Event-based Classification with Recurrent Spiking Neural Networks on Low-end Micro-Controller Units

Boretti, Chiara; Prono, Luciano; Frenkel, Charlotte; Indiveri, Giacomo; Pareschi, Fabio; Mangia, Mauro; Rovatti, Riccardo; Setti, Gianluca

DOI
10.1109/ISCAS46773.2023.10181998

Publication date
2023

Document Version
Final published version

Published in
ISCAS 2023 - 56th IEEE International Symposium on Circuits and Systems, Proceedings

Event-based Classification with Recurrent Spiking Neural Networks on Low-end Micro-Controller Units

Chiara Boretti[‡], Luciano Prono[‡], Charlotte Frenkel[§], Giacomo Indiveri[¶], Fabio Pareschi[‡,†], Mauro Mangia[*,†], Riccardo Rovatti[*,†], and Gianluca Setti[‖,*]

[‡]DET, Politecnico di Torino, Italy - Email: {chiara.boretti, luciano.prono, fabio.pareschi}@polito.it

[§]Microelectronics Department, Delft University of Technology, The Netherlands - Email: c.frenkel@tudelft.nl

[¶]Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland - Email: giacomo@ini.uzh.ch

[*]DEI, [†]ARCES, University of Bologna, Italy - Email: {mauro.mangia, riccardo.rovatti}@unibo.it

[‖]CEMSE, King Abdullah University of Science and Technology (KAUST), Saudi Arabia - Email: gianluca.setti@kaust.edu.sa

2023 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-6654-5109-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISCAS46773.2023.10181998

***Abstract*—Due to its intrinsic sparsity both in time and space, event-based data is optimally suited for edge-computing applications that require low power and low latency. Time varying signals encoded with this data representation are best processed with Spiking Neural Networks (SNN). In particular, recurrent SNNs (RSNNs) can solve temporal tasks using a relatively low number of parameters, and therefore support their hardware implementation in resource-constrained computing architectures. These premises propel the need of exploring the properties of these kinds of structures on low-power processing systems to test their limits both in terms of computational accuracy and resource consumption, without having to resort to full-custom implementations. In this work, we implemented an RSNN model on a low-end, resource-constrained ARM-Cortex-M4-based Micro Controller Unit (MCU). We trained it on a down-sampled version of the N-MNIST event-based dataset for digit recognition as an example to assess its performance in the inference phase. With an accuracy of 97.2%, the implementation has an average energy consumption as low as 4.1 µJ and a worst-case computational time of 150.4 µs per time-step with an operating frequency of 180 MHz, so the deployment of RSNNs on MCU devices is a feasible option for small image vision real-time tasks.**

I. Introduction

Deep Neural Networks (DNNs) are structures capable of solving complex tasks with the use of trainable parameters that can be optimized to fit on low-end devices for edge computing tasks. Indeed, the use of DNNs on mobile Internet of Things (IoT) devices is of great interest, with numerous potential applications [1], [2]. Many works on DNNs applied to edge computing, tiny machine learning (tinyML) and mobile computing have been proposed, ranging from augmented reality [3], natural language processing [4], computer vision [5] to compressed sensing for biomedical signals [6].

In computer vision, event-based encoding of data is an interesting approach to solving machine vision and object detection tasks, when applied to edge computing [7]. Indeed, this type of data representation is intrinsically sparse both in space and time, thus allowing for both a large reduction of the memory footprint necessary to store the information and the minimization of the energy cost due to its acquisition. Event-based signals can be produced for example by Dynamic Vision Sensors (DVS), that are asynchronous cameras capable of responding only to the temporal contrast changes in the visual scene. The first prototype of a DVS sensor was developed in 2002 [8] and since then event-based versions of widely used datasets and new collections of samples have been created [9]–[15].

