the network. This allows for faster computations and therefore lower power consumption and latency [38].

3.2. Neuron Models

Spiking neurons can be modelled in different ways, depending on the application. The most relevant models are here introduced.

3.2.1. Leak Integrate & Fire

The most commonly used neuron model is the Leak Integrate and Fire (LIF). Similarly to real neurons, LIF neurons also have a membrane potential that increases as the neuron receives more spikes. The membrane potential ”leaks”, meaning that it decreases at a certain rate after receiving an input. If the rate of spikes is high enough and the potential increases enough to exceed a certain threshold, the neuron ”fires” a spike to its neighboring neurons. Figure 3.1 shows how the membrane potential increases, when more spikes are received. Equation 3.1 shows the how LIF neurons are modelled in sinabs [39], a SNN simulator. The membrane potential at step $[t+1]$ is given as the sum of the potential $V_{\text{mem}}(t)$ and the sum of all input currents $\sum z(t)$. τ_{mem} is the membrane potential time constant and V_{min} is the lower bound for membrane potential. The leaking behaviour of the neuron is given by the τ_{mem} factor, which represents how fast the membrane potential decays. A larger time constant means a slower response. When $V_{\text{mem}}(t)$ exceeds the threshold, the potential goes back to the reset value.

$$\begin{aligned}
&V_{\text{mem}}(t+1) = \max\left(\alpha V_{\text{mem}}(t) + (1-\alpha)\sum z(t), V_{\text{min}}\right) \\
&\alpha = \exp\left(-1/\tau_{\text{mem}}\right) \\
&\text{if } V_{\text{mem}}(t) >= V_{\text{th}}, \text{ then } V_{\text{mem}} \to V_{\text{reset}}
\end{aligned} \tag{3.1}$$

3.2.2. Integrate & Fire

Another diffused neuron model is Integrate and Fire (IF). The difference with LIF is that the membrane potential does not decay after spiking. The IF behaviour can be described by setting the time constant equal to infinity, so that the α term is cancelled out and Equation 3.2 [39] is obtained. Even without time constant, a leaking behaviour can still be implemented by using biases. A bias is a constant term that is subtracted from the output of a layer at a certain frequency. Bias terms can serve as linear leaks for a certain layer, as they help decreasing the membrane potential and allow for short term memory in the network. An example of how the membrane potential behaves for IF neurons with leaks is shown in Figure 5.5.

$$\begin{aligned}
&V_{\text{mem}}(t+1) = V_{\text{mem}}(t) + \sum z(t) - \text{bias} \\
&\text{if } V_{\text{mem}}(t) >= V_{\text{th}}, \text{ then } V_{\text{mem}} \to V_{\text{reset}}
\end{aligned} \tag{3.2}$$

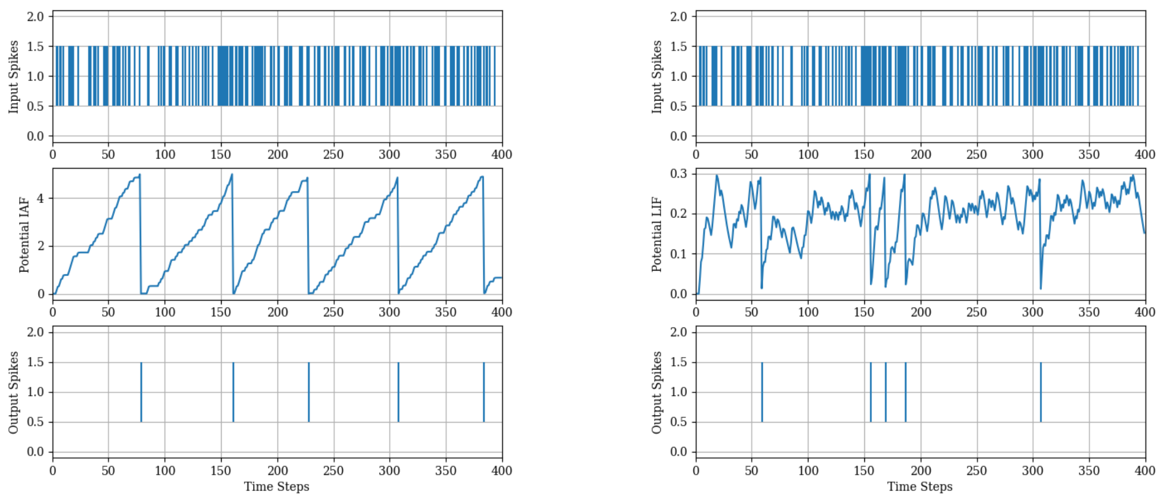


Figure 3.2: IF and LIF membrane potentials after input spikes. The neurons were simulated using sinabs [39]

Figure 3.2 shows the simulated response of a LIF and a IF neuron to the same set of input spikes. As seen from the membrane potential graphs, LIF neurons are able to capture more complex temporal features of the inputs. Other neuron models have been implemented for different applications. Some of the more biologically plausible models, such as the Hodgkin-Huxley model [40], have been used to simulate accurately biological neurons behaviour [41]. However, LIF is commonly the most popular choice for neuromorphic chips, as it presents a complex enough behaviour to be utilized for SNNs tasks [42].

3.3. Hyperparameters & Training

Training SNN represents one of the main challenges in the field of neuromorphic computing. Traditional training methods, such as backpropagation for ANN or backpropagation through time for RNN, cannot be applied to SNN, because of the discontinuous nature of spikes. For simple non-time-dependent problems such as image classification, an ANN can be trained and then converted to SNN. This method is known as *shadow training* and it allows for efficient learning without significant loss of accuracy [43, 44]. However, training a SNN directly allows the network to iterate spiking hyperparameters, such as thresholds, leaks and time constants. In order to do so, other techniques need to be used.

3.3.1. Surrogate Gradients

One of the main challenges of training SNNs is the non-linearity of the spiking inputs, which means that classic backpropagation through time cannot be used directly. For supervised and self-supervised learning problems, gradient-based optimization is required because it allows to adjust the weights and learn according to a loss function. To implement backpropagation, Surrogate Gradients (SG) are introduced as a substitute to the non-linear activation functions. SG are smooth functions that are substituted to the derivatives of a spiking function [45]. In this way, the network can be trained with gradient-based optimization algorithms. Zenke et al. [46, 45] studied the robustness of surrogate gradient learning and showed that SG can achieve performances comparable to ANNs.

The first algorithm to perform backpropagation on SNN is *Spikeprop* by Bohte et al. [47], which was applied to the classic XOR problem. The aim of *Spikeprop* is to associate certain target firing times to certain input patterns and then compare the actual firing time. Following, the *Slayer* network proposed by Shrestha et al. [48] included a loss function based on both spiking times and the number of output spikes for a given interval.

3.3.2. Learnable Hyperparameters

Besides the different neuron models and surrogate functions, other hyperparameters can be chosen to shape the response of the input received by the neuron and the output produced by it. Changing parameters such as the threshold, the bias, the time constant and the reset function has an impact on the learning of the network that shall be discussed.

By increasing the time constant, the response is stretched out in the time direction. Having a larger time constant implies having a slower charge of the potential and a slower decay, as shown in Figure 3.3. In biological brains the membrane parameters vary across different regions [49]. Incorporating variable and learnable time constants, can make the SNN robust to a wider range of inputs and temporal features can be better captured. Wei Fang et al. [50] showed that incorporating learnable time constants makes the network less sensitive to initial conditions and accelerates the training. In a similar way, SNN can also benefit from learnable thresholds. Having a lower threshold leads to a more frequently spiking neuron. Siqi Wang et al. [51] showed that varying the threshold makes the network converge faster.

Another parameter to be specified is the reset function for the neurons. It can be either hard or soft. The first one brings the function back to zero after spiking, while the second one subtracts the threshold value from the potential. Implementing a hard reset function can be helpful for cancelling the error from surrogate gradients, since the potential is reset to zero and the error does not build up [45].

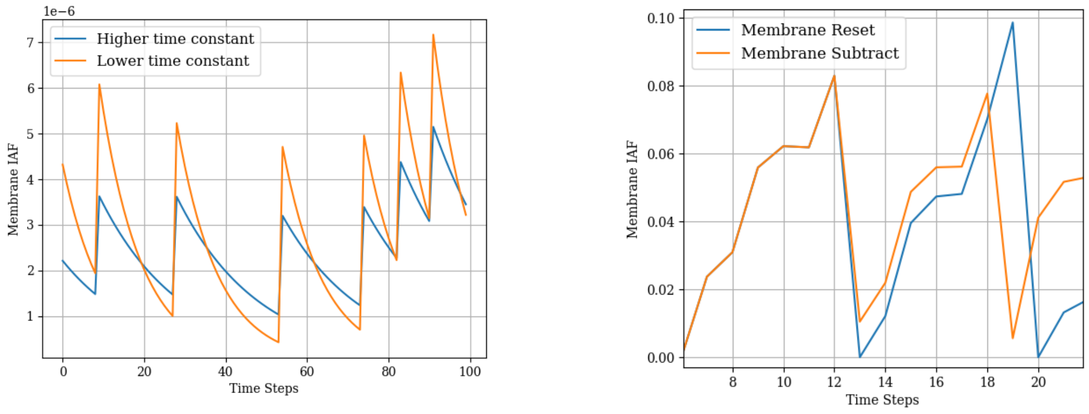


Figure 3.3: Effect of time constant and reset function on membrane potential. This image was generated using sinabs [39].

3.4. Recurrency in Spiking Neural Networks

One of the main characteristics of SNN is that the membrane potential serves as a short term memory, which allows it to perform time integration. The input spikes to the network can either be fed directly or accumulated over a time interval and fed in the network together. In either case, the network needs to extract temporal features from the inputs to compute the outputs. Another family of neural networks with an internal memory mechanism is Recurrent Neural Networks (RNN). Recurrent connections between layers allow the information to be also fed back inside the network, creating a loop-like structure. These architectures are especially suitable to process sequences of data.

In the work of F. Paredes-Vallés, J. J. Hagenaars and G. de Croon [52], both ANN and SNN with recurrent connections were developed and compared to the non-recurrent architectures of Zhu et al. [53] and [54]. The recurrent networks were able to produce high quality event-based optical flow estimates. This shows the advantage of including recurrent connections in SNN.

It is important to point out that SNN already have a form of intrinsic recurrence. This is because the state of each neuron depends on both the spike inputs and the previous states [45]. However, when dealing with IF neurons with only bias, having additional recurrent connections could potentially lead to better performance.

4 Event-Based Optical Flow

This chapter treats event-based optical flow determination. Both supervised learning (Section 4.1) and self-supervised learning methods (Section 4.2) are treated. Section 4.3 will treat previously used devices for estimating optical flow.

4.1. Supervised Learning

Supervised learning algorithms require labelled data or targets for training. The difference between the output of the network and the target is calculated and used to optimize the weights with backpropagation [37]. Supervised learning of SNN has been widely used for classification tasks.

Regarding the field of ego-motion estimation, Gehrig et al. [55] trained a SNN in supervised manner to perform temporal regression on angular velocity data in all three axis. The loss function used is the time-integral over the euclidean distance between the predicted angular velocity and ground truth angular velocity (Equation 4.1). This work proved that it is possible to train SNN on continuous-time regression tasks.

$$L = \frac{1}{T_1 - T_0} \int_{T_0}^{T_1} \sqrt{\boldsymbol{e}(t)^\top \boldsymbol{e}(t)} \mathrm{d}t \tag{4.1}$$

In a similar way, SNN can be trained with ground truth optical flow as a target. Javier Cuadrado et al. [**10.3389/fnins.2023.1160034**] used a loss function to calculate the error between ground truth optical flow components and the output of the network. The first term of the loss function calculates the magnitude error (Equation 4.2), while the second one calculates the direction error (Equation 4.3).

$$L_{\text{mod}} = \frac{\sum^{N_{\text{pixels}}} \sqrt{(\text{ pred }_x - gt_x)^2 + (\text{ pred }_y - gt_y)^2}}{N_{\text{pixels}}} \tag{4.2}$$

$$L_{\text{ang}} = \frac{\sum^{N_{\text{pixels}}} a\cos(c\theta)}{N_{\text{pixels}}} \quad c\theta = \frac{\overrightarrow{gt} \cdot \overrightarrow{\text{pred}} + \epsilon}{|\overrightarrow{gt}| \cdot |\overrightarrow{\text{pred}}| + \epsilon} \tag{4.3}$$

4.2. Self-Supervised Learning

Supervised learning methods for SNN have developed significantly in the latest years. However, one persisting issue is the acquiring labelled data. Self-supervised methods were introduced to overcome this issue and retrieve information from the input data directly.

Yu et al. [21] proposed a self-supervised method to learn optic flow that uses a weighted average of two loss functions, the photometric and the smoothness function. The method consists in taking two temporarily adjacent images and an initial estimate of the flow (u, v). By applying optic flow vector to the second image, the inverse warped image can be computed. This warped image is then compared to the first image to calculate the error and correct the flow. The photometric loss (Equation 4.4) is calculating the difference between the image at time t and the warped image. Note that with this self-supervised method, no labeling of the data is required.

Additionally, the second loss function (Equation 4.5) encourages smoothness in the estimated optical flow field. It is based on the assumption that neighboring pixels should have similar flow values. The smoothness function is meant to regularize the output flow. The total cost function is a weighted sum of the two functions (Equation 4.6). Note that the function $\rho(x)$ is the Charbonnier function, commonly used in optic flow estimation because of its robustness to outliers. An overview of the method is shown in Figure 4.1.

$$\ell_{\text{photometric}}\ (u, v; I_t, I_{t+1}) = \sum_{x,y} \rho\left(I_t(x,y) - I_{t+1}\left(x + u(x,y), y + v(x,y)\right)\right) \tag{4.4}$$

$$\ell_{\text{smoothness}}(u, v) = \sum_{x,y} \sum_{i,j \in \mathcal{N}(x,y)} \rho(u(x,y) - u(i,j)) + \rho(v(x,y) - v(i,j)) \tag{4.5}$$

$$L_{\text{total}} = \ell_{\text{photometric}} + \lambda \ell_{\text{smoothness}} \quad \rho(x) = \left(x^2 + \epsilon^2\right)^{\alpha} \tag{4.6}$$

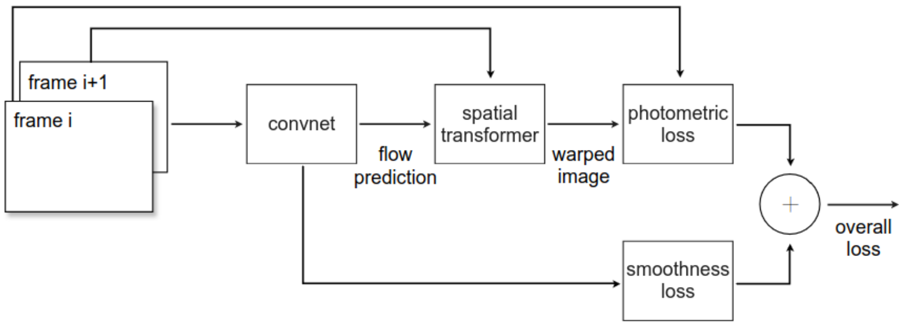


Figure 4.1: Overview of the self-supervised approach [21]

Zhu et al. [56] used the loss function in Equation 4.6 in a self-supervised deep learning pipeline for optic flow estimation. To train the network, the input events from an event based camera and the corresponding grayscale images from the same camera were used. During training, the events are fed into the network. The grayscale images before and after the event time window are used to calculate the loss function.

A different approach was proposed by Gallego et al. [57], which solely relied on the events stream and did not require grayscale images. Consider a set of positive and negative events in a certain spatio-temporal neighborhood as in Figure 4.2. The events triggered by the same moving edges are expected to follow the same trajectories. The translational displacement of the pixels can be described by Equation 4.7, where (u, v) are the flow components.

$$\begin{pmatrix} x_i' \\ y_i' \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \end{pmatrix} + (t - t_i') \begin{pmatrix} u(x_i, y_i) \\ v(x_i, y_i) \end{pmatrix} \tag{4.7}$$

Calculating the correct flow can be interpreted as finding the best fitting trajectory that passes through the events generated by the same moving edge. Figure 4.2 (left) shows events on a x-y plane over time. The horizontal lines connect events from the same edges at different times.

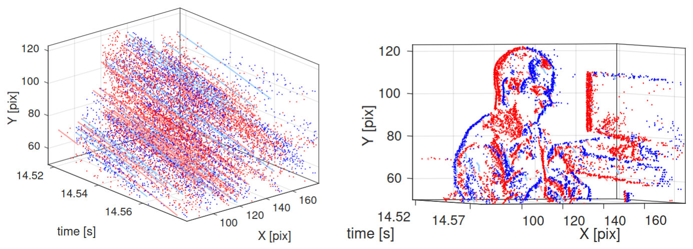


Figure 4.2: Events trajectories (left), Image with warped events (right) [57].

By summing the polarities along a certain trajectory and computing the variances for each sum, the distance of each point from the trajectory can be found. To sum the events, they first need to be warped, meaning that they need to be transposed to a reference time using the flow components estimation. The contrast maximization can also be seen as a function to minimize motion blur. The more correct the flow is, the less blurry the image will be. Figure 6.3 (left) shows the contrast maximization function plotted over the optic flow components θ.

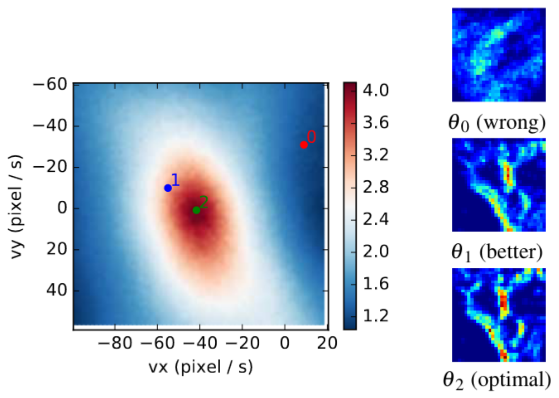


Figure 4.3: Contrast as a function of optic flow components (left). Warped events (right) [57].

Zhu et al. [54], proposed a similar approach but based on the loss function of Mitrokhin et al. [58]. To compute the contrast maximization loss function, first the events are separated by polarity. Following, the average timestamp image is generated at each pixel for each polarity.

$$T_{p'}(\mathbf{x};\mathbf{u}|t_{\text{ref}}) = \frac{\sum_j \kappa(x - x'_j)\kappa(y - y'_j)t_j}{\sum_j \kappa(x - x'_j)\kappa(y - y'_j) + \epsilon} \quad \kappa(a) \quad = \max(0, 1 - |a|) \tag{4.8}$$

The loss function proposed by Zhu et al. [54] has 2 components. The first one aims to minimize the sum of squares of the average timestamp at each pixel for each polarity T_+, T_- (Equation 4.8). To make the loss function convex, in Paredes-Vallés et al. [52] the contrast maximization function was scaled with the number of pixels with at least one warped event (Equation 4.9). The second term of the total loss function is a local smoothness regulator as in Equation 4.5.

$$\ell_{\text{contrast}}(t_{\text{ref}}) = \frac{\sum_{\mathbf{x};\mathbf{u}} T_+(\mathbf{x};\mathbf{u}|t_{\text{ref}})^2 + T_-(\mathbf{x};\mathbf{u}|t_{\text{ref}})^2}{\sum_{\mathbf{x}}[n(\mathbf{x}') > 0] + \epsilon} \tag{4.9}$$

4.3. Onboard Applications for MAV

SNN require specific hardware that can perform neuromorphic computing with artificial neurons and synapses. Neuromorphic chips have been used for a variety of applications in the field of robotics [59, 60, 61].

4.3.1. Intel Loihi

Regarding micro-air vehicles applications, the Loihi chip produced by Intel [62], has been used on board of drone by Stagsted et al. [63]. This work proposed a SNN based PID controller for a drone constrained to a single degree of freedom. Dupeyroux et al. [64] used the same chip for vertical motion control and autonomous landing. The proposed SNN calculates the thrust command based on the divergence of the ventral optic flow field. Following, in the other work by Paredes-Vallés et al. [65], the Loihi chip was used in a fully neuromorphic pipeline to control the drone motion using optical flow. The pipeline takes EBC data and outputs low-level control commands to perform hovering, landing and lateral maneuvers. To reduce the computational effort, the network is used on the four corners, instead of the full image.

