EA-Based SNN Hyperparameter Tuning

1st Anant Sahoo
Department of EECS
University of Tennessee, Knoxville
Knoxville, United States
asahoo@vols.utk.edu

2nd Deep Patel
Department of EECS
University of Tennessee, Knoxville
Knoxville, United States
dpate125@vols.utk.edu

3rd Sulaiman Mohyuddin
Department of EECS
University of Tennessee, Knoxville
Knoxville, United States
smohyud1@vols.utk.edu

***Abstract*—Spiking Neural Networks (SNNs) are biologically inspired models known for their energy efficiency and temporal processing capabilities. However, their performance on classification tasks is highly sensitive to hyperparameter choices, which are often difficult to optimize using traditional methods. In this project, we investigate whether Evolutionary Algorithms (EAs) can effectively optimize SNN hyperparameters to improve classification accuracy. Using the LEAP library, we evolved high-level training parameters—including batch size, number of hidden neurons, membrane decay rate (beta), and number of simulation time steps—on the KMNIST dataset, a more complex variant of MNIST focused on Japanese character recognition. Our best evolved configuration achieved a test accuracy of approximately 88%, surpassing the baseline model's 85% despite being trained for only one epoch. Visualizations of the evolutionary process and parameter-specific accuracy trends revealed both intuitive and unexpected outcomes, such as non-monotonic performance with respect to time steps and diminishing returns at higher beta values. These findings demonstrate the potential of EAs for discovering non-obvious yet effective hyperparameter settings in SNNs. Future work will explore larger population sizes, extended training epochs, and additional EA configurations to further refine performance.**

***Index Terms*—Spiking Neural Networks, Evolutionary Algorithms, Hyperparameter Optimization, Neuroevolution, LEAP, KMNIST, Classification, snnTorch, Genetic Algorithms.**

I. Introduction

Spiking Neural Networks (SNNs) have increased in popularity in recent years due to their biologically inspired architecture and efficiency benefits [1]. In this project, we focus on improving the performance of SNNs for image classification tasks by tuning their hyperparameters. Specifically, we investigate the use of Evolutionary Algorithms (EAs) to optimize these parameters. Our central research question is: Can evolutionary algorithms effectively optimize hyperparameters in spiking neural networks to improve performance on classification tasks? The dataset we used is KMNIST, which classifies groups of cursive Japanese characters into 10 distinct classes [2]. The extra layer of complexity that KMNIST offers over MNIST leaves more room for the evolutionary process to improve the baseline model.

While this general question has been addressed in various forms throughout the literature, it remains open-ended due to the wide range of possible methods and design choices. This flexibility makes the problem highly exploratory and well-suited for experimentation. We chose to pursue this approach not only because EAs are particularly effective for navigating large, non-differentiable, and multimodal search spaces, but also because of our prior experience using the LEAP software in earlier coursework. Building on prior experience with the framework, we were motivated to explore its application in a novel context—hyperparameter optimization for Spiking Neural Networks.

II. Related Work

Spiking Neural Networks (SNNs) have gained increased attention in recent years due to their advantages over traditional Artificial Neural Networks (ANNs) [1]. Unlike ANNs, SNNs rely on in-memory and local computations while leveraging time as a computational resource. When deployed on neuromorphic hardware, these properties make SNNs highly energy- and memory-efficient, scalable, and inherently parallelizable [3, 4]. While SNNs have shown promising results on classification tasks, their performance can often be improved further through careful hyperparameter tuning [5]. Identifying an optimal combination of hyperparameters remains a challenge, as their effectiveness is highly sensitive to both model architecture and dataset [4].

To address this, we explore the use of Evolutionary Algorithms (EAs) for hyperparameter optimization. EAs are particularly effective for problems with complex, non-differentiable search spaces. They offer a robust method for discovering near-optimal or global solutions through stochastic exploration of multimodal parameter landscapes [6]. This search process is commonly referred to as neuroevolution and has been explored over the last 3 decades. Network weights, architectures, learning rules, and input features are commonly targeted as the parameters to optimize [7].