When working with event-based data, a promising approach is to use Spiking Neural Networks (SNNs) [16]–[18]. These networks leverage the possibility of representing data sparsely both in time and space because each spiking neuron is a dynamical system and thus possess a state variable, as it integrates the inputs in its membrane potential variable. In machine vision, SNNs processing event-based data have been shown to solve many tasks reliably and efficiently, ranging from classification [19], [20] or hand-gesture recognition [10], [21], [22] to object recognition [23], [24] and optical flow estimation [25].

Among different SNN models, Recurrent Spiking Neural Networks (RSNNs) [26] are of great interest, as their recurrent nature extends the simple SNN's ability to process temporal signals to time-scales that go beyond the time constants of individual elements in the network [27]. In this work we present an RSNN implemented on a low-end Micro Controller Unit (MCU) that is highly resource-constrained both in terms of memory footprint and power consumption, inspired by recently proposed RSNN hardware-constrained designs [28]. By deploying an RSNN on a low-end MCU, we demonstrate the suitability of these structures on low-cost, low-budget commercial devices, which would allow for fast prototyping and deployment times compared to more complex custom hardware implementations.

In Section II, we describe the RSNN model we are using in this implementation. In Section III, we present the N-MNIST dataset on which we test the RSNN, we explain how the RSNN is trained, and we report the network performance in terms of accuracy. In Section IV-A we describe the MCU implementation and we show the results in terms of computational time, energy consumption and memory footprint.

This study was carried out within the FAIR - Future Artificial Intelligence Research and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1555 11/10/2022, PE00000013). This manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

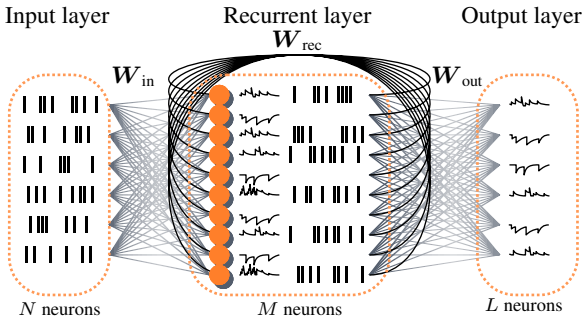

Fig. 1. Scheme of the RSNN model employed in this work, composed of an input layer (for feeding input spikes), a hidden recurrent layer composed of LIF neurons and an output layer composed of leaky integrators.

II. Recurrent SNN Architecture

The RSNN that we are using for our implementation on MCU makes use of Leaky Integrate and Fire (LIF) neurons, following the time-discrete models presented in [26] and used in the custom implementation proposed in [28].

A. Model description

The RSNN is composed of an input layer, a hidden recurrent layer and an output layer, as illustrated in Fig. 1.

The input layer is represented by a series of external input spikes that are fed to the interconnections leading to the recurrent layer. These input spikes are encoded as a series of vectors $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^T$ defined for any discrete time step $t \in \{1, \cdots, T\}$, with $\boldsymbol{x}^t \in \{0,1\}^N$ where N is the number of input neurons. The value 1 represents a spike at time t, while 0 indicates the absence of activity.

Conversely, the LIF neurons in the hidden recurrent layer are entities retaining membrane potentials $\boldsymbol{v}^t \in \mathbb{R}^M$ which can generate spiking outputs $\boldsymbol{z}^t \in \{0,1\}^M$, where M is the number of recurrent neurons. The potentials $\boldsymbol{v}^t$ are updated at each time step following the rule

$$v_j^{t+1} = \alpha v_j^t + \sum_{i=1}^{N} w_{ji}^{\text{in}} x_i^t + \sum_{i=1, i\neq j}^{M} w_{ji}^{\text{rec}} z_i^t - \theta z_j^t \quad \text{for } j \in 1, \ldots, M \tag{1}$$

where v_j^t is the value of the potential of the j-th neuron, $\alpha \in (0,1)$ is a damping factor, w_{ji}^{in} is the value at row j and column i of the input weights matrix $\boldsymbol{W}^{\text{in}}$, x_i^t is the i-th value of input spikes vector $\boldsymbol{x}^t$, w_{ji}^{rec} is the value at row j and column i of the recurrent weights matrix $\boldsymbol{W}^{\text{rec}}$, z_i^t is the i-th value of output spikes vector $\boldsymbol{z}^t$ and θ is the firing threshold parameter.

