Machine Learning in Applications 2023/GU01 - Bioinspired Spatial Data

Francesco Scalera, Giuseppe Pellegrino, Peter ALHachem, Andrea Pani
Politecnico di Torino

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Machine Learning in Applications 2023/GU01 - Bioinspired Spatial Data

***Abstract*—Spiking Neural Networks (SNNs) represent an innovative approach to artificial neural network modeling, drawing inspiration from the communication patterns of biological neurons through spikes or action potentials. These networks exhibit significant potential in low-power application environments, particularly in the domain of medical diagnosis using personalized patient data. In this study, we explore the conversion of a Convolutional Neural Network (CNN) optimized through evolutionary strategies into an SNN in order to perform a classification of miRNA expression datasets of cancer patients, aiming to retain maximum accuracy while enhancing the network's suitability for resource-constrained settings.**

Considering the diversity of optimization methods for artificial neural networks and the availability of Python frameworks for creating and converting SNNs, we developed two distinct pipelines, emphasizing the effectiveness of different implementation approaches and the complexity of proposed solutions. The first pipeline involves PyTorch for CNN architecture, NNI framework for hyperparameter optimization, and SNNTorch for SNN conversion. The second pipeline employs the Fast-Denser framework for automatic search and deployment of potentially deep artificial neural networks (DANNs) and NengoDL for SNN conversion.

Identifying an optimal neural network configuration that balances accuracy and parameter efficiency is crucial in various applications. Among the various configurations evaluated, we identify a spiking CNN denoted as SNN_PT_2, emerged as a strong contender for this classification task, thanks to its balance in memory footprint, classification accuracy and energy consumption.

I. Introduction

Images are a widely used and studied type of data in the various fields of Artificial Intelligence (AI) like Machine Learning (ML) and Deep Learning (DL), due to the patterns and textures that can be identified hidden in the spatial relationships of its components. Furthermore, different types of data, like gene expression from the biological data domain, inherently carries spatial correlation and can therefore be treated and analyzed as an image-like structure. The most common and widely used models to carry out an image classification task are Convolutional Neural Networks (CNN), as their architecture consisting of kernels and averaging operations successfully captures the spatial relationship of images. Despite their successes, CNNs are usually heavy in parameter size and computational power required during inference, making them not tailored to be deployed on so called "Edge Devices" like sensors.

Spiking Neural Networks (SNN) are a type of artificial neural network model inspired by the way biological neurons communicate through spikes or action potentials. As they use an event-driven approach, they use much less computational power and have therefore shown promising results in "edge applications" like in human wearable devices, which have potential to revolutionize medical diagnosis using personalized patient data.

In this work, we start from a dataset of miRNA expressions from cancer patients: first, we treat them as input to an optimized CNN architecture to find a baseline model; then we try to convert the model into a Spiking Neural Network, with the goal to retain the maximum accuracy possible while making the model more suitable to low power application environments.

II. Background

In recent years, the landscape of machine learning and neural network architectures has witnessed a remarkable shift. While Convolutional Neural Networks (CNNs) have dominated the field, delivering groundbreaking results in tasks such as image classification, a novel technology model has been steadily gaining recognition and momentum: Spiking Neural Networks (SNNs). This shift reflects a growing acknowledgment within the scientific and research communities that SNNs offer a unique and promising approach to mimic the brain's intricate processes. The quest for more brain-like computing models intensifies, the limitations of traditional feedforward neural networks have become increasingly evident. CNNs, although powerful, are fundamentally different from the brain's neural architecture, which operates through the spiking activity of neurons.

In this section, we will indulge in several definitions explored within prior research conducted additional to a clinical observation of frameworks used that solidifies our conceptual implementation of Spike Neural Networks (SNN).

A. *miRNA*

miRNA, or microRNA, is a type of small non-coding RNA molecule involved in the regulation of gene expression in various biological processes. As much as its role in gene expression is crucial, its dysregulation has been linked in numerous studies with the development and progression of various diseases, most notably cancer. In cancer, aberrant miRNA expression can contribute to the uncontrolled growth of cells, evasion of apoptosis (programmed cell death), invasion, metastasis, and angiogenesis. For this reason, miRNA expression is currently used as a molecular biomarker incorporated into the cancer diagnosis procedure but as more and more miRNA data is collected into databases, its analysis on large scale becomes increasingly difficult without resorting to automatized processes. For these reasons, Machine learning techniques can provide useful insights into miRNA correlation with tumors and potentially aid the medical professionals into the diagnosis process.

