DynSNN: A Dynamic Approach to Reduce Redundancy in Spiking Neural Networks

Fangxin Liu[1], Wenbo Zhao[1], Yongbiao Chen[1], Zongwu Wang[1], Fei Dai[2,†]
1. Shanghai Jiao Tong University 2. Institute of System Engineering, Academy of Military Sciences
{liufangxin, zhaowenbo, chenyongbiao0319, wangzongwu}@sjtu.edu.cn, daifei08@163.com

***Abstract*—Current Internet of Things (IoT) embedded applications use machine learning algorithms to process the collected data. However, the computational complexity and storage requirements of existing deep learning methods hinder the wide availability of embedded applications. Spiking Neural Networks (SNN) is a brain-inspired learning methodology that emerged from theoretical neuroscience, as an alternative computing paradigm for enabling low-power computation. Since these IoT devices are usually resource-constrained, compression techniques are crucial in the practical application of SNNs. Most existing methods directly apply pruning methods from artificial neural networks (ANNs) to SNNs, while ignoring the distinction between ANNs and SNNs, thus inhibiting the potential of pruning methods on SNNs. In this paper, inspired by the topology of neuronal co-activity in the neural system, we propose a dynamic pruning framework (dubbed DynSNN) for SNNs, enabling us to seamlessly optimize network topology on the fly almost without accuracy loss. Experimental results on a wide range of classification applications show that the proposed method achieves almost lossless for SNN on MNIST, CIFAR-10, and ImageNet datasets. Moreover, it reaches a $\sim 0.3\%$ accuracy loss under 34% compression rate on CIFAR and ImageNet, and achieves 60% compression rate with no accuracy loss on MNIST, which reveals remarkable structure refining capability in SNNs.**

***Index Terms*—Spiking Neural Network, Dynamic Network, Accuracy, Edge Devices**

I. Introduction

Artificial Neural Networks (ANNs) achieve significant performance over many tasks, e.g., image recognition [1], natural language processing [2], game playing [3], etc. However, this success comes with the ever-increasing computational costs. The high computational costs make ANNs difficult to deploy in resource-constrained edge devices [4]–[7]. To improve computational efficiency, Spiking Neural Networks (SNNs) are proposed as a promising alternative to traditional deep learning approaches since they perform event-driven information processing [8], [9]. The high energy efficiency of SNNs stems from two-fold: 1) the sparse spike signals are transmitted and processed in SNNs; 2) the spike signals enable SNNs to replace the expensive Multiply-Accumulate (MAC) operations in ANNs with additions [10].

The key difference between ANN and SNN is the concept of time [11], [12]. Specifically, ANN performs the feed-forward pass once to complete a single inference, while SNN needs to perform the feed-forward pass over multiple time steps to complete a single inference. However, the requirement of neuron computation over the multiple time steps processing lessens the energy benefits of SNNs [8], [9], [13]. This is because existing software frameworks or SNN hardware adopt the time-driven execution mechanism, which makes the energy consumption and execution speed of SNN proportional to the number of time steps [14], [15]. With a large number of time steps, the SNN implementation not only increases the latency and energy consumption, but also introduces the frequent memory access of the membrane potential [16], [17]. To tackle this, one approach is to design a combination of SNN-ANN hybrid network, which can be leveraged to obtain the same accuracy as corresponding SNN in fewer time steps [13], [14]. Another route for efficient SNN computation is to use pruning techniques to significantly improve hardware performance by reducing the synapses of linked neurons [18], [19].

Several works have explored the applicability of pruning methods for SNNs. These approaches use pruning to reduce the number of synapses (similar to weights in ANN), and most of the analysis still focuses only on small-scale datasets (such as MNIST) or a certain type of SNN (e.g., directly training SNNs). In addition, these methods are similar to pruning applied in ANN, which requires fine-tuning to recover accuracy. More significantly, the feasibility of compressing the core computational units (i.e., neurons) in SNNs has not been explored in any of these prior works. Therefore, if the number of neurons involved in computation can be appropriately reduced in the SNN, there is scope to improve the energy efficiency of the SNN further since such neuron compression is directly correlated with computation.

Inspired by the organizational principle for integration and interaction of neurons in the neuroscience [20] and dynamic neural networks, in this paper, we propose a dynamic pruning method of neurons to improve the energy efficiency of SNN. SNNs have potential in terms of computational efficiency and characterization capability if they can dynamically adjust their structure while processing the input signal. This is because neurons are the basic computational units of SNN, enabling decreasing the number of neurons directly reduces the need for PE in SNN implementation. In addition, our pruning method is not restricted by the structural pruning applied in ANNs since the processing elements (PEs) are executed independently and in parallel in the SNN implementation.

Our contributions can be summarized as follows:

- We propose DynSNN, a SNN pruning framework for

Corresponding author: Fei Dai.

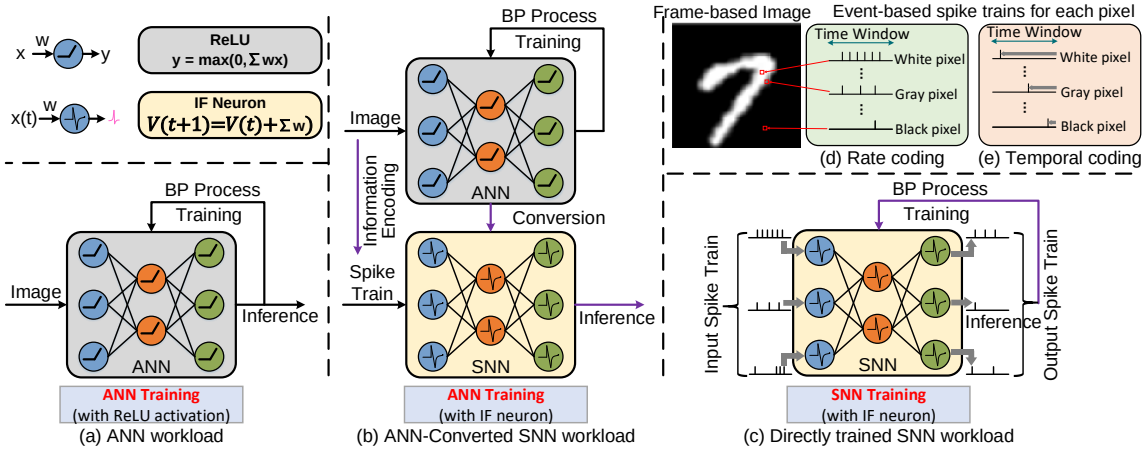


Fig. 1. Illustration of the workload varying in SNN types for spiking neural networks and an example about the image is converted into the spike train by various encoding methods. (a) general ANNs; (b) ANN-converted SNNs; (c) directly training SNNs using error backpropagation; (d) rate coding scheme; (e) temporal coding scheme. The time window represents the length of the spike train, which is equal to the number of time steps.

exploring the efficient and accurate SNN by removing neurons on the fly to make the trade-off between application accuracy and execution cost.

- We explore the feasibility of DynSNN by applying it to a wide variety of SNNs and discover the different operating mechanisms.
- The evaluation shows that directly trained SNN using DynSNN can achieve 60% compression rate without accuracy loss during training, and ANN-converted SNN can achieve 34% averagely compression rate with $< 0.3\%$ accuracy loss.

II. SNN Preliminaries

Biologically-inspired spiking neural networks are regarded as the third generation of neural networks developed to process information more similar to biological neural networks [21], [22]. SNNs have attracted the concentration of researchers with their rich information in the spatio-temporal domain, diverse coding mechanisms, and event-driven advantages. Unlike ANNs, SNNs are potentially capable of dealing with large volumes of data and using sparse binary spike events for information representation, leading to more power-efficient computation and communication primitives in specialized neuromorphic hardware.

In SNNs, neurons and synapses serve as the basic processing units and basic storage elements, respectively. The information is exchanged and transmitted among neurons via discrete action potentials or spikes [23]. Specifically, the typical structure of the neuron consists of three main parts: dendrite, soma, and axon [24]. The dendrite collects input spikes from other neurons and transmits them to the soma; the soma acts as a central processor, generating spikes (i.e., action potentials) when the membrane potential exceed a certain threshold caused by accumulating received spikes. The spikes propagate along the axon and are transmitted to the next neuron through synapses located at the end of the axon [5], [25].

Current SNNs can be categorized into ANN-converted SNN and directly trained SNNs consisting of supervised and unsupervised learning. For unsupervised learning, the mainstream learning method utilizes the spike timing-dependent plasticity rule (STDP). However, it is limited to shallow SNNs with small layers and yields much lower accuracy than ANNs on complex datasets (e.g., only 66.23% on CIFAR-10 [26]). On the other hand, the supervised methods represented by error backpropagation with surrogate functions can achieve better performance than the unsupervised ones, but they still can not provide compatible results with ANNs in large datasets.

Directly adapting the parameters of ANNs into SNNs, known as ANN-converted SNNs [9], [27]–[29] are converted from the pre-trained ANNs by replacing the activation function (e.g., Rectified Linear Unit (ReLU)) in ANNs with neuron function. Its network structure is usually the same as the source ANN, and network parameters are transformed from the source ANNs by simple operations such as scaling. By this method, the state-of-the-art methods for training ANN can be used to construct ANN-Converted SNNs and achieve competitive accuracy and the widest applicability, even on large datasets such as ImageNet.

III. DynSNN Framework

Overfitting due to parameter redundancy is a well-known problem for neural networks, including ANNs and SNNs. The pruning method is a successful way to avoid this problem. Here, we use pruning in SNNs. That is, we selectively mask some useless or even negatively acting neurons during the propagation of SNNs. The reduction in the number of neurons results in significant improvement in the efficiency of the SNN inference phase.

Recent research published in Nature Neuroscience [20] suggests that when new memories are created, it takes forgetting old experiences to free up storage space. This suggests that neurons act in different roles from each other. Inspired by this observation, we propose a bio-inspired pruning framework combined with dynamic neural networks, called DynSNN. Unlike general pruning methods that migrate directly from the ANN domain, DynSNN explores the contribution of neurons

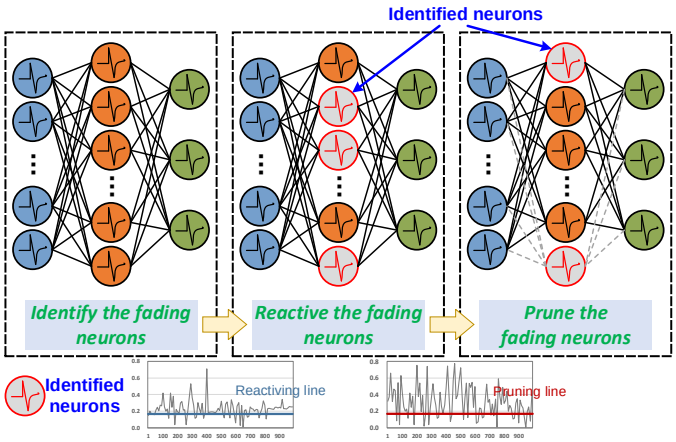


Fig. 2. Schematic of DynSNN applied in the SNN training process.

to the network's overall performance from the SNN itself. This aims to reduce the useless or even negatively acting neurons to improve the overall performance of the network and reduce the energy consumption of the SNN.

High-activity principal cells form the core of each memory, while low-activity cells enable their crosstalk. This reveals an organizational principle for continuous integration and interaction of memories. Based on these, to further measure the cellular activity (i.e., neuron activity in the SNN), we count the spike fired rate for each neuron as follows:

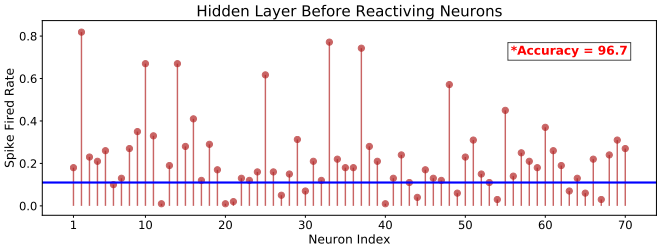
$$FSR = \frac{\#\text{fired spikes}}{\#\text{time steps}} \tag{1}$$

A higher value of the neuron activity means the larger number of spikes fired by the neuron, resulting in a greater contribution of this neuron to the whole SNN. If the activity of a neuron is low, we consider that this neuron has a low contribution to the SNN and can be eliminated, and we call these neurons as "fading neurons". In this case, the SNN that contains fading neurons may not achieve optimal performance.

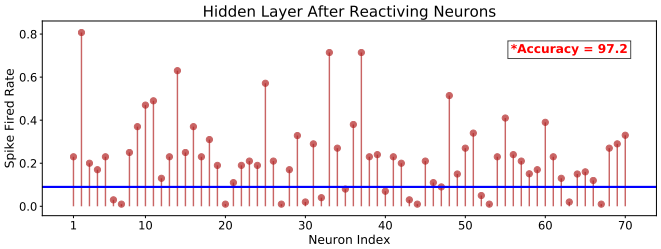
The goal of DynSNN is to learn the patterns of the network topology in each SNN, such that if a new SNN was introduced, our design could recognize the better network topology to which it belongs. This process of identifying the FSR can occur either in SNN training or in inference. Specifically, the synapses connecting the neurons in the SNN may be small or even negative (based on the backpropagation of the weight update), making it difficult for the accumulated membrane voltage of the neuron to reach the threshold voltage over time steps. This inhibits the firing spike frequency of the neuron, which means the neuron transmits less information or even no information. Then it can be considered as an "fading neuron" that has no contribution to the final prediction of SNNs.

For general pruning methods (both ANN and SNN), connection pruning based on magnitude thresholding (i.e., weight pruning) leads to unstructured sparsity that is difficult to be exploited in the hardware implementation. Fortunately, for DynSNN, this is not a problem. The reason is that in the hardware implementation of SNN, neurons are used as the base computational unit, i.e., the neuron calculation is mapped to the PE. In this way, directly reducing neurons in DynSNN not only does not add additional overhead but also directly reduces hardware resources.

Therefore, in the inference phase, we identify the fading neurons by the threshold determined during training and mask off these neurons from the SNN computation. To further improve the performance of SNN, we apply DynSNN in the training phase, as shown in Fig. 2, where we reactivate neurons by randomly initializing the weights connecting the neuron after identifying the fading neurons. If the neuron still shows the fading trend after the reactivation, these neurons are removed from the network topology.



(a)



(b)

Fig. 3. The change in firing rate value before and after applying DynSNN to neurons in the hidden layer for the directly trained SNN with three-layer fully connected layers.

It is worth mentioning that our method has more significant performance improvement for SNNs based on time-based encoding. The spike train generated by a time-based encoded SNN belongs to one-hot coding, i.e., the neuron generates the spike train with at most a single spike signal, such that this signal carries a large amount of information (spatio-temporal information). If the SNN has some fading neurons in the network topology due to the learning problem, limiting their training results. In this case, with our DynSNN, these neurons can be given a second chance to be reactivated and adjust the topology of the network again. Fig. 3 represents the values of the firing rate of a part of neurons in the hidden layer obtained from the directly trained SNN. The directly trained SNN consists of 3 fully connected layers (784-1000-10); the x-axis represents the neuron index, and the y-axis represents the firing rate value. The blue line in Fig. 3(a) and Fig. 3(b) indicate the value of the firing rate in the hidden layer that needs to be reactivated and those that need to be pruned, respectively. The reported accuracy is the classification accuracy of the SNN trained on the MNIST dataset.

TABLE I
DYNSNN PERFORMANCE ON VARIOUS SNNS.

SNN	Type	Network Structure	Encoding Method	DataSet	Acc. of Baseline	Acc. of DynSNN Reactived	Acc. of DynSNN Pruned	Acc. Drop	Comp. Rate of Topology
SNN-T [30] (ICASSP)	SNN training	784-340-10	Temporal	MNIST	97.9%	98.3%	98.0%	+0.1	53.71%
STiDi-BP [31] (Neurocomputing)	SNN training	784-500-10	Temporal	MNIST	97.4%	98.1%	97.7%	+0.3	62.33%
BP-STDP [32] (Neurocomputing)	SNN training	784-1000-10	Rate	MNIST	96.6%	97.2%	96.9%	+0.3	64.81%
RMP-SNN [28] (CVPR)	ANN-to-SNN	ResNet-20	Rate	CIFAR-10	91.36%	-	91.13%	-0.23	37.13%
VGGNet [29] (Frontiers in Neuroscience)	ANN-to-SNN	VGG-16	Rate	ImageNet	69.96%	-	69.65%	-0.31	31.71%

IV. EVALUATION AND DISCUSSION

A. Experimental Setup

We describe the functionality of the proposed DynSNN using a Pytorch framework. To assess the performance of the proposed DynSNN, we apply the DynSNN over two types of SNN, including directly trained SNN and ANN-converted SNN, on the image recognition benchmarks, namely MNIST, CIFAR-10, and ImageNet datasets. For the MNIST dataset, we consider directly trained SNN with the three-layer fully connected SNN network structure (Input-Flatten (784)-340/500/1000-IF-10-IF) as in previous works. For the directly trained SNN, we randomly initialize the synaptic weights in the ranges $[1, 10]$ for input and hidden layers and $[20, 50]$ for the classification layer. We apply DynSNN in the SNN training process. On the ImageNet and CIFAR-10 datasets, we use ANN-converted SNN proposed in our previous work to evaluate our DynSNN. We use a Poisson encoder in the open-source SNN framework SpikingJelly [33] to convert the static image into a spike train over the time steps. For the ANN-converted SNN, We adopt the identical weights initialization as [34] and train ANN without bias terms and batch normalization, according to [27], [35]. Upon ANN-SNN conversion, we apply DynSNN on the inference.

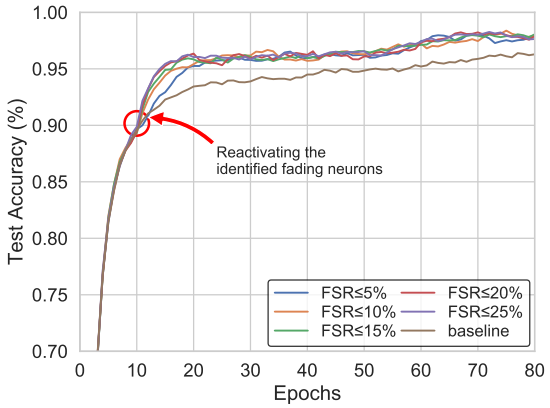


Fig. 4. Accuracy of SNN with three-layer fully connected layers, trained using spike-based error backpropagation on MINIST dataset, versus the number of identified fading neurons.

B. Discussion

Tab. I summarizes the compression performance of various SNN on MNIST, CIFAR-10 and ImageNet datasets. Directly trained SNNs composed of three fully-connected layers, using DynSNN, offer the opportunity to adjust the network topology again during training, yielding higher accuracy. Specifically, before the pruning, DynSNN first sets the FSR for identifying the fading neurons. Then, it exploits the identified fading neurons to adjust the network topology. At the end of the training, DynSNN prunes the neurons that are still in the fading state in the adjusted topology according to FSR. The results show that DynSNN in hybrid mode can achieve high classification accuracy and fast training. Moreover, it can achieve a high compression rate ($> 60\%$) with accuracy lossless. Tab. I also shows the classification accuracy of ANN-converted SNN using DynSNN in the inference. In terms of compression performance, DynSNN can prune ANN-converted SNNs with ResNet-20 and VGG-16 network structure $> 34\%$ with $< 0.3\%$ averagely accuracy drop. It can enable energy- and memory-efficient inference in edge devices for low-complexity tasks. We also compare our proposed DynSNN to other state-of-the-art pruning methods of SNNs, and the results are listed in Tab. II, from which we can find that the proposed method outperforms previous works. Note that for Deep R, ADMM-based and Grad R still prune the connectivity (i.e., the same as pruning weights in the ANN), while our approach is based on the activity of neurons. Once a neuron is removed from the network topology, all the connectivities (i.e., synapse weights) connected to that neuron are also pruned. In summary, the performance of DynSNN is comparable to Grad R and much better than Deep R and ADMM-based methods. Fig. 4 visualizes the classification accuracy of DynSNN using various FSR for reactivating the identified fading neurons. Our evaluation shows that SNN without DynSNN takes a long time to converge and yields worse performance. However, DynSNN identifies the fading neurons according to the FSR and exploits these fading neurons to optimize the training, enabling the SNN with larger FSRs to take less time to converge. The SNN performance is better than that without DynSNN.

TABLE II
PERFORMANCE COMPARISON BETWEEN DYNSNN AND PREVIOUS PRUNING WORKS ON MINIST DATASET.

Pruning Methods	SNN Type	Arch.	Acc. (%)	Acc. Drop(%)	Comp. Rate
Deep R [36]	SNN training	LSNN	93.70	+2.70	88%
ADMM [37]	SNN training	LeNet-5 like	99.07	-0.43	60%
Grad R [35]	SNN training	3 FC	98.92	-0.33	74.29%
this work	**SNN training**	**3 FC**	**99.23**	**-0.02**	**57.4%**
this work	**SNN training**	**3 FC**	**98.98**	**-0.27**	**69.7%**
this work	**ANN-to-SNN**	**LeNet**	**99.15**	**-0.35**	**61.5%**

ACKNOWLEDGMENT

We thank Wu Wen Jun Honorary Doctoral Scholarship, AI Institute, Shanghai Jiao Tong University.

REFERENCES

[1] Y. Chen, S. Zhang, and Z. Qi, "Maenet: Boosting feature representation for cross-modal person re-identification with pairwise supervision," in *Proceedings of the 2020 International Conference on Multimedia Retrieval*, 2020, pp. 442–449.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] F. Liu, W. Zhao, Y. Zhao, Z. Wang, T. Yang, Z. He, N. Jing, X. Liang, and L. Jiang, "Sme: Reram-based sparse-multiplication-engine to squeeze-out bit sparsity of neural network," in *Proceedings of the 39th IEEE International Conference on Computer Design*, 2021.

[5] L. Deng, Y. Wu, X. Hu, L. Liang, Y. Ding, G. Li, G. Zhao, P. Li, and Y. Xie, "Rethinking the performance comparison between snns and anns," *Neural Networks*, vol. 121, pp. 294–307, 2020.

[6] F. Liu, W. Zhao, Z. Wang, T. Yang, and L. Jiang, "Im3a: Boosting deep neural network efficiency via in-memory addressing-assisted acceleration," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021, pp. 253–258.

[7] F. Liu, W. Zhao, Z. He, Y. Wang, Z. Wang, C. Dai, X. Liang, and L. Jiang, "Improving neural network efficiency via post-training quantization with adaptive floating-point," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5281–5290.

[8] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[9] F. Liu, W. Zhao, Y. Chen, Z. Wang, and L. Jiang, "Spikeconverter: An efficient conversion framework zipping the gap between artificial neural networks and spiking neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[10] N. Rathi, A. Agrawal, C. Lee, A. K. Kosta, and K. Roy, "Exploring spike-based learning for neuromorphic computing: Prospects and perspectives," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 902–907.

[11] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling spike-based backpropagation for training deep neural network architectures," *Frontiers in neuroscience*, vol. 14, p. 119, 2020.

[12] L. Deng, H. Tang, and K. Roy, "Understanding and bridging the gap between neuromorphic computing and machine learning," *Frontiers in Computational Neuroscience*, vol. 15, 2021.

[13] S. Singh, A. Sarma, N. Jao, A. Pattnaik, S. Lu, K. Yang, A. Sengupta, V. Narayanan, and C. R. Das, "Nebula: a neuromorphic spin-based ultra-low power architecture for snns and anns," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 363–376.

[14] W. Ponghiran and K. Roy, "Hybrid analog-spiking long short-term memory for energy efficient computing on edge devices," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 581–586.

[15] S. Narayanan, K. Taht, R. Balasubramonian, E. Giacomin, and P.-E. Gaillardon, "Spinalflow: an architecture and dataflow tailored for spiking neural networks," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 349–362.

[16] H. Lee, C. Kim, Y. Chung, and J. Kim, "Neuroengine: A hardware-based event-driven simulation system for advanced brain-inspired computing," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 975–989. [Online]. Available: https://doi.org/10.1145/3445814.3446738

[17] A. Khodamoradi, K. Denolf, and R. Kastner, "S2n2: A fpga accelerator for streaming spiking neural networks," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 194–205.

[18] S. S. Chowdhury, I. Garg, and K. Roy, "Spatio-temporal pruning and quantization for low-latency spiking neural networks," *arXiv preprint arXiv:2104.12528*, 2021.

[19] N. Rathi, P. Panda, and K. Roy, "Stdp-based pruning of connections and weight quantization in spiking neural networks for energy-efficient recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 668–677, 2018.

[20] G. P. Gava, S. B. McHugh, L. Lefèvre, V. Lopes-dos Santos, S. Trouche, M. El-Gaby, S. R. Schultz, and D. Dupret, "Integrating new memories into the hippocampal network activity space," *Nature neuroscience*, vol. 24, no. 3, pp. 326–330, 2021.

[21] A. Taherkhani, A. Belatreche, Y. Li, G. Cosma, L. P. Maguire, and T. M. McGinnity, "A review of learning in biologically plausible spiking neural networks," *Neural Networks*, vol. 122, pp. 253–272, 2020.

[22] F. Liu, W. Zhao, Y. Chen, Z. Wang, T. Yang, and L. Jiang, "Sstdp: Supervised spike timing dependent plasticity for efficient spiking neural network training," *Frontiers in Neuroscience*, vol. 15, 2021.

[23] P. Rashvand, M. R. Ahmadzadeh, and F. Shayegh, "Design and implementation of a spiking neural network with integrate-and-fire neuron model for pattern recognition," *International Journal of Neural Systems*, vol. 31, no. 03, p. 2050073, 2021.

[24] W. He, Y. Wu, L. Deng, G. Li, H. Wang, Y. Tian, W. Ding, W. Wang, and Y. Xie, "Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences," *Neural Networks*, vol. 132, pp. 108–120, 2020.

[25] J. L. Lobo, J. Del Ser, A. Bifet, and N. Kasabov, "Spiking neural networks and online learning: An overview and perspectives," *Neural Networks*, vol. 121, pp. 88–100, 2020.

[26] G. Srinivasan and K. Roy, "Restocnet: Residual stochastic binary convolutional spiking neural network for memory-efficient neuromorphic computing," *Frontiers in neuroscience*, vol. 13, p. 189, 2019.

[27] B. Han and K. Roy, "Deep spiking neural network: Energy efficiency through time based coding," in *Proc. IEEE Eur. Conf. Comput. Vis.(ECCV)*, 2021, pp. 388–404.

[28] B. Han, G. Srinivasan, and K. Roy, "Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 13 558–13 567.

[29] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in neuroscience*, vol. 13, p. 95, 2019.

[30] I. M. Comsa, T. Fischbacher, K. Potempa, A. Gesmundo, L. Versari, and J. Alakuijala, "Temporal coding in spiking neural networks with alpha synaptic function," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8529–8533.

[31] M. Mirsadeghi, M. Shalchian, S. R. Kheradpisheh, and T. Masquelier, "Stidi-bp: Spike time displacement based error backpropagation in multilayer spiking neural networks," *Neurocomputing*, vol. 427, pp. 131–140, 2021.

[32] A. Tavanaei and A. Maida, "Bp-stdp: Approximating backpropagation using spike timing dependent plasticity," *Neurocomputing*, vol. 330, pp. 39–47, 2019.

[33] W. Fang, Y. Chen, J. Ding, D. Chen, Z. Yu, H. Zhou, Y. Tian, and other contributors, "Spikingjelly," https://github.com/fangwei123456/spikingjelly, 2020.

[34] M. Hardt and T. Ma, "Identity matters in deep learning," *arXiv preprint arXiv:1611.04231*, 2016.

[35] Y. Chen, Z. Yu, W. Fang, T. Huang, and Y. Tian, "Pruning of deep spiking neural networks through gradient rewiring," in *IJCAI*, 2021.

[36] G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass, "Long short-term memory and learning-to-learn in networks of spiking neurons," in *NeurIPS*, 2018, pp. 795—-805.

[37] L. Deng, Y. Wu, Y. Hu, L. Liang, G. Li, X. Hu, Y. Ding, P. Li, and Y. Xie, "Comprehensive snn compression using admm optimization and activity regularization," *arXiv preprint arXiv:1911.00822*, 2019.