# Synaptic delays for temporal pattern recognition

Kévin Constantin[1], Amélie Gruel[2], and Jean Martinet[2]

[1] Dept. of Computer Science, Polytech Nice-Sophia, Université Côte-d'Azur, 930 Rte des Colles 06410 Biot, France
kevin.constantin@etu.unice.fr
[2] SPARKS (Scalable and Pervasive softwARe and Knowledge Systems), I3S CNRS, Les Algorithmes - bât. Euclide B 06900 Sophia Antipolis, France
{agruel,jmartinet}@i3s.unice.fr

**Abstract.** Spiking Neural Networks (SNNs) are a special type of artificial neural network in which neurons communicate through sequences of spikes. Unlike deep convolutional networks, spiking neurons only fire when their activation level reaches a certain threshold value, making the network asynchronous and well-suited for handling temporal data like video. SNNs do not use stochastic gradient descent and backpropagation for inference, but rather have neurons connected through synapses that use biologically-inspired learning mechanisms to update synaptic weights or delays. The development of event-based cameras, modeled after the retina, has increased the potential for SNNs to be used in computer vision. These cameras record events of significant pixel intensity changes rather than continuously measuring pixel intensity like traditional cameras, and SNNs can take advantage of their asynchronous nature to process this type of data. This paper focuses on the training part of SNNs with the addition of visual indicators on the learning of delays and weights, as well as measures on the effectiveness of the training. From this work, we will be able to verify the usability of a learned model, or compare in an objective way SNNs specialized in motion recognition.

**Keywords:** Spiking Neural Networks · Delay learning · Temporal pattern recognition.

## 1 State of the art

### 1.1 Computer Science and Biology

Understanding the human brain has always been an important subject of study in the world of research. The motivations are diverse, including understanding the mechanisms of thought, the causes of mental disorders or the mechanisms of memory, with that of learning.

It is with the objective of understanding the latter that, nearly 80 years ago, Colluch and Pitch [1] published a scientific paper on the potential functioning of human neurons, by modeling a network of neurons using electrical circuits. This was the first time that a neural network was talked about. Through this research,

they showed that a neural network could in principle compute any function that a computer could compute, suggesting that the human brain could be seen as a computational device.

Research on neural networks continued in an attempt to solve problems, and a few years later, Frank Rosenblatt [2] published the Perceptron; the first trainable neural network, consisting of an input layer of neurons, an output layer, and an intermediate layer. All these neurons are linked and weighted by weights. It is during training (i.e. learning) that these weights are automatically adjusted.

It is in this context that Maass [5] proposes an alternative to artificial neural networks by taking more inspiration from the real functioning of neurons in the brain, by proposing an architecture that makes neurons communicate not in a continuous way but in the form of an impulse (Ponulak [6]) when a threshold potential is exceeded. SNNs are born. He hypothesizes that SNNs are particularly well adapted to motion recognition using Reichardt [3] detectors, the mechanism in insects capable of detecting motion.

Neuromorphic processors [15], are designed to work with SNNs. These processors are highly parallel, which allows for great energy efficiency, and they have faster execution speeds than traditional processors. Their arrival marks a significant advancement in bio-inspired computer science.

### 1.2 Computer Vision, Spatio-Temporal Recognition and Delay Learning

Currently, computer vision tasks such as image segmentation, object detection or optical character recognition are largely based on so-called convolution models (CNN) [7]. Over the years, the proposed models offer better results, but too often at the cost of models that are always bigger and/or require a larger amount of training data than the previous models [8]. The CNNs being supervised trained models (data for which the expected output is known), the human costs for the manual annotation of the data become exorbitant.

Another more reasonable approach is to use models capable of unsupervised learning, which is what SNNs allows. Thanks to bio-inspired learning rules such as Spike-Timing Dependent Plasticity (STDP) [10], the model is able to modulate its parameters by itself according to the inputs it receives, thus solving the problem of data annotation and energy cost (offering at the same time the possibility to do embedded activities).