5 Neuromorphic Hardware & Software

This chapter will treat neuromorphic hardware (Section 6.4) and software (Section 6.3) used for the master thesis project.

5.1. Hardware

Synsense is another cutting-edge company in neuromorphic computing field [1]. Synsense develops both software and hardware tools for implementation of SNN. For this project, a speck2e from Synsense will be used [66]. This chip combines an EBC with an event-processing chip into one board. Other Synsense chips with similar interfaces have already been used for classification problems. To our knowledge Synsense chips have not been used yet for ego-motion estimation. The main applications of speck2e include gesture control, smart tracking, fall detection, lane detection, sign recognition, driver attention tracking, obstacle detection and object tracking [66]. The Speck2e chip architecture is described by four different components (Figure 5.1, [67]):

- **Convolutional Cores**
  The chip has a maximum of 9 convolutional processing layers and each one has a leak operation. The maximum neuron capacity is 0.32 million. The neurons have a read-add-check spike-write operation (Figure 5.2), so that the incoming stream of events is read and summed to the already present potential.
- **Dynamic Visual Sensor (DVS)**
  The DVS consists of 128x128 individually operating event-based vision pixels. The pixels encode the incoming photon flux temporally on a logarithmic intensity scale.
- **Pre-Processing Block**
  The pre-processing core after the DVS can be used to filter the polarities, mirror/rotate the image or cutting the image to define a region of interest out of the 128x128 pixels. This core can be connected to the DVS or also process events from an external source.
- **Readout Core** The readout core transforms the incoming stream of events into readable data. It can output 16 different classes or moving averages, which can be calculated on different average lengths.

5.2. Software

Sinabs [39] and Samna [68] are the 2 main Python packages developed to interact with Synsense chips. The first one is a PyTorch-based library for designing and testing SNN architectures, while the latter is a C++ tool with Python extension to interact with the Synsense devices.

Sinabs provides functions to simulate IF, LIF, Exponential Leak and Adaptive Leak neurons. The networks can be built with `torch.nn.Sequential` and uploaded together with the configuration. They can be either SNNs or ANNs converted to spiking layers.

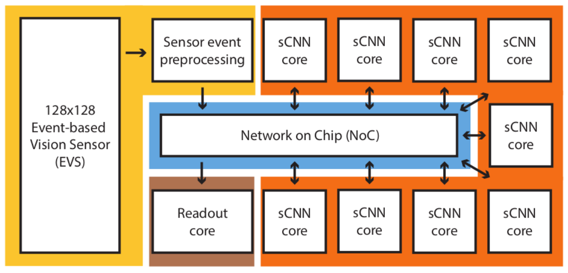


Figure 5.1: Speck Architecture [67]

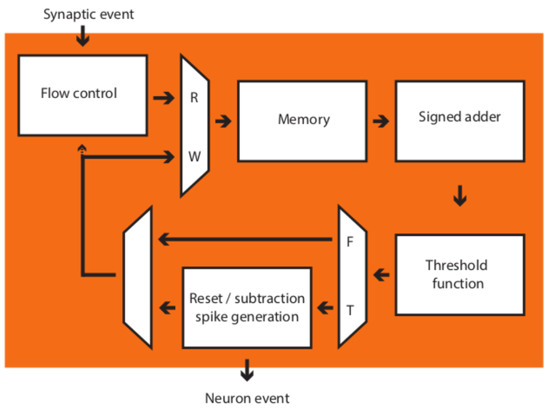


Figure 5.2: Neuron compute unit [67].

The Samna library provides all the necessary tools to interact and edit the configuration of the devices. In order to connect to a device, there is 3 parts to set up:

- *Devkit Configuration*. It is an object containing the configuration for the CNN layer, the DVS layer and the Readout layer. The configuration encloses all the details not only about network architecture, but also about the DVS and pre-processing layer.
- *Samna Graph*. It defines the event stream flow inside the chip with a system of filter nodes (Figure 5.3). Nodes can be *sources* (only outputting event streams) or *sinks* (only receiving event streams). The input node can be used to write custom made events into the chip, instead of using the DVS output. The output node is used to read data from the chip.
- *Visualizer* (optional). It can be used show the input signal, the output of the network or the real-time power consumption. The output of the DVS is 128x128 pixels (Figure 5.4).

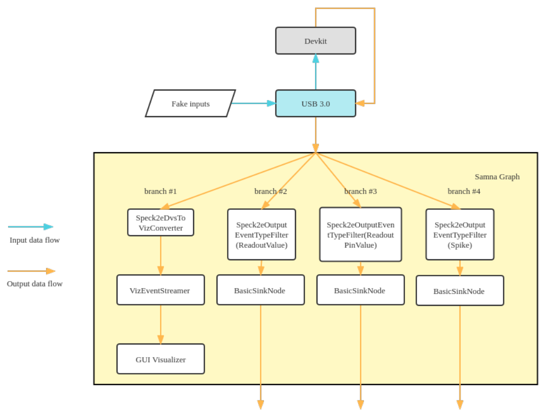


Figure 5.3: Samna graph [68].

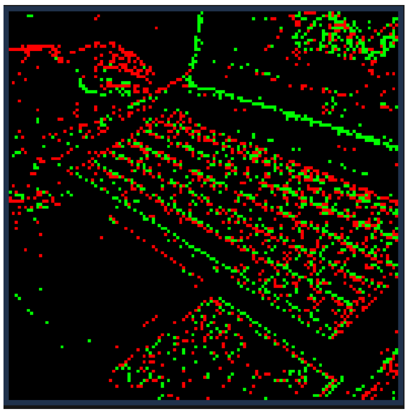


Figure 5.4: Speck2e visualizer. Red pixels are negative changes in brightness and green are positive. The image was generated using the visualizer provided by Samna [68].

5.2.1. Editing & Uploading Devkit Configuration

Uploading a configuration on a Synsense device can be done by converting the network to a `dynapcnn` object and then calling the method `.to(device_name)`. In this way, a simple samna graph with input-writing and output-reading is built. If a more complex configuration is required, the `samna_config` object needs to be modified.

Inside the configuration there is a list of `CNNLayerConfig` objects, where each element contains all the properties for a convolutional core of the chip. For each convolutional layer, one can specify the high and low thresholds, padding, stride, kernel size and dimensions. In terms of neuron properties, the object contains the initial neuron values, weights and biases. Additionally the `return_to_zero` boolean characterizes the reset function of the neuron, thus if it is set to true, the potential returns to zero after spiking.

The chip also contains a slow clock with a certain frequency to be specified. By setting the `leak_enable` to True, the bias terms are subtracted from the potential every slow clock cycle. For instance if the slow clock frequency is set to 1, every second the bias will be subtracted from the potential. This mechanism is what allows for a leaky behaviour of the neuron. In the documentation it is specified that including bias might significantly increase the power consumption

Finally, each layer has 2 possible destinations that can be enabled. By default, each layer forwards to the next in order, however one can specify the second destination to be different. If the second destination is set to the same index of the layer, a recurrent connection is built. Note that this step needs to be done manually, as it is not possible to build a spiking recurrent neural network with sinabs and automatically convert it to a speck-compatible configuration. The recurrent connections have to be redefined one by one before uploading the configuration with `.apply_configuration()`.

To show an example, a simple SNN was built and uploaded to the speck2e. A set of input spikes was written and the resulting potential of one of the neurons was read. Figure 5.5 shows the neuron membrane potential, the input and the output spikes. The bias effect can be observed for instance at approximately 3.5 sec, when the potential decreases linearly. When the neuron receives spikes but does not exceeds the threshold and no more spikes arrive, a clock cycle passes and the bias is subtracted. With the help of bias, a short-term memory effect can be obtained. As seen in Figure 5.5, the potential increases after every spike, but it exceeds the threshold only when the rate of spikes is sufficiently high.

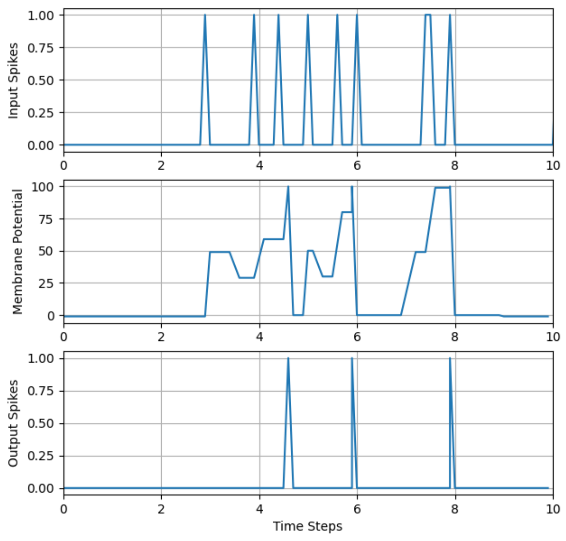


Figure 5.5: Input spikes, membrane potential and output spikes of a neuron over time.

5.2.2. Synaptic Operations

In the Sinabs [39] documentation, it says that Synsense devices have a bandwidth limit on synaptic operations (synops). The number of synops shall not exceed a few millions of neurons per layer or the layer reaches saturation and it is not able to output spikes. Using the `sinabs.synopcounter.SNNAnalyzer` tool, the number of synops can be estimated for a network operation. Because of the saturation limit of the layers, it might be useful to include the maximum number of synops in the loss function when training for a certain task. Althogh the model might work in software, if the number of synops is excessive, the same network will perform poorly on hardware.

Part III

Preliminary Evaluation of Integrate & Fire Neurons for Optical Flow

6 Methodology

This part of the report includes the design process of a speck2e-compatible network that is able to determine optical flow from an event-based dataset. In this chapter, the methodology of the design process is documented. section 6.1 will give an overview of the starting point for the network's architecture. section 6.2 explains the two different datasets that will be used and their purpose. section 6.3 will explain the software framework used of the project, including the different training hyperparameters and tools used to interact with Synsense devices. section 6.4 will outline the hardware implementation part of the project and the network constraints provided by the speck2e.

6.1. Network Architecture

Designing a speck2e-compatible architecture for estimating optical flow requires an iterative process. An initial network configuration will be used and then updated step by step, in order to be within the hardware constraints. The initial network architecture is the one described in Paredes-Vallés et al. [65] and it consists in three *encoders*, one pooling layer and one prediction layer. Each encoder has a forward layer with kernel size (3,3), stride (2,2), padding (1,1) and an increasing number of output channels (e.g. from 2 to 32 in the first encoder). Moreover, a recurrent layer with kernel size (3,3), stride (1,1) and padding (0,0) is used to take into account the previous state. After the three encoders, a convolutional pooling layer with kernel size (23,23) is used to reduce the output size from 23 to 1. Finally, a non-spiking layer is used to predict the 8 components of optical flow, which represent the vertical and horizontal components for each of the 4 sections of the image. A diagram of the network can be seen in Figure 6.1.

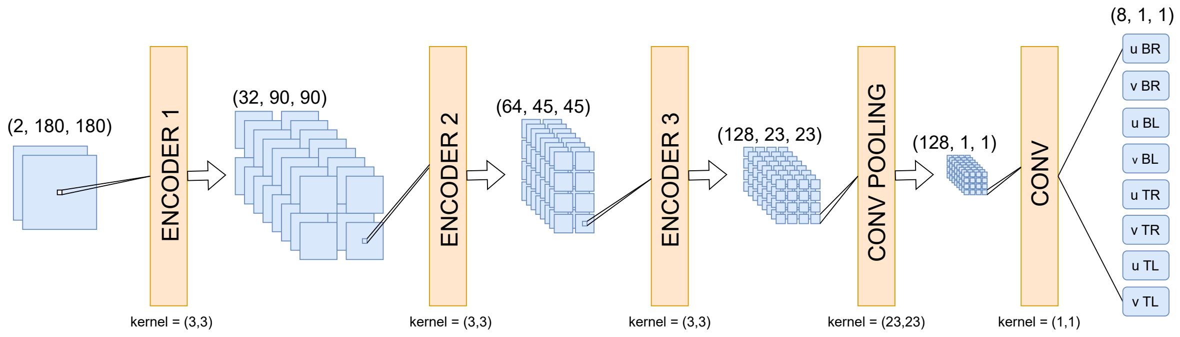


Figure 6.1: Network architecture with LIF neurons. The outputs are the u and v components of the flow in the bottom right (BR), bottom left (BL), top right (TR) and top left (TL) sections.

The network configuration is described by high and low level hyperparameters. The first ones describe the general structure, such as the number of layers, input and output channels and recurrency mechanisms, while the latter ones are related to individual neurons characteristics, for instance thresholds, leaks and/or biases. During the iterative process, the hyperparameters will be updated to design a new architecture that is able to fulfill the task and can be hosted on the speck2e.

6.2. Training & Testing Datasets

The network will be trained on two main datasets. The first one will be addressed as *SR dataset* and it includes 40 minutes of event data, which are split into 25 minutes for training and 15 for testing. This dataset includes translational and rotational motion in multiple directions and at different speeds. The SR dataset will be used to quantify the network's accuracy with respect to ground truth data and assess its performance. The second dataset is provided by the University of Zurich and it will be denoted as *Davis dataset* [69]. This dataset includes two main video sequences of 55 seconds each. It contains translational and rotational motion, however due to its limited size, it is not as complex and diversified as the SR dataset. The Davis dataset will be used to train the network faster and check if it is learning properly. This will allow to save computational time and make conclusions more promptly.

The data is stored in h5 files, where each event is saved with its timestamp, coordinates and polarity. Before training the network, the data is split in sequences of 5 seconds. Each sequence is divided in frames which contain events accumulated over time windows of 5 milliseconds. Asynchronous training of SNN is commonly used to avoid running the network at too high frequencies and save computational power and/or time. When uploaded on the neuromorphic device, the network on the chip will operate at a much higher rate.

6.3. Software

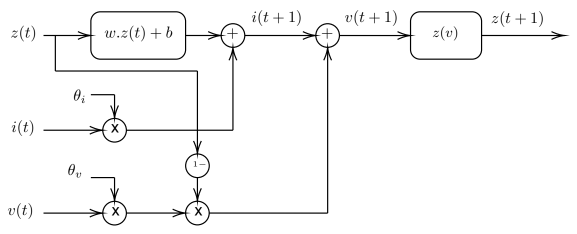
The network will be trained in self-supervised manner using PyTorch built modules and neuron models. These will be substituted later with sinabs modules, as they are more representative of the behaviour of Synsense devices.

6.3.1. PyTorch Framework

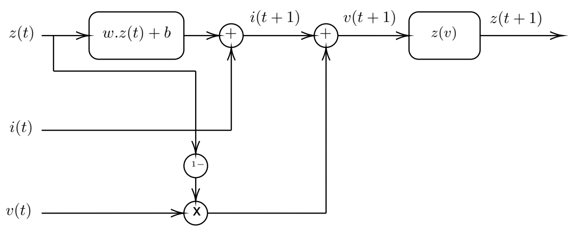
For training the network a PyTorch framework is used. This includes neuron models, loss function calculation and performance testing tools.

Neuron Models

Each layer in the SNN can be visualized as a convolutional layer plus a spiking activation function. The layer contains weights and biases, which define how much the input influences the potential of the neuron and therefore its output. The spiking function outputs a 1 or a 0 at every step, depending whether the potential exceeded the threshold or not. A schematic of the neurons is provided in Figure 6.2a and 6.2b.



(a) LIF neuron diagram.



(b) IF neuron diagram

$z(t)$ is the input spikes vector, $i(t)$ and $v(t)$ are the neurons' current and potential values. θ_i and θ_v stand for the current and potential leak respectively and they are multiplied by the previous state values. The leak represent how much information of the previous state is preserved in the following step. A higher leak value means that the current or potential decreases more slowly and more information is maintained through time. Additionally, the potential is multiplied by a reset term $(1-z)$, so that if a neuron spiked in the previous step ($z=1$), the potential is reset to zero.

The spikes are fed in the convolutional layer, where they are multiplied by the weights and summed with the biases. The output of the convolutional layer represents the change in current due to the spikes, which is added to the previous state current. The new current $i(t+1)$ is summed with the previous potential to compute the new potential $v(t+1)$. Finally, the spiking function $z(v)$ checks if the potential is higher than the threshold and transmits spikes to the following layer.

Training Settings

The network is trained in backpropagation through time (BPTT). The event data is fed in the network in batches of 16 sequences for the SR dataset and 8 for the Davis dataset. A single batch contains a sequence of 5 seconds, which is fed in the network frame by frame and each frame contains the events accumulated over 5 milliseconds. The network is trained over 100 epochs, with Adam optimizer and learning rate of 0.001. Moreover, a data augmentation technique is used on the dataset. The individual frames of events are flipped horizontally, vertically and the polarities are switched.

Self-Supervised Loss Function

As explained in chapter 4, the loss function for self-supervised learning is composed by two terms: *contrast maximization* and *flow smoothing*. The first term is extracting information from the event data to find the best fitting flow vectors, while the second one regularizes the smoothness of the flow.

After 5 forward passes, the loss term is computed and the gradients are backpropagated to update the weights. For each of the 5 frames, the events are transposed in time using the last flow vectors estimation. Essentially, the 5 frames are used to reconstruct the events at a reference time. The 5 reconstructions are then summed together and, if the flow estimation is correct, the pixels should be aligning without any blur. In reality, the first optical flow estimates will not be correct and will therefore result in a blurred reconstructed image (Figure 6.3). In order to measure this blur and minimize it, the density of events per pixels is calculated. If the flow estimation is accurate, the reconstructed events will be aligning on the same pixel.

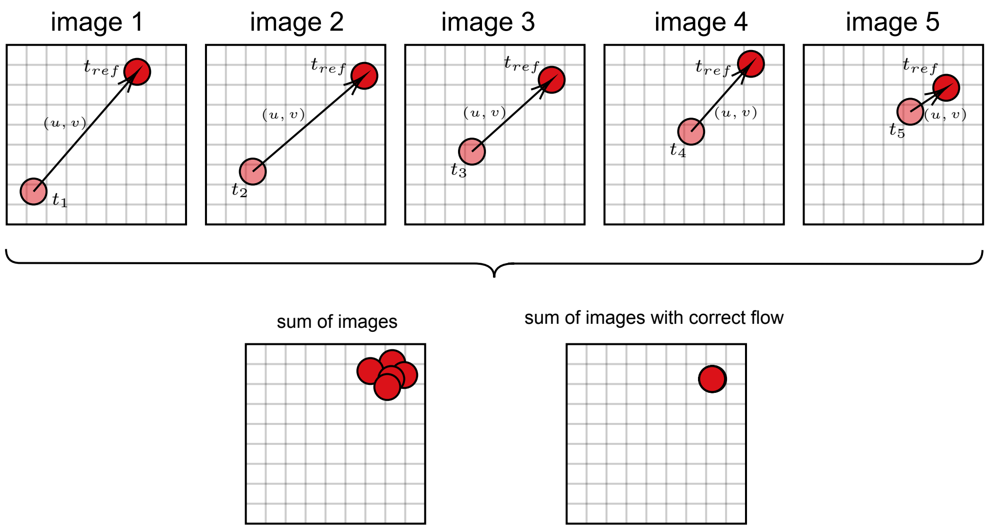


Figure 6.3: Contrast maximization computation scheme.

6.3.2. Sinabs & Samna

Sinabs and Samna are python packages provided by Synsense to train and test chip-compatible architectures. Sinabs is used to simulate and train SNN, while Samna is a C++ based package to interact and upload configurations on the Synsense devices.

Spiking Activation Function

The IF spiking functions will be simulated using the `IAFSqueeze` sinabs module, which works as an activation function for a convolutional layer. The internal state of the module is described only by the potential value, since there are no leak terms and the current is summed directly to the previous state. The following parameters can be defined:

- `spike_threshold` (*Tensor*) - Maximum allowed potential value (set to 1.0 by default).

- `spike_fn` (*Callable*) - Function defining the spiking output. If set to `MultiSpike`, a neuron will be able to produce multiple spikes in given time step. If set to `SingleSpike`, a neuron will produce at most one spike per time step.
- `reset_fn` (*Callable*) - Function defining the reset phase of the potential. If set to `MembraneSubtract()`, the threshold value is subtracted from the potential after spiking. If set to `MembraneReset()`, the potential returns to zero after spiking.
- `surrogate_grad_fn` (*Callable*) - Function used to define the gradients during training. For `SingleSpike`, it should be set to `SingleExponential`, while for `MultiSpike`, it should be set to `PeriodicExponential`.
- `min_v_mem` (*Tensor*) - Lower bound for membrane potential, clipped at every time step.

For direct training of IF networks, a multi-spike behaviour is preferable, as it describes better what happens on the chip. Within a given time step, a neuron will output multiple spikes sequentially if the synaptic input is an integer times higher than the threshold. During training it is important to monitor the spiking activity, since convolutional core on the chip has a limit on the number of synaptic operations per second (synops/s). The synops are calculated using the sinabs function `SNNAnalyzer`. To make sure the limit is not exceeded, the number of synops/s per layer can be included in the loss function during training.

Network Configuration
Before uploading the trained network to the device, the weights, biases and thresholds need to be discretized using the `DynapcnnNetwork` function. This is done because the chip has a limited resolution to represent the its parameters (R-speck-6, R-speck-7). Note that when the parameters are quantized, the quantization range is unique per layer, because the relation between weights and threshold is meaningful for the synaptic output of a layer.

Before uploading a trained model on the chip, a configuration object `CnnLayerConfig` is created. This contains all the relevant information of the network including weights, biases, thresholds, initial neurons state, reset mechanism type, layers dimensions and destinations.

6.4. Hardware

The device used for the experiments is the speck2e from Synsense, an event-driven neuromorphic chip with fully asynchronous digital circuit and integrated Dynamic Vision Sensor (DVS). Moreover, it is a low-power (< 5 mW) and low-latency (<50 ms) system. The speck2e circuit is able to support large-scale SNN for various computer vision tasks, such as sign recognition, smart tracking and obstacle detection. The network requirements for the speck2e are listed in Table 6.1.

Table 6.1: Speck2e requirements.

Requirement ID	Description
R-speck-1	Maximum of 9 convolutional layers (including pooling).
R-speck-2	Maximum number of neurons: 32k.
R-speck-3	Maximum input dimension: 128x128
R-speck-4	Maximum feature output size: 64x64
R-speck-5	Maximum feature number: 1024
R-speck-6	Weight resolution: 8 bit
R-speck-7	Neuron state resolution: 16 bit
R-speck-8	Maximum kernel size: 16x16
R-speck-9	Stride: {1, 2, 4, 8} independent in X/Y
R-speck-10	Padding: [0..7] independent in X/Y
R-speck-11	Pooling: 1:1, 1:2, 1:4
R-speck-12	Fanout: 2
R-speck-13	Frequency: 1 GHz
R-speck-14	Synaptic Operations limit per core: 10 millions synops/s
R-speck-15	Maximum number of channels in readout layer: 15.

6.5. Recurrency Types

The initial architecture in Figure 6.1, has internal recurrency, which means that the membrane potential is first summed with the synaptic input from the recurrent layer and then passed through the spiking function. This is done so that the influence of the recurrent layer is included in the potential already and the spikes are only used to communicate between the encoders. However, the recurrency available on the speck2e is external, which means that every core consists in a convolutional layer and a spiking layer. The cores are connected and send information to each other with spikes.

The network will be tested with different external recurrency mechanisms both in the PyTorch framework and on the chip. Samna does not directly support recurrent architectures, which means that the mapping of the layers' destinations will be done manually. According to R-speck-1, each layer can be connected to maximum 2 other layers. This allows for two main options for recurrency mechanisms.

6.5.1. Recurrency type S

The first recurrency type involves 2 layers. The first one increases the number of channels to double, while the second one has equal number of input and output channels. The second layer output is fed back and summed with the new incoming input. Note that the number of input and output channels is the same in the recurrent layer.

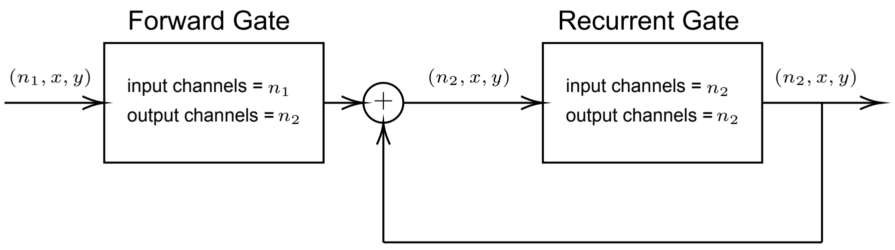


Figure 6.4: Recurrency type S scheme.

```python
def forward(self, input_, prev_state):

    # FIRST GATE
    # rescale input if spiking
    if self.spiking:
        input_ = input_ * self.scale_ff

    # forward pass
    ff = self.ff(input_)
    ff = self.activation_ff(ff)

    # generate empty prev_state, if None is provided
    if prev_state is None:
        batch, _, height, width = ff.shape
        state_shape = (batch, self.hidden_size, height, width)
        prev_state = torch.zeros(*state_shape, dtype=ff.dtype, device=ff.device)

    # SECOND GATE
    input_rec = ff + prev_state
    # rescale input if spiking
    if self.spiking:
        input_rec = input_rec * self.scale_rec

    # forward pass
    out  = self.rec(input_rec)
    out = self.activation_rec(out)

    return out, out
```

Figure 6.5: Code implementation of recurrency type S.

6.5.2. Recurrency type C

This other type of recurrency requires 3 layers. This system works with concatenation of outputs. The output of the forward layer is concatenated with the output of the recurrent layer and then fed in the second forward layer. Note that the second forward layer increases the number of channels to n_3, while the recurrent layer decreases it again to n_2, in order to concatenate it with the new input. This means that n_3 shall always be twice as large as n_2, so that half of the inputs to the second forward layer are coming from the new input and the other half from the previous input.

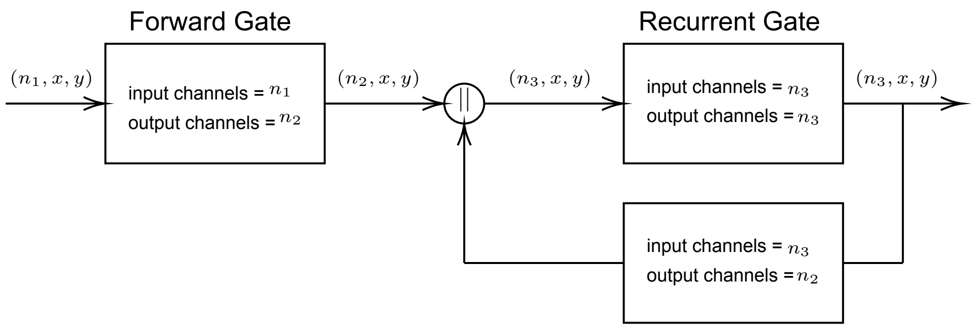


Figure 6.6: Recurrency type C scheme.

```python
def forward(self, input_, prev_state):
    # FIRST GATE
    if self.spiking:
        input_ = input_ * self.scale_ff1
    ff1 = self.activation_ff1(self.ff1(input_))

    # generate empty prev_state, if None is provided
    if prev_state is None:
        batch, _, height, width = ff1.shape
        state_shape = (batch, self.hidden_size, height, width)
        prev_state = torch.zeros(*state_shape, dtype=ff1.dtype, device=ff1.device)

    concat = torch.cat([ff1, prev_state], dim=1)

    # SECOND GATE
    if self.spiking:
        concat = concat * self.scale_ff2
    out = self.activation_ff2(self.ff2(concat))

    # RECURRENT GATE
    rec_input = out.clone()
    if self.spiking:
        rec_input = out * self.scale_rec
    state = self.activation_rec(self.rec(rec_input))

    return out, state
```

Figure 6.7: Code implementation of recurrency type C.

6.6. Network Naming Convention

To address the different network types and state their main characteristics, the following naming convention is used: *network type - number of encoders - recurrency type*. For instance a non-spiking convolutional neural network with 2 encoders and recurrency type S will be addressed as RNN-2-A. The networks will all be convolutional and the different tested types are the following:

- LIF = Leaky Integrate & Fire Spiking Convolutional Neural Network
- IF = Integrate and Fire Spiking Convolutional Neural Network
- RNN = Recurrent Convolutional Neural Network

7 Network Design & Training Strategies

In this chapter, different network configurations and training methods will be explored. This step serves as a preliminary design of a speck2e-compatible network, that shall be able to determine optical flow from both the SR and Davis datasets. This chapter explain LIF networks and why they are suitable for learning flow in section 7.1. Following section 7.2 will introduce discuss the use of biases as a substitute to leaks in the neuron model. section 7.3 and 7.4 will introduce the different network configurations and the ANN-to-SNN conversion method. The performance of each configuration will be assessed in section 7.5, 7.6 and 7.7.

7.1. Leak Integrate & Fire Recurrent Network

LIF neurons have two short term memory systems that allow for temporal feature extraction. The first one is the leak inside the neurons, which actively reduces the membrane potential. The leak is a measure of how much of the previous state is retained and how much is dispersed in absence of inputs. The second system is the recurrent links in the network. When a neurons spikes, that information is also sent backwards and taken into account in the following computations. These two memory systems allow LIF networks to be trainable on time-dependant tasks. The information transmitted between neurons is not only in spikes that are sent in the recurrent connections, but it also resides in the membrane potential. For instance, when a neuron's potential increases but does not exceed the threshold, the leak system filters out that useless accumulated information. Without a leak system in the neurons, there is a need for a new short term memory mechanism.

7.2. Integrate & Fire Recurrent Network with Biases

The neurons available on the speck2e are only IF, however a bias term can be introduced. The bias is a trainable parameter that is added to the output of a convolutional layer. On the chip itself, this term is added periodically at a specified frequency, while for training, the term is added every time an image is passed through the network. By clamping the biases to negative values during training, the potential can be reduced in a similar way as with leaks. However, when a IF network with biases is trained, it is not able to learn optical flow.

7.2.1. Analysis and Assessment of Biases

By comparing the neuron models, this phenomenon can be better understood (Figure 7.1a and 7.1b). The new input coming from the synaptic connections is first passed through a convolutional layer `self.ff`, where the weights and biases are applied. The current of the previous state decays with the `leak_i` factor and together with the new current, it forms the new current `i_out`. Finally, the potential value is updated with the current and the previous potential state decreases with the `leak_v` factor. The new potential value `v_out` is fed in the `self.spike_fn`, which outputs 0 or 1, depending whether `v_out` is lower or higher than the threshold.

```python
def forward(self, input_, prev_state):
    # input current
    ff = self.ff(input_)
    i, v, z, _ = prev_state

    # get leaks
    leak_i = torch.sigmoid(self.leak_i)
    leak_v = torch.sigmoid(self.leak_v)

    # current update
    i_out = i * leak_i + ff

    # voltage update
    v_out = v * leak_v * (1 - z) + i_out

    # spike output
    z_out = self.spike_fn(v_out, thresh)

    return z_out, [i_out, v_out, z_out, ff]
```

(a) LIF neuron model code.

```python
def forward(self, input_, prev_state):
    # input current
    ff = self.ff(input_)
    v, z, _ = prev_state

    # potential update
    v_out = v + ff
    v_out = torch.clamp_max(v_out, self.thresh)
    v_out = torch.clamp_min(v_out, -self.thresh)

    # spike output
    z_out = self.spike_fn(v_out, thresh)

    # reset voltage with new spike
    v_out = v_out * (1 - z_out)

    return z_out, [v_out, z_out, ff]
```

(b) IF neuron model code.

The leak terms essentially represent how much of the previous state needs to be taken into account in the future and how much is ”forgotten”. During training, this quantity is only multiplied by the previous state, therefore it impacts only the resetting phase of the neuron. In IF neurons, the bias is not multiplied but subtracted from the potential, thus its reducing effect is not only applied on the previous state but also on the new incoming current. Although the biases can be trained, they are not able to actively reduce the potential as promptly as a leak, especially because they are not paired to the previous state directly. As a result, in some scenarios the neurons end up spiking uncontrollably and the bias is not able to restore the potential, while in other scenarios the bias might be too large and prevent the neurons from spiking at all.

The effect of biases on learning can be better understood by looking at the channels of the encoders over time. Each encoder in the network has a certain number of input and output channels. The number of channels is representative of the number of features that each layer is capturing in the image. Figure 7.2 shows a channel of the first encoder at 4 different time steps. An input sequence is fed in the network and the channel is observed over time, while the events propagate through the network. The channel is expected to change as the input sequence moves in different directions. However, because the bias is not able to recover the potential, most of the neurons start spiking continuously until the end. As more inputs are fed in, the already spiking neurons continue spiking and more neurons start following the same trend. Towards the end of the sequence, almost all the neurons are spiking continuously and no meaningful pattern is detected. This proves the inadequacy of IF neurons with biases for training.

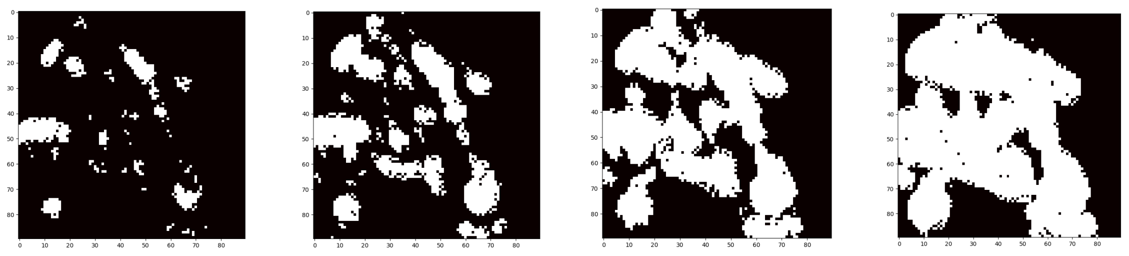


Figure 7.2: Feature number 1 of 32 of the first encoder over time.

7.3. Recurrent Convolutional Neural Network

ANN-to-SNN conversion is commonly used technique used for training IF SNN without dealing with surrogate gradients. Instead of a SNN, an ANN with ReLU activation functions is trained and then converted. The ReLU functions are then substituted with IF spiking functions. This method works for simple tasks such as the MNIST, however it results in an inevitable drop in accuracy. Note that with this training strategy, the only memory mechanism inside the network is provided by the recurrent connections. Having ReLU activation in every layer, especially in the recurrent ones, can cause exploding gradients. For instance if a layer returns a large output, that same output is fed back through the network and causes an even larger output. This can be monitored by decreasing the learning rate, in order to avoid excessive weight updates.

7.3.1. Recurrency Types and Architecture

Two different recurrency types were presented in section 6.5. type S requires two convolutional layers, a forward and a recurrent one. The forward layer increases the number of channels and decreases the shape of the images, therefore it encodes the relevant information of the new incoming inputs. The recurrent layer maintains the same number of channels and image shape, however it feeds the output back to itself, in order to save the relevant information of the previous inputs.

type C uses two forward layers and the output of the second one is first passed through the recurrent layer and then concatenated with the output of the first forward layer. This entails that half of the inputs to the second forward layer will be coming from the first forward layer and half from the recurrent one. Because the output is fed into an additional layer before being fed back, this type of recurrency is expected to be more stable and regularized in training and less prone to exploding gradients.

The previous information is then added to the new incoming one. The network will have two main encoders. Table 7.1, 7.2 and 7.3 show the features of all layers for the three architectures that will be presented in this chapter: RNN-3-S, RNN-2-S, RNN-2-C. The first two will be useful to observe the impact on performance of reducing the number of encoders, which is expected to decrease redundant connections and synaptic operations. The latter two are used to compare the recurrency types. type C has more parameters than type S, thus it is expected to perform better. The performances of the networks will also be compared to their spiking equivalent. Note that when switching from 2 to 3 encoders, the stride of the second encoder doubles. This is done to decrease the output size, so that the kernel of the pooling layer does not exceed the limit of 16 (R-speck-8). Moreover, the number of channels in every layer is decreased to the minimum, in order to limit the number of synaptic operations.

Table 7.1: Network Architecture for RNN-3-S on SR dataset.

	Layer	In Channels	Out Channels	In Size	Out Size	Kernel	Stride	Padding
Encoder 0	Fwd.	2	4	180	90	3	2	1
	Rec.	4	4	90	90	3	1	0
Encoder 1	Fwd.	4	8	90	45	3	2	1
	Rec.	8	8	45	45	3	1	0
Encoder 3	Fwd.	8	16	45	23	3	2	1
	Rec.	16	16	23	23	3	1	0
	Pooling	16	8	23	1	23	1	0
	Prediction	8	8	1	1	1	1	0

Table 7.2: Network Architecture for RNN-2-S on SR dataset.

	Layer	In Channels	Out Channels	In Size	Out Size	Kernel	Stride	Padding
Encoder 0	Fwd.	2	4	180	90	3	2	1
	Rec.	4	4	90	90	3	1	0
Encoder 1	Fwd.	4	8	90	23	3	4	1
	Rec.	8	8	23	23	3	1	0
	Pooling	8	8	23	1	23	1	0
	Prediction	8	8	1	1	1	1	0

Table 7.3: Network Architecture for RNN-2-C on SR dataset.

	Layer	In Channels	Out Channels	In Size	Out Size	Kernel	Stride	Padding
Encoder 0	Fwd. 1	2	4	180	90	3	2	1
	Fwd. 2	8	8	90	90	3	1	1
	Rec.	8	4	90	90	3	1	0
Encoder 1	Fwd. 1	8	16	90	23	3	4	1
	Fwd. 2	32	32	23	23	3	1	1
	Rec.	32	16	23	23	3	1	0
	Pooling	32	8	23	1	23	1	0
	Prediction	8	8	1	1	1	1	0

7.4. Integrate & Fire Recurrent Network

As explained in section 7.1, LIF neurons extract temporal information with the help of both the leaks and the recurrent connections. Now with IF neurons the potential activity does not behave as a short term memory system, which means that all the valuable information among layers is encoded in the spikes and the recurrent connections are used to remember previous states. Commonly, ANN are converted to spiking networks by substituting the ReLU activation functions with a spiking activation function. The thresholds of such function are usually set to $\pm$ 1.0. These conversion settings work well for non time-dependant tasks such as image classification and pattern recognition, where recurrent connections are usually not required, however, when applied to RNNs, the information in every layer tends to accumulate, without transmitting spikes to the following layers. If the information between layers is not promptly transmitted, the recurrent connections are not going to be activated and therefore the network will not be able to produce correct predictions.

7.4.1. Weights Re-scaling

Recurrent connections only activate if the neuron spikes, thus setting the threshold too high with respect to the scale of the synaptic inputs results in losing relevant information. For this reason, it is important to re-scale thresholds and weights to be in the right range. By observing the range of the output of the ReLU functions in every layer, a common threshold can be defined for every encoder. The threshold should be low enough to allow some spikes to pass through at approximately every frame of 5 ms. Setting the threshold to high will result in loss of information over time, while setting it too low will make the layer produce more spikes than needed. Note that finding the exact threshold value to allow the minimum number of spikes to pass, while still encoding the relevant information, would be a quite challenging and difficult to test optimization problem. For this reason the thresholds will be selected with an educated guess.

For instance, in the first layer of the first encoder, the outputs are in the range 10^{-1}, thus the upper and lower thresholds are set to $\pm$ 0.1 for all layers of the first encoder. The weights of the following layer are then re-scaled by 0.1. This pattern of re-scaling the weights by the threshold value of the previous layer is repeated for all layers, as shown in Figure 7.3. In the second encoder the synaptic output range is in the order of 10^{-2}, thus the threshold is set to $\pm$ 0.01.

