Neural coding: adapting spike generation for embedded hardware classification

Nassim Abderrahmane, Benoît Miramond
Université Côte d'Azur, CNRS, LEAT, France
firstname.lastname@univ-cotedazur.fr

***Abstract*—Recent literature considers that Spiking Neural Networks are now a serious alternative to Formal Neural Networks for embedded artificial intelligence. The changes in the information coding and the elementary neural computation make them more efficient than FNNs in terms of power consumption and chip surface occupation. However, these results are often based on simple neural network topologies with basic data-sets. In this paper, we study the behavior of Spiking Convolutional Neural Networks when applied to two different classification tasks. To do so, we analyze the spiking activity on both MNIST and GTSRB data-sets using different rate-based and temporal coding schemes. Notably, the Spike Select method is confronted to First Spike and Jittered Periodic methods in terms of prediction accuracy and spiking activity. Finally, we conclude about spike generation within spiking CNNs for embedded hardware classification.**

***Index Terms*—Spike Generation, Temporal Coding, Rate Coding, Spiking Neural Networks, Convolutional Neural Networks, Embedded Systems, Neuromorphic Computing**

I. Introduction

Convolutional Neural Network (CNN) is the most used deep learning architecture. It is inspired from the biological visual perception mechanism of living beings. Indeed, Hubel and Wiesel have found, in 1959, that animals have cells in their visual cortex that detect the light present in their receptive field [1]. Inspired by this finding, Kunihiko Fukushima proposed in 1980 the neocognitron, which is considered as the CNN's predecessor[2]. A decade later, LeCun et al. [3] proposed the revolutionary framework of CNN that is presented as a neural network of several layers of different types, called "LeNet-5". This Artificial Neural Network (ANN) is specifically used to ensure the handwritten digits classification task.

More than a decade later, several models have been proposed, they came with improvements and facilitation in training CNNs. Krizhevsky et al. proposed one of them, AlexNet [4], it has similar structure to LeNet-5 but with more layers. This model was very successful, it brought out many methods coming from other works that improved classification performances. Recent advances on CNNs can be found in the paper proposed by Jiuxiang Gu et al. [5].

Meanwhile, CNNs have become more popular with applications in several domains such as image and video recognition, image classification, medical image analysis, and natural language processing. This success can be awarded to two factors. First, computing capabilities of modern CPU/GPU based computers that accelerated implementation and inference stages. Second, the huge amount of available open source labeled data for training CNNs, which increased the number of applications and contributed to the improvement of CNN models.

The achievements of CNNs on image classification have given them the leading role in machine learning algorithms and Artificial Intelligence (AI) research. Indeed, more and more applications such as smart devices, IoT or autonomous vehicles require embedded and efficient implementation. However, it is difficult to integrate them in low power systems because their implementation on CPU/GPU is resource-intensive. Therefore, the solution is designing dedicated systems fitting the parallel and distributed computing aspects of neural algorithms. Recent literature considers the third generation of neural algorithms, i.e. Spiking Neural Networks (SNNs), as the alternative to Formal Neural Networks (FNNs) for embedded artificial intelligence. In this context, L. Khacef *et. al* [6] have shown that SNNs are around 50% more efficient in terms of power consumption and chip surface occupation compared to FNNs for simple topologies. To implement their SNNs, they have used an approach consisting in mapping FNNs to SNNs, which is found in several works. In [7] work, a thorough description of this method is presented, where they have applied it to classify data coming from asynchronous sensors. Similarly but with static images, P. U. Diehl *et. al* [8] adopted the same approach and studied the spiking activity of SNNs generated using rate coding. On the other hand, H. Mostafa [9] proposed a supervised learning algorithm based on temporal coding instead of rate-based coding. Also, S. R. Kheradpisheh *et. al* in [10], used temporal coding and proposed a Spiking Deep Neural Network (SDNN) that consists in an STDP-based CNN combined with a Support Vector Machine classifier. Recently, more exhaustive works have studied the integration of deep SNNs in embedded systems [11], [12].

Nevertheless, none of these works have studied all aspects of implementing convolutional SNNs for realistic embedded applications simultaneously. Indeed, these aspects concern the neural coding (temporal and rate-based paradigms), the neural model (FC-based SNNs and CNNs) and the hardware architectural design (parallel and multiplexed computing). In this paper, we propose to extend these works for better covering embedded applications through the use of spiking CNNs with various neural coding schemes. A design space exploration of hardware SNNs framework is proposed within new neural coding techniques characterized by a low number of generated spikes while maintaining the same neural model. Indeed, for the design of SNNs in hardware, we follow the design

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

flow framework illustrated in figure 1, where we first build a neural network using a learning framework (TensorFlow, Torch, Caffe, CNTK or N2D2) to extract the SNN's parameters and topology.