A. EONS

One established approach to applying EAs to SNNs is Evolutionary Optimization for Neuromorphic Systems (EONS). EONS is designed to generate optimized Spiking Recurrent Neural Networks (SRNNs) for specific tasks and neuromorphic platforms. It begins with a population of candidate networks and iteratively applies evaluation, selection, and reproduction to generate improved generations. EONS searches for optimal configurations of nodes, edges, and associated neuron parameters [8]. Schuman et al. demonstrated EONS across several tasks, including classification on the wine, iris, and breast cancer datasets from scikit-learn. Each SNN's fitness was determined by its classification accuracy on the training data. Their results showed that EONS achieved more consistent performance compared to baseline models such as MLP and Reservoir, with only the Whetstone approach slightly outperforming EONS in some cases [9].

B. LEAP

Library for Evolutionary Algorithms in Python (LEAP) is an extensive and modular EA framework that aims to provide an easy-to-use toolkit for users to solve problems with EAs, for researchers to implement novel approaches, and for making EA concepts accessible for students [10]. Unlike EONS, LEAP is not tailored specifically for SNNs, but has proven effective for evolving network hyperparameters. In prior work by Parsa et al. [11], LEAP was used to evolve weights, delays, and thresholds in SNNs for the aforementioned scikit-learn datasets. Their implementation varied EA hyperparameters such as mutation rate, crossover rate, and tournament size across two case studies—one with a narrow search space and another with a broader one. While LEAP and EONS yielded comparable accuracy results, LEAP provided greater flexibility and adaptability in representing diverse network configurations.

While the LEAP implementation in Parsa et al. [11] focused on evolving low-level neuron parameters such as weights, delays, and thresholds, our methodology centers on optimizing high-level training hyperparameters that significantly affect model performance and convergence. Our binary representation of the genome encoded the batch size, the number of hidden layer nodes, the beta value, and the time steps. Once these parameter values were extracted from the binary string, the SNN was trained, and the resulting testing accuracy was computed as the final fitness value. Moreover, we fixed the values of the EA hyperparameters and trained the SNN for one epoch.

III. Methodology

A. Algorithm Selection and Justification

To evolve the hyperparameters of spiking neural networks (SNNs), we utilized Leap [10]. LEAP provides a flexible framework for evolving genomes across a wide variety of problem domains, including neural network optimization. Previous work has demonstrated its effectiveness in SNN-related tasks, and we extend that capability to hyperparameter tuning specifically. We chose LEAP for its ease of setup and customization, allowing us to define genome encodings and search strategies efficiently. For our experiments, we set a population size of 10, a mutation rate of 0.01, a crossover rate of 0.3, and used tournament selection with a tournament size of 3. The algorithm was run for 15 generations.

B. System Design and Experimental Setup

Our SNN model was implemented using the snnTorch library and consists of a straightforward architecture with three fully connected layers: an input layer, a hidden layer, and an output layer. Between the input and hidden layers, and again between the hidden and output layers, we inserted leaky integrate-and-fire (LIF) neuron layers to produce spiking behavior. The input layer size was set to match the number of pixels in the KMNIST images (28×28), and the output layer contained ten neurons corresponding to the ten class labels.

During training, the input data was divided into mini-batches. For each mini-batch, the network was simulated over a fixed number of time steps. At each step, spikes were accumulated and passed through the network. After the final time step, the spike outputs were summed, and the resulting tensor was passed to a standard cross-entropy loss function. We used the Adam optimizer from PyTorch to perform gradient updates. This process was repeated across the entire dataset for a single training epoch. The SNN implementation is based on code from Jason Esghraghian's presentation for the Open Neuromorphic initiative [12].

C. Parameter Space and Evaluation Criteria

We focused on evolving four hyperparameters: the number of hidden neurons, batch size, beta (the membrane decay constant in LIF neurons), and the number of simulation time steps. These were selected due to their significant influence on both the temporal and spatial learning behavior of SNNs. The hidden neuron count was varied between 200 and 1000 in increments of 20. Batch size ranged from 64 to 256 in steps of 32. beta was varied between 0.5 and 0.99 using 32 evenly spaced values. Time steps were explored between 10 and 50 in increments of 10.