Thanks to this mechanism, SNNs have since been tested and used in many cases, notably for visual feature learning [9], pattern detection [11, 12].

Because of their asynchronous nature, SNNs are also particularly well adapted to the use of event-driven cameras [4] whose principle resides in the capture of luminosity changes in the screen, returning only information on the location of the pixels that have been altered, as well as a very precise time associated with these changes.

Oudjail's [13] paper focuses on the ability to recognize motion patterns with data encoded in AER (Adress Event Representation) format, namely used in

event-driven cameras. They showed fast convergence for 2 of the 3 tested patterns.

Nadafian [14] presents a method for adjusting synaptic delays in SNNs in an unsupervised manner that allows neurons to learn to recognize repetitive spatio-temporal movements.

Real applications are being studied, in particular for use in autonomous vehicles [16]. Installed on a neuromorphic processor, the SNNs correspond perfectly to the constraints of embedded systems (power consumption, responsiveness, size).

## 2 Material and methods

This work follows and is based on the code written in Python by Hugo Bulzomi, which implements the architecture presented by Nadafian [14] for training SNNs for motion recognition by convolution layers in an unsupervised way.

For the construction of the model, the code uses PyNN [18], an interface adapted for Python using a lower level simulator; NEST [19], for modeling (among others) neurons and synapses.

The training of the model is based on input data corresponding to well-defined movements (4 directions: $\pm 135°$ and $\pm 45°$) which are randomly generated at a fixed time interval.

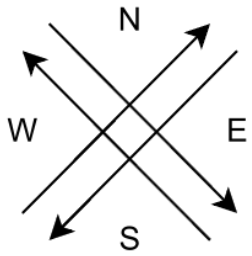


**Fig. 1.** Visual representation of the 4 directions : $\pm 135°$ and $\pm 45°$

### 2.1 Visualization of delays and weights

To visually appreciate the potential convergence of the different convolution layers, we have taken the values of delays and weights for each convolution layer, to place them in a color matrix gradient from yellow to red, where yellow corresponds to a value greater than red. The delays are given in milliseconds (ms), and the weights have no particular unit.

It is hoped that during training and successive iterations, it will be possible to observe, (if there is a convergence of the different convolution layers) a decrease of the delays (towards red) and an increase of the weights (towards yellow) at

the level of the pattern in the kernel, and an increase of the delays (towards yellow) and a decrease of the weights (towards red) in the rest of the kernel.

### 2.2 Training phase efficiency : Metrics

In order to verify in a more objective and scientific way the convergence of the convolution kernels towards the recognition of the motions given in input, metrics were set up, those traditionally used in classification tasks: precision, recall and score-f1 whose equations and meaning are reminded below.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

$$\text{F1-Score} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

Where :

- TP (True Positive) : Spike where it was supposed to be
- FP (False Positive) : Spike but the input motion did not correspond
- FN (False Negative) : No spike while the input motion corresponded

- Precision : Correct spike rate compared to those found
- Recall : Did we find every spike we should have found ?

The objective will be to calculate these metrics for each convolution layer and thus have an overview of the efficiency of the training. The general idea is to observe the spikes produced at the input and output of each convolution filter of the model and to compare the expected (input spikes) and the obtained (output spikes).

**Link a convolution layer to a motion :** The first thing to do was to be able to relate a convolution layer to the direction in which the layer has specialized. We know that a layer has specialized when its kernel of delays and weights (we were particularly interested in delays) represents a pattern, that this pattern has significantly lower delays than the delays outside this pattern, and that within the pattern the delays are not homogeneous but have a progressive trend.

Each motion is encoded in a function that checks the above conditions, and thus each convolution layer is tested for each motion, to be able to associate in the form of a key-value, the convolution layer and the associated motion.