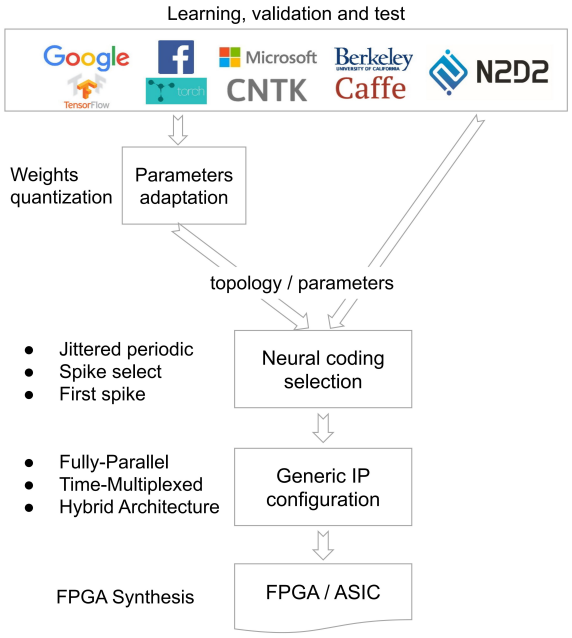

Fig. 1: Design space exploration of hardware SNNs framework

Then, we select a coding method and an architectural model to generate the SNN architecture on FPGA. Here, we intervene in the neural coding part by adapting the methods proposed in [11] to spiking CNNs. Moreover, we study the impact of neural coding on the performance and the efficiency of spiking CNNs in the perspective of their implementation onto different neuromorphic hardware chips [13], [14], [11], [12]. Indeed, for validating this adaptation of the spike generation methods, we use, in addition to MNIST data-set [15], the German Traffic Sign Recognition Benchmark (GTSRB)[16]. The different explored coding methods are First Spike, Spike Select and Jittered Periodic. This work is an extension of the previous work [11] to the case of convolutional topologies where we evaluate the impact of spike-select coding compared to classical rate coding.

The remaining of the paper is composed of three sections. First, in section II we present the CNN's structure with brief definition of the most used layers. In the same section, the transcoding method and neural coding techniques are defined. Second, in section III, results of coding techniques on CNNs are shown, then we discuss them in section IV. Finally, we present conclusions and some future works in section V.

II. Material and Methods

A. Spiking versus formal neural coding

In this work, we adopt the transcoding approach, consisting in transforming neural networks from formal to spiking domain. This approach has been already studied in [7][8][11] but few studies have explored the impact of spike coding on both accuracy and power consumption with spiking CNNs.

To do this mapping of CNNs, we first train the neural model in formal domain then export the resulting parameters to be used with the spiking CNN. Second, we test both formal and spiking CNNs with the testing data-set. Finally, if accuracy results are satisfying, then these networks are ready for inference.

Note that there exist learning techniques directly applicable in spiking domain, such as SpikeProp or STDP [17] [9] [10] [18]. However, we focus on training in a supervised manner and with feed-forward network topologies using back-propagation learning algorithm. Indeed, this learning method is more mature and is largely deployed [6][11][19][20][21]. It is currently a more serious competitor to the standard CNN approaches than other, very promising but exploratory, learning methods.

For the neuron model, among several models identified in neuroscience literature and in a machine learning context, we use spiking neurons that are most often based on a simple (Leaky) Integrate and Fire (IF) model. This neuron model is used in the transcoding method to substitute the Perceptron of the originate CNN. The efficiency of IF neurons compared to the Perceptrons is due to their computation rules. Equations 1 and 2 depict these computation rules, the first equation for formal coding and the second one for spike coding.

$$y_j^l(t) = f(s_j^l(t)), \quad s_j^l(t) = \sum_{i=0}^{N_{l-1}-1} w_{ij} * y_i^{l-1}(t) \qquad (1)$$

Where $y_j^l(t)$ is the output of j^{th} neuron of layer l, $f()$ is a non-linear activation function, $s_j^l(t)$ is the membrane potential of j^{th} neuron of layer l and w_{ij} the synaptic weight between i^{th} neuron of layer $l-1$ and j^{th} neuron of layer l.

$$\gamma_j^l(t) = \begin{cases} 1 \text{ if } s_j^l(t) \geq \theta \\ 0 \text{ otherwise} \end{cases},$$
$$p_j^l(t) = \begin{cases} s_j^l(t) \text{ if } s_j^l(t) \leq \theta \\ s_j^l(t) - \theta \text{ otherwise} \end{cases}, \qquad (2)$$
$$s_j^l(t) = p_j^l(t-1) + \sum_{i=0}^{N_{l-1}-1} (w_{ij} * \gamma_i^{l-1}(t))$$

Where $\gamma_j^l(t)$ is the spiking output of j^{th} neuron of layer l, $p_j^l(t)$ is the membrane potential of j^{th} neuron of layer l, and θ is the activation threshold of j^{th} neuron of layer l.

In a context of hardware implementation, it turns out that the presence of multiplications and a non-linear function $f()$ in equation 1 are resource-intensive, whereas only additions and comparison are present in equation 2 which are more adapted to hardware integration.