B. *snnTorch*

snnTorch, an innovative framework, has been meticulously crafted to cater to the unique demands of training Spiking Neural Networks (SNNs) within the realm of machine learning. At its core, snnTorch places a sharp focus on the gradient-based training of SNNs, harnessing the power of mathematical optimization to facilitate efficient learning.

This framework stands on the robust shoulders of PyTorch, a renowned platform celebrated for GPU acceleration and the streamlined computation of gradients. By leveraging PyTorch's capabilities, snnTorch brings forth a potent synergy, enabling both researchers and developers to embark on the training journey of intricate SNN models with the familiarity and might of the PyTorch ecosystem.

snnTorch isn't just a training tool; it's a comprehensive toolkit, equipped with an array of features and utilities tailored to the intricacies of SNN architecture. It facilitates the design and optimization of SNNs, paving the way for the creation of models that reflect the intricacies of biological neural networks.

One of its standout virtues is its efficiency in training and deployment on GPUs, a testament to its commitment to performance and scalability. What snnTorch achieves with distinction is bridging the divide between conventional Deep Learning (DL) and the realm of SNNs. This bridging potentially opens doors to groundbreaking avenues in AI research and development, where the realms of artificial and biological intelligence intersect and innovate. With snnTorch, the journey into the intriguing world of SNNs becomes not just accessible but truly transformative.

C. *Nengo*

Nengo serves as a valuable tool for the transformation of Convolutional Neural Networks (CNNs) into Spiking Neural Networks (SNNs). It provides a versatile Python library that simplifies the process of converting and simulating these neural models. Nengo's primary focus lies in offering an intuitive interface that streamlines the creation of both spiking and non-spiking neural simulations.

One of its key strengths is its adaptability, allowing users to define custom neuron types and learning rules, a crucial aspect when transitioning from traditional CNNs to SNNs. Additionally, Nengo can efficiently interface with hardware for input, making it a versatile choice for a wide range of neural network architectures.

In Nengo, we have three main building blocks: ensembles, nodes, and connections, which correspond to the framework principles of representation, transformation, and dynamics. When you combine these building blocks, you create networks and models. Additionally, Nengo has a feature called a "probe" that helps gather data during simulations.

Furthermore, Nengo's flexibility extends to its compatibility with various neural simulators, making it a valuable tool for researchers and developers looking to explore the possibilities of SNNs in different computational environments.

D. *FastDENSER*

The uniqueness of FastDENSER [1] lies in its elegant two-layer representation strategy. The outer layer skillfully encodes the overarching structural framework of the neural network, while the inner layer encodes the intricate hyperparameters associated with each network layer. This dual-layer approach empowers FastDENSER with the ability to intelligently navigate both the broad architecture and the fine-grained details of the network.

One standout feature of the FastDENSER module is its ability to efficiently manage learning time as it generates solution candidates. It's noteworthy that the time invested in training the initial generation of solution candidates is significantly smaller when compared to the later generations of deeper solutions. This elegant time scaling aligns seamlessly with addressing the challenge of static epoch definition, ensuring that the algorithm adapts dynamically to the complexities of the optimization landscape. FastDENSER thus represents a remarkable advancement in automated deep neural network design, offering both efficiency and adaptability in the pursuit of optimal model architectures.

E. *NNI*

NNI, or Neural Network Intelligence, is a sophisticated framework used in the field of machine learning, specifically for the crucial task of fine-tuning a model's hyperparameters. Hyperparameters are like the dials and switches of a machine learning model, and getting them just right can greatly impact the model's performance.

What sets NNI apart is its strategic approach to hyperparameter tuning. Rather than blindly exploring hyperparameters, it allows users to define these parameters precisely. Moreover, it offers a selection of sampling methods tailored to each hyperparameter, acknowledging that different parameters may require different tuning strategies.

Now, the ingenious part is NNI's use of Annealing. Imagine you're trying to find the highest point on a mountain. Initially, you might wander around randomly. But as you gain elevation and perspective, you become more deliberate, focusing on the peak. NNI operates similarly but in the hyperparameter space.

At the beginning of tuning, NNI takes random samples across the hyperparameter range. As the training progresses and it observes which settings result in better model performance, NNI becomes more focused. It gradually shifts its attention toward the promising areas of the hyperparameter space.

This dynamic strategy is grounded in the concept of the "response surface," representing how the model performs based on different hyperparameter settings. By navigating this surface intelligently, NNI converges towards the optimal hyperparameters, enhancing the overall performance of machine learning models. In essence, NNI acts as a skilled guide, efficiently steering the tuning process towards superior model configurations.

Example of reference to the previous Section Introduction I

III. Materials and Methods

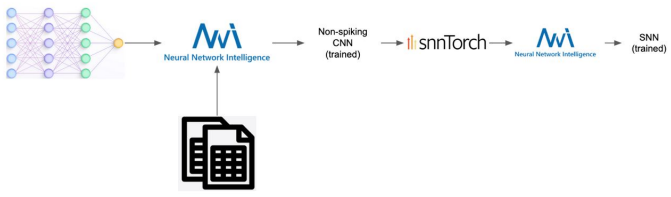
In this section we focus on describing the dataset used for this task, the model developed and frameworks and strategies that were used in the training process.

To analyze various approaches to the dataset classification task at hand, we explored different solutions to conduct a comparative study between different convolutional neural networks, both spiking and non-spiking. Our objective is to highlight the main differences between the various types of networks not only in terms of classification accuracy but also from a perspective that allows us to assess the computational resources and their memory usage in terms of trainable parameters. The workflow we followed consists of the following steps:

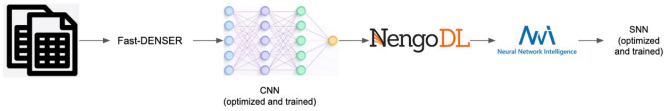
1) Find an optimized topology for a non-spiking CNN.
2) Convert the corresponding network into an SNN by replacing the so-called "rate" neurons (e.g., ReLU activation function) that output continuous values with neurons that produce a discrete output known as spikes.
3) Analyze and evaluate the different architectures in terms of accuracy and memory usage.

Considering that the current literature offers various methods for optimizing an ANN, along with the various Python frameworks available for creating Spiking Neural Networks or converting them from non-spiking models, we have developed two distinct pipelines, as showed in Fig. 1 , in order to investigate the effectiveness of different implementation approaches and assess the complexity of the proposed solutions.

1) The first pipeline involves creating a CNN architecture using PyTorch and optimizing its hyperparameters through the NNI framework, followed by the conversion into a spiking model using the Python SNNTorch package.
2) The second pipeline utilizes the grammar-based general-purpose framework Fast-Denser for the automatic search and deployment of potentially deep artificial neural networks (DANNs). This method optimizes a network architecture developed using TensorFlow Keras and then converts it into an SNN using the Python NengoDL package.



(a) PyTorch+SNNTorch pipeline.

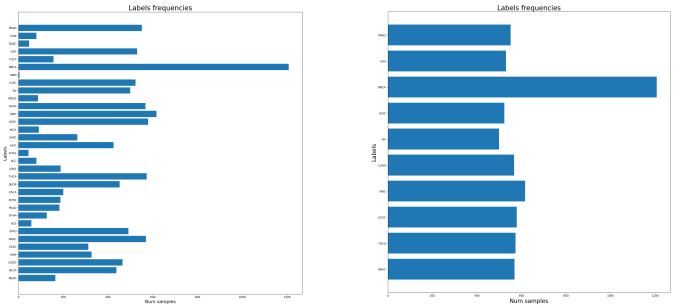


(b) Keras+NengoDL pipeline.

Fig. 1: Schema of the two adopted workflow pipelines

A. *Dataset and Data processing*

The dataset used in our experiment is comprised of miRNA gene expression samples taken from the Cancer Genome Atlas Program (TCGA). The samples belong to one of 33 tumor classes and have 1881 gene expression features. As initial experiments proved the task of classifying all of the 33 cancers very difficult due to highly unbalanced classes, we choose to focus our attention only on the 10 most frequent classes. To reduce the high variance and impact of extreme values in classification, a frequently encountered problem when working with biological data, we normalized the dataset using a z-score and then min-max scaled it to bring all the values in the range [0,1].



(a) Class frequencies plot for entire dataset (33 classes).

(b) Class frequencies plot for 10 most frequent classes.

Fig. 2: Class frequencies plots.

B. *PyTorch + SNNTorch pipeline*

*1) **Non-spiking network schema and hyperparaemeter optimization**:*

As seen in [2], a convolutional neural network (CNN) has proven to be effective for the classification of miRNA biomarkers. In this case, the proposed architecture consists of three convolutional layers, each followed by a Rectified Linear Unit (ReLU) and a max-pooling layer. Following these layers, there is a flattening layer and a dense layer.

The hyperparameters to be optimized include the number of filters, denoted as w_i, and the kernel size, denoted as wd_i, for each convolutional layer, as well as the window size, h_i, for each MaxPooling layer. In this regard, we have identified a set of independent hyperparameters $\{w_1, w_2, w_3, wd_1, h_1\}$ and a set of parameters dependent on h_1 and wd_1. This distinction was made to properly define the hyperparameters for the layers following the first one, which includes $\{wd_2, h_2, wd_3, h_3\}$. The hyperparameter tuning of the first set is carried out using the NNI toolkit, utilizing its built-in annealing tuner, which conducts a search in a predefined search space. Subsequently, for each trial performed with different combinations of hyperparameters from the first set, those from the second set are generated using Algorithm 1, which adequately returns a valid network architecture for the subsequent training. A summary of the hyperparameters contained in the search spaces employed for the Hyperparameter Optimization (HPO) is presented in Table I.

Each trial of the experiment involves training the network for 50 epochs, followed by an evaluation of the model on

a test set. This number of epochs has been chosen as the best compromise to achieve adequate accuracy in the shortest training time possible. The network is trained using the Adam optimizer with a constant learning rate of 0.001, and the Cross-Entropy Loss function is selected as the loss function to minimize.

After conducting the NNI experiment, two architectures were identified, which were subsequently converted into SNNs in the following steps: one architecture achieved the best accuracy result, referred as CNN_PT_1 hereafter, while the other represents the best compromise between accuracy, the number of parameters, and training time (CNN_PT_2 hereafter). In Appendix A the value found for each hyperparameter has been reported.

Algorithm 1 Generation of dependent hyperparameters set.

function GENERATEPARAMETERS(input_size, wd1, h1)
  $conv1\_out \leftarrow ((\text{input\_size} - 1 * (wd1 - 1) - 1) + 1)$
  $s1 \leftarrow ((\text{conv1\_out} - 1 * (h1 - 1) - 1)/h1 + 1)$
  $h2 \leftarrow \text{random}(2, s1)$
  $wd2 \leftarrow \text{random}(2, s1)$
  $conv2\_out \leftarrow ((s1 - 1 * (wd2 - 1) - 1) + 1)$
  $s2 \leftarrow ((\text{conv2\_out} - 1 * (h2 - 1) - 1)/h2 + 1)$
  while $s2 < 2$ **do**
    $h2 \leftarrow \text{random}(2, s1)$
    $wd2 \leftarrow \text{random}(2, s1)$
    $conv2\_out \leftarrow ((s1 - 1 * (wd2 - 1) - 1) + 1)$
    $s2 \leftarrow ((\text{conv2\_out} - 1 * (h2 - 1) - 1)/h2 + 1)$
  end while
  if $s2 = 2$ **then**
    $h3 \leftarrow 1$
    $wd3 \leftarrow 2$
  else
    $wd3 \leftarrow \text{random}(2, s2)$
    $h3 \leftarrow \text{random}(2, wd3)$
    $s3 \leftarrow (((s2 - wd3 + 1)/h3) + 1)$
    if $s2 = wd3$ or $(s2 - wd3 < h3)$ **then**
      $h3 \leftarrow 1$
    end if
  end if
  return $h2, wd2, h3, wd3$
end function

*2) **Conversion from a non-spiking CNN to an SNN and Hyperparameter Optimization**:*

After finding hyperparameters that optimize the architecture of a CNN (see Appendix A), we proceeded to convert it into the corresponding spiking version by replacing the "rate" neurons (ReLU) with Leaky Integrate-and-Fire (LIF) neurons. In summary, these neurons take inputs and, instead of passing them directly to an activation function, integrate the input over time with a leakage effect, similar to an RC circuit. If the integrated value exceeds a threshold, the LIF neuron emits a spike.

To conduct the experiments that we will describe later, we tried different types of LIF neurons supported by the SNNTorch package:

- 1st-order model (*Leaky*)
- Lapicque's RC model (*Lapicque*)
- Recurrent 1st-order model (*RLeaky*)

In this case, the experiments were conducted on two SNNs: one derived from the conversion of the CNN architecture with fewer parameters (referred to as SNN_PT_2), and the other derived from the conversion of the architecture that achieved the best test accuracy result (referred to as SNN_PT_1). At this point, two different types of network training were chosen for the experiments:

- Training to minimize the Mean Squared Error (MSE) Count Loss function.
- Training to minimize the Cross-Entropy Count Loss function, with weight transfer from the corresponding non-spiking CNN architecture that was previously trained.

In this phase, we also performed Hyperparameter Optimization (HPO) using the NNI toolkit. We selected hyperparameters to optimize not only related to the architecture of the spiking network but also to the way the network training is conducted. For example, in this case, we considered hyperparameters such as the learning rate and the number of steps (representing how long the input is presented to the network) in the set of parameters for optimizing the SNN. Each trial of the experiment involves training the network for 30 epochs, using the Adam optimizer for all trials, followed by an evaluation on test set.

C. *Keras + NengoDL pipeline*

*1) **Non-spiking Architecture Search**:*

FastDENSER [1] is an integral part of the **DENSER** Algorithm (Deep Evolutionary Network Structured Representation) [3], that represents a pioneering leap in automated deep neural network design. The methodology relies on evolutionary computation, where a group of individuals (population) undergoes iterative refinement over a specific number of generations. These individuals encapsulate Deep Artificial Neural Network structures (DANNs) and must be transformed into intelligible models to evaluate their effectiveness (fitness). Mutations are introduced to the population in order to encourage progress from one generation to the next. Specifically, Fast-DENSER makes use of a $(1 + \lambda)$-Evolutionary Strategy (ES): the succeeding generation consists of the most outstanding individual (elite), and λ mutations derived from it. The output of the framework is a fully-trained DANN, tailored to the considered problem.

The network structure is divided into three key components:

- Hidden Layers: This segment governs the sequence of evolutionary units used to construct Dynamic Artificial Neural Networks (DANNs). It is user-defined as an ordered list of evolutionary units. Each position in this list holds the grammar's non-terminal symbols, along with the specified range of allowable evolutionary units (minimum and maximum).
- Output: This component specifies the rule employed to construct the output layer of the network.

- Macro Blocks: These encompass the global network settings, including the choice of learning strategies.

Collectively, the network structure and the associated grammar collaboratively define the search space for this methodology.

In Fast-DENSER, the domain is established using a Backus-Naur Form (BNF) grammar, which serves to encode the hyper-parameters associated with each evolutionary unit. The structure of a grammar can be formally described through a 4-tuple representation: $G = (N, T, P, S)$, where: N represents the set of nonterminal symbols, T represents the set of terminal symbols, P represents the collection of production rules in the form $x ::= y$, where x belongs to N and y belongs to N or T, and S denotes the initial symbol.

At the conclusion of the experiment, we generated 15 successive generations, each comprising 10 distinct individuals. The phenotype is encapsulated within a singular string that delineates the network, its parameters, and the corresponding hyperparameters. For the ensuing phase, we opted for a model, that represent the best compromise between test accuracy and number of parameters, referred as CNN_K hereafter.

*2) **Conversion into spiking CNN and Hyperparameter Optimization**:*

We chose to implement the conversion to a Spiking Neural Network using the Nengo framework, taking account the integration with networks built using TensorFlow's Keras framework. Thanks to its specialized libraries, Nengo facilitates the straightforward conversion of a classical network into a spiking version, allowing for full customization of activation neuron types and their associated parameters. The primary objective of this phase was to ensure that the spiking network achieved performance comparable to its classical counterpart. To achieve this, we once again leveraged the NNI framework.

During the pipeline of this phase, we initially performed pre-training of the CNN to maximize performance. Subsequently, we conducted training of the CNN, with the network converted to Nengo without yet making it spiking. Finally, we completed the conversion to a spiking network and evaluated performance on the testing dataset.

In the process of training the network on Nengo, we applied 10 epochs using stochastic gradient descent as the optimizer. From our experiments, we observed that the model's performance significantly improved with sufficiently high scale firing rates. Without this scale firing rates, a notable reduction in the number of spikes was observed, as can be seen in Fig. 3.

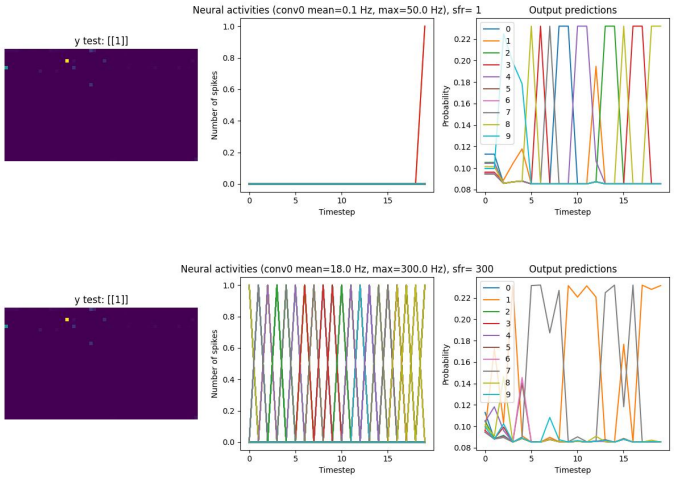


Fig. 3: Comparison of the effect of scale firing rates applied to the SNN_K network on the same test input. On the top one is applied a scale firing rates of 1 while in the bottom is applied a scale firing rates of 300

Network	Hyperparameters	Description
CNN_PT	w_1 w_2 w_3	Number of filters in convolutional layers
	wd_1 wd_2 wd_3	Kernel size of convolutional layers
	h_1 h_2 h_3	Window size of max pooling layers
SNN_PT_1, SNN_PT_2 (Training with MSE Count Loss)	beta	Decay rate of membrane potential
	learning_rate	Step size for weights update in each learning iteration
	num_steps	Repeating inputs for a certain number of temporal steps
	neuron_type	Type of LIF neurons applied to the network
	population_coding	Boolean flag indicating whether a neuron ensemble is performing
	neurons_per_classes	Number of neurons involved in population coding
SNN_PT_1, SNN_PT_2 (Training with Cross-entropy Count Loss)	beta learning_rate num_steps neuron_type	
SNN_K	syapse	A synaptic filter used to process the output of all neurons
	num_steps	
	activation	Type of Nengo spiking neurons applied to the network
	scale_firing_rate	This parameter amplifies the firing rates of spiking neurons

TABLE I: Hyperparameters description of NNI experiments performed on different architectures

IV. Results and Discussion

In classification problems, the focus is often on classification accuracy as the sole evaluation metric for a model. Despite its undeniable importance, it is necessary to introduce other network evaluation parameters for a more comprehensive analysis. Regarding the neuro-inspired solutions that we have developed, a comparison in terms of accuracy with non-spiking models alone may not be sufficient. In this context, we also wanted to analyze the impact on memory usage (in terms of the number of parameters) and the training time required to achieve adequate classification accuracy. Regarding the latter metrics evaluated, it is known that this measure is strongly dependent by the hardware where the training is performed. For this reason, we decided to consider training time as evaluation metric just to give an idea of time effort that different architectures needs, but it could change significantly depending on the hardware at hand. In this work, all the experiments are performed using a NVIDIA GTX 1080Ti GPU for models training.

Table II summarizes the evaluation metrics used and the corresponding results of the networks developed using the two previously described work pipelines.

Specifically, concerning classification accuracy, which remains a fundamental metric for this comparison, we observe that among the non-spiking models, the CNN architectures derived from the PyTorch pipeline achieve better results compared to those developed using the Fast-Denser algorithm. For example, CNN_PT_1 achieves a test accuracy of 74.57%

compared to the 73.09% of the CNN_K.

Having identified a second architecture, CNN_PT_2, that represents a good compromise between the number of parameters influencing memory usage, test accuracy, and training time, it can be noted that it achieves a test accuracy of 72.24%, a result comparable to that achieved by CNN_PT_1 and CNN_K but with significantly fewer parameters. Analyzing the results of the converted SNNs, it is observed that, in general, networks trained by minimizing the Cross-Entropy Count Loss function with weight transfer from the non-spiking network perform better compared to those trained by minimizing the MSE Count Loss without weight transfer. For example, the spiking network derived from the CNN_PT_1 architecture achieves 64.57% test accuracy when trained with MSE Count Loss, as opposed to the 74.89% achieved by the same network trained using Cross-Entropy Count Loss. Additionally, the second training strategy generally requires less time than the first. For these reasons, in the table II are only reported experiments that involve Cross-Entropy Count Loss for the spiking models.

At this point, considering only the experiments involving training with Cross Entropy Count Loss, a comparison needs to be made between the non-spiking architectures and their corresponding spiking versions. Regarding the PyTorch + SNNTorch work pipeline, in general, the spiking versions maintain the same number of parameters but manage to achieve slightly higher test accuracy compared to the non-spiking version. From the results of the experiments, a set of hyperparameters that improve the test accuracy of the corresponding non-spiking model is identified, but it sometimes leads to a considerable increase in the model's training time. The increase in training time is certainly related to the number of steps, an hyperparameter representing how long the input is presented to the network. For each spiking architecture, in addition to finding a set of hyperparameters that yields the best possible test accuracy, the table reports another set representing a trade-off between achieved accuracy, training time, and the number of steps.

In the Keras + NengoDL pipeline, the spiking model achieves slightly lower results compared to the non-spiking model (70.24% vs. 73.09%).

Combining the information obtained from the above results, it is clear that identifying the best network in terms of accuracy with the least possible number of parameters and not considerably increased training time is a priority.

Certainly, the spiking version of CNN_PT_2, referred to as SNN_PT_2, is a strong candidate to meet these requirements. For this model, hyperparameters have been identified that allow achieving a 75.06% test accuracy. Furthermore, conducting a deeper analysis reveals other hyperparameters with a lower number of steps, significantly reducing the training time while slightly decreasing accuracy to 74.41%.

Network name	Test accuracy (%)	Parameters	Training time
CNN_PT_1	74.57	3350772	4m52s
CNN_PT_2	72.24	80736	37s
SNN_PT_1	*best*: 74.89	3350772	37min30s
	tradeoff acc./timestep: 74.52		9min39s
SNN_PT_2	*best*: 75.06		8min23s
	tradeoff acc./timestep: 74.41	80736	1min36s
CNN_K	73.09	255357	55 s
SNN_K	70.24	255357	/

TABLE II: Summary of the evaluated metrics. The reported values are obtained with the optimal hyperparameters configuration for each network.

V. Conclusions and future works

We have introduced an approach for classifying cancer types using miRNA biomarkers. Our methodology involves converting a CNN into a spiking neural network (SNN) and optimizing and training it using two different workflows and implementations. These approaches have been tested on a real-world dataset, which includes 1,881 biomarkers for 6,214 patients.

Through multiple optimization experiments, we have explored the characteristics and performance of various neural networks, both spiking and non-spiking. Notably, among these SNNs, the spiking architecture derived from the CNN with a smaller number of parameters, called SNN_PT_2, represents an optimal solution for achieving high classification accuracies within a reasonable training time. In addition, as can be seen in [4], in general the spiking networks shows a significant reduction in energy consumption with respect non-spiking model, allows them to be used in the domain of on-edge applications.

In future work, a more in-depth analysis of the dataset could be conducted to mitigate class imbalance effects. Expanding the experiments to the entire dataset may lead to improved results, with a potential reevaluation of the hyperparameter search space considered in this work.

References

[1] F. Assunção, N. Lourenço, B. Ribeiro, and P. Machado, "Fast-denser: Fast deep evolutionary network structured representation," *SoftwareX*, vol. 14, p. 100694, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S235271102100039X

[2] A. Lopez-Rincon, A. Tonda, M. Elati, O. Schwander, B. Piwowarski, and P. Gallinari, "Evolutionary optimization of convolutional neural networks for cancer mirna biomarkers classification," *Applied Soft Computing*, vol. 65, pp. 91–100, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494617307615

[3] F. Assunçao, N. Lourenço, P. Machado, and B. Ribeiro, "Denser: deep evolutionary network structured representation," *Genetic Programming and Evolvable Machines*, pp. 1–31, 2018.

[4] V. Fra, E. Forno, R. Pignari, T. C. Stewart, E. Macii, and G. Urgese, "Human activity recognition: suitability of a neuromorphic approach for on-edge aiot applications," *Neuromorphic Computing and Engineering*, vol. 2, no. 1, p. 014006, feb 2022. [Online]. Available: https://dx.doi.org/10.1088/2634-4386/ac4c38

APPENDIX

In this appendix, we have listed all the hyperparameter values that were found through our experiments using the NNI framework.

Network	Hyperparameters	Values
CNN_PT_1	w_1	142
	w_2	226
	w_3	220
	wd_1	51
	wd_2	101
	wd_3	2
	h_1	8
	h_2	58
	h_3	1
CNN_PT_2	w_1	12
	w_2	85
	w_3	229
	wd_1	35
	wd_2	38
	wd_3	2
	h_1	12
	h_2	56
	h_3	1

TABLE III: Hyperparameters set for non-spiking architectures developed in PyTorch+SNNTorch pipeline.

Network	Hyperparameters	Values
SNN_PT_1 (Training with MSE Count Loss)	beta	0,75
	learning_rate	0,015
	num_steps	5
	neuron_type	lapicque
	population_coding	True
	neurons_per_classes	50
SNN_PT_2 (Training with MSE Count Loss)	beta	0,75
	learning_rate	0,001
	num_steps	5
	neuron_type	leaky
	population_coding	True
	neurons_per_classes	75

TABLE IV: Hyperparameters set for spiking architectures developed in PyTorch+SNNTorch pipeline, trained with MSE Count Loss.

Network	Hyperparameters	Values
SNN_PT_1 (Training with Cross-Entropy Count Loss), Best accuracy	beta	0,8
	learning_rate	0,0015
	num_steps	20
	neuron_type	lapicque
SNN_PT_1 (Training with Cross-Entropy Count Loss), Best compromise accuracy/timestep	beta	0,9
	learning_rate	0,002
	num_steps	5
	neuron_type	lapicque

TABLE V: Hyperparameters set for spiking architectures SNN_PT_1 developed in PyTorch+SNNTorch pipeline and trained with Cross-Entropy Count Loss.

Network	Hyperparameters	Values
SNN_PT_2 (Training with Cross-Entropy Count Loss), Best accuracy	beta	0,95
	learning_rate	0,0025
	num_steps	75
	neuron_type	lapicque
SNN_PT_2 (Training with Cross-Entropy Count Loss) Best compromise accuracy/timestep	beta	0,85
	learning_rate	0,0015
	num_steps	10
	neuron_type	lapicque

TABLE VI: Hyperparameters set for spiking architectures SNN_PT_2, developed in PyTorch+SNNTorch pipeline and trained with Cross-Entropy Count Loss.

Network	Hyperparameters	Values
SNN_K	synapse	0,01
	num_steps	90
	activation	1
	scale_firing_rate	300

TABLE VII: Hyperparameters set for spiking architecture developed in Keras+NengoDL pipeline.