It has been assumed so far that the convolution kernels are specialized in motion recognition. But it can happen that in case of bad training (for various reasons like movements to represent too complex, not enough time to train to converge ...) that the delay kernel of a layer does not represent anything in particular (or represents several movements at the same time). In this case, we will say that the convolution layer represents an indeterminate motion, and that it cannot be evaluated by the metric.

**Gather input data :** When generating the input data, we know what type of motion is given to the model (but the model does not know it because it is not supervised). We will therefore collect this information with the temporal information (when the motion starts and ends) to finally have a set of temporal intervals associated with the motions they represent.

**Gather output data :** From the collected time intervals, we will extract the spikes that occurred on each convolution layer in this same interval, allowing a margin of a few milliseconds beyond the interval to take into account the delays of the kernel in particular.

**Metrics computation :** For each time interval given by the input and the direction it represents, we check for each layer if we have a spike or not (previous part). If there is no spike we make sure that the layer does not represent the real direction of the input (part on the link between convolution layer and motion). If there is a spike, we record a false negative for this layer.

If there is at least one spike, we also check that the layer where there is the spike corresponds to the input direction, if it is the case then it is a true positive, otherwise a false positive.

Finally we gather all the records (TP, FP, FN) for each layer and we calculate the metrics with the equations given above.

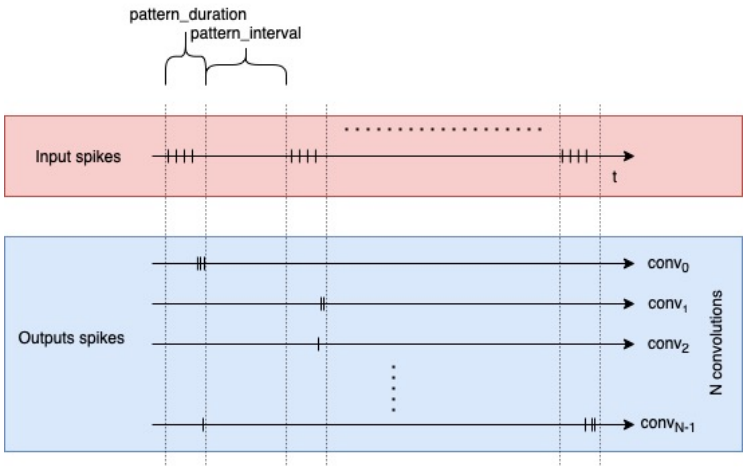


**Fig. 2.** Representation of time intervals and associated spikes

# 3 Results

## 3.1 Delays and weights training results

Here are the results obtained at the end of the training of the different models tested.