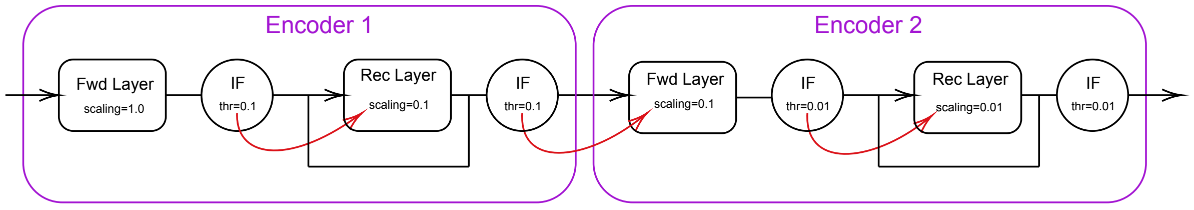


Figure 7.3: Weight re-scaling diagram.

Note that the spiking functions `IAFSqueeze` need to be in `MultiSpike` mode, which means that more than one spike can be produced at every time step, if the synaptic input is large enough. Note that on the speck, the neurons do not produce multiple spikes all at once but one after the other sequentially, as all the computations occur asynchronously. Using `MultiSpike` allows to account for the fact that within a given time window, a neuron can produce multiple spikes one after the other. The spiking layer's reset function is set to `MembraneReset` instead of `MembranSubtract`, in order to avoid potential accumulation and unnecessary spikes.

Figure 7.4 shows the output of the ReLU function (continuous) compared to the output of the `IAFSqueeze` function (discrete) and the resulting spike rate. The idea of ANN-to-SNN conversion is using only the spikes to transmit information through layers and therefore reducing the resolution or sampling hte information. For instance, if the input of a neuron equal to 0.421 and the threshold of the neuron is set to 0.1, 4 spikes will be transmitted to all the neighboring neurons.

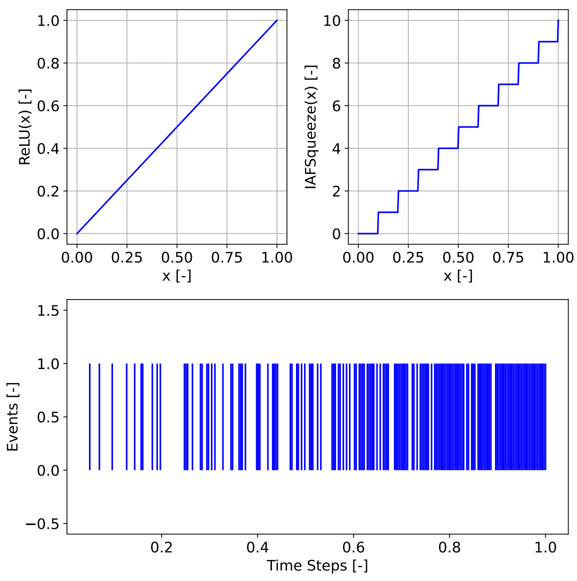


Figure 7.4: Activation functions and resulting spike rate output.

7.5. Results on SR Dataset

In order to compare the different networks, three metrics will be considered: Signal-to-Noise Ratio (SNR), Average End-point Error (AEE) and Ratio of Squared Average Timestamps (RSAT). The prediction of the network compared to the ground truth optical flow can be observed in Figure 7.5 for RNN-2-S and IF-2-S and in Figure 7.6, for RNN-2-C and IF-2-C.

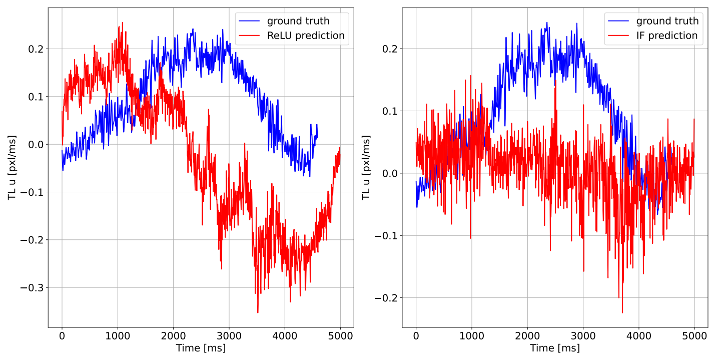


Figure 7.5: Ground truth compared to network prediction for RNN-2-S and IF-2-S.

The converted network IF-2-S performs poorly compared to the ground truth and to its RNN counterpart. This is probably due to the conversion and re-scaling system, which reduces the accuracy of the network and increases the noisiness of the predictions.

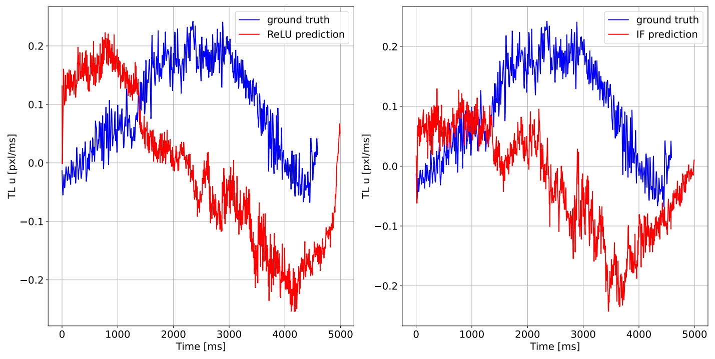


Figure 7.6: Ground truth compared to network prediction for RNN-2-C and IF-2-C.

RNN-2-C and IF-2-C seem to have a more comparable performance, although the differences can still be seen. In some instances, IF-2-C appears to be closer to the ground truth than RNN-2-C. All the other network predictions compared to ground truth can be seen in Appendix A. In order to properly assess the networks, some performance metrics need to be introduced.

7.5.1. Signal-to-Noise Ratio

To calculate the SNR, a sequence of 5.0 s from the testing dataset is considered. The network outputs 8 optical flow predictions, a horizontal and a vertical component for each of the 4 corners of the image. To distinguish the power of the signal from the noise, the predictions are transposed to the frequency domain using Fast Fourier Transform (FFT). In this way, the main signal frequency can be identified by observing the peaks. The frequency range of the main signal is defined by observing the FFT of the ground truth signal and it is approximately $2.5 \cdot 10^{-3}$ Hz. Using Equation 7.1 the average SNR can be calculated for each flow vector signal and compared to the ground truth.

$$\text{SNR} = 10 \cdot \log_{10}\left(\frac{P_{\text{signal}}}{P_{\text{noise}}}\right) \tag{7.1}$$

Figure 7.7, 7.8 and 7.9 show the FFT of the ground truth signal compared to LIF-3, RNN-3-S, RNN-2-S, IF-2-S, RNN-2-C and IF-2-C predictions. On average the ground truth has a SNR of 12.8 dB. The noise of the prediction signals is expected to increase when the network is converted to spiking, since the resolution of the information between layers decreases and the output prediction deteriorates.

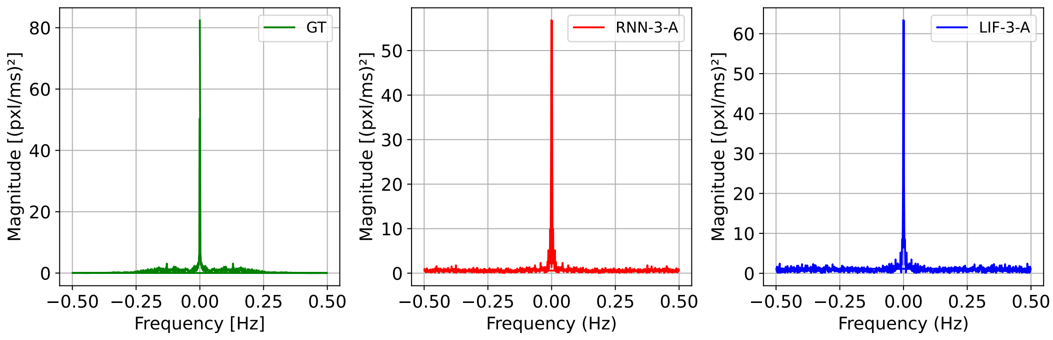


Figure 7.7: FFT of ground truth, RNN-3-S and LIF-3 output signals for top left horizontal component.

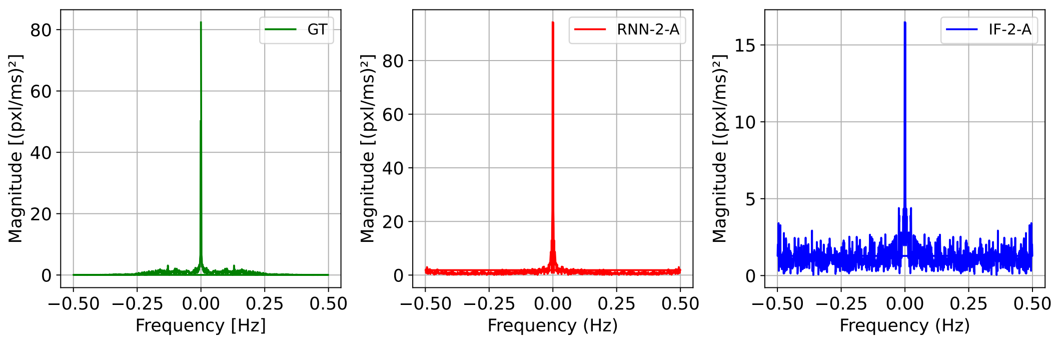


Figure 7.8: FFT of ground truth, RNN-2-S and IF-2-S output signals for top left horizontal component.

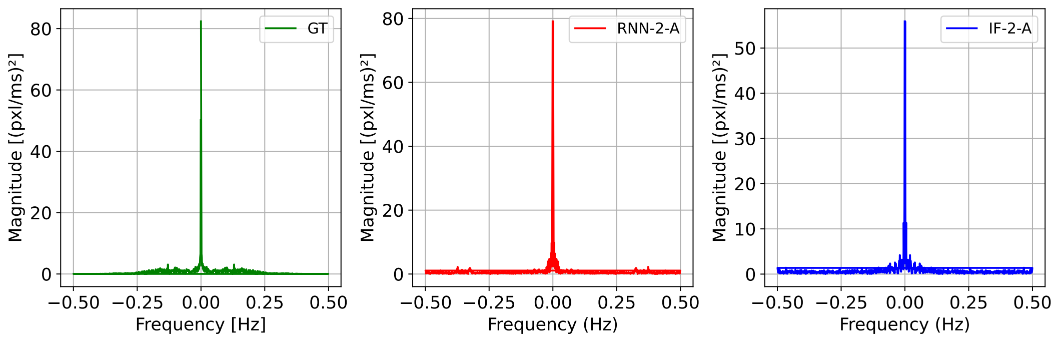


Figure 7.9: FFT of ground truth, RNN-2-C and IF-2-C output signals for top left horizontal component.

7.5.2. Average End-point Error

The Average Endpoint Error (AEE) is a metric used to evaluate the accuracy of optical flow algorithms. Every pixel in the image is transposed in space using the ground truth vectors and the predicted vectors. The euclidean distance between the two pixels' locations is calculated and the average of all errors is taken.

7.5.3. Ratio of the Squared Averaged Timestamps

The Ratio of Squared Average Timestamps (RSAT) is the is the ratio of the squared sum of the per-pixel and per-polarity average timestamp of the image of warped events and that of the image of (non-warped) events. The lower the value of this metric, the better the optical flow estimate. Note that this metric is sensitive to the number of input events. Table 7.4 summarizes the results for all the trained networks. The discussion of results will follow in section 7.7

Table 7.4: SNR, AEE and RSAT for the different network types on a SR dataset sequence.

Configuration	SNR (dB)	AEE	RSAT
LIF-3	9.09	0.156	0.929
RNN-3-S	10.8	0.152	0.918
RNN-2-S	9.11	0.185	0.949
IF-2-S	-0.877	0.165	0.971
RNN-2-C	12.2	0.166	0.925
IF-2-C	6.98	0.149	0.945

7.6. Results on Davis Dataset

Results on the Davis dataset cannot be compared to ground truth, thus the performance of the network is evaluated by visual inspection. The size of the dataset sequences is reduced from 128 pixels to 90, in order to further reduce the number of synaptic operations for hardware implementation. Figure 7.10 shows the output of the network on a horizontal and a vertical motion sequence. The frames on top show the color-coded optical flow representation, according to the color map in Figure A.13.

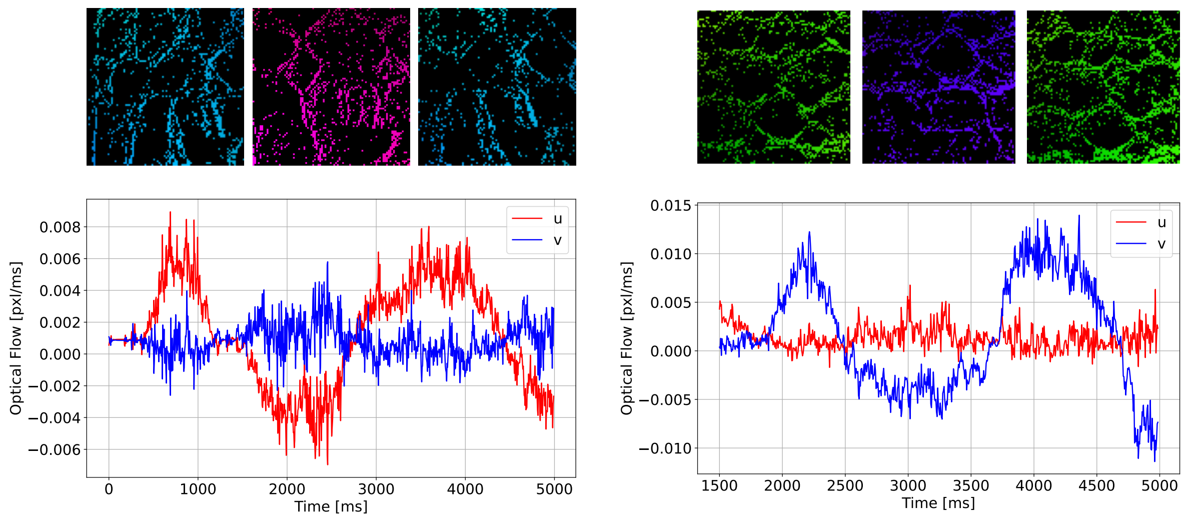


Figure 7.10: RNN-2-C optical flow vectors for the Davis dataset sequence.

In order to quantify the increase in noise after conversion, the SNR can be calculated by defining the main frequency of the signal from the RNN output and compare it to the IF network output.

SNR results

7.7. Results Discussion

The results obtained from the tested networks shall be discussed, in order to assess their performance and decide which configuration will be used for the hardware implementation.

7.7.1. Neuron Model: LIF vs. RNN vs. IF

The new networks' performance is comparable to the LIF network. As expected, RNN-3-S performs better than LIF-3 in all metrics. This is due to the higher information resolution between layers. Moreover, LIF networks have a higher level of complexity with respect to RNNs, because of the the additional hyperparameters, such as leaks and thresholds. When the RNN is reduced to 2 encoders only, the performance inevitably drops and, similarly when the network is converted to IF-2-S.
With RNN-2-C, the network is able to produce less noisy signals, at the cost of a small drop in accuracy compared to LIF. When converted, the IF-2-C becomes more noisy, but it shows a lower AEE value.

7.7.2. Number of Encoders: 3 vs. 2

Reducing the number of encoders from 3 to 2 undoubtedly influences performance, because less parameters are used. However the drop in performance in SNR, AEE and RSAT is not significant and the network is still able to recognize fairly accurately the general direction of the flow. Having more encoders would also mean having more redundancy and more synaptic operations in the network, therefore choosing a less complex configuration is preferred. Additionally, 3 encoders with recurrency type C means a total of 10 convolutional cores is needed, which is more than the speck2e has (R-speck-1).

7.7.3. Recurrency Type: Sum vs. Concatenation

With the same number of encoders, recurrency type C is more accurate than type S in terms of AEE and RSAT, because it has more parameters. Moreover, type C outputs less noisy predictions, which is most likely due to the additional convolutional layer in the recurrent connection. This extra layer processes the previous state and it serves as a filter for non-relevant information. With type S recurrency, the previous state is directly summed with the new incoming state and then processed, therefore the convolutional layer does not distinguish between past and present information. type C behaves similarly to GRU connections, where the reset and update gates distinguish decide how much of the previous state information should be kept.

In terms of hardware compatibility, the both type S and B are suitable for the speck2e. However, one aspect to consider is that the network on chip runs asynchronously at a much higher frequency, while in simulation it is tested synchronously. With type S this represents a problem, because if the output spikes of a layer are fed in the same layer immediately after and summed to the new incoming ones, the layer will reach the limit of operations much faster and it might not be functional. The extra layer in the type C configuration delays the output spikes before feeding them back and also it concatenates with the new incoming input instead of being summed. This allows for better regularization of the network's activity and it prevents the risk of reaching the synaptic operations limit.

7.7.4. Network Conversion

Converting RNN to IF inevitably lowers the SNR and makes the prediction less defined. When the network is converted, the resolution of the information between layers is reduced and therefore output signal becomes less defined. In type S (SNR = -0.877 dB), the conversion has way more impact on the SNR than in type C (SNR = 6.98 dB). This could be due to the extra layer in the recurrent connection, as previously specified in subsection 7.7.3.

It is interesting to notice that when converting RNN-2-S to IF-2-S and RNN-2-C to IF-2-C, the AEE becomes lower while the RSAT increases. The reason for the better AEE, could be that a more noisy prediction is better at generalizing the motion. Even though a less noisy prediction could be more defined, it might still be quite far from the ground truth. Figure 7.6 shows the RNN-2-C and IF-2-C predictions compared to the ground truth. The gap between the RNN prediction and the ground truth seems wider than for the IF network.

7.7.5. Conclusion

The network configuration that will be hosted on the speck2e will be IF-2-C. type C recurrency is considered more suitable for hardware applications because of the asynchronous nature of the neuromorphic device and the higher level of complexity which results in better predictions.

8 Hardware Implementation

In this chapter the network implementation on the speck2 is treated. In section 8.1, the quantization of the network's parameters is explained. Section 8.2 treats different strategies to reduce the number of synaptic operations in the layers.

8.1. Quantized Network

Quantization is the process of reducing the precision of a neural network model's parameters for hosting it on hardware. It involves converting floating-point representations to fixed-point or integers. This is done to optimize memory usage and computational efficiency and it comes at the cost of a loss in precision. In order to properly asses the on-chip network performance, the output of the speck2e is compared to the model with quantized parameters.

8.1.1. Parameters Quantization

The quantization of weights in thresholds is done with the `DynapcnnNetwork` sinabs function. The synaptic weights can be represented by a maximum of 8 bits (R-speck-6), hence the range of weights in every layer is [-127,127], while membrane potentials and thresholds have a 16 bits resolution, thus the range is [-32767,32767]. The neurons are initiated at average potential value (1 in quantized parameters). The effect of quantizing the parameters can be seen in Figure 8.1.

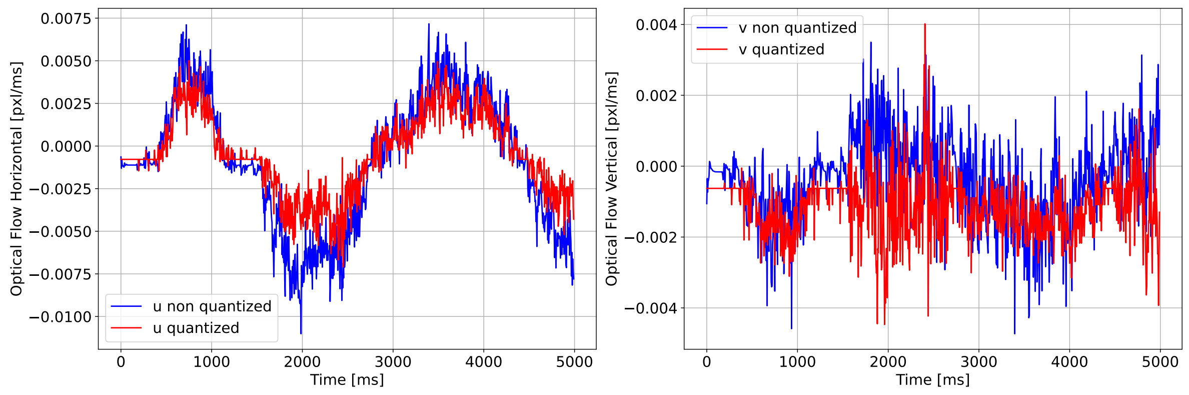


Figure 8.1: Non-quantized network versus quantized network.