Moreover, the spiking outputs generated by the hidden recurrent neurons behave according to

$$z_j^t = \mathcal{H}\left(v_j^t - \theta\right) \quad \text{for } j \in 1, \ldots, M \tag{2}$$

where $\mathcal{H}(\cdot)$ is the step Heaviside function defined as

$$\mathcal{H}\left(v_j^t - \theta\right) = \begin{cases} 0 & \text{for } v_j^t \leq \theta \\ 1 & \text{for } v_j^t > \theta \end{cases} \tag{3}$$

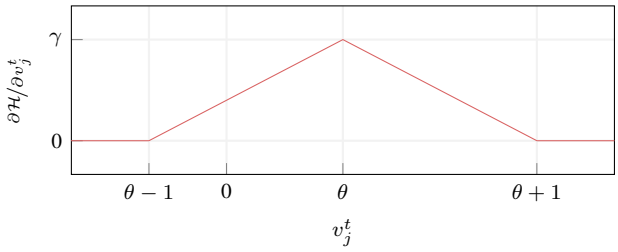

Fig. 2. Plot of the pseudo derivative function in (5).

The damping term α of (1) has the effect of decreasing the membrane potential exponentially over time, such that in absence of input spikes, it leaks to zero.

Then, the second and third terms on the right-hand-side of (1) integrate the input spikes $\boldsymbol{x}^t$ and the hidden recurrent spikes $\boldsymbol{z}^t$, respectively. When a spike is present, the value of the corresponding weight is added to the neuron potential. While the input spikes are sent to the network from an external source, the hidden recurrent spikes are generated from (2) and then fed back to hidden neurons through recurrent connections, excluding self-recurrence.

Finally, the Heaviside function in (2) simply describes the firing mechanism of the neuron: when the neuron potential grows bigger than the threshold value θ, a spike is produced (i.e., $z_j^t = 1$) and the neuron potential is reset, thanks to the presence of the last term on the right-hand-side of (1).

The output layer of the network comprises neurons modeled as non-firing leaky integrators with dynamics determined by the following equation:

$$y_k^{t+1} = \kappa y_k^t + \sum_{i=0}^{L} w_{ji}^{\text{out}} z_i^t \quad \text{for } k \in 1, \ldots, L \tag{4}$$

where y_k^{t+1} is the k-th output of the network, $\kappa \in (0,1)$ is a damping factor and w_{ji}^{out} is the value at row j and column i of the output weights matrix $\boldsymbol{W}^{\text{out}}$. The output neuron is fed by the output spikes of the hidden recurrent layer $\boldsymbol{z}^t$.

The damping factors α and κ define the leakage time constants of the recurrent and output layer units, respectively τ_{rec} and τ_{out}, as $\alpha = \exp\left(-\Delta t/\tau_{\text{rec}}\right)$ and $\kappa = \exp\left(-\Delta t/\tau_{\text{out}}\right)$, where Δt is the time interval between two discrete time-steps, i.e. the temporal resolution.

The RSNN is trained with the Back Propagation Through Time (BPTT) algorithm, it is thus important that the gradients are defined everywhere. As (2) introduces a discontinuity, a pseudo derivative function is employed as suggested in [26], defined as

$$\frac{\partial \mathcal{H}}{\partial v_j^t} \triangleq \gamma \max\left(0, 1 - \left|v_j^t - \theta\right|\right) \tag{5}$$

where γ is a factor controlling the strength of the pseudo derivative, as illustrated in Fig. 2.

III. Dataset and Performance

We employ the RSNN model to solve an event-based classification task, in the domain of event-based image processing

applications. Training and preliminary tests are performed off-line, within a PyTorch-based framework.