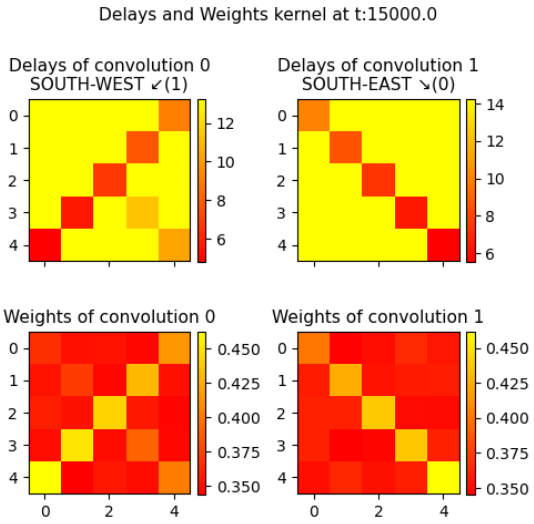


**Fig. 3.** Kernel values of 2 convolution layers training on -135° and -45° directions.

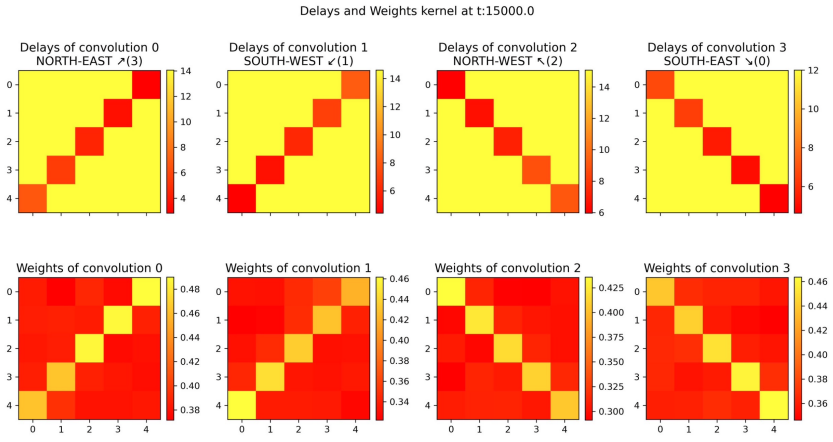


**Fig. 4.** Kernel values of 4 convolution layers training on ±135° and ±45° directions.

In order to verify the hypotheses on the evolution of delays and weights, here are the evolution of delays and weights of 2 pixels on the convolution layer 0 (previous figure), the first one on the top right which is not on the pattern, and the second one on the end of the pattern.

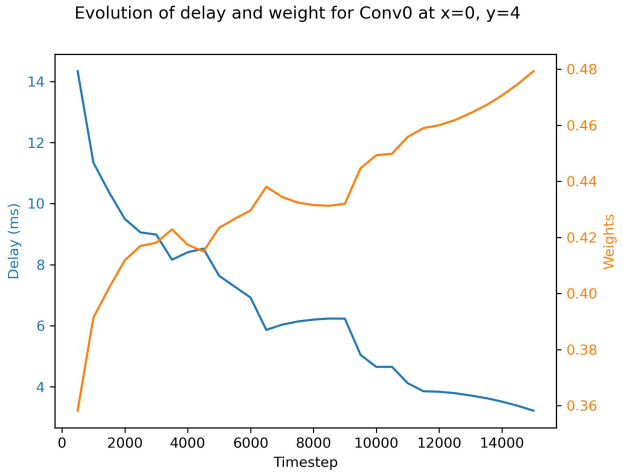


**Fig. 5.** Delay and weight values on Conv0 at (0;4), in the pattern.

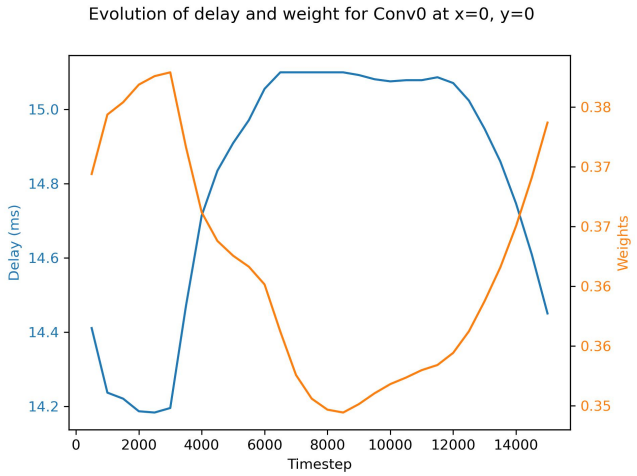


**Fig. 6.** Delay and weight values on Conv0 at (0;0), outsite the pattern.

### 3.2 Metrics

Based on these same models, here are the results concerning the metrics.

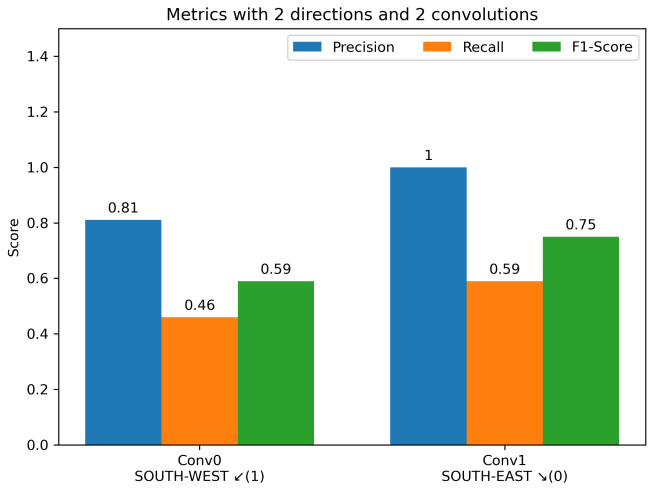


**Fig. 7.** Metrics for an SNN model with 2 convolution layers and training on -135° and -45° input directions.

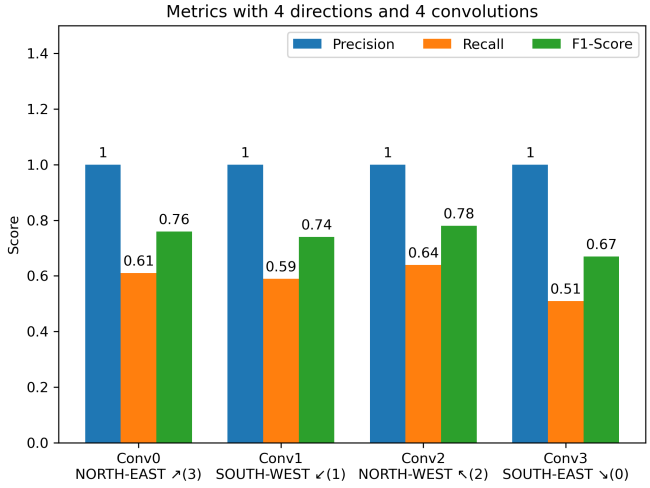


**Fig. 8.** Metrics for an SNN model with 4 convolution layers and training on ±135° and ±45° input directions.

## 4 Discussion

The visualization of the convolution kernels, shows well the convergence of the different layers of convolution towards a pattern which is proposed to him.

But during the experiments it was observed, and sometimes in a more pronounced way, what we see on the figure 3.1 at the level of convolution 0, a double specialization like below.

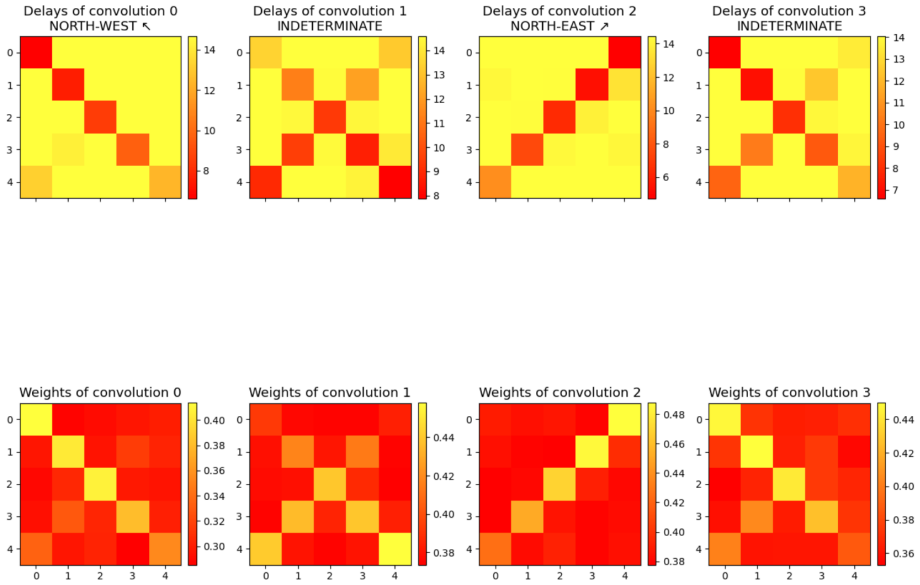


**Fig. 9.** Kernel values of an SNN with 4 convolution layers, training on ±135° and ±45° directions.

This behavior is normally supposed to be avoided thanks to what is called lateral inhibition. Lateral inhibition is a mechanism for regulating neuronal activity that consists of inhibiting the activity of neighboring neurons when a given neuron is active.

When a neuron is spiking, it sends an inhibition signal to its neighboring neurons to reduce their activity and prevent them from spiking as well. This lateral inhibition normally limits the neuronal response to specific stimuli and thus improves the specialization of neurons to recognize distinct movements.

Some feedbacks blame PyNN for sometimes not doing its job. A solution would be to implement this lateral inhibition yourself.

Regarding the evolution of the delays and weights, we find the expected evolution of the latter when the position of the pixel is important or not.

But there is an interesting thing concerning the pixel outside the pattern, its delay increases and its weight decreases but we observe towards the end a

reversal of the trends (which remains to be relativized compared to the values of the pixel on the pattern, there is a big margin of difference at the level of the delay in particular). If the training had continued, would this have been prolonged or is it temporary? Same question about the pixel on pattern, will there be a glass ceiling for the delay or will it tend to infinity if the training is?

According to the metrics, current learning is almost not wrong on the prediction of a given motion in input (accuracy). On the other hand, we observe that the models fail quite often, almost half the time for some layers, to recognize a motion when there is one (recall).

### 4.1 Future works

**Test on noisy input data :** For the moment, the input data generated represents "perfect" movements, i.e. there are no extraneous elements that could hinder the training of the network. It may be interesting, as it is probably more realistic with reality, to train the network on noisy data and to check if the convolution layers still manage to specialize for a particular motion.

**Increase the number of directions :** For the purpose of using the model for real data, we can easily imagine that we are in a world with more than 4 directions. A model taking into account 8 directions (adding: 90° North, -90° South, 0° East, 180° West) is a good start.

Eventually, if we want to observe more complex directions like 60°, 30° or 120°, we will quickly be blocked by the size of the convolution kernel (which is currently 5*5). Increasing the size of the convolution kernel is unavoidable if we want to represent more directions, and we will then have to find a kernel size large enough for what we want to achieve, but small enough to reduce the computation time.

A first implementation was done for 8 directions, and the results quickly showed the difficulty for the convolution layers to specialize in a particular motion. We think that this is due to the size of the convolution kernel (5*5) and that by increasing it, we should have more satisfactory results.

**Apply the model to real-life input data :** It would be interesting to confront the network to real data (data from an event camera) on several aspects.

The first one is to try to learn only from these data and to look at the quality of the learning. Will there be convergence of certain types of movement or on the contrary will the learning have totally failed?

Second aspect, from a model trained on generated data (the current case) make inference with the model learned on real data, and check the presence of spike on the layers concerned by the movements produced on the camera.

**Implement the model in a neuromorphic processor :** To improve the speed of execution of the code in particular, on a SpiNNaker [17] processor.

## References


1. McColloch, W.S., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. In: Bulletin of Mathematical Biophysics, vol. 5, pp. 115–133. (1943). https://doi.org/10.1007/BF02478259
2. Rosenblatt, F.: The Perceptron - A perceiving and recognizing automaton. In: Cornell Aeronautical Laboratory, 85-460-1. (1957).
3. Reichardt, W.: Evaluation of optical motion information by movement detectors. In: J. Comp. Physiol, vol. 161, pp. 533–547. (1987). https://doi.org/10.1007/BF00603660
4. Lichtsteiner, P., Posch, C., Delbruck, T.: A 128× 128 120 dB 15 us Latency Asynchronous Temporal Contrast Vision Sensor. In : IEEE Journal of Solid-State Circuits, vol. 43, pp. 566–576 (2008). https://doi.org/10.1109/JSSC.2007.914337
5. Maass, W.: Networks of Spiking Neurons: The Third Generation of Neural Network Models. In: Electron Colloquium Computational Complexity, TR96. (1996).
6. Ponulak, F., Kasinski, A.: Introduction to spiking neural networks: Information processing, learning and applications. In: Acta Neurobiol Exp 2011, vol. 71, pp. 409–433. (2011).
7. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324. (1998). https://doi.org/10.1109/5.726791.
8. Zhao, Z., Zheng, P., Xu, S., Wu, X.: Object Detection with Deep Learning: A Review. In: arXiv. (2018). https://doi.org/10.48550/arxiv.1807.05511
9. Masquelier, T., Thorpe, S.: Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity. In: PLoS computational biology, vol. 3, pp. e31. (2007).https://doi.org/10.1371/journal.pcbi.0030031
10. Caporale, N., Yang, D.: Spike timing-dependent plasticity: a Hebbian learning rule. In: Annual review of neuroscience, vol. 31, pp. 25–46. (2008). https://doi.org/10.1146/annurev.neuro.31.060407.125639
11. Bichler, O., Querlioz, D., Thorpe, S J., Bourgoin, Jean-Philippe., Gamrat, Christian.: Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. In: Neural Networks, vol. 32, pp. 339–348. (2012). https://doi.org/10.1016/j.neunet.2012.02.022
12. Hopkins, M., Pineda-Garcia, G., Bogdan, P. A., and Furber, S. B.: Spiking neural networks for computer vision. In: Interface Focus. (2018). https://doi.org/10.1098/rsfs.2018.0007
13. Oudjail, V., Martinet, J.: Bio-inspired Event-based Motion Analysis with Spiking Neural Networks. In: Proceedings Of The 14th International Joint Conference On Computer Vision, Imaging And Computer Graphics Theory And Applications, VISAPP, vol. 4, pp. 389–394. (2019). https://doi.org/10.5220/0007397303890394
14. Nadafian, A., Ganjtabesh, M.: Bio-plausible Unsupervised Delay Learning for Extracting Temporal Features in Spiking Neural Networks. In: CoRR. (2020). https://doi.org/10.48550/arXiv.2011.09380
15. Merolla, P., Arthur, J., Alvarez-Icaza, R., Cassidy, A., Sawada, J., Akopyan, F., Jackson, B., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S., Appuswamy, R., Taba, B., Amir, A., Flickner, M., Risk, W., Manohar, R., Dharmendra S.M.: A million spiking-neuron integrated circuit with a scalable communication network and interface. In: Science, vol. 345, pp. 668–673. (2014). https://doi.org/10.1126/science.1254642

16. Viale, A., Marchisio, A., Martina, M., Masera, G., Shafique, M.: CarSNN: An Efficient Spiking Neural Network for Event-Based Autonomous Cars on the Loihi Neuromorphic Research Processor. In: arXiv. (2021). https://doi.org/10.48550/arxiv.2107.00401
17. Furber, S., Bogdan, P.: SpiNNaker: A Spiking Neural Network Architecture. In: Boston-Delft: Now Publishers. (2020). https://doi.org/10.1561/9781680836523
18. Davison, AP., Brüderle, D., Eppler, JM., Kremkow, J., Muller, E., Pecevski, DA., Perrinet, L., Yger, P.: PyNN: a common interface for neuronal network simulators. In: Front. Neuroinform. vol. 2:11 (2009). https://doi.org/10.3389/neuro.11.011.2008
19. Fardet, T., Vennemo, S., Mitchell, J., Mørk, H., Graber, S., Hahne, J., Spreizer, S., Deepu, R., Trensch, G., Weidel, P., Jordan, J., Eppler, J., Terhorst, D., Morrison, A., Linssen, C., Antonietti, A., Dai, K., Serenko, A., Cai, B., Kubaj, P., Gutzen, R., Jiang, H., Kitayama, I., Jürgens, B., Konradi, S., Albers, J. & Plesser, H.: NEST 2.20.1. In: Zenodo. (2020). https://doi.org/10.5281/zenodo.4018718