8.2. Synaptic Operations Analysis

The speck2e device has a limit on synaptic operations (synops) per second per convolutional core (R-speck-14). Every core runs at a frequency of 1 MHz, therefore it can process at most one spike every microsecond. If two events share the same timestamp, the core will process them sequentially.

When the number of synops exceeds the limit, the events accumulate and information is lost in time. When the RNN-2-C configuration is uploaded on the speck2e, the activity of the individual cores can be observed. Figure 8.2 shows the total number of spikes over time produced by the first encoder's layers. Layer 0 is the first forward layer, layer 1 is the second and layer 2 is the recurrent one. It is clear that each layer reaches a saturation point when an overflow of input spikes arrives. To avoid this, the network's architecture and training has to be revisited.

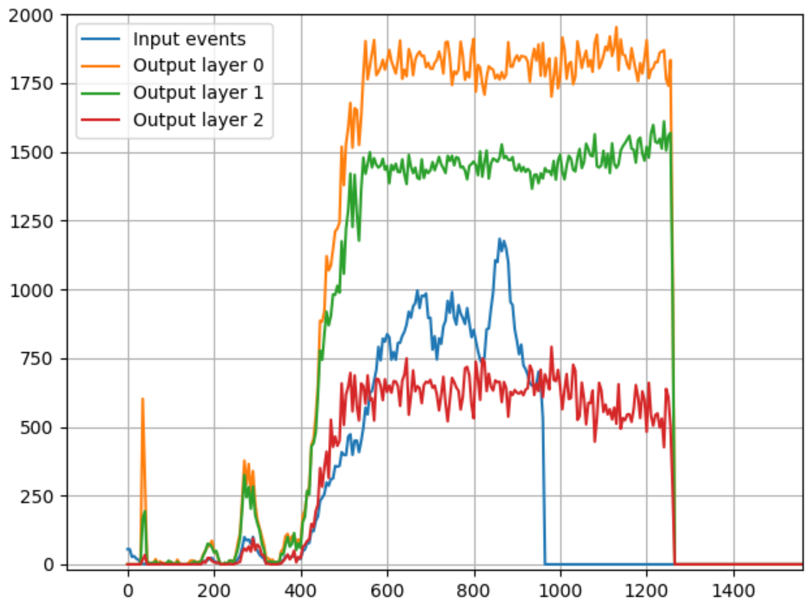


Figure 8.2: Number of synaptic operations for IF-2-C configuration.

When minimizing the synops, two layer characteristics shall be considered: the total number of operations and the maximum number of operations. Section 7.4.1 explained how the conversion and re-scaling method relies on spike rate encoded information. This means that individual neurons need to output multiple spikes in a time step, to better represent the information sent to neighboring neurons. With the sinabs activation function `IAFSqueeze`, a neuron sends multiple spikes if the synaptic weight is larger than the threshold. When the weights are significantly larger than the threshold, the model performance starts deviating from the on-chip behaviour. This occurs because the spikes are received and processed asynchronously on the chip, unlike in simulation. Without regularization, some neurons can end up spiking hundreds of times per step. For these reasons, it is fundamental to monitor and regularize both the total number of operations and the maximum number of spikes per neuron. This can be achieved by modifying the network's architecture and/or introducing an regularization term in the loss function.

8.2.1. Stride

As seen in Table 7.3, the first encoder has stride 2 and the second has stride 4. This was done to reduce the pooling layer's kernel size from (23,23) to (12,12), given the limit of 16 (R-speck-8). To minimize the synops, the stride of the first encoder can be changed to 4 as well. By halving the size of the images in the second encoder, the total and maximum synops are reduced to approximately a third (Figure 8.4). Obviously this affects the prediction of the network, since the information between layers is down-sampled. Figure 8.3 shows on the top the total number of spikes per layer and on the bottom the maximum number of spikes produced by a single neuron.

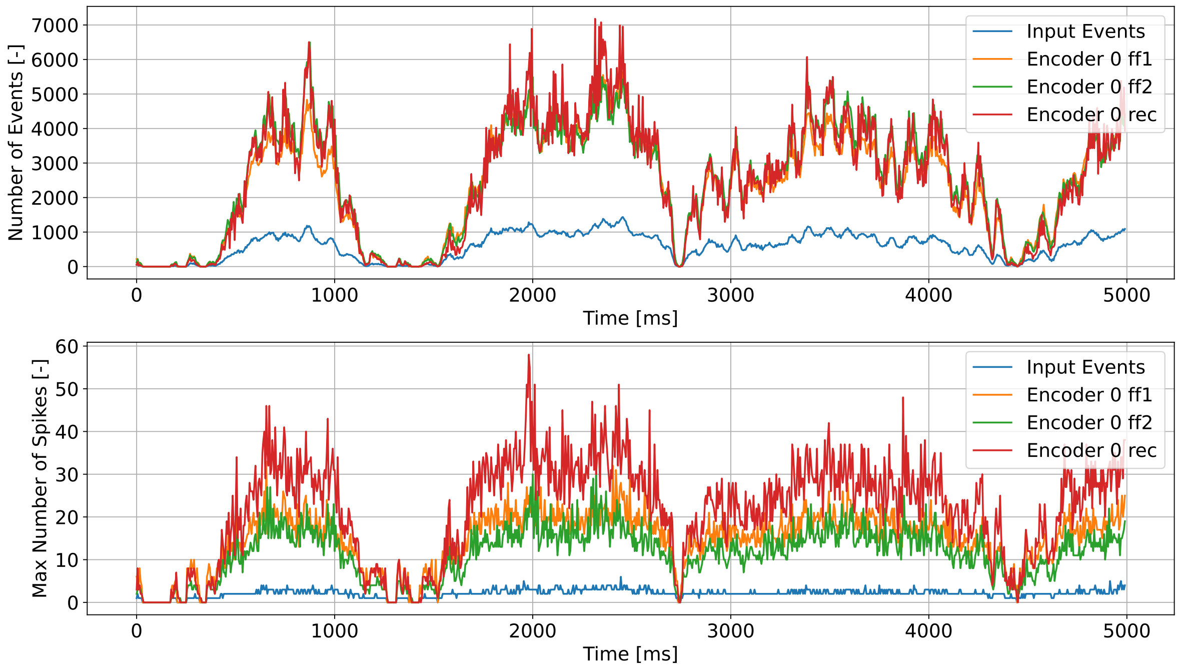


Figure 8.3: Spike activity in RNN-2-C with stride 2 in first encoder.

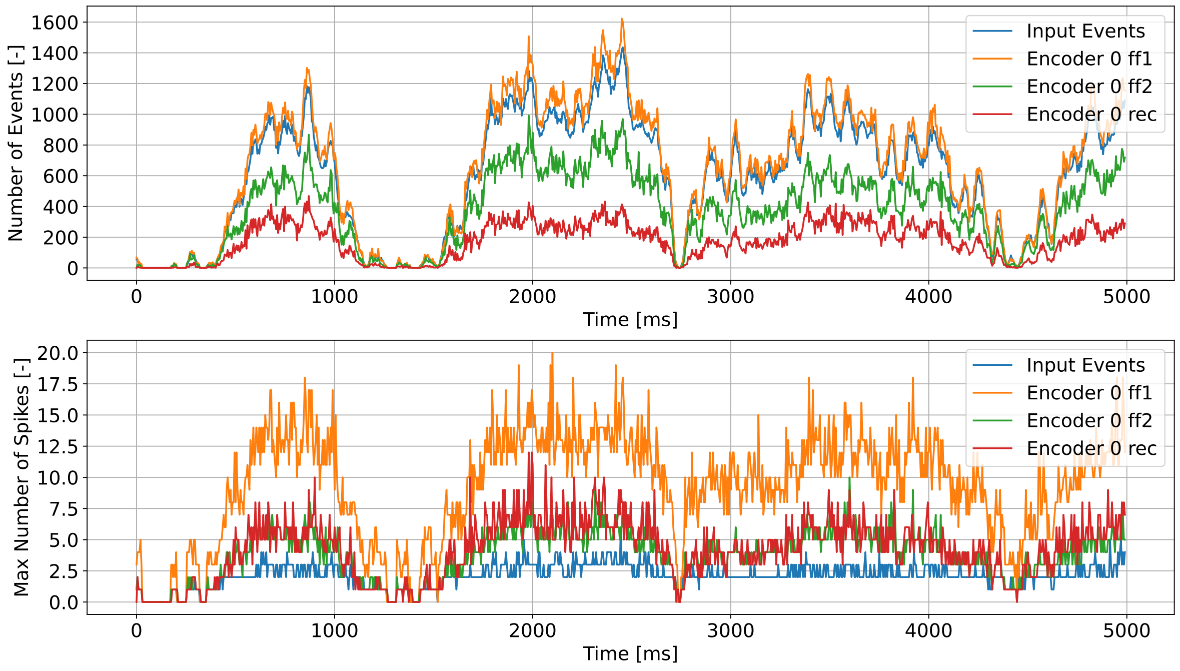


Figure 8.4: Spike activity RNN-2-C with stride 4 in first encoder.

8.2.2. Number of Channels

Increasing the number of channels can make the network's activity more sparse. Choosing a configuration with more neurons means having more neurons doing less work, therefore less operations per neuron overall. To prove this, two new network architectures are trained and tested. RNN-2-C has 4 output channels in the first layer of the first encoder. The new architectures will have 8 and 3 output channels instead.

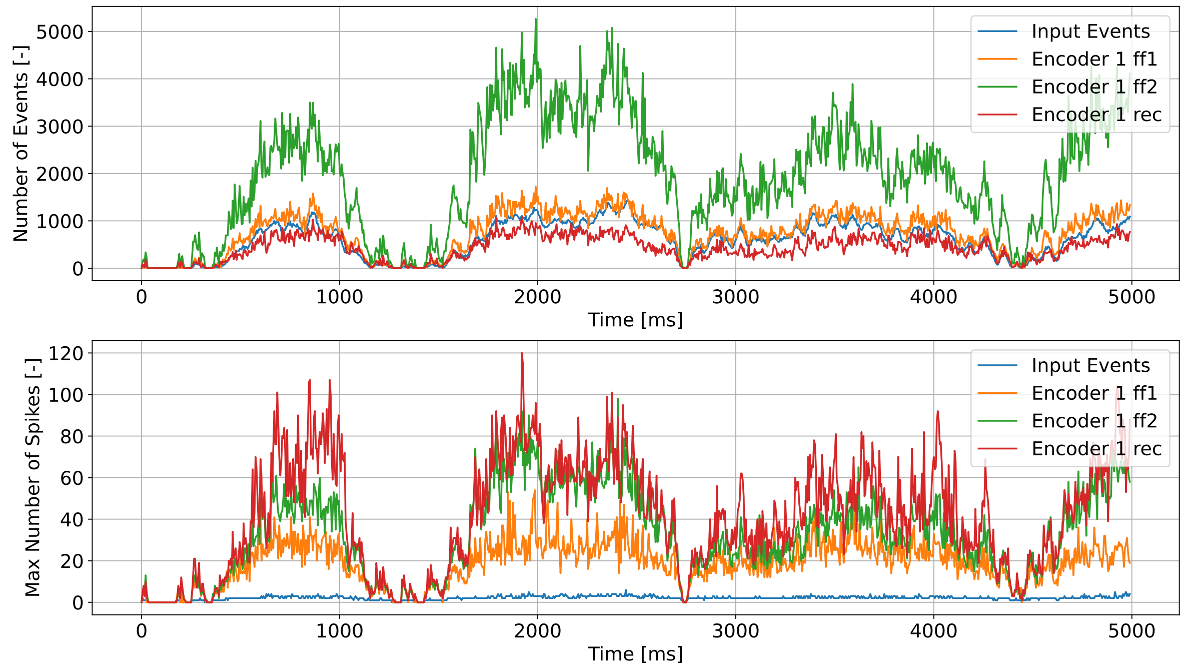


Figure 8.5: RNN-2-C with increased number of channels.

From Figure 8.5 and 8.6 it is clear that increasing the number of channels results in more synops especially in the first layer of the second encoder. However the maximum number of spikes is decreased, hence more neurons are doing less work. Although this change is not significant, if combined with regularization term in the loss function, a more sparse activity could be achieved. Note that the number of channels can be increased up to 1024 (R-speck-5), however the limit on the total number of neurons is 32k (R-speck-2).

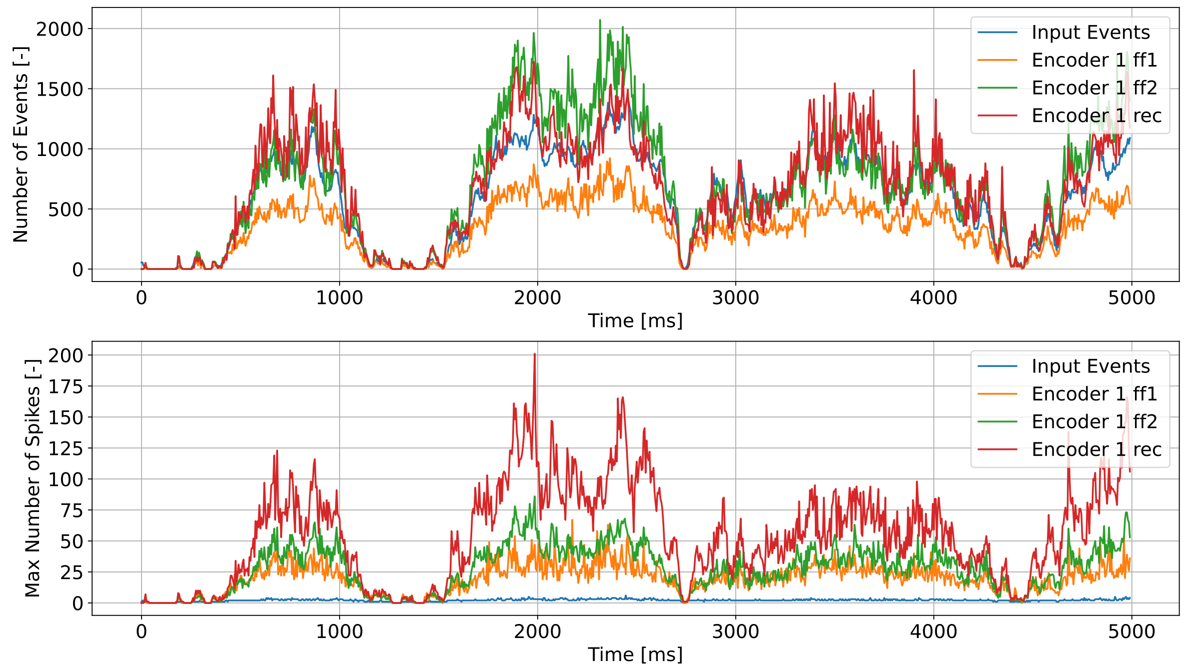


Figure 8.6: RNN-2-C with decreased number of channels.

8.2.3. Early Stopping

During training, the network establishes the synaptic weights to optimize for minimal loss. However, in this case the minimum loss is often achieved in the first few epochs. By letting the network train more, additional connections are formed, which make the predictions more accurate. These redundant weights are useful for making the network more precise, however they increase the number of operations significantly. If the training is stopped earlier and the model is still able to estimate the flow accurately, a more operations efficient result can be obtained.

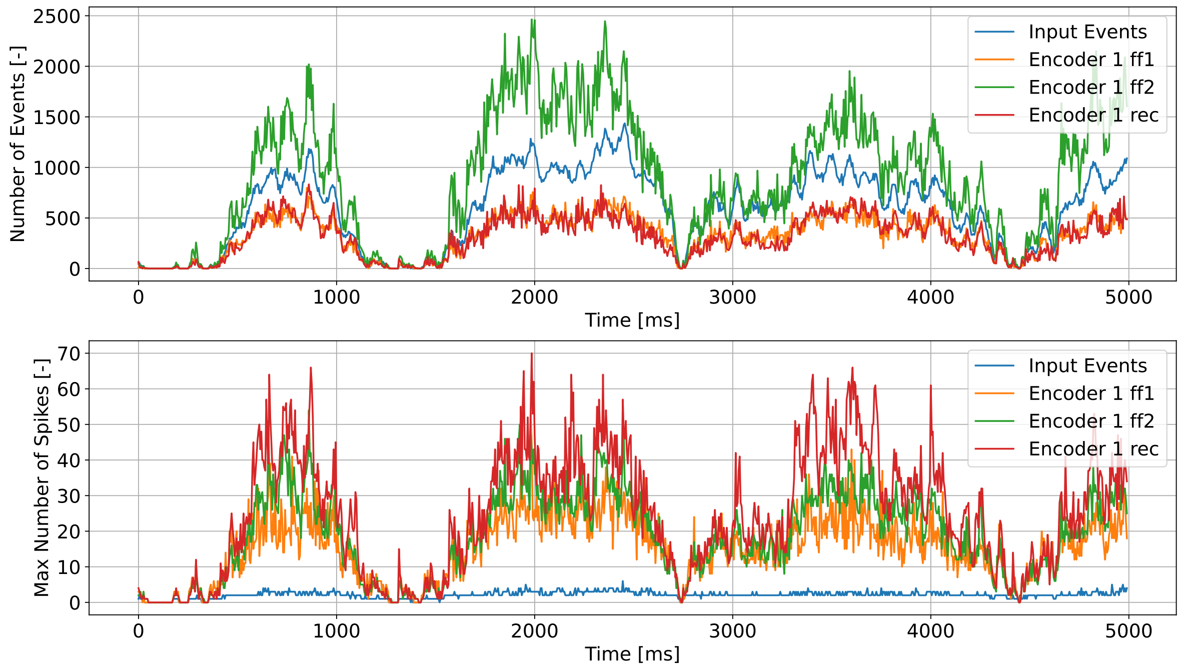


Figure 8.7: RNN-2-C after 80 epochs.

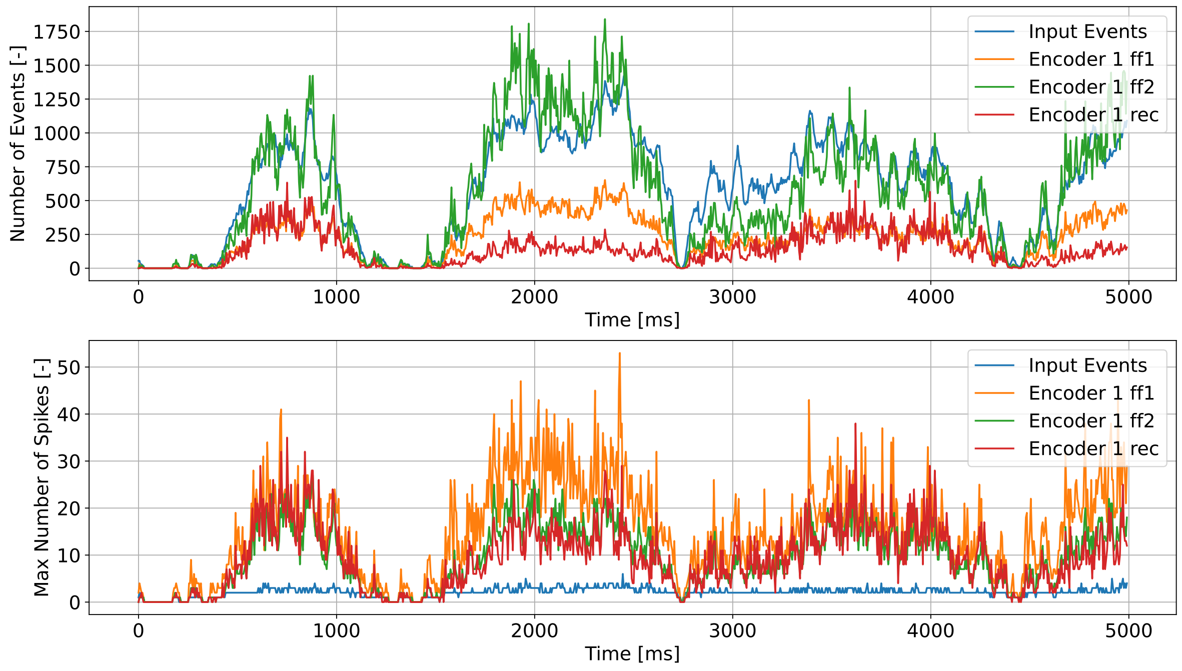


Figure 8.8: RNN-2-C after 10 epochs.