A. N-MNIST event-based dataset for digit recognition

The N-MNIST dataset [29] is the neuromorphic event-based version of the well-known MNIST dataset [30]. It is composed of a total of $70\,000$ samples ($55\,000$ for training, 5000 for validation and $10\,000$ for testing). Each sample is labeled with a number from 0 to 9 and is a collection of events with a duration of about $350\,\mathrm{ms}$, a time resolution of about $1\,\mu\mathrm{s}$ and a spatial resolution of 34×34 pixels. Each event is represented as positive ($+1$) or negative (-1), depending on the direction of variation of the associated pixel. The way events are collected takes inspiration from the biological phenomenon known as saccade, defined as a quick and simultaneous movement of the eye between multiple phases of fixation [31]. In particular, each digit image from the MNIST dataset is displayed on a monitor and then recorded by a motor-driven DVS camera, moving in three "saccadic" movements, along different directions in a triangular alignment.

For this demonstration, the resolution of the recordings is down-sampled to 17×17 with a remapping of all the events in order to decrease the number of input neurons required in the RSNN and consequentially to reduce the memory footprint and computational effort on MCU. For the same reason, the number of time-steps T is reduced to 300 by discarding any event eventually exceeding $300\,\mathrm{ms}$ and by down-sampling the time resolution to $\Delta t = 1\,\mathrm{ms}$, i.e., the events contained in each $1\,\mathrm{ms}$ time bin are merged together. This value of time resolution is in line with most of the SNN tasks, that typically do not benefit from lower values of Δt [21], [28], [32].

B. Training setup and preliminary performance

We train and test an RSNN with $M = 100$ recurrent neurons and $L = 10$ output neurons, as the number of classes to be recognized. The hyperparameters selected for the network are $\theta = 0.6$, $\tau_{\text{rec}} = 250\,\mathrm{ms}$, $\tau_{\text{out}} = 20\,\mathrm{ms}$ and $\gamma = 0.3$.

To account for the input event polarity, a pair of input neurons is associated to each pixel in the N-MNIST dataset: one neuron for the positive events and the other for the negative events associated with that position. Therefore, we set the number of input neurons of the RSNN to $N = 578$, defined as the number of pixels in the sample (17×17) multiplied by two to include both positive and negative events. An example of input sample is illustrated in Fig. 3.

The RSNN is trained following a standard BPTT procedure [33] with the Adam optimizer [34] and a cross-entropy (CE) loss function between the output potentials $\boldsymbol{y}^t$ and the target $\boldsymbol{y}^t_{\text{true}}$ label averaged for all the time-steps $t \in \{1, \dots, T\}$. For validation/test, the selected class is considered as the one corresponding to the output neuron with the highest average output potential.

In order to minimize the spiking activity of the recurrent neurons, a regularization contribute is added to the loss function computed as the L^2 norm of the spike-trains generated by the recurrent neurons (i.e., $\boldsymbol{z}^t$), averaged for all time steps. By introducing this contribution, the number of spikes generated while inferring the complete test set is reduced by about 90%.

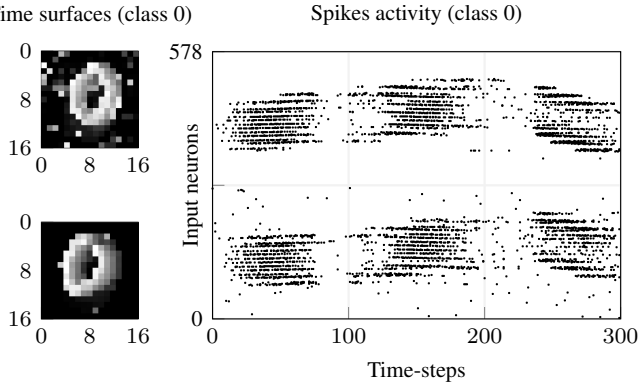

Fig. 3. An example of sample of class 0. On the left, the time surfaces for positive and negative events, obtained by integrating all the events over $300\,\mathrm{ms}$. On the right, the spiking activity of the sample.

This introduces a significant increase of the sparsity of the hidden layer activity without impeding classification accuracy.