To encode this parameter space, we used a 17-bit genome. Specifically, the first 3 bits represented the batch size, the next 6 bits encoded the hidden layer size, followed by 5 bits for the beta value, and the remaining 3 bits for the number of time steps. Each genome was decoded into its corresponding numerical value and then scaled to hyperparameter values, ensuring each decoded value mapped to the defined search space.

The fitness of each individual genome was determined by training the SNN using the decoded hyperparameters and evaluating its accuracy on the test set. This test accuracy served as the fitness score. The evolutionary algorithm then selected individuals for reproduction, applied crossover and mutation, and generated the next population. This process continued until all generations were completed.

To compare the effectiveness of the evolved hyperparameters, we also evaluated a fixed baseline configuration obtained from Esghraghian's presentation [12]. The baseline used a batch size of 128, 1000 hidden neurons, a beta of 0.95, and 20 time steps. This allowed us to evaluate whether the evolutionary process could surpass manually selected hyperparameters with minimal training.

IV. Results

A. Performance Metrics and Observations

To evaluate the effectiveness of evolutionary hyperparameter optimization, we used classification accuracy on the KMNIST test set as our performance metric. Each genome in the population was assigned a fitness value equal to this test accuracy, and the evolutionary algorithm was run for 15 generations. To determine the benefits of this approach, we compared accuracy results from evolved models with those using a manually selected baseline configuration.

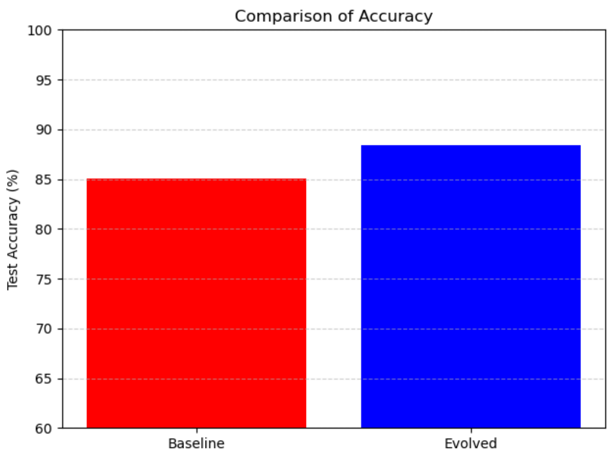

Fig. 1. Test accuracy comparison between baseline and evolved hyperparameters.

As shown in Fig. 1, the evolved hyperparameter set achieved a noticeably higher accuracy of approximately 88%, compared to 85% from the baseline configuration. The optimal parameters found through evolution were batch size of 128, 880 LIF neurons, 0.942 for the beta value, and 20 for the number of time steps. While the baseline parameters were chosen based on established good practices in prior SNN tutorials, the evolved configuration surpassed them by exploring a larger search space. This outcome demonstrates the capability of evolutionary algorithms to uncover more optimal hyperparameter combinations that may not be intuitive or easily guessed.

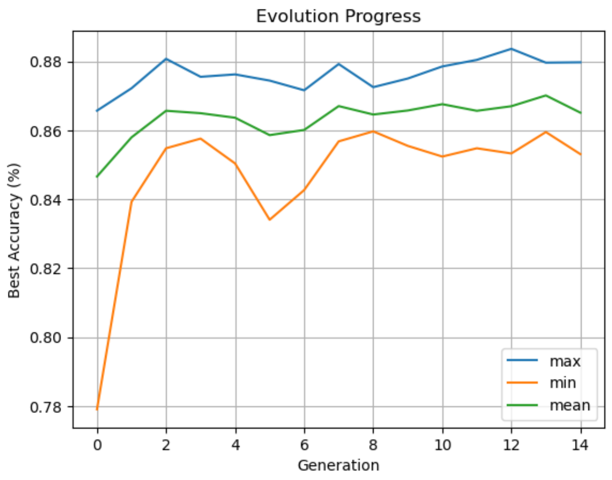

Fig. 2. Accuracy evolution across generations: minimum, maximum, and mean fitness.

Fig. 2 tracks the evolutionary algorithm's progress across 15 generations. The max accuracy steadily improves from 86.5% to 88.3%, while the mean also trends upward, indicating consistent population-wide improvement. A significant increase in minimum accuracy during early generations suggests successful elimination of poor-performing candidates. Interestingly, after generation 10, both mean and max values start to plateau. This may suggest convergence or exploitation dominating over exploration, potentially highlighting a need for more aggressive mutation strategies or diversity-preserving mechanisms in later generations.

B. Visualizations and Interpretation

To better understand how each hyperparameter contributed to model performance, we visualized accuracy trends for individual parameters while holding others approximately constant. Each plot below presents the average test accuracy for models sharing the same value of one hyperparameter.

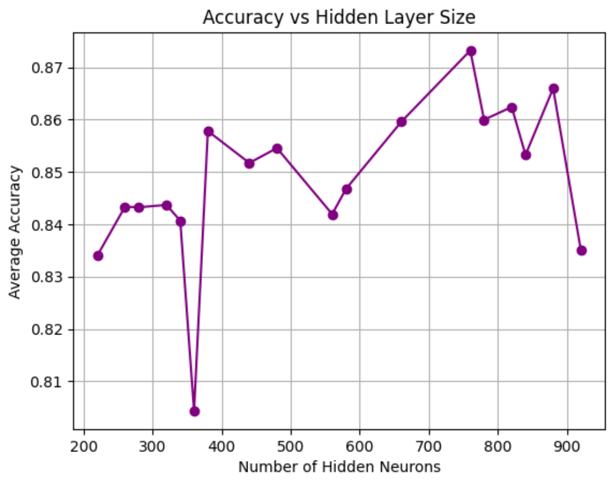

Fig. 3. Average accuracy across different hidden layer sizes.

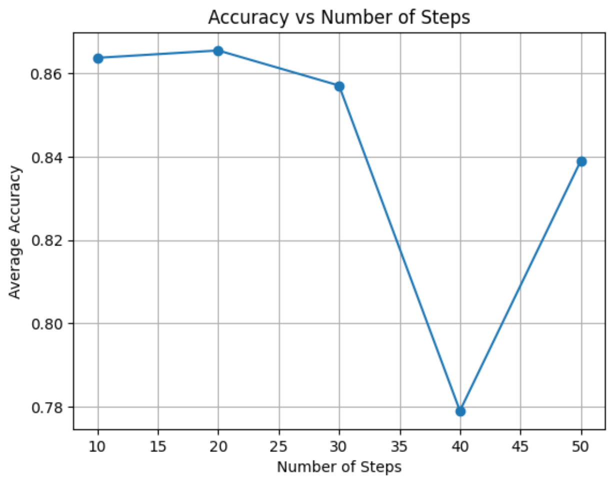

Fig. 4. Average accuracy across different numbers of time steps.

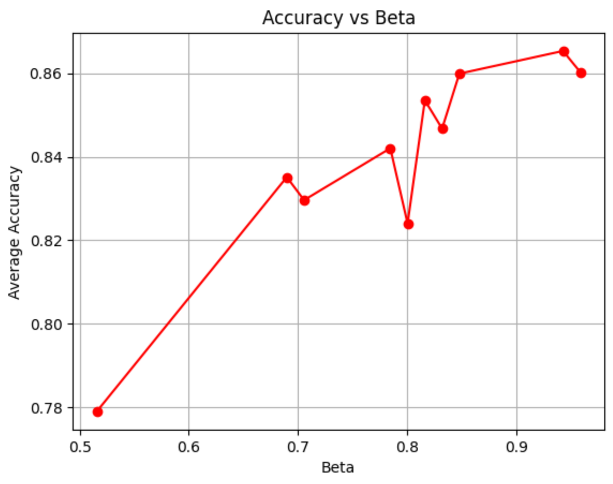

Fig. 5. Average accuracy across different beta values.

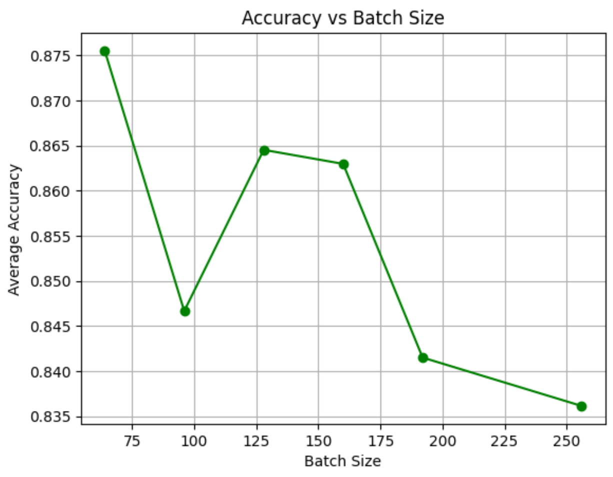

Fig. 6. Average accuracy across different batch sizes.

In Fig. 3, accuracy generally increases with the number of hidden neurons. Performance is lowest near 360 neurons but improves as the network size increases, peaking around 760–860 neurons. This aligns with the expectation that more hidden units provide greater representational capacity. However, the drop near 920 neurons is unexpected. One possible explanation is overfitting due to limited training epochs (only one epoch per run), or unstable gradients in larger networks when combined with other hyperparameters like low beta or high step counts. This suggests that while increasing model capacity helps up to a point, it must be balanced with temporal constraints and training duration.

Fig. 4 shows that accuracy improves slightly from 10 to 20 time steps, then declines sharply at 40 steps before partially recovering at 50. This non-monotonic trend suggests an optimum around 20 steps. The significant accuracy drop at 40 steps is unexpected. A likely explanation is that too many simulation steps accumulate noisy or redundant spikes, degrading the signal-to-noise ratio. Since loss is computed on the sum of spikes across time, extended simulation may lead to the differences between classes vanishing if all neurons spike excessively. This highlights a counterintuitive insight: more temporal resolution does not always improve performance in SNNs.

Fig. 5 indicates that increasing beta generally correlates with higher accuracy. A higher beta results in slower membrane decay, allowing neurons to integrate spikes over longer periods. This improves temporal context memory and increases the likelihood of correct spiking activity. The curve flattens beyond beta = 0.9, suggesting diminishing returns. A brief dip around beta = 0.8 could be due to interaction effects with other hyperparameters (e.g., low time steps), which shorten the effective integration window. These results emphasize the importance of properly tuning beta in tasks like SNN classification.

Fig. 6 reveals a decreasing trend in accuracy as batch size increases. Smaller batches (e.g., 64) yield better accuracy, while performance drops for batches above 128. This may be attributed to increased gradient noise in smaller batches acting as implicit regularization, helping avoid overfitting during the brief 1-epoch training phase. Larger batches, while more computationally efficient, may oversmooth the gradient and reduce exploration during optimization. The results suggest that for short training periods, smaller batch sizes are preferable.

Across all plots, the results demonstrate that: 1. Hyperparameters interact in nontrivial ways, 2. Some trends (e.g., higher hidden neurons, higher beta) align with expectations, 3. Other outcomes (e.g., accuracy drop at 40 steps) were unexpected and merit deeper investigation.

Overall, the visualizations provide strong evidence that evolutionary algorithms are effective in discovering performant, and occasionally surprising, hyperparameter configurations for spiking neural networks.

V. Discussion and Future Directions

A. Key Findings and Insights

Our results demonstrate that Evolutionary Algorithms (EAs), when applied to high-level training hyperparameters of Spiking Neural Networks (SNNs), can lead to measurable improvements in classification accuracy. We focused on evolving four specific parameters: batch size, number of hidden neurons, membrane decay rate (beta), and number of simulation time steps. Despite training all models for only one epoch, the best evolved configurations consistently outperformed the baseline configuration derived from Eshraghian's recommended defaults. On average, the evolved models achieved test accuracies that were approximately 3% higher than the baseline model, highlighting the potential of EAs to discover effective hyperparameter settings even under constrained training conditions.

In evaluating the individual influence of each hyperparameter, we observed that increasing the number of hidden neurons generally resulted in improved accuracy. This aligns with expectations, as larger hidden layers often allow the model to learn richer internal representations. However, this benefit plateaued beyond a certain point, suggesting diminishing returns and the potential for overfitting or computational inefficiency if the network is oversized. Batch size, on the other hand, did not exhibit a consistent pattern—smaller and mid-sized batches occasionally outperformed larger ones, indicating that optimal batch size may be dependent on interactions with other parameters such as beta or time steps.

A particularly interesting trend emerged with the number of time steps used to simulate spiking dynamics. Accuracy improved from 10 to 20 steps but dropped significantly at 40 before partially recovering at 50. This non-monotonic trend suggests that extending the simulation window does not always yield better results. One possible explanation is that excessively long simulations may accumulate redundant or noisy spikes, which can reduce class separability and degrade overall performance. This effect was especially pronounced given that spike outputs were summed over time before loss calculation, making the network vulnerable to signal saturation. These findings underscore the importance of tuning temporal dynamics carefully in SNNs rather than assuming longer simulations are inherently better.

Finally, the impact of beta values, which govern the membrane potential decay in LIF neurons, revealed another nuanced relationship. Mid-range beta values (e.g., around 0.85 to 0.95) tended to produce more reliable results, while extreme values on either end led to more unstable accuracy trends. This suggests that both too much memory (high beta) and too little memory (low beta) can hinder network responsiveness and learning. Together, these observations support the hypothesis that EAs can uncover non-obvious yet effective combinations of parameters and highlight the necessity of exploring the full range of design tradeoffs in SNNs.

B. Limitations and Challenges

While the results are encouraging, our approach has several limitations. First, the network was trained for only one epoch, primarily due to time and resource constraints. This limited the model's opportunity to fully converge and may have affected the stability of accuracy comparisons between configurations. In a longer training setting, the performance of both the baseline and evolved models might shift or converge more closely.

Additionally, the evolutionary algorithm parameters (mutation rate, crossover rate, population size, and tournament selection) were kept constant throughout the experiment. It is possible that tuning the EA itself could yield further performance improvements. Another constraint was the size of the parameter space and the resolution of the binary encoding. Due to the fixed genome length, we used discrete ranges for each hyperparameter, which may have prevented discovering finer-grained optimal values.

C. Proposed Future Work

Given additional time and computational resources, several directions could be explored to build upon this research. A clear limitation in our current setup was the use of a relatively small population size. Due to resource constraints, the evolutionary algorithm was restricted to a gene pool of only 10 individuals per generation. Increasing this population size would allow for broader sampling of the hyperparameter space and better diversity, potentially leading to the discovery of higher-performing models. Similarly, extending the number of generations beyond 15 would enable further refinement of top individuals and promote more robust evolutionary convergence.

Another key area for improvement is training duration. All SNN models in this study were trained for just one epoch. Extending training to multiple epochs would likely yield stronger performance, allowing the networks more time to learn from the data. Moreover, our current configuration fixed EA parameters such as mutation rate, crossover probability, and tournament size. Future work could systematically explore these parameters to assess how different evolutionary dynamics influence convergence and performance.

Finally, expanding the genome representation to include additional hyperparameters—such as learning rate, optimizer type, or neuron-specific characteristics—could yield deeper insights into which model-level configurations most influence SNN learning on classification tasks.

VI. Conclusion

This work demonstrates the effectiveness of evolutionary algorithms in tuning high-level hyperparameters of spiking neural networks for image classification tasks. By applying an evolutionary search over parameters such as batch size, hidden layer width, membrane decay rate, and simulation time steps, we achieved consistent improvements in classification accuracy over a manually defined baseline, despite limiting training to a single epoch. The evolved configurations achieved test accuracies up to 3% higher, highlighting the potential of evolutionary approaches in navigating the complex and sensitive hyperparameter spaces of SNNs.

Our results reinforce that SNN performance is highly dependent on careful tuning, especially with respect to temporal parameters like time steps and decay rates. Moreover, the non-linear and sometimes counterintuitive relationships discovered, such as the performance drop at excessive simulation time steps, emphasize the need for automated optimization techniques when working with neuromorphic models.

While constrained by training time and population size, our work provides an effective demonstration of how evolutionary methods can complement neuromorphic computing research. Future efforts involving longer training durations, expanded parameter ranges, and improved evolutionary strategies may further enhance performance and extend these findings to more complex datasets or hardware platforms.

Acknowledgment

The authors would like to thank the Department of Electrical Engineering and Computer Science at the University of Tennessee, Knoxville, for providing a supportive research envi-

ronment and facilitating valuable discussions that contributed to this work.

References

[1] S. Chen *et al.*, “A survey on evolutionary spiking neural networks,” *arXiv preprint* arXiv:2406.12552, Jun. 2024. [Online]. Available: https://arxiv.org/abs/2406.12552

[2] rois-codh, “Repository for Kuzushiji-MNIST, Kuzushiji-49, and Kuzushiji-Kanji,” GitHub, https://github.com/rois-codh/kmnist [Online]. Available: https://github.com/rois-codh/kmnist. [Accessed: May 11, 2025]. :contentReference[oaicite:0]index=0

[3] M. Pfeiffer and T. Pfeil, “Deep learning with spiking neurons: Opportunities and challenges,” *Frontiers in Neuroscience*, vol. 12, p. 774, Oct. 2018. doi: 10.3389/fnins.2018.00774.

[4] T. Firmin, P. Boulet, and E.-G. Talbi, “Parallel hyperparameter optimization of spiking neural networks,” *Neurocomputing*, vol. 609, p. 128483, 2024. doi: 10.1016/j.neucom.2024.128483.

[5] T. Iakymchuk, A. Rosado-Muñoz, J. F. Guerrero-Martínez, M. Bataller-Mompeán, and J. V. Francés-Víllora, “Simplified spiking neural network architecture and STDP learning algorithm applied to image classification,” *EURASIP Journal on Image and Video Processing*, vol. 2015, no. 1, p. 4, Jan. 2015. doi: 10.1186/s13640-015-0059-4.

[6] S. Ding, H. Li, C. Su, J. Yu, and F. Jin, “Evolutionary Artificial Neural Networks: A Review,” Artificial Intelligence Review, vol. 39, no. 3, pp. 251–260, Jun. 2011. doi:10.1007/s10462-011-9270-6

[7] Xin Yao, ”Evolving artificial neural networks,” in Proceedings of the IEEE, vol. 87, no. 9, pp. 1423-1447, Sept. 1999, doi: 10.1109/5.784219. keywords: Artificial neural networks;Intelligent systems;Competitive intelligence;Evolutionary computation;Artificial intelligence;Transfer functions;Computer networks;Intelligent networks;Algorithm design and analysis;Adaptive systems,

[8] C. D. Schuman, J. P. Mitchell, R. M. Patton, T. E. Potok, and J. S. Plank, ”Evolutionary Optimization for Neuromorphic Systems,” Proc. 2020 Annu. Neuro-Inspired Comput. Elements Workshop (NICE ’20), New York, NY, USA: ACM, 2020, Art. no. 2, pp. 1–9, doi:

[9] C. D. Schuman, G. Bruer, A. R. Young, M. Dean and J. S. Plank, ”Understanding Selection And Diversity For Evolution Of Spiking Recurrent Neural Networks,” 2018 International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro, Brazil, 2018, pp. 1-8, doi: 10.1109/IJCNN.2018.8489149.

[10] M. A. Coletti, E. O. Scott, and J. K. Bassett, ”Library for evolutionary algorithms in Python (LEAP),” Proc. 2020 Genetic and Evolutionary Computation Conf. Companion (GECCO ’20), New York, NY, USA: ACM, 2020, pp. 1571–1579, doi: 10.1145/3377929.3398147.

[11] M. Parsa, S. R. Kulkarni, M. Coletti, J. Bassett, J. P. Mitchell and C. D. Schuman, ”Multi-Objective Hyperparameter Optimization for Spiking Neural Network Neuroevolution,” 2021 IEEE Congress on Evolutionary Computation (CEC), Kraków, Poland, 2021, pp. 1225-1232, doi: 10.1109/CEC45853.2021.9504897.

[12] open-neuromorphic, ”Hands-on Session SNNtorch 230302,” GitHub repository, Mar. 2023. [Online]. Available: https://github.com/open-neuromorphic/hands-on-session-snntorch-230302