In addition to their greater ease of integration onto neuromorphic hardware, spiking CNNs naturally take advantage of event-driven computation. This paradigm assumes that only

events are processed, reducing the global chip activity to the units triggered by the flow of events. But results obtained with standard rate-coding schemes do not ensure that spike coding generates less information than activity-based coding. Indeed, rate-coding generally imposes additional latency in the presentation of samples that counterbalances spike coding. This is why we are interested in the following in exploring other forms of coding that are more sparsely distributed and therefore more efficient for low-power embedded AI applications.

B. Convolutional Neural Networks

Convolutional Neural Networks are algorithms used for various AI applications. They are multi-layer ANNs composed of a various number of layers with different types. The most commonly used to build a CNN are convolution, pooling and fully-connected layers.

Convolution layer: The convolution layer is composed of kernels or filters that are applied to an input image or Feature Map (FM) to extract specific features. To do so, each kernel has its unique weights that are defined after the learning process and applied at different positions of the input image/FM. The number of these positions correspond to the size of the convolution layer's output feature map that is computed using: $size = W * W$, where W is defined in equation 3.

$$W = (N - F + 2 * P)/S + 1 \tag{3}$$

Where F is the receptive field width (filter width), P is the padding, S is the stride and N is the input FM width. As shown in figure 2, LeNet-5 CNN is used for classifying traffic signs found in GTSRB data-set. Here, there are three convolution layers, one connected to input and two others connected to the first and second pooling layers. The first layer is composed of 6 convolution kernels with the size of 5x5 that slide over the input image using a stride equals to 1. The output of this convolution layer is 6x28x28, i.e. 6 FMs of 28x28 size, this is obtained by applying the formula in equation 3.

Pooling layer: Pooling layers, also called sub-sampling layers, are used to reduce the size of input FMs. Actually, they are composed of average or maximum pooling kernels that are applied to input FMs using similar sliding technique to convolution layer. Therefore, to compute its output feature map size we use the same formula found in 3. For example, in the third layer from the left in figure 2, applying 6 pooling kernels (of 2x2 size and with a stride of 2) on 6x28x28 FMs results in feature maps of 6x14x14 size.

Fully-Connected layer: Fully-connected layers are generally located at last stages of a CNN network. They come to process the output FM of the last CNN convolution or pooling layer, they are considered as the network's classifier. These layers are composed of neurons connected to all previous layer nodes.

C. Neural coding methods

Alongside the section, three different spike generation techniques used to encode data into spiking events with spiking CNNs are presented. The methods are: Jittered Periodic, First Spike and Spike Select.

Note that, the same generation process is used with both gray level and RGB formats of input data. When applied to RGB images, the process is applied to the three channels of the image independently.

1) Jittered Periodic: Jittered Periodic is a rate-based coding method, it transforms analog input data to a spiking train. For example, the transcoding of a gray scale image will generate train spike with a rate proportional to its intensity or brightness.

With Jittered periodic an input information ($value$) is converted to spike domain as follows:

- First, defining some parameters that are used for the conversion process: f_{min} & f_{max} the minimum and maximum frequency parameters and s_{dev} the relative standard deviation.
- Second, using the formula defined in equation 4, the input information referred as $value$ is transformed to a period.
- Afterwards, the time step is calculated using equation 5.
- Finally, the time step is integrated to the previous emitted spike's time to get the next spike emission according to equation 6.

$$period = 1/(f_{max} + (1 - \mid value \mid) * (f_{min} - f_{max})) \tag{4}$$

$$\Delta t = f_{Udist}(f_{Ndist}(period, s_{dev})) \tag{5}$$

$$t = t_{previous} + \Delta t \tag{6}$$

Note that $f_{Ndist}()$ and $f_{Udist}()$ are the Random Normal Distribution and Uniform Distribution functions.

This process is repeated until the classification process is finished. Indeed, for the class selection, we are using the Terminate Delta procedure that consists in enacting the classification process when the difference between the most and second most spiking neurons is equal to *delta* spikes.

2) First Spike: First Spike method, compatible with the IF-neuron as in rate-coding, is used to represent input information using at most one spike. Therefore, it is an intermediate version between temporal and rate codings since it uses only one spike per pixel and IF-neuron within the SNN.

In fact, the generation process is similar to Jittered periodic method which is presented in section II-C1. However, it differs by the fact that only the first generated spike per pixel is kept and therefore the generation process is applied at most once for each input pixel. Doing so, the number of spikes propagated in the network should be drastically reduced but will induce accuracy losses.

3) Spike Select: The steps to follow for spike generation with Spike Select are similar to Jittered Periodic method. However, within this method the SNN's spiking activity over the network is regulated to optimize the hardware architecture utilization. Statistical results illustrated in figure 7 show that

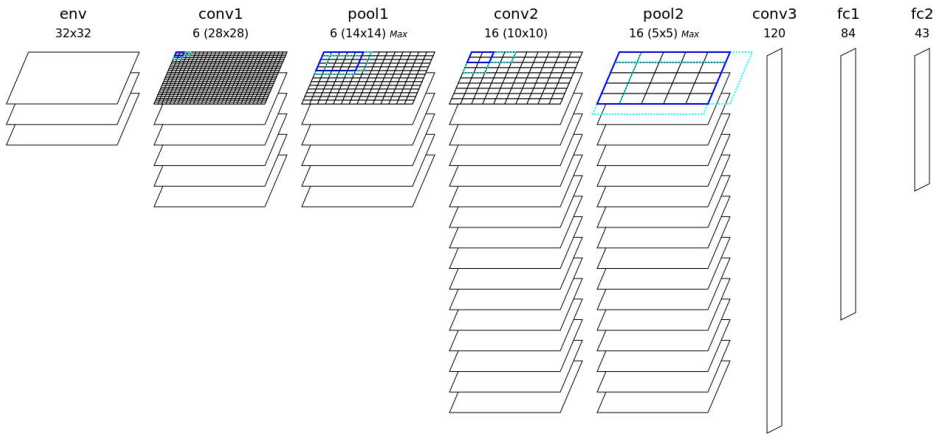

Fig. 2: LeNet-5 CNN network architecture used for classifying traffic signs found in GTSRB benchmark – generated from the N2D2 framework [22]

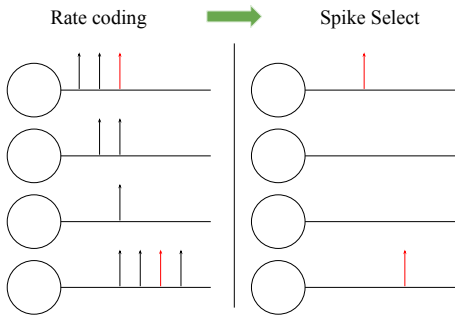

Fig. 3: Filtering spikes with Spike Select method. Rate coding neurons threshold is equal to 1 and that of spike select neurons is equal to 3.

most of this spiking activity is located at output of the first convolution layer. Therefore, this is the layer which is the most solicited for computing.

In the purpose of reducing outgoing spikes of this convolution layer, we propose with Spike Select method to set a filter within this layer. In fact, reducing outgoing spikes from this stage makes deeper layers receiving less events.

Actually, a simple manner to integrate this filter is feasible by just raising the IF-neurons membrane threshold which is set by default to unity. Doing so, we are able to reduce neurons output spikes. For example, in figure 3 a filter is applied to some neurons by increasing their potential threshold from 1 to 3, which caused a reduction of output spikes from 10 to 2.

Moreover, the terminate delta is also reduced since fewer spikes are propagated in deeper layers of the SNN and therefore only few spikes are necessary for classification.

It has been already shown in [11] that by using Spike Select the amount of spikes propagating in the SNN is drastically reduced. However, this study did not covered CNNs and was based on SNNs with only FC layers. This work extends the study to the case of spiking CNNs and evaluates the impact of using spike select compared to classical rate-coding.

III. Results

In this section, we experimented the different coding methods (Spike Select, Jittered Periodic and First Spike) on spiking CNNs. To do so, first, we used two training data-sets, the handwritten digits MNIST data-set and the traffic signs GTSRB data-set.

On one hand, MNIST data-set is made of 70 000 gray-level images, 60 000 for learning and validation, 10 000 images for testing [15].

On the other hand, GTSRB database has 43 classes spread on 51840 colored (RGB) image samples (figure 4), 50% of them is used for training 25% for validation and the remaining 25% is used for testing. The images have different sizes ranging from 15x15 to 250x250, therefore re-scaling the input within the CNN is mandatory[16].

Fig. 4: Examples of the German Traffic Sign Recognition Benchmark – GTSRB [16]

We defined different CNN topologies for both data-sets. First, four models were compared for using with MNIST data-set that are *Conv29*, *ConvPool*, *LeNet* and *ConvFc*. Where *Conv29* is a CNN with a 1D input with a size of 29x29x1 followed by 2 convolution layers, 2 pooling layers and 2 Fc layers raising to the topology: "29x29x1-32c4s1-32p2s2-48c5s1-48p3s3-200-10"; *ConvFc* is a CNN composed of only convolution and fully-connected layers (28x28x1-16c4s2-24c4s2-150-10) and *ConvPool* is composed of only convolution and pooling layers (28x28x1-12c5s1-12p2s1-64c5s1-64p2s1-10).

On another hand, 4 topologies with two gray-level (indexed by "-1D") and two RGB (indexed by "-3D") are built for

TABLE I: Classification accuracy results of different SNNs on MNIST data set. Note that, all the networks are in spiking domain.

Network	Accuracy (%)
In [8]: *784-2*(1200)-10*	98.60
In [9]: *784-800-10*	97,55
In [6]: *784-300-10*	95.37
In [23]: *784-300-10*	95.40
In our previous work [11]: *784-300-10*	97.74
In our previous work [11]: *784-2*(300)-10*	98.00
In our previous work [11]: *784-3*(300)-10*	98.24
In this paper: *Conv29*	99.21

GTSRB data-set. *LeNet-1D* and *Conv-1D* the gray level networks with "32x32x1-6c5s1-6p2s2-16c5s1-16p2s2-120c5s1-84-43" and "29x29x1-32c4s1-32p2s2-48c5s1-48p3s3-200-43" topologies. *LeNet-3D* and *Conv-3D* are the colored input networks with "32x32x3-6c5s1-6p2s2-16c5s1-16p2s2-120c5s1-84-43" and "32x32x3-6c5s1-6p2s2-16c5s1-16p2s2-120c5s1-84-43" topologies.

Finally, we tested these networks using the coding methods presented in section II and recorded their results in terms of accuracy and spiking activity. Note that in this work, we are studying neural coding impact on the energy-efficiency of Spiking CNN architectures, and that is why we are principally focusing on reducing the spikes propagating in the network while keeping approximately the same accuracy. Therefore, the metrics used to discriminate the coding methods are accuracy and spiking data propagation.

A. Accuracy results

The defined CNNs were tested on corresponding databases (MNIST & GTSRB) using the neural coding methods presented earlier. Here, we present prediction accuracies obtained from these experiments.

Note that networks are first trained and validated in formal domain and then transformed to spiking domain. Moreover, these results represent averages of several runs.

TABLE II: Different classification accuracy results on GTSRB data-set

Network	Accuracy (%)
Human [16]	97.98
LeNet-5 in [24] – formal	92.68
Random forests [25] – formal	96.14
Architecture_32 [24] – formal	97.79
This work: Conv-3D – spiking	97,83

In tables I and II, accuracy results obtained in this paper are compared to different networks that we found in literature for MNIST and GTSRB data-sets. On one hand, with MNIST data-set, we obtained slightly higher accuracy compared to others. On the other hand, with GTSRB data-set, we get lower results compared to human[16] and Architecture_32 network [24] but higher than the other networks.

Tables III and IV represent respectively accuracy results of CNNs using 3 different coding methods on MNIST and GTSRB data-sets.

TABLE III: Accuracy results of 4 different CNN models on MNIST data using 4 different coding techniques

Coding method	CNN model			
	Conv29	ConvPool	LeNet	ConvFc
Jittered Period	99,16	**99,21**	99,15	**99,09**
Spike Select	**99,21**	97,07	99,13	98,68
First Spike	87,21	96,56	90,87	96,16
Formal domain	99,16	99,18	**99,18**	**99,09**

TABLE IV: Accuracy results of 4 different CNN models on GTSRB data using 4 different coding techniques

Coding method	CNN model			
	Conv-1D	Conv-3D	LeNet-1D	LeNet-3D
Jittered Period	**96,68**	97,75	**95,76**	**96,19**
Spike Select	96,56	**97,83**	95,72	96,18
First Spike	21,57	45,49	25,46	68,61
Formal domain	96,8	97,86	95,84	96,25

On one hand, we remark that the First Spike method performs lower results on MNIST data-set with more than 9% accuracy loss with Conv29 and LeNet CNNs. Moreover, a huge loss is recorded when confronted to GTSRB data-set with scores varying from 21.57% to 68.61%. This method shows competitive spiking activity compared to others, but its important losses in accuracy makes it not viable for classification applications. Moreover, histograms on figure 5 illustrate this difference more clearly, especially for GTSRB data-set.

On the other hand, Jittered Periodic and Spike Select methods are more accurate and present equivalent records compared to formal CNNS. Therefore, in the following we compare these methods in terms of spiking activity and prediction performance.

B. Spike Select versus Jittered Periodic

Indeed, Spike Select and Jittered Periodic are more promising for classification regarding their performances. Here, we analyze and compare the spiking activity using these methods with CNN networks. In fact, in a context of embedded applications, the energy consumption of an SNN hardware implementation is directly proportional to the number of spikes it generates [26]. Therefore, this spiking activity is represented as the average number of spikes propagated over the SNN for one image.

In figure 6, the total number of spikes generated per image propagated on different CNNs with MNIST and GTSRB data-sets is illustrated. These results show that Spike Select is more efficient, it generates less spikes than Jittered Periodic for all CNN topologies and data-sets. Moreover, on GTSRB data-set, it is generating around 50% or more less spikes while keeping similar recognition rates, as mentioned in tables IV and V.

IV. Discussion

A. Efficiency of spike coding

In this paper, we have explored neural coding within spiking convolutional neural networks in a context of embedded classification. The objective of this study was to extend a design

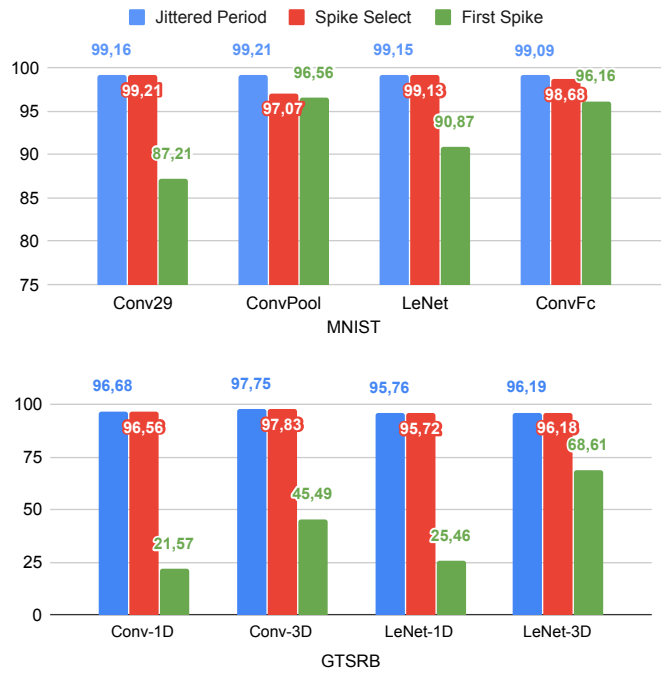

Fig. 5: Spiking CNNs accuracy performance on MNIST and GTSRB data-sets using different coding methods

TABLE V: Spike Select versus Jittered Periodic: spikes distribution over spiking gray-level and RGB LeNet CNN layers using GTSRB data-set

Layer	LeNet-2D		LeNet-3D	
	JitPeriod	SpikeSel	JitPeriod	SpikeSel
Input	3295	3110	7692	6698
Conv1	12581	3028	15048	3553
Pool1	4160	1046	4956	1207
Conv2	4585	1463	4799	1542
Pool2	2209	723	2346	762
Conv3	115	37	91	29
Fc1	77	23	67	20
Output	44	8	28	5
Total	27066	**9438**	35027	**13816**

space framework of hardware spiking neurons, presented in figure 1. In fact, this framework allows the implementation of SNNs in neuromorphic hardware while having the ability of adapting the chip to the application constraints. To do so, users have possibility to choose the neural coding technique, the ANN topology and the hardware architectural model.

Nevertheless, this framework initially covered SNNs composed of only fully-connected layers, which are not sufficient to conceive applications needing other layer types, like convolution and pooling. Therefore, we have adapted some spike generation techniques to be used with spiking CNNs and analyzed their behavior. In this context, we focused on two factors: the prediction accuracy and the spiking activity. Moreover, different experiments are made using various CNN topologies on two different classification tasks to verify integrability of spiking CNNs on neuromorphic hardware.

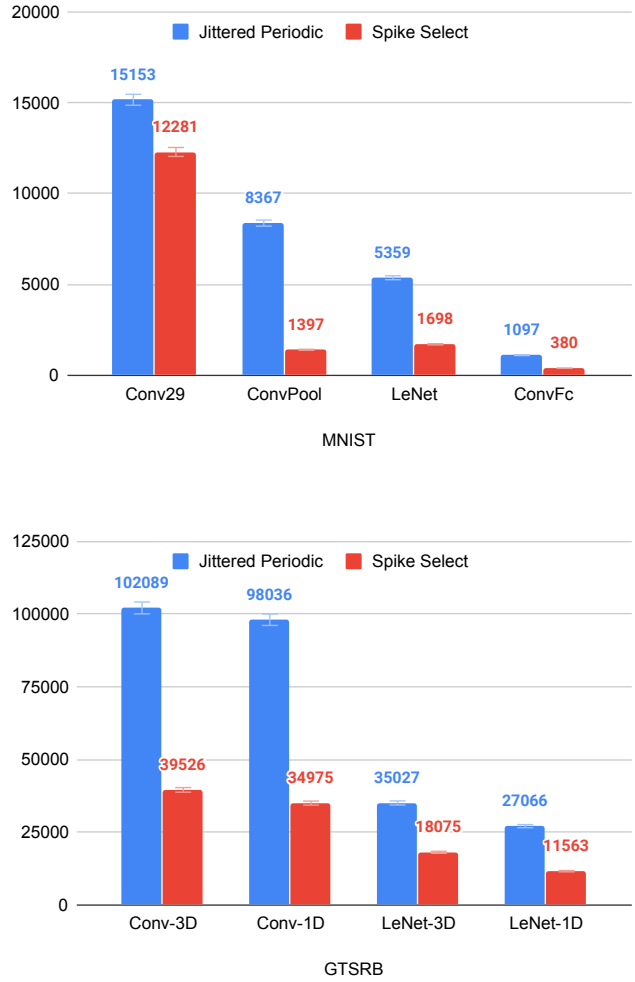

Fig. 6: Total number of spikes propagated in different network layers using Spike Select and Jittered Periodic methods

We have seen that First Spike method is not adequate for spiking CNNs because it presents important accuracy losses compared to CNNs in formal domain. Therefore, we have selected Spike Select and Jittered Periodic as potential solutions for spike generation in convolutional SNNs. In this way, we have compared their accuracy records and spiking activities on two data-sets (MNIST and GTSRB) using different topologies. In terms of recognition rate, both methods perform results equivalent to formal CNNs scores. Whereas, in terms of spiking activity, with Spike Select method we are able to regulate the spiking distribution over the network to correlate with results found in [11] work. Moreover, the amount of spikes propagating over network's layers is drastically reduced using this method, refer to figure 6. Therefore, these findings make Spike Select a promising solution for embedded AI-application when dealing with spiking CNNs. In addition, combining this coding scheme with the hybrid architecture, as described in the next section, would offer a better latency-

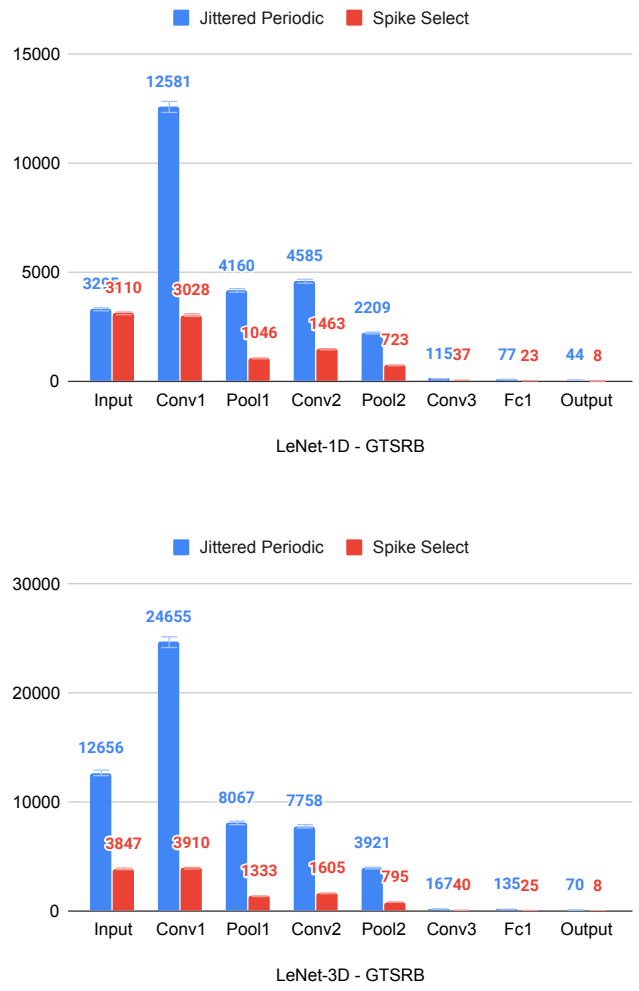

Fig. 7: Spike Select versus Jittered Periodic: Spiking data distribution over LeNet layers for GTSRB data-set

resources trade-off, including spiking CNNs to the proposed framework.

B. Hardware perspectives

Before concluding about these methods, let us analyze the spikes distribution on LeNet-5 network using both methods, this is illustrated on figure 7 and table V. Actually, these results confirm that Spike Select is more promising than Jittered Periodic, where fewer propagated spikes.

Moreover, the regulation of spiking distribution over SCNN's layers resulted from Spike Select, which is similar to the one in [11], makes spiking CNNs compatible with the hybrid architecture proposed in the same work (labelled as Generic IP in Figure 1), and hence get benefit of its structure.

The hybrid architecture is composed of a spike generator, a neural core, several NPUs and a terminate delta module, refer to figure 8. Its different modules communicate in an event-based fashion using three signals, *event* to conduct the

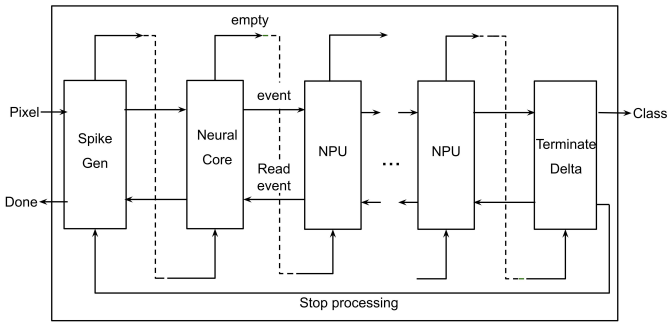

Fig. 8: Complete Hybrid Architecture schematic diagram

spiking event, *empty* to indicate the presence of spikes and *read event* to indicate that an event is being retrieved for processing. Within this architecture, an input pixel is processed by the *spike gen* cell that generates spiking events using one of the coding methods presented in this paper. Afterwards, the first convolution layer neurons process these events in a parallel fashion using the *neural core* module. Then, in the remaining layers, each one is implemented by the mean of a Neural Processing Unit, communicate their events in the same event-based fashion. Indeed, these NPUs are processing data sequentially, where an FC layer having N neurons is spending N cycles to process a single event. Finally, the last NPU is connected to a *terminate delta* module that verifies the enacting of the classification procedure. In fact, this is a hybrid architecture because it includes a fully-parallel part presented by the neural core module in the first stage and multiplexed ones in the following layers through the usage of NPUs.

As shown in figure 7, most spikes are processed by "Conv1" layer and therefore it is the most solicited one. Therefore, having a fully parallel implementation of this layer would accelerate the processing in the network. In addition, since few spikes are propagated in the other layers, then multiplexing computing would not affect significantly the processing time.

Due to the event-based nature of neuromorphic architectures, it is difficult to estimate the latency of spiking CNNs. Nevertheless, despite the fact that several spikes are executed in parallel in those architectures, knowing the total number of spikes processed by a spiking CNN give a partial latency estimation. In this context, the total number of spikes generated by the 3D and 2D LeNet networks when processing GTSRB testing dataset is indicated in table V. We notice that, compared to Jittered Periodic generated spikes, only 34.87% on LeNet-1D and 39.44% on LeNet-3D of these spikes are generated using Spike Select. Therefore, we expect a reduction of the computation time and thus energy consumption using this method with convolutional SNNs.

V. Conclusion

There are different rungs of neuromorphic chip design: neural model, high level modeling and RTL design. In this work, we studied neural coding in spiking CNNs to integrate them to our SNN's implementation framework.

In this context, we compared performances of three different coding techniques in terms of recognition rate and spiking activity. In terms of spiking activity, it turns up that Spike Select is more efficient, where, it drastically reduces the network's activity. Moreover, by the use of this method we notice a regulation of the spiking data distribution, where most spikes are generated by in the input layer and few of them propagate in the remaining ones. Indeed, this spiking distribution would benefit from our hybrid architecture with fully-parallel input layer and multiplexed deeper layers. Therefore, in the light of these findings one can say that, the "Spike Select & Hybrid Architecture" combination would be an effective solution for embedded AI applications.

The perspectives of the current work can be expressed in two steps. First, it is essential to implement spiking CNNs in hardware using a hybrid architectural structure and confront it with the state-of-the-art neuromorphic hardware architectures. Doing so, we would be able to conclude about spiking neural networks and their use for embedded AI-applications. Second, in order to take full advantage of the event-driven property of SNNs, the Spiking CNN has to be adapted, in both algorithmic model and hardware structure, to process efficiently asynchronous input sensors. Indeed, systems with Event-Based Sensors (EBS) input source are more efficient in terms of number of input spiking events, because EBS sensors are emitting only information about the changes between two consecutive scenes (frames) which directly impacts processing time and thus energy consumption.

Therefore, if the neuromorphic architecture supports both convolutional SNNs and event-based input sensors it could be a one of the best solutions for embedded AI-applications.

REFERENCES

[1] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of Physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[2] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[3] Y. LeCun *et al.*, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), pp. 396–404, Morgan-Kaufmann, 1990.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, May 2017.

[5] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, and G. Wang, "Recent advances in convolutional neural networks," *CoRR*, vol. abs/1512.07108, 2015.

[6] L. Khacef, N. Abderrahmane, and B. Miramond, "Confronting machine-learning with neuroscience for neuromorphic architectures design," in *International Joint Conference on Neural Networks (IJCNN)*, July 2018.

[7] J. A. Perez-Carrasco, Bo Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, Shouchun Chen, and B. Linares-Barranco, "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing–Application to Feedforward ConvNets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2706–2719, 2013.

[8] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *ieee*, pp. 1–8, IEEE, 2015.

[9] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3227–3235, 2018.

[10] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018.

[11] N. Abderrahmane, E. Lemaire, and B. Miramond, "Design space exploration of hardware spiking neurons for embedded artificial intelligence," *Neural Networks*, vol. 121, pp. 366 – 386, 2020.

[12] L. Maguire, T. McGinnity, B. Glackin, A. Ghani, A. Belatreche, and J. Harkin, "Challenges for large-scale implementations of spiking neural networks on fpgas," *Neurocomputing*, vol. 71, no. 1, pp. 13 – 29, 2007. Dedicated Hardware Architectures for Intelligent Systems Advances on Neural Networks for Speech and Audio Processing.

[13] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, pp. 652–665, May 2014.

[14] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[16] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, vol. 32, pp. 323 – 332, 2012.

[17] J. C. Thiele, O. Bichler, and A. Dupret, "Event-based, timescale invariant unsupervised online deep learning with stdp," *Frontiers in Computational Neuroscience*, vol. 12, p. 46, 2018.

[18] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, "Combining STDP and reward-modulated STDP in deep convolutional spiking neural networks for digit recognition," 2018.

[19] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[20] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47 – 63, 2019.

[21] N. Abderrahmane and B. Miramond, "Information coding and hardware architecture of spiking neural networks," in *Euromicro Conference on Digital System Design (DSD)*, 2019.

[22] O. Bichler, D. Briand, V. Gacoin, and B. Bertelone, *Neural Network Design and Deployment*, 2017. https://github.com/CEA-LIST/N2D2.

[23] Z. Du, D. D. B. Rubin, Y. Chen, L. Hel, T. Chen, L. Zhang, C. Wu, and O. Temam, "Neuromorphic accelerators: A comparison between neuroscience and machine-learning approaches," in *48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015.

[24] W.-J. Yang, C.-C. Luo, P.-C. Chung, and J.-F. Yang, "Simplified neural networks with smart detection for road traffic sign recognition," in *Advances in Information and Communication* (K. Arai and R. Bhatia, eds.), (Cham), pp. 237–249, Springer International Publishing, 2020.

[25] F. Zaklouta, B. Stanciulescu, and O. Hamdoun, "Traffic sign classification using k-d trees and random forests," in *The 2011 International Joint Conference on Neural Networks*, pp. 2151–2155, July 2011.

[26] Y. Cao, Y. Chen, and D. Khosla, "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.