Figure 8.7 and 8.8 show that the layers' activity is way lower in the earlier epochs. The second forward layer of the second encoder is almost doubling its number of output spikes.

8.2.4. Synops Loss Term

The exact limit value is not uniquely determined for all cores, as it depends on the its size, the weights and the number of incoming spikes. However, Figure 8.2 gives a rough indication of how many spikes can be handled without accumulation. To minimize the total number of spikes, an additional term in the loss function is included. The synops loss term is given in Equation 8.1.

$$\text{loss} = \sum_{k=1}^{N} \frac{(\text{total output})_k}{(\text{threshold})_k (\text{\# parameters})_k} \tag{8.1}$$

The total output of every layer at every step is the total number of spikes that a layer produces. During training the network is using ReLU functions and the output of every neuron is a floating point. However the range of weights values for every layer changes and, if it is not rescaled, the deeper layers with lower weight values will have less importance in the cost function. Moreover, layers with more parameters need to have a higher spike rate to send more information, thus the layer output is also scaled by the number of parameters. The weight of the synops loss term on the total loss function has to be tuned, in order to achieve a balance that allows to learn optical flow properly and with the minimum number of operations.

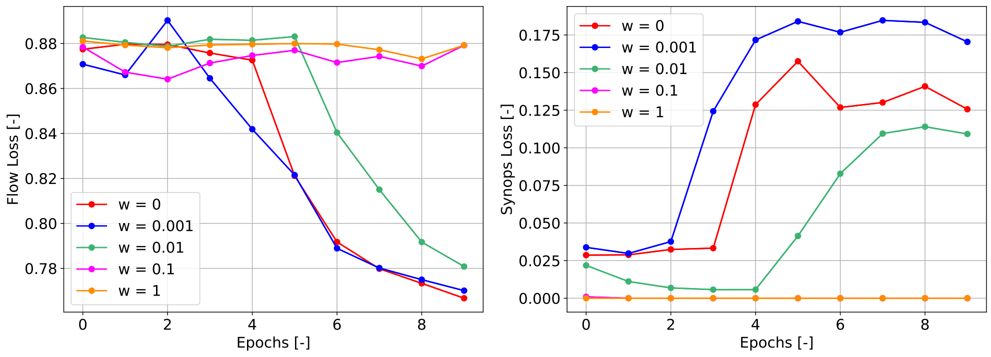


Figure 8.9: Flow and synops loss during training with different weighting.

Figure 8.9 shows the flow loss and synops loss over 10 epochs. By looking at the synops loss, it is clear that 0.01 is an optimal value for the weight, as it allows the flow loss to converge to a slightly higher value, while keeping the synops lower. If the weight is increased to 0.1 (pink line), the network minimizes the synops to zero and it is prevented from learning flow. If it is decreased to 0.001 (blue line), its influence is almost unnoticeable and the synops increase to more than necessary.

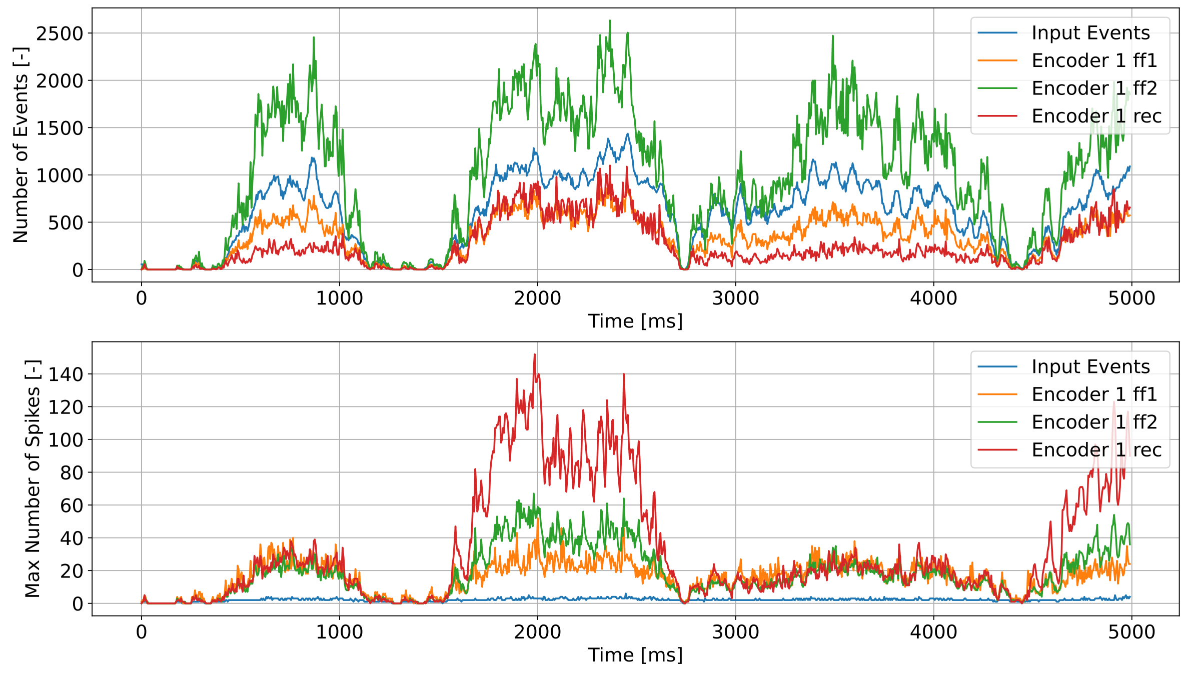


Figure 8.10: RNN-2-C without synops loss term.

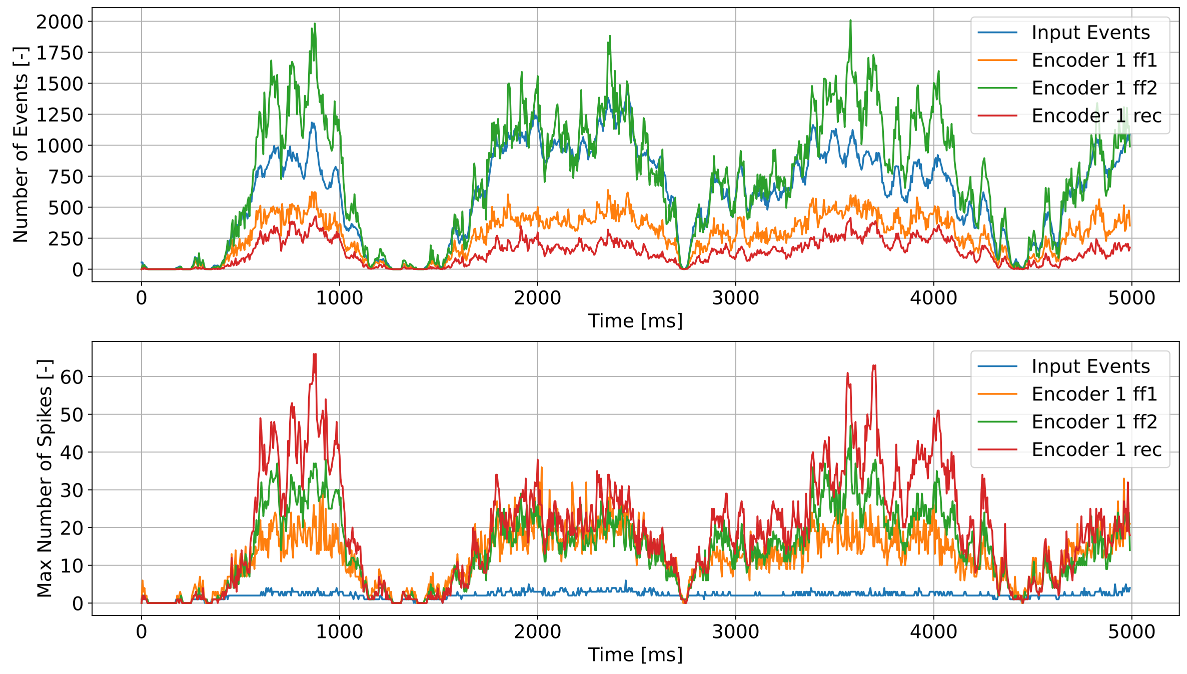


Figure 8.11: RNN-2-C with synops loss weight of 0.01.

8.2.5. Final Network Configuration

In the previous sections, it was showed how different hyperparameters influence the number of synaptic operations. Following these guidelines, a new architecture can be designed for the speck2e. The final configuration is similar to RNN-2-C (Table 7.3), but with a few updates:

- The stride of first layer in Encoder 0 is increased from 2 to 4. As explained in subsection 8.2.1, this forces the network to decrease the information sent between layers.

- Kernel size of pooling layer is decreased from 12 to 6, as a result of the increase in stride.
- The number of output channels of first layer in Encoder 0 increased from 4 to 6. Having a more sparse activity results in less spikes per neuron.
- Number of input channels of first layer in Encoder 1 increased from 8 to 12.
- Number of output channels of pooling layer is increased from 8 to 15. According to R-speck-15, the maximum number of output channels in the readout layer is 15 and all of them shall be used to maximize the number of features obtained from the chip.

An overview of all the configuration details is displayed in Table 8.1. This is the network configuration to be used for the following experiments on the speck2e.

Table 8.1: Network Architecture for RNN-2-C-speck.

	Layer	In Channels	Out Channels	In Size	Out Size	Kernel	Stride	Padding
Encoder 0	Fwd. 1	2	6	90	23	3	4	1
	Fwd. 2	12	12	23	23	3	1	1
	Rec.	12	6	23	23	3	1	0
Encoder 1	Fwd. 1	12	16	23	6	3	4	1
	Fwd. 2	32	32	6	6	3	1	1
	Rec.	32	16	6	6	3	1	0
	Pooling	32	15	6	1	6	1	0
	Prediction	15	8	1	1	1	1	0

References

[1] *Neuromorphic Intelligence I& Application Solutions | Synsense*. URL: https://www.synsense.ai/.

[2] Javaan Chahl, Mandyam Srinivasan, and Shaowu Zhang. “Landing Strategies in Honeybees and Applications to Uninhabited Airborne Vehicles”. In: *I. J. Robotic Res.* 23 (Feb. 2004), pp. 101–110. DOI: 10.1177/0278364904041320.

[3] Harald E. Esch and John E. Burns. “Distance Estimation by Foraging Honeybees”. In: *Journal of Experimental Biology* 199.1 (Jan. 1996), pp. 155–162. ISSN: 0022-0949. DOI: 10.1242/jeb.199.1.155. eprint: https://journals.biologists.com/jeb/article-pdf/199/1/155/3107314/jexbio_199_1_155.pdf. URL: https://doi.org/10.1242/jeb.199.1.155.

[4] Mandyam Srinivasan. “Honeybees as a Model for the Study of Visually Guided Flight, Navigation, and Biologically Inspired Robotics”. In: *Physiological reviews* 91 (Apr. 2011), pp. 413–60. DOI: 10.1152/physrev.00005.2010.

[5] S. S. Beauchemin and J. L. Barron. “The Computation of Optical Flow”. In: 27.3 (Sept. 1995), pp. 433–466. ISSN: 0360-0300. DOI: 10.1145/212094.212141. URL: https://doi.org/10.1145/212094.212141.

[6] Haiyang Chao, Yu Gu, and Marcello Napolitano. “A Survey of Optical Flow Techniques for Robotics Navigation Applications”. In: *Journal of Intelligent I& Robotic Systems* 73 (May 2013). DOI: 10.1007/s10846-013-9923-6.

[7] Bruce Lucas and Takeo Kanade. “An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI)”. In: vol. 81. Apr. 1981.

[8] Berthold K.P. Horn and Brian G. Schunck. “Determining optical flow”. In: *Artificial Intelligence* 17.1 (1981), pp. 185–203. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(81)90024-2. URL: https://www.sciencedirect.com/science/article/pii/0004370281900242.

[9] Bruno Herissé et al. “Landing a VTOL Unmanned Aerial Vehicle on a Moving Platform Using Optical Flow”. In: *IEEE Transactions on Robotics* 28.1 (2012), pp. 77–89. DOI: 10.1109/TRO.2011.2163435.

[10] Kevin Jones, T.C. Lund, and Max Platzer. “Fixed and flapping wing aerodynamics for micro air vehicle applications”. In: *Progress In Astronautics and Aeronautics* (Jan. 2001).

[11] Robert Ross, J. Devlin, and Song Wang. “Toward Refocused Optical Mouse Sensors for Outdoor Optical Flow Odometry”. In: *IEEE Sensors Journal* 12 (June 2012), pp. 1925–1932. DOI: 10.1109/JSEN.2011.2180525.

[12] Stephen Griffiths et al. “Maximizing miniature aerial vehicles - Obstacle and terrain avoidance for MAVs”. In: *IEEE ROBOTICS AND AUTOMATION MAGAZINE. SUBMITTED FOR REVIEW* 1 (Jan. 2007). DOI: 10.1007/978-1-4020-6114-1_7.

[13] Guido C H E de Croon. “Monocular distance estimation with optical flow maneuvers and efference copies: a stability-based strategy”. In: *Bioinspiration I& Biomimetics* 11.1 (Jan. 2016), p. 016004. DOI: 10.1088/1748-3190/11/1/016004. URL: https://dx.doi.org/10.1088/1748-3190/11/1/016004.

[14] Edward H. Adelson and James R. Bergen. “Spatiotemporal energy models for the perception of motion”. In: *J. Opt. Soc. Am. A* 2.2 (Feb. 1985), pp. 284–299. DOI: 10.1364/JOSAA.2.000284. URL: https://opg.optica.org/josaa/abstract.cfm?URI=josaa-2-2-284.

[15] David J. Heeger. “Optical flow using spatiotemporal filters”. In: *International Journal of Computer Vision* 1.4 (Jan. 1988), pp. 279–302. ISSN: 1573-1405. DOI: 10.1007/BF00133568. URL: https://doi.org/10.1007/BF00133568.

[16] David Fleet and Allan Jepson. "Computation of component image velocity from local phase information". In: *International Journal of Computer Vision* 5 (Aug. 1990), pp. 77–104. DOI: 10.1007/BF00056772.

[17] Farid Kendoul, Isabelle Fantoni, and Kenzo Nonami. "Optic flow-based vision system for autonomous 3D localization and control of small aerial vehicles". In: *Robotics and Autonomous Systems* 57.6 (2009), pp. 591–602. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2009.02.001. URL: https://www.sciencedirect.com/science/article/pii/S0921889009000396.

[18] Alexey Dosovitskiy et al. "FlowNet: Learning Optical Flow with Convolutional Networks". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2758–2766. DOI: 10.1109/ICCV.2015.316.

[19] Anurag Ranjan and Michael J. Black. "Optical Flow Estimation Using a Spatial Pyramid Network". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2720–2729. DOI: 10.1109/CVPR.2017.291.

[20] Aria Ahmadi and Ioannis Patras. "Unsupervised convolutional neural networks for motion estimation". In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 1629–1633. DOI: 10.1109/ICIP.2016.7532634.

[21] Jason J. Yu, Adam W. Harley, and Konstantinos G. Derpanis. "Back to Basics: Unsupervised Learning of Optical Flow via Brightness Constancy and Motion Smoothness". In: *CoRR* abs/1608.05842 (2016). arXiv: 1608.05842. URL: http://arxiv.org/abs/1608.05842.

[22] Zhe Ren et al. "Unsupervised Deep Learning for Optical Flow Estimation". In: Feb. 2017.

[23] Christoph Posch et al. "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output". In: *Proceedings of the IEEE* 102.10 (2014), pp. 1470–1484. DOI: 10.1109/JPROC.2014.2346153.

[24] Hanme Kim, Stefan Leutenegger, and Andrew J. Davison. "Real-Time 3D Reconstruction and 6-DoF Tracking with an Event Camera". In: *European Conference on Computer Vision*. 2016.

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

[26] Ziwei Wang et al. *A Linear Comb Filter for Event Flicker Removal*. 2022. arXiv: 2205.08090 [cs.CV].

[27] Enea Ceolini et al. "Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing". In: *Frontiers in Neuroscience* 14 (2020). ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00637. URL: https://www.frontiersin.org/articles/10.3389/fnins.2020.00637.

[28] Arnon Amir et al. "A Low Power, Fully Event-Based Gesture Recognition System". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[29] Garrick Orchard et al. "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades". In: *Frontiers in Neuroscience* 9 (2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00437. URL: https://www.frontiersin.org/articles/10.3389/fnins.2015.00437.

[30] Amos Sironi et al. *HATS: Histograms of Averaged Time Surfaces for Robust Event-based Object Classification*. 2018. arXiv: 1803.07913 [cs.CV].

[31] Javier Cuadrado et al. "Optical flow estimation from event-based cameras and spiking neural networks". In: *Frontiers in Neuroscience* 17 (2023). ISSN: 1662-453X. DOI: 10.3389/fnins.2023.1160034. URL: https://www.frontiersin.org/articles/10.3389/fnins.2023.1160034.

[32] Timo Stoffregen et al. "Event-Based Motion Segmentation by Motion Compensation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019.

[33] Timo Stoffregen and Lindsay Kleeman. *Simultaneous Optical Flow and Segmentation (SOFAS) using Dynamic Vision Sensor*. 2018. arXiv: 1805.12326 [cs.CV].

[34] David Drazen et al. "Toward real-time particle tracking using an event-based dynamic vision sensor". In: *Experiments in Fluids* 51 (Nov. 2011), pp. 1465–1469. DOI: 10.1007/s00348-011-1207-y.

[35] Zhenjiang Ni et al. “Asynchronous event-based high speed vision for microparticle tracking”. In: *Journal of microscopy* 245 (Nov. 2011), pp. 236–44. DOI: 10.1111/j.1365-2818.2011.03565.x.

[36] Xavier Lagorce et al. “Asynchronous Event-Based Multikernel Algorithm for High-Speed Visual Features Tracking”. In: *IEEE Transactions on Neural Networks and Learning Systems* 26.8 (2015), pp. 1710–1720. DOI: 10.1109/TNNLS.2014.2352401.

[37] André Grüning and Sander M Bohte. “Spiking neural networks: Principles and challenges.” In: *ESANN*. Bruges. 2014.

[38] Jason K. Eshraghian et al. *Training Spiking Neural Networks Using Lessons From Deep Learning*. 2023. arXiv: 2109.12894 [cs.NE].

[39] *Sinabs (Sinabs Is Not A Brain Simulator).* URL: https://sinabs.readthedocs.io/en/1.2.8/index.html.

[40] Lyle Long and Guoliang Fang. “A review of biologically plausible neuron models for spiking neural networks”. In: *AIAA Infotech@ Aerospace 2010* (2010), p. 3540.

[41] Arindam Basu, Csaba Petre, and Paul Hasler. “Bifurcations in a silicon neuron”. In: *2008 IEEE International Symposium on Circuits and Systems*. 2008, pp. 428–431. DOI: 10.1109/ISCAS.2008.4541446.

[42] Catherine D Schuman et al. “A survey of neuromorphic computing and neural networks in hardware”. In: *arXiv preprint arXiv:1705.06963* (2017).

[43] Christoph Stöckl and Wolfgang Maass. “Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes”. In: *Nature Machine Intelligence* (2021). DOI: 10.1038/s42256-021-00311-4.

[44] Garrick Orchard et al. “Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades”. In: *Frontiers in Neuroscience* 9 (2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00437. URL: https://www.frontiersin.org/articles/10.3389/fnins.2015.00437.

[45] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. “Surrogate Gradient Learning in Spiking Neural Networks”. In: *CoRR* abs/1901.09948 (2019). arXiv: 1901.09948. URL: http://arxiv.org/abs/1901.09948.

[46] Friedemann Zenke and Tim P. Vogels. “The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks”. In: *Neural Computation* 33.4 (Mar. 2021), pp. 899–925. ISSN: 0899-7667. DOI: 10.1162/neco_a_01367. eprint: https://direct.mit.edu/neco/article-pdf/33/4/899/1902294/neco_a_01367.pdf. URL: https://doi.org/10.1162/neco%5C_a%5C_01367.

[47] Sander M. Bohte, Joost N. Kok, and Han La Poutré. “Error-backpropagation in temporally encoded networks of spiking neurons”. In: *Neurocomputing* 48.1 (2002), pp. 17–37. ISSN: 0925-2312. DOI: https://doi.org/10.1016/S0925-2312(01)00658-0. URL: https://www.sciencedirect.com/science/article/pii/S0925231201006580.

[48] Sumit Bam Shrestha and Garrick Orchard. “SLAYER: Spike Layer Error Reassignment in Time”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/82f2b308c3b01637c607ce05f52a2fed-Paper.pdf.

[49] Christof Koch, Moshe Rapp, and Idan Segev. “A Brief History of Time (Constants)”. In: *Cerebral Cortex* 6.2 (Mar. 1996), pp. 93–101. ISSN: 1047-3211. DOI: 10.1093/cercor/6.2.93. eprint: https://academic.oup.com/cercor/article-pdf/6/2/93/968752/6-2-93.pdf. URL: https://doi.org/10.1093/cercor/6.2.93.

[50] Wei Fang et al. “Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks”. In: (Nov. 2020).

[51] Siqi Wang, Tee Hiang Cheng, and Meng-Hiot Lim. “LTMD: Learning Improvement of Spiking Neural Networks with Learnable Thresholding Neurons and Moderate Dropout”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 28350–28362. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/b5fd95d6b16d3172e307103a97f19e1b-Paper-Conference.pdf.

[52] Federico Paredes-Vallés, Jesse J. Hagenaars, and Guido de Croon. “Self-Supervised Learning of Event-Based Optical Flow with Spiking Neural Networks”. In: *CoRR* abs/2106.01862 (2021). arXiv: 2106.01862. URL: https://arxiv.org/abs/2106.01862.

[53] Alex Zihao Zhu et al. “EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras”. In: *CoRR* abs/1802.06898 (2018). arXiv: 1802.06898. URL: http://arxiv.org/abs/1802.06898.

[54] Alex Zihao Zhu et al. “Unsupervised Event-based Learning of Optical Flow, Depth, and Egomotion”. In: *CoRR* abs/1812.08156 (2018). arXiv: 1812.08156. URL: http://arxiv.org/abs/1812.08156.

[55] Mathias Gehrig et al. “Event-based angular velocity regression with spiking networks”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 4195–4202.

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

[57] Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. “A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3867–3876. DOI: 10.1109/CVPR.2018.00407.

[58] Anton Mitrokhin et al. “Event-Based Moving Object Detection and Tracking”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1–9. DOI: 10.1109/IROS.2018.8593805.

[59] Carlo Michaelis, Andrew B. Lehr, and Christian Tetzlaff. “Robust Trajectory Generation for Robotic Control on the Neuromorphic Research Chip Loihi”. In: *Frontiers in Neurorobotics* 14 (2020). ISSN: 1662-5218. DOI: 10.3389/fnbot.2020.589532. URL: https://www.frontiersin.org/articles/10.3389/fnbot.2020.589532.

[60] Raphaela Kreiser, Alpha Renner, and Yulia Sandamirskaya. “Error-driven learning for self-calibration in a neuromorphic path integration system”. In: Aug. 2019.

[61] Raphaela Kreiser et al. “Error estimation and correction in a spiking neural network for map formation in neuromorphic hardware”. In: May 2020, pp. 6134–6140. DOI: 10.1109/ICRA40945.2020.9197498.

[62] *Loihi 2: A New Generation of Neuromorphic Computing*. URL: https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html.

[63] Rasmus Stagsted et al. “Towards neuromorphic control: A spiking neural network based PID controller for UAV”. In: *Robotics: Science and Systems 2020*. RSS, July 2020. ISBN: 9780992374761. DOI: 10.15607/rss.2020.xvi.074. URL: https://doi.org/10.5167/uzh-200415.

[64] Julien Dupeyroux et al. “Neuromorphic control for optic-flow-based landing of MAVs using the Loihi processor”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 96–102. DOI: 10.1109/ICRA48506.2021.9560937.

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

[66] *Speck: Event-Driven Neuromorphic SoC | Synsense*. URL: https://www.synsense.ai/products/speck-2/.

[67] Ole Richter et al. *Speck: A Smart event-based Vision Sensor with a low latency 327K Neuron Convolutional Neuronal Network Processing Pipeline*. 2023. arXiv: 2304.06793 [cs.NE].

[68] *Samna*. URL: https://pypi.org/project/samna/.

[69] Elias Mueggler et al. “The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM”. In: *CoRR* abs/1610.08336 (2016). arXiv: 1610.08336. URL: http://arxiv.org/abs/1610.08336.

A Additional Results

A.1. Prediction vs. Ground Truth Optical Flow

A.1.1. LIF-3

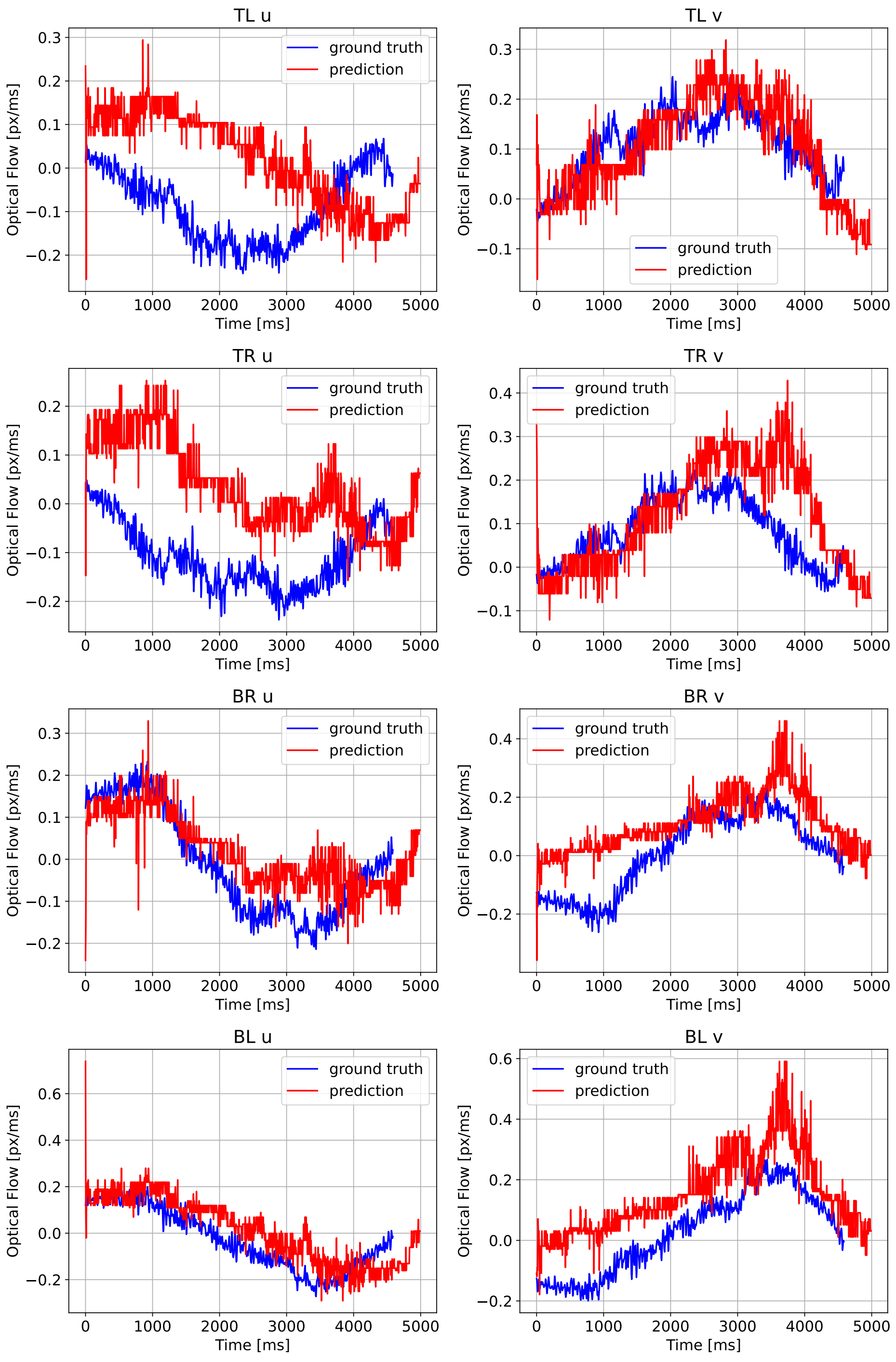


Figure A.1: LIF-3 optical flow prediction.

A.1.2. RNN-3-S

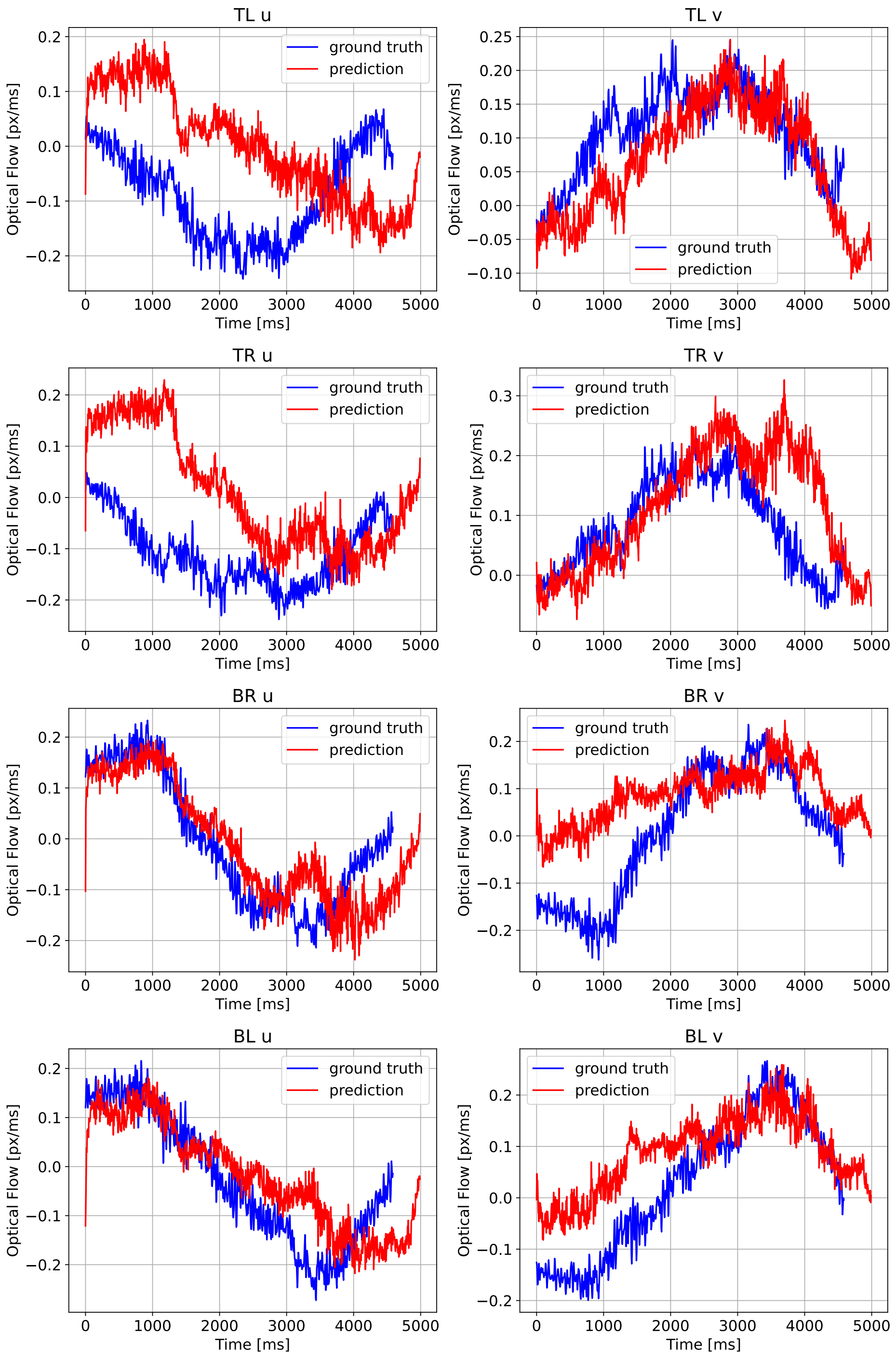


Figure A.2: RNN-3-S optical flow prediction.

A.1.3. RNN-2-S

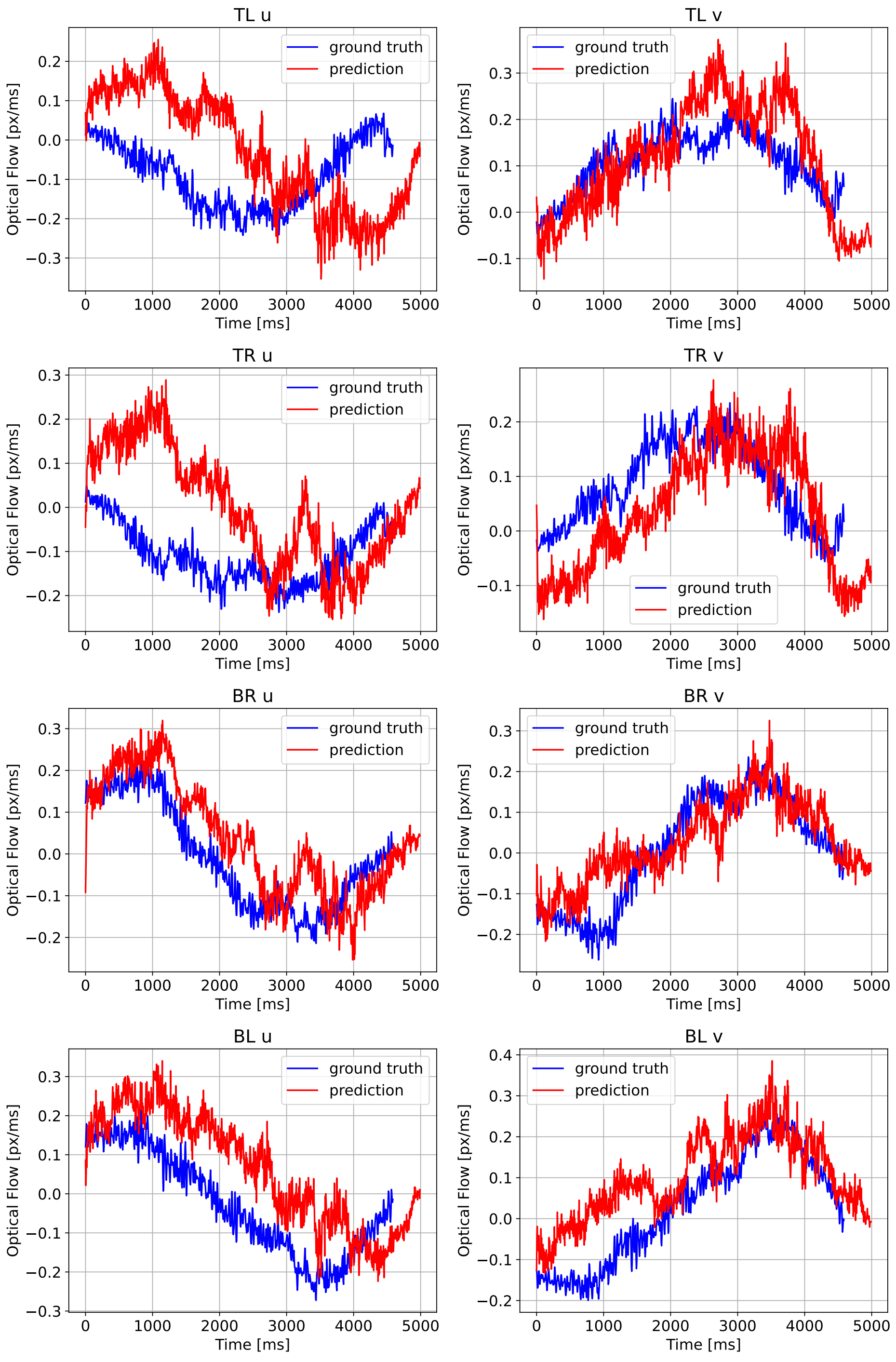


Figure A.3: RNN-2-S optical flow prediction.

A.1.4. IF-2-S

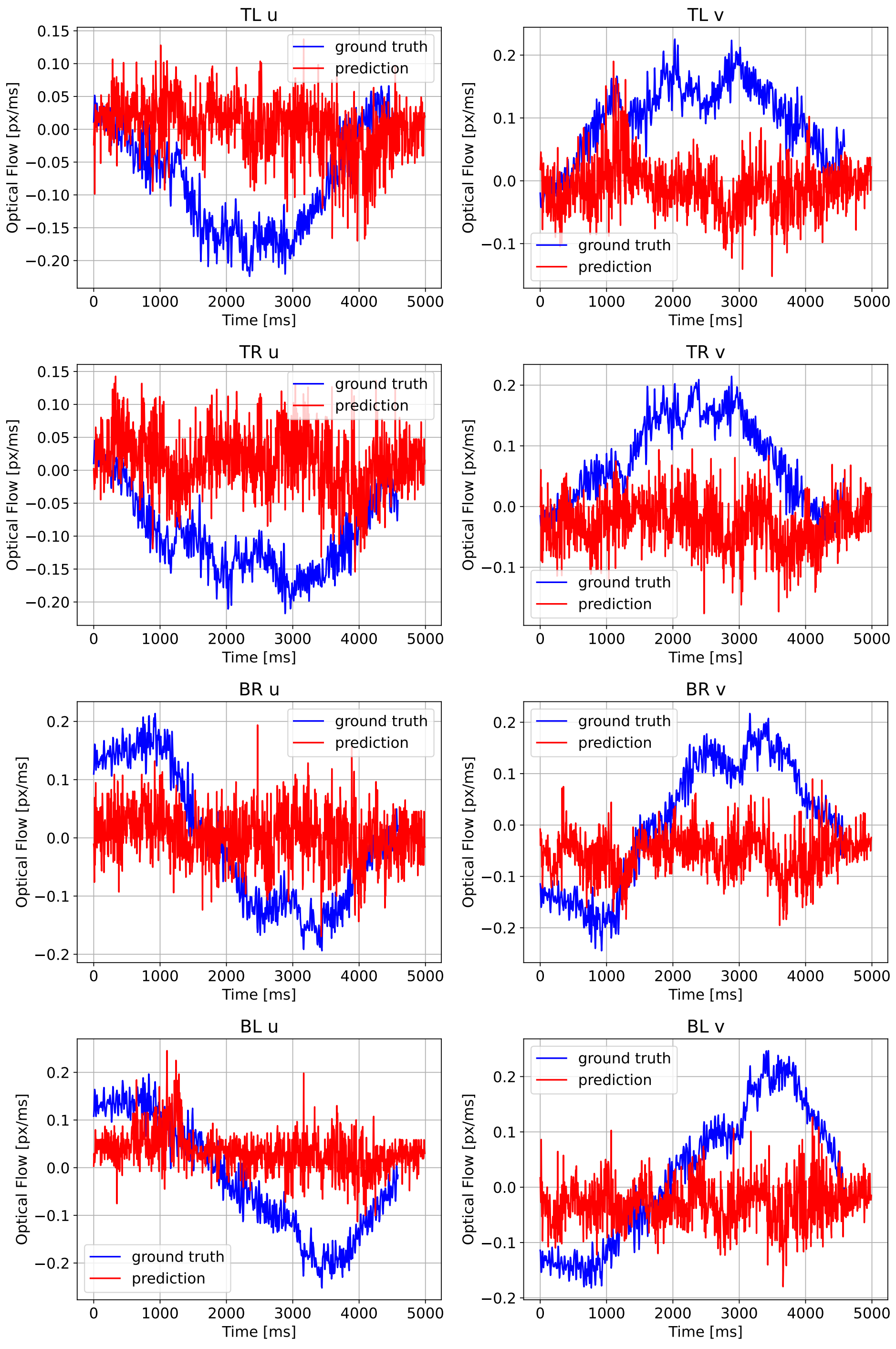


Figure A.4: IF-2-S optical flow prediction.

A.1.5. RNN-2-C

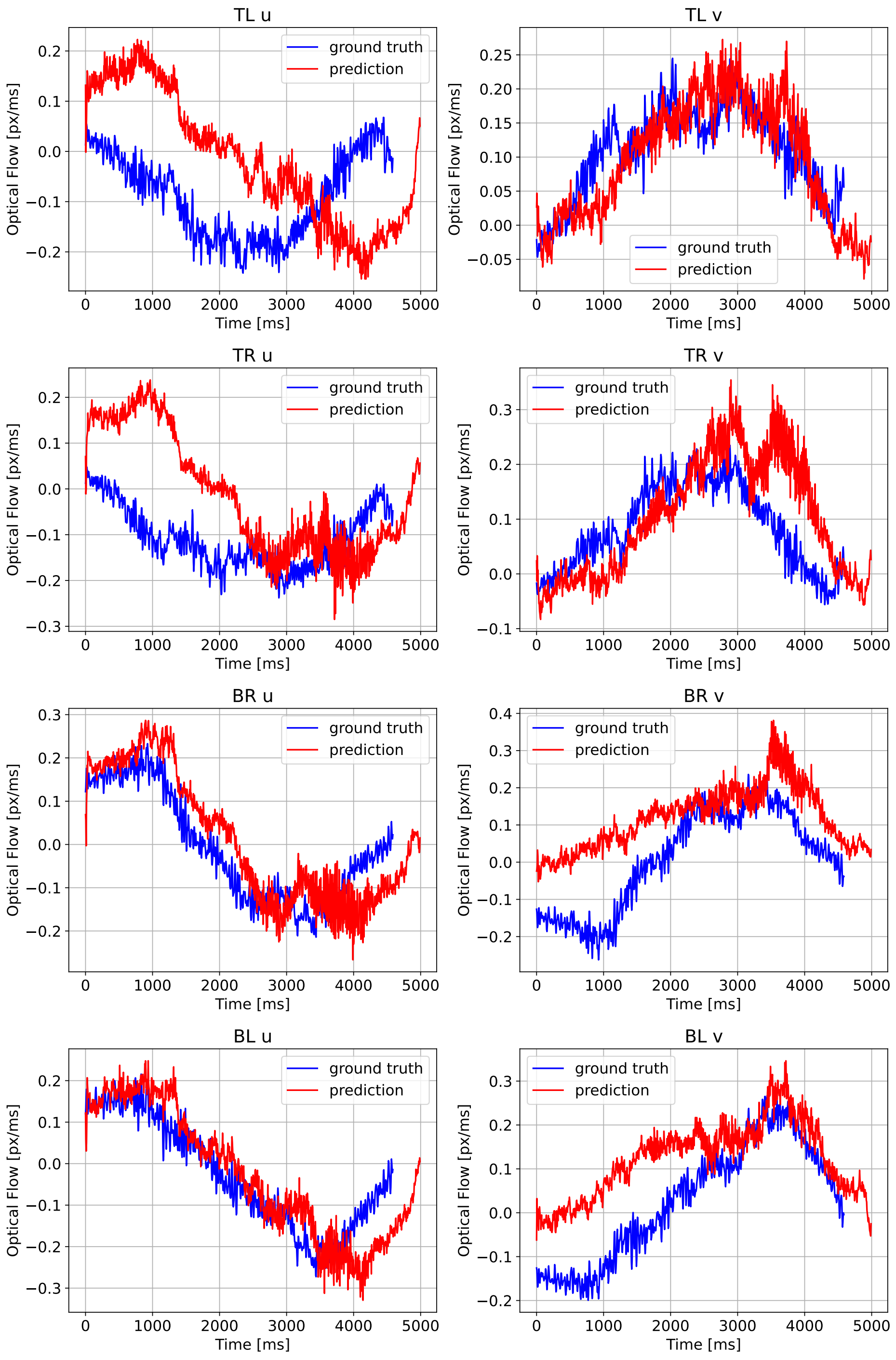


Figure A.5: RNN-2-C optical flow prediction.

A.1.6. IF-2-C

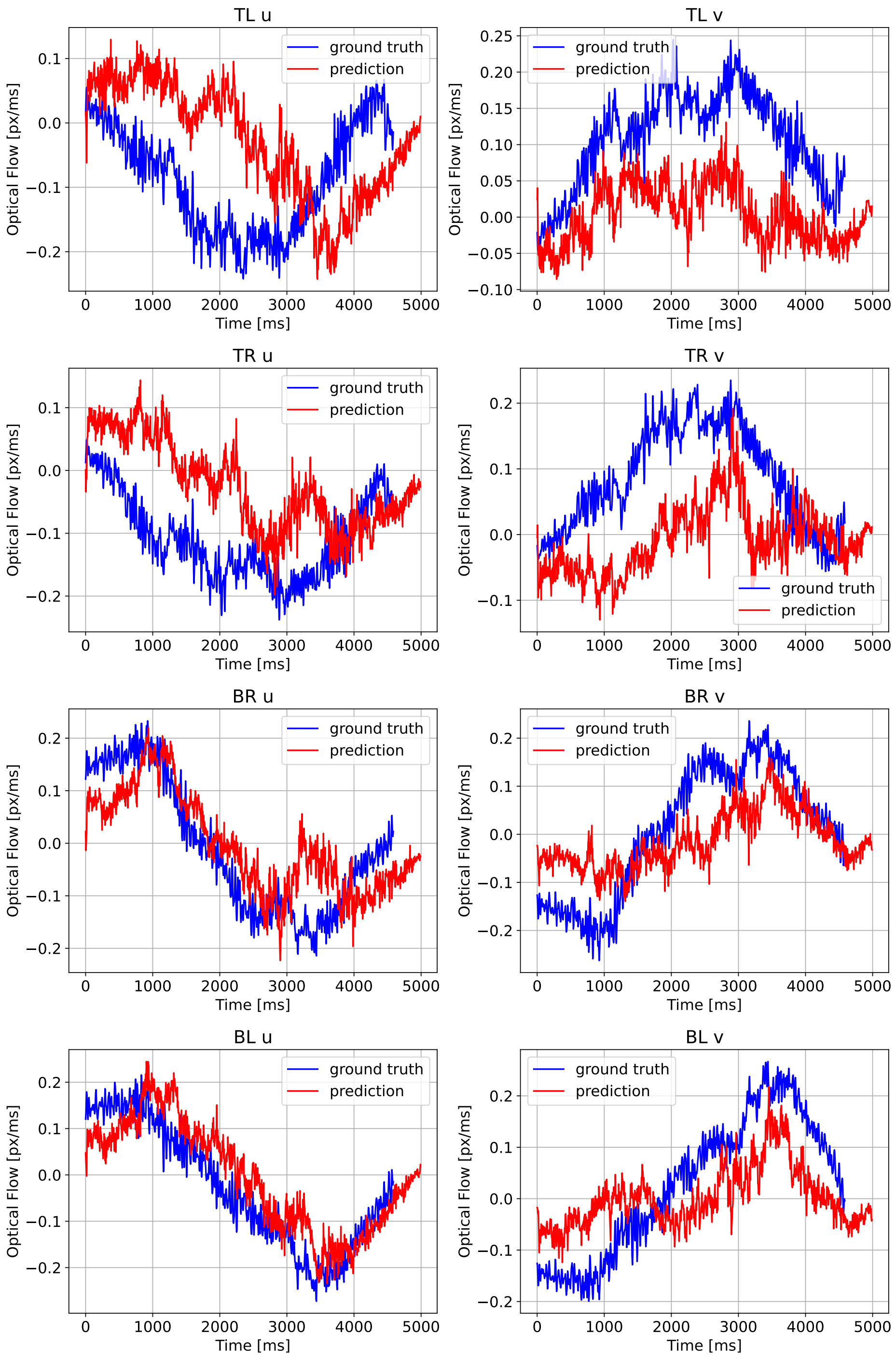


Figure A.6: IF-2-C optical flow prediction.

A.2. Spike Activity on Speck2e

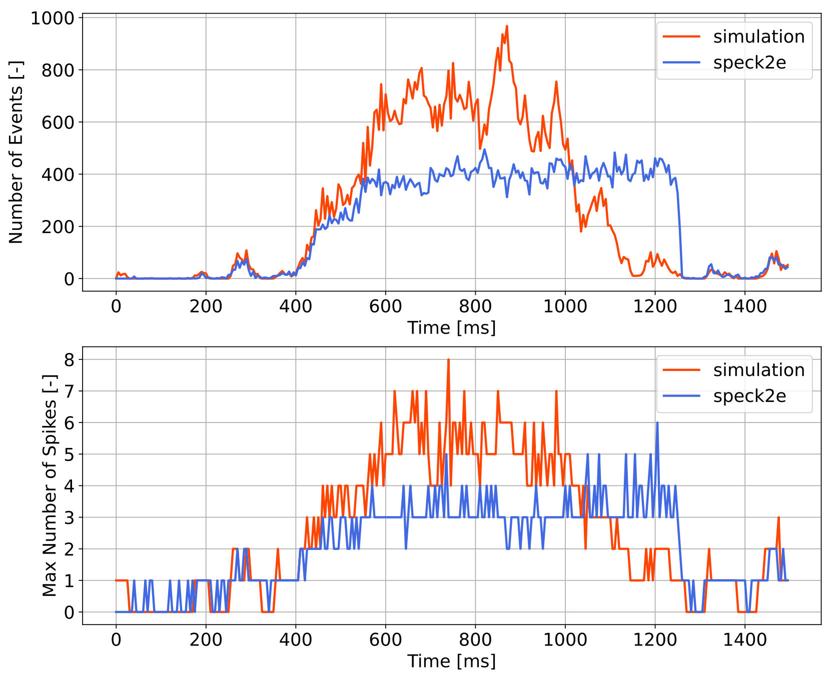


Figure A.7: Spike activity of second forward layer of encoder 0.

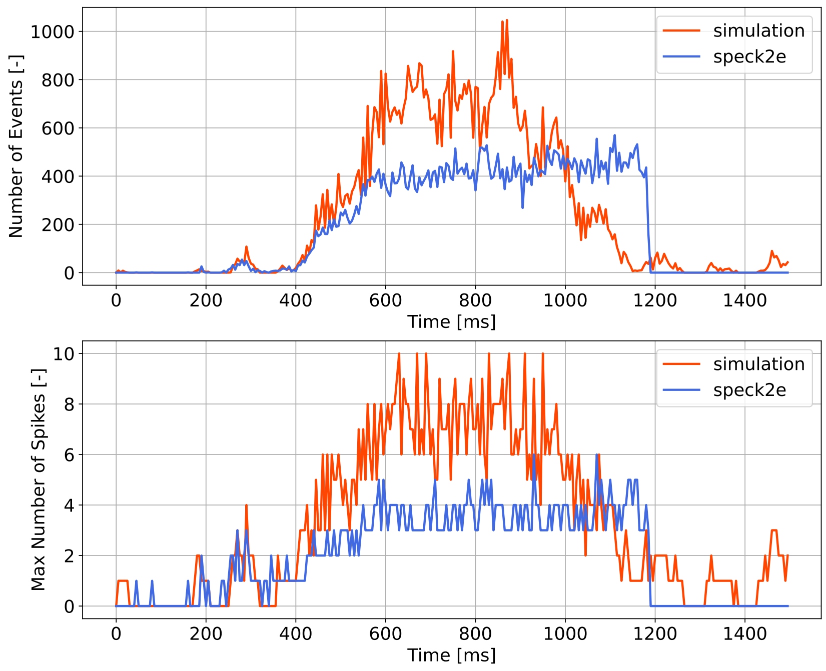


Figure A.8: Spike activity of recurrent layer of encoder 0.

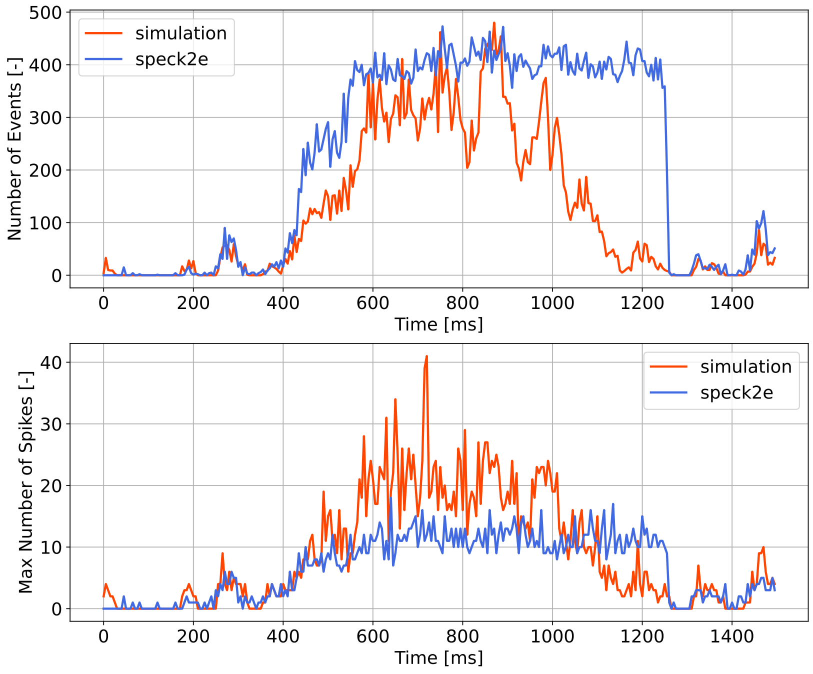


Figure A.9: Spike activity of first forward layer of encoder 1.

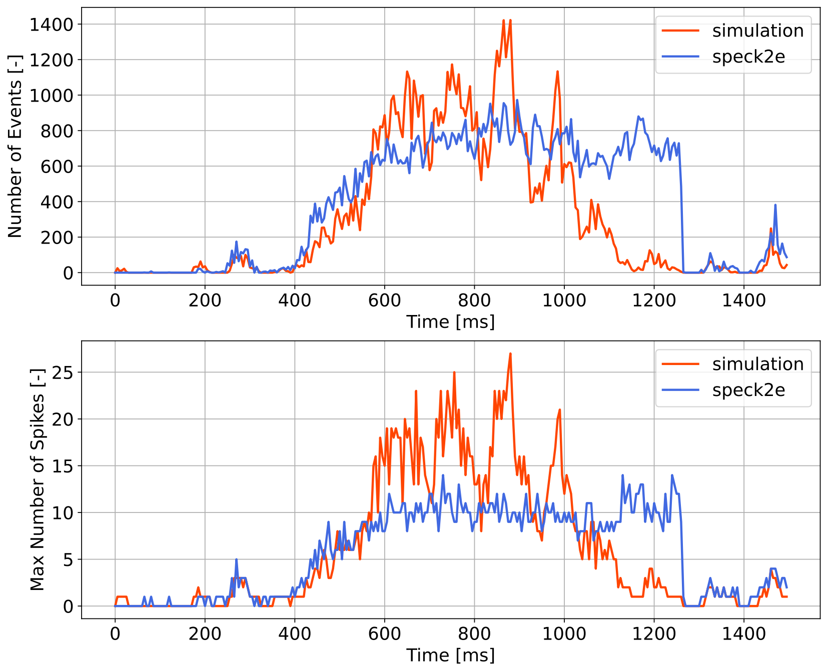


Figure A.10: Spike activity of second forward layer of encoder 1.

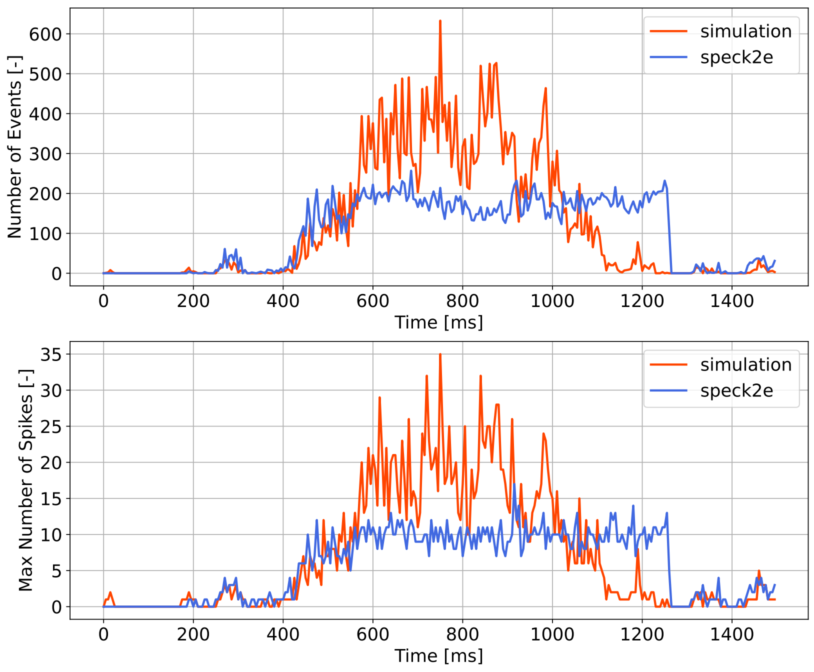


Figure A.11: Spike activity of recurrent layer of encoder 1.

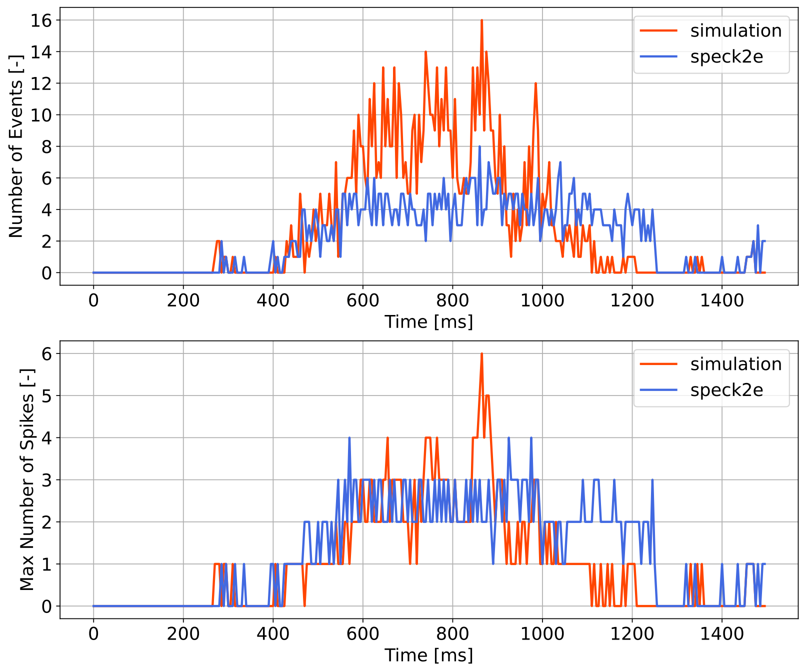


Figure A.12: Spike activity of pooling layer.

A.3. Optical Flow Color Map

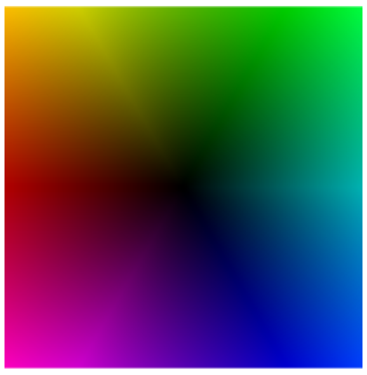


Figure A.13: Optical flow color map.