Additionally, to use quantized parameters in the resulting network, we employ quantization-aware training techniques. Because of this, during training we perform fake quantization [35], i.e., forward pass is performed using dynamically quantized 8-bit weights.

With a batch size of 5 and a learning rate of 10^{-4}, after 50 epochs the accuracy we achieve on the test set is 97.2 %, which is near state-of-the-art performance on the N-MNIST dataset with more complex structures [36]–[39].

IV. Implementation and Performance

A. Implementation on MCU

The RSNN model described in Section II-A is implemented by software on an ARM-Cortex-M4-based MCU, namely the STM32F767ZI. This device features an SRAM of $512\,\mathrm{kB}$ and a maximum operating frequency of $216\,\mathrm{MHz}$. Spike updates are applied column-first, i.e., for each spike the potential update is applied sequentially to all neuron potentials. The leakage is applied to all neurons potentials at each time-step. C code[1] is compiled with gcc and optimized by means of -Cfast and -loop-unroll options. STM32 AXI interface is enabled along with data and instruction caches.

Each input event sent to the MCU is encoded in 32 bits, with 16 bits used for the input neuron address and 16 bits that indicate the number of time steps from the previous spike (that can be associated with any input neuron). Conversely, recurrent spikes are generated internally at each time step and are encoded as an array of 8-bit values indicating the recurrent neuron address.

Internally, the weights are encoded with 8 bits (sign bit with 7 fractional bits) while the neuron potentials are encoded with 32 bits (sign bit, 16 integer bits and 15 fractional bits). Given the bits alignment used for neuron potentials, no overflow detection mechanism is required as the numerical range is far greater than what can be reached in practice.

[1]GitHub repository at https://github.com/SSIGPRO/ucrsnn.

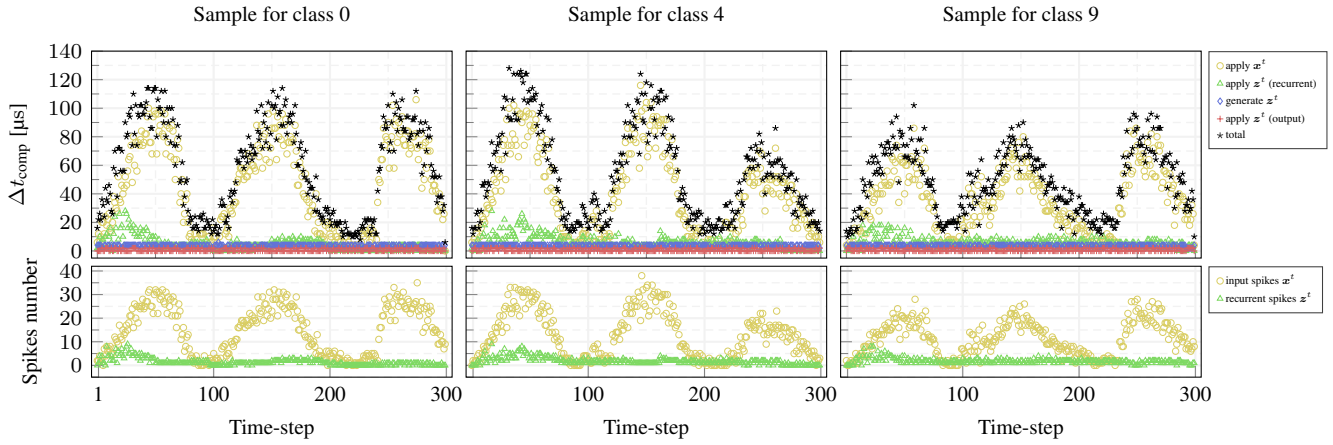

Fig. 4. Computational time per time-step Δt_c for example inferred samples with $f_{\text{CLK}} = 144\,\text{MHz}$, compared with the input and recurrent spiking activity. The computational time is split in different contributions, and the contributions due to the application of the leakage are not shown as they are below 1 µs. The greatest contributions are due to the input spikes whose number is high compared to the recurrent spikes. Their trend is coherent with the three movements performed by the DVS camera while recording N-MNIST.

B. Performance

In order to assess the performance of the RSNN implementation on MCU, the full test set of 10 000 samples of 300 time-steps each is inferred, resulting in the same accuracy measured in Section III, and the computational time required by each operation is measured with an on-chip counter.

Fig. 4 shows the computational time per time-step Δt_c for different samples with an operating frequency $f_{\text{CLK}} = 144\,\text{MHz}$, along with the spiking activity of the model. Computational time is split among its different contributions, i.e., application of leakages to recurrent and output neurons, application of input spikes and recurrent spikes to the recurrent neurons and to the output neurons, and recurrent spikes generation. Most of the computational time is due to the application of the spikes to the neuron potentials, which is of course proportional to the number of spikes to be integrated. On the other hand, other contributions are almost negligible.

Tab. I shows the average and worst-case power consumption of the MCU, the computational time for a single time-step, and the energy consumed by the device for the update of a time-step. Power consumption is calculated with the values declared in the datasheet with an internal supply voltage of 1.2 V, STM32 AXI interface and cache enabled, all peripherals disabled[2]. The average and worst-case computational times are evaluated over the time-steps for the samples, while the energy consumption is simply the product of power consumption and time.

Worst-case computational times are about 1 ms for the lowest operational frequency tested, while it can reach values as low as 150.4 µs for a higher f_{CLK} of 180 MHz, enabling the use of this type of neural network models on low-cost, low-resource devices with fast deployment times when compared to custom hardware implementations.

[2]Power consumption is actually the sum of two contributions, the principal one given by the internal power supply, the other one from the external power supply of 1.7 V that absorbs and average of 1 mA and 2 mA in the worst-case.

TABLE I
POWER CONSUMPTION OF STM32F767ZI, WITH THE COMPUTATIONAL TIME AND ENERGY CONSUMPTION FOR A SINGLE TIME-STEP UPDATE OF THE RSNN

f_{CLK}	P_{typ}	$\Delta t_{\text{c, avg}}$	$E_{\text{c, avg}}$
25 MHz	13.1 mW	292.5 µs	3.8 µJ
60 MHz	26.8 mW	121.5 µs	3.3 µJ
144 MHz	58.7 mW	50.1 µs	3.1 µJ
169 MHz	83.6 mW	43.5 µs	3.7 µJ
180 MHz	99.5 mW	40.6 µs	4.1 µJ
f_{CLK}	P_{max}	$\Delta t_{\text{c, worst-case}}$	$E_{\text{c, worst-case}}$
25 MHz	18.5 mW	1082.4 µs	21.9 µJ
60 MHz	34.1 mW	444.7 µs	15.9 µJ
144 MHz	70.1 mW	207.0 µs	14.8 µJ
169 MHz	98.1 mW	161.3 µs	16.1 µJ
180 MHz	116.5 mW	150.4 µs	17.8 µJ

Finally, the memory footprint of the RSNN model on MCU is mostly due to its parameters, which, in this work, use 68.8 kB of space.

V. CONCLUSION

We trained and tested an RSNN model to solve an event-based machine vision task, using the N-MNIST dataset for digit recognition. The implementation of the models on a low-cost, low-power MCU device with strong resource constraints, successfully validated the approach proposed. The benefits of this approach include the use of standard commercial devices and fast prototyping and deployment times. With an accuracy of 97.2%, the worst-case computational time per time-step can be as low as 150.4 µs with an average energy consumption per time-step of 4.1 µJ at $f_{\text{CLK}} = 180\,\text{MHz}$. This shows that it is possible to employ software-based RSNNs for inference in real-time image vision applications using resource constrained hardware.

REFERENCES

[1] M. Merenda, C. Porcaro, and D. Iero, "Edge Machine Learning for AI-Enabled IoT Devices: A Review," *Sensors*, vol. 20, no. 9, p. 2533, Jan. 2020. doi:10.3390/s20092533

[2] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019. doi:10.1109/JPROC.2019.2921977

[3] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *12th annual international conference on Mobile systems, applications, and services (MobiSys '14)*, Jun. 2014, pp. 68–81. doi:10.1145/2594368.2594383

[4] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma, "FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network," in *32nd International Conference on Neural Information Processing Systems (NIPS'18)*, Dec. 2018, pp. 9031–9042.

[5] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The Design and Implementation of a Wireless Video Surveillance System," in *21st Annual International Conference on Mobile Computing and Networking (MobiCom '15)*, Sep. 2015, pp. 426–438. doi:10.1145/2789168.2790123

[6] L. Prono, M. Mangia, A. Marchioni, F. Pareschi, R. Rovatti, and G. Setti, "Deep Neural Oracle With Support Identification in the Compressed Domain," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 458–468, Dec. 2020. doi:10.1109/JETCAS.2020.3039731

[7] G. Gallego *et al.*, "Event-Based Vision: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 154–180, Jan. 2022. doi:10.1109/TPAMI.2020.3008413

[8] J. Kramer, "An on/off transient imager with event-driven, asynchronous read-out," in *2002 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2, May 2002, pp. II–II. doi:10.1109/ISCAS.2002.1010950

[9] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 X 128 120db 30mw asynchronous vision sensor that responds to relative intensity change," in *2006 IEEE International Solid State Circuits Conference - Digest of Technical Papers*, Feb. 2006, pp. 2060–2069. doi:10.1109/ISSCC.2006.1696265 ISSN: 2376-8606.

[10] A. Amir *et al.*, "A Low Power, Fully Event-Based Gesture Recognition System," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 7388–7397. doi:10.1109/CVPR.2017.781 ISSN: 1063-6919.

[11] Y. Park *et al.*, "A Wireless Power and Data Transfer IC for Neural Prostheses Using a Single Inductive Link With Frequency-Splitting Characteristic," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 15, no. 6, pp. 1306–1319, Dec. 2021. doi:10.1109/TBCAS.2021.3135843

[12] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "CIFAR10-DVS: An Event-Stream Dataset for Object Classification," *Frontiers in Neuroscience*, vol. 11, 2017. doi:10.3389/fnins.2017.00309

[13] C. Scheerlinck, H. Rebecq, T. Stoffregen, N. Barnes, R. Mahony, and D. Scaramuzza, "CED: Color Event Camera Dataset," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jun. 2019, pp. 1684–1693. doi:10.1109/CVPRW.2019.00215 ISSN: 2160-7516.

[14] M. Gehrig, W. Aarents, D. Gehrig, and D. Scaramuzza, "DSEC: A Stereo Event Camera Dataset for Driving Scenarios," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4947–4954, Jul. 2021. doi:10.1109/LRA.2021.3068942

[15] E. Perot, P. de Tournemire, D. Nitti, J. Masci, and A. Sironi, "Learning to Detect Objects with a 1 Megapixel Event Camera," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 16 639–16 652.

[16] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, Aug. 2005. ISBN 978-0-262-54185-5 Google-Books-ID: fLT4DwAAQBAJ.

[17] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International Journal of Neural Systems*, vol. 19, no. 04, pp. 295–308, Aug. 2009. doi:10.1142/S0129065709002002

[18] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, Mar. 2019. doi:10.1016/j.neunet.2018.12.002

[19] A. Samadzadeh, F. S. T. Far, A. Javadi, A. Nickabadi, and M. H. Chehreghani, "Convolutional Spiking Neural Networks for Spatio-Temporal Feature Extraction," Jan. 2021. doi:10.48550/arXiv.2003.12346

[20] B. Han and K. Roy, "Deep Spiking Neural Network: Energy Efficiency Through Time Based Coding," in *Computer Vision – ECCV 2020*, 2020, pp. 388–404. doi:10.1007/978-3-030-58607-2_23

[21] A. M. George, D. Banerjee, S. Dey, A. Mukherjee, and P. Balamurali, "A Reservoir-based Convolutional Spiking Neural Network for Gesture Recognition from DVS Input," in *2020 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2020, pp. 1–9. doi:10.1109/IJCNN48605.2020.9206681 ISSN: 2161-4407.

