Confronting machine-learning with neuroscience for neuromorphic architectures design

Lyes Khacef, Nassim Abderrahmane, Benoît Miramond
University Côte d'Azur / LEAT / CNRS UMR 7248
Sophia Antipolis, France
firstname.lastname@unice.fr

***Abstract*—Artificial neural networks are experiencing today an unprecedented interest thanks to two main changes: the explosion of open data that is necessary for their training, and the increasing computing power of today's computers that makes the training part possible in a reasonable time. The recent results of deep neural networks on image classification has given neural networks the leading role in machine learning algorithms and artificial intelligence research. However, most applications such as smart devices or autonomous vehicles require an embedded implementation of neural networks. Their implementation in CPU/GPU remains too expensive, mostly in energy consumption, due to the non-adaptation of the hardware to the computation model, which becomes a limit to their use. It is therefore necessary to design neuromorphic architectures, i.e. hardware accelerators that fit to the parallel and distributed computation paradigm of neural networks for reducing their hardware cost implementation. We mainly focus on the optimization of energy consumption to enable integration in embedded systems. For this purpose, we implement two models of artificial neural networks coming from two different scientific domains: the multi-layer perceptron derived from machine learning and the spiking neural network inspired from neuroscience. We compare the performances of both approaches in terms of accuracy and hardware cost to find out the most attractive architecture for the design of embedded artificial intelligence.**

***Index Terms*—artificial neural networks, neuromorphic architectures, hardware accelerator, power consumption**

I. Introduction

A large number of embