Replication study: Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks

Krishnendu Bose, Alexander Ditz, Hannah Köster, Tim Kapferer, Hanna Willkomm

Group 7

1 Introduction

Spiking Neural Networks (SNNs) offer a more biologically plausible approach to information processing compared to traditional artificial neural networks such as Deep Convolutional Neural Networks (DCNNs) (Shirsavar, Vahabie, & Dehaqani, 2023). Furthermore, SNNs can be more energy efficient than DCNNs when implemented on specialized neuromorphic hardware (Kheradpisheh, Ganjtabesh, Thorpe, & Masquelier, 2017; Mozafari, Kheradpisheh, Masquelier, Nowzari, & Ganjtabesh, 2018), as they are event-driven and energy consumption in the network is primarily due to the neurons that spike. DCNNs require computation across all neurons leading to more energy consumption (Wang, Wang, Kong, & Zhang, 2020).

In this report, we focus on an SNN adapted from Mozafari, Ganjtabesh, Nowzari, Thorpe, and Masquelier (2019). The authors introduced a bio-inspired SNN for digit recognition trained using their custom Spike-timing Dependent Plasticity (STDP) and Reward-based STDP (R-STDP) learning rules. The network was trained and evaluated on the MNIST dataset and reached a test accuracy of 97.2%, making it the second-best performing SNN trained with STDP-based learning rules at the time.

This project aims to replicate those findings using the SpikingJelly framework (Fang et al., 2023) for the implementation. Our first objective is to evaluate the model's performance on the MNIST dataset (LeCun, Cortes, & Burges, 2010). Following the approach in Mozafari et al. (2019), we will convert the MNIST images into spike trains by applying six different difference of Gaussian (DoG) filters and using intensity-to-latency encoding. Next, we aim to modify the network and training pipeline to work with the Neuromorphic-MNIST (N-MNIST) dataset (Orchard, Jayawant, Cohen, & Thakor, 2015), which is a widely used benchmark for neuromorphic vision (Lee, Delbruck, & Pfeiffer, 2016). The N-MNIST dataset consists of the same samples as MNIST but is recorded using an event-based camera, which captures visual information by detecting changes in light intensity at each pixel. When such a change is detected, the camera emits an event for that pixel, resulting in data that is already encoded in the spike domain. This event-based recording allows for highly efficient and low-latency data processing, making N-MNIST an ideal dataset for evaluating the performance of neuromorphic models.

By reimplementing the original model using SpikingJelly, we seek to validate the original results, demonstrate the robustness across different implementations or SNN frameworks, and explore how the input encoding used by Mozafari et al. (2019) compares against input encoding through recording with an event-based vision sensor with regards to the testing accuracy of a model trained on these.

All code and materials used are made publicly available under the following URL: `https://github.com/Cl4ryty/synaptic_plasticity`

2 Methods

2.1 Datasets

In this project, we utilize the MNIST and N-MNIST datasets to train and evaluate the recognition performance of our model.

2.1.1 MNIST

The MNIST dataset (LeCun et al., 2010) is a well-known benchmark in machine learning, particularly for image classification tasks. It consists of 60,000 training and 10,000 test images of handwritten

digits. Examples of the MNIST are displayed in Figure 1a.

2.1.2 N-MNIST

The N-MNIST dataset (Orchard et al., 2015) is a spiking version of the original MNIST dataset, capturing temporal patterns and spatiotemporal dependencies. N-MNIST comprises the same number of samples as MNIST. The spatiotemporal patterns captured by the N-MNIST dataset are illustrated in Figure 1b.

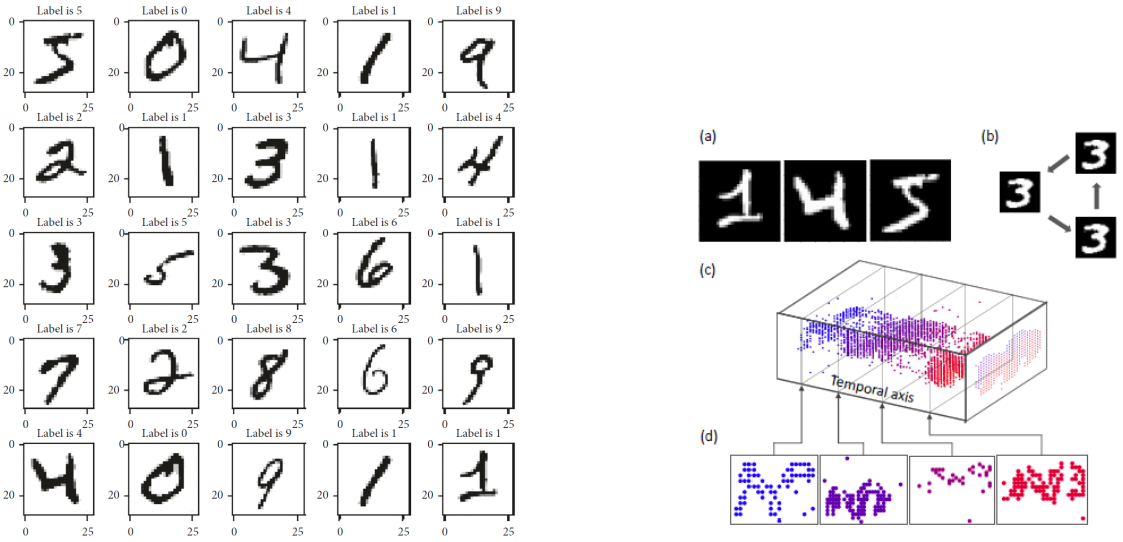

Figure 1: Visual comparison of the MNIST and N-MNIST datasets. (a) A sample of handwritten digits from the MNIST dataset, showcasing its static 28x28 pixel grayscale images. (b) The N-MNIST dataset illustrates the spatio-temporal pattern of the same digits, which are captured as dynamic sequences of events over time, providing an additional temporal dimension. Figures adapted from Nandhini Abirami et al. (2021) and He et al. (2020), respectively.

2.2 Network

The network consists of three spiking convolutional layers, each followed by a pooling layer. STDP is applied to the first two layers, while R-STDP is used for the final layer. The first convolutional layer uses 30 channels with a 5x5 kernel followed by max pooling and padding. The second layer increases the feature depth to 250 channels with a 3x3 kernel and similar processing. The final layer features 200 channels with a 5x5 kernel. The network classifies digits based on the maximum membrane potential in the last layer.

2.2.1 Neuron model

All neurons in the network are Integrate-and-Fire (IF) neurons which integrate all input current by adding it to the neuron's potential and spike once a set threshold is reached. However, unlike common IF neurons, the ones used in this network do not reset their potential after firing but keep integrating inputs into the potential while not emitting any further spikes after the initial one. These neurons can thus spike maximally one time if their threshold is exceeded, but no more than that, regardless of how much input they receive. As this project aims to replicate and extend upon the findings of Mozafari et al. (2019), this neuron model was primarily chosen as it matches the one used in their study, with the reasoning also remaining the same as that of the original authors, namely that it is a computationally efficient neuron model and therefore suitable for training a larger network in this performance-oriented task. Limiting the number of spikes one neuron can produce to just one spike serves the same purpose as it allows for a much simpler learning algorithm, further reducing computational requirements for training the network as compared to one consisting of traditional IF neurons.

2.2.2 Learning algorithm

The choice of the learning algorithm is based on the work of Mozafari et al. (2019), using their proposed version of STDP and R-STDP, a short explanation of which is provided in the following

with the reader also being directed to the original paper for more details and the reasoning behind the choices and assumptions made by the algorithms.

The general learning rule or algorithm is the same for both STDP are R-STDP, with different parameter values leading to the different weight updates. Specifically, the weight update is performed according to the following equations:

$$\Delta w_{ij} = \delta_{ij}(w_{ij})(1 - w_{ij}),$$

$$\delta_{ij} = \begin{cases} \alpha\phi_r a_r^+ + \beta\phi_p a_p^- & \text{if } t_j - t_i \leq 0, \\ \alpha\phi_r a_r^- + \beta\phi_p a_p^+ & \text{if } t_j - t_i > 0, \text{ or neuron j never fires,} \end{cases} \tag{1}$$

with δ_{ij} being the main update rule and the rest a scaling factor intended to stabilize the training and keep the weights within the interval [0, 1]. To ensure that they do not grow beyond these bounds, an additional clipping is also applied after each update. It is to be noted that while this clipping is not mentioned by Mozafari et al. (2019), it is present in the SpykeTorch implementation and we elected to give preference to the following details from the implementation over descriptions from the paper wherever they diverged.

What differentiates the STDP and R-STDP update is the value of the α, β, and ϕ parameters, with α and ϕ_r being set to one for all STDP updates and the other two controlling factors set to 0 such that for STDP updates

$$\delta_{ij} = \begin{cases} a_r^+ & \text{if } t_j - t_i \leq 0, \\ a_r^- & \text{if } t_j - t_i > 0, \text{ or neuron j never fires.} \end{cases} \tag{2}$$

For R-STDP α and β are set to 1 and 0 or 0 and 1 for rewards and punishments, respectively, while $\phi_p = N_{\text{hits}}/N$ and $\phi_r = N_{\text{misses}}/N$, with N_{hits} and N_{misses} being the number of correctly or incorrectly classified samples in the previous batch of N samples. This results in larger positive updates for rewards if the model has classified a larger number of samples incorrectly in the last batch and similarly also regulates the weight updates in case of punishment such that this leads to a kind of performance-based annealing of the weight updates, with more drastic changes during worse performance and smaller updates once the network performs better.

The values for the other parameters are given in Table 1 and are the same that are provided by Mozafari et al. (2019).

Note that since either α or β is zero when the other is set to one, this learning rule can be simplified for implementation by implementing only half of it and performing STDP updates with different parameter values for R-STDP depending on the reward, which can be thought of as a normal STDP update in the case of a reward and an anti-STDP update in case of a negative reward or punishment. The general learning rule can thus be described as follows:

$$\delta_{ij} = \begin{cases} \alpha\phi_r a_r^+ & \text{if } t_j - t_i \leq 0, \\ \alpha\phi_r a_r^- & \text{if } t_j - t_i > 0, \text{ or neuron j never fires,} \end{cases} \tag{3}$$

The first line of this can be interpreted as describing long-term potentiation while the second line, conversely, would lead to long-term depression provided that $a_r^+ > 0$ and $a_r^- < 0$. For the anti-STDP update in the case of a negative reward for the R-STDP update $a_r^+ = a_p^-$ and $a_r^- = a_p^+$, effectively reversing the previously described roles and leading to depression if the pre-neuron fires before the post neuron and potentiation otherwise.

Table 1: Parameter values for each layer

Layer	a_r^+	a_r^-	a_p^+	a_p^-	**k**	**r**
S1	0.004	-0.003	0	0	5	3
S2	0.004	-0.003	0	0	8	2
S3	0.004	-0.003	0.0005	-0.004	1	0

3 Results

To compare the performance of the different models training and testing accuracies were logged for each training epoch. Training accuracies are provided as a convenience for judging model fitting but

are not used to compare model performance, for which only testing accuracies are considered. The results of the reimplementation were compared with the results achieved by the SpykeTorch model. Training and testing accuracies for all models are plotted in Figure 2 with the maximum accuracy of 97.2% reached by Mozafari et al. (2019) plotted as a baseline. However, this accuracy was only reached after training a considerable amount of more epochs than was possible for us to train due to limited time and computational resources, which is why the epoch-by-epoch performance of the SpykeTorch model is also displayed as a reference. As can be seen, while the accuracy of our reimplemented model improves quickly in the first few epochs and improving at a similar rate as the SpykeTorch model, it consistently performs at around ten percent less accuracy compared to that model, reaching a maximum accuracy of 87.48% at epoch 15 while the SpykeTorch model reaches a maximum accuracy of 96.7% at epoch 20 in the training timespan. The model trained on N-MNIST performs worse, barely increasing in accuracy after the second epoch and reaching a maximum accuracy of 49.97% at epoch 20.

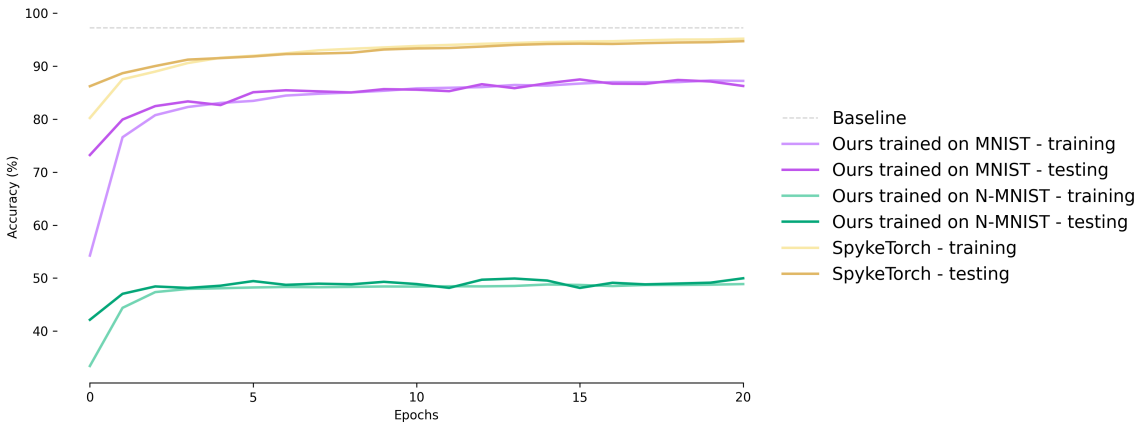

Figure 2: Training and testing accuracies for all models across epochs

4 Discussion

Due to the extended time required for network training on the available computational resources and the tight project deadlines, we were unable to train the networks as extensively as in the original study by Mozafari et al. (2019). Consequently, it is unsurprising that our networks did not achieve the test accuracy reported by Mozafari et al. (2019). However, it is still possible to compare the performance across the epochs that all models reached during training. As discussed in the results section, our reimplemented network consistently underperforms compared to the SpykeTorch implementation, even when comparing accuracies for the same number of epochs. One potential reason for this discrepancy is a major limitation of our project: we were only able to train a single model for each variant. Proper comparison requires multiple training runs for each configuration to minimize the effects of random initializations. Moreover, the efficiency of our implementation could be significantly improved. Future work should focus on using more vectorized operations and fully leveraging GPU capabilities during training, which would allow for more extensive training within the given time constraints. On the positive side, our implementation in the SpikingJelly framework offers significant flexibility. It allows for easier modification or replacement of individual components, such as the neuron model or plasticity algorithm. This flexibility contrasts with the SpykeTorch implementation, where assumptions are so deeply ingrained that altering or testing these components would be difficult, if not impossible, without rewriting the entire library.

5 Contribution

KB: Writing some parts of the report, proofreading, editing. AD: Writing (Introduction, Methods, References). HK: Writing (Neuron model, learning algorithm, results, discussion), review and editing (report and code documentation), implementation. TK: Code implementation and documentation; report review and editing. HW: Code implementation and documentation.

References

Fang, W., Chen, Y., Ding, J., Yu, Z., Masquelier, T., Chen, D., ... Tian, Y. (2023). Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence. *Science Advances*, *9*(40), eadi1480. Retrieved from https://www.science.org/doi/abs/10.1126/sciadv.adi1480 doi: 10.1126/sciadv.adi1480

He, W., Wu, Y., Deng, L., Li, G., Wang, H., Tian, Y., ... Xie, Y. (2020, 08). Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences. *Neural Networks*, *132*. doi: 10.1016/j.neunet.2020.08.001

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S., & Masquelier, T. (2017, 12). Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, *99*. doi: 10.1016/j.neunet.2017.12.005

LeCun, Y., Cortes, C., & Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, *2*.

Lee, J., Delbruck, T., & Pfeiffer, M. (2016, 11). Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, *10*. doi: 10.3389/fnins.2016.00508

Mozafari, M., Ganjtabesh, M., Nowzari, A., Thorpe, S., & Masquelier, T. (2019, 05). Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks. *Pattern Recognition*, *94*. doi: 10.1016/j.patcog.2019.05.015

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari, A., & Ganjtabesh, M. (2018, 05). First-spike-based visual categorization using reward-modulated stdp. *IEEE Transactions on Neural Networks and Learning Systems*, *PP*, 1-13. doi: 10.1109/TNNLS.2018.2826721

Nandhini Abirami, R., Durai Raj Vincent, P. M., Srinivasan, K., Tariq, U., Chang, C.-Y., & Sarfraz, D. S. (2021, jan). Deep cnn and deep gan in computational visual perception-driven image analysis. *Complex.*, *2021*. Retrieved from https://doi.org/10.1155/2021/5541134 doi: 10.1155/2021/5541134

Orchard, G., Jayawant, A., Cohen, G., & Thakor, N. (2015, 11). Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, *9*. doi: 10.3389/fnins.2015.00437

Shirsavar, S., Vahabie, A., & Dehaqani, M.-R. (2023, 03). Models developed for spiking neural networks. *MethodsX*, *10*, 102157. doi: 10.1016/j.mex.2023.102157

Wang, G., Wang, R., Kong, W., & Zhang, J. (2020, 11). The relationship between sparseness and energy consumption of neural networks. *Neural Plasticity*, *2020*, 1-13. doi: 10.1155/2020/8848901