[22] E. Ceolini, C. Frenkel, S. B. Shrestha, G. Taverni, L. Khacef, M. Payvand, and E. Donati, "Hand-gesture recognition based on emg and event-based camera sensor fusion: A benchmark in neuromorphic computing," *Frontiers in Neuroscience*, vol. 14, p. 637, 2020.

[23] Y. Cao, Y. Chen, and D. Khosla, "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, May 2015. doi:10.1007/s11263-014-0788-3

[24] S. Kim, S. Park, B. Na, and S. Yoon, "Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, pp. 11 270–11 277, Apr. 2020. doi:10.1609/aaai.v34i07.6787

[25] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan, "Asynchronous frameless event-based optical flow," *Neural Networks*, vol. 27, pp. 32–37, 2012. doi:https://doi.org/10.1016/j.neunet.2011.11.001

[26] G. Bellec, F. Scherr, A. Subramoney, E. Hajek, D. Salaj, R. Legenstein, and W. Maass, "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Communications*, vol. 11, no. 1, p. 3625, Jul. 2020. doi:10.1038/s41467-020-17236-y

[27] B. Yin, F. Corradi, and S. M. Bohté, "Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks," *Nature Machine Intelligence*, vol. 3, no. 10, pp. 905–913, Oct. 2021. doi:10.1038/s42256-021-00397-w

[28] C. Frenkel and G. Indiveri, "ReckOn: A 28nm Sub-mm2 Task-Agnostic Spiking Recurrent Neural Network Processor Enabling On-Chip Learning over Second-Long Timescales," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, Feb. 2022, pp. 1–3. doi:10.1109/ISSCC42614.2022.9731734 ISSN: 2376-8606.

[29] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades," *Frontiers in Neuroscience*, vol. 9, 2015. doi:10.3389/fnins.2015.00437

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. doi:10.1109/5.726791

[31] K. R. Gegenfurtner, "The Interaction Between Vision and Eye Movements," *Perception*, vol. 45, no. 12, pp. 1333–1357, Dec. 2016. doi:10.1177/0301006616657097

[32] Y. Xing, G. Di Caterina, and J. Soraghan, "A New Spiking Convolutional Recurrent Neural Network (SCRNN) With Applications to Event-Based Hand Gesture Recognition," *Frontiers in Neuroscience*, vol. 14, 2020. doi:10.3389/fnins.2020.590164

[33] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990. doi:10.1109/5.58337

[34] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 2017. doi:10.48550/arXiv.1412.6980

[35] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8BERT: Quantized 8Bit BERT," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, Dec. 2019, pp. 36–39. doi:10.1109/EMC2-NIPS53020.2019.00016

[36] R. Vaila, J. Chiasson, and V. Saxena, "Feature Extraction using Spiking Convolutional Neural Networks," in *Proceedings of the International Conference on Neuromorphic Systems (ICONS '19)*, New York, NY, USA, 2019, pp. 1–8. doi:10.1145/3354265.3354279

[37] X. Cheng, Y. Hao, J. Xu, and B. Xu, "LISNN: Improving Spiking Neural Networks with Lateral Interactions for Robust Object Recognition," in *Twenty-Ninth International Joint Conference on Artificial Intelligence*, vol. 2, Jul. 2020, pp. 1519–1525. doi:10.24963/ijcai.2020/211 ISSN: 1045-0823.

[38] S. B. Shrestha and G. Orchard, "SLAYER: spike layer error reassignment in time," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*, Red Hook, NY, USA, 2018, pp. 1419–1428.

[39] W. Zhang and P. Li, "Temporal Spike Sequence Learning via Backpropagation for Deep Spiking Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 12 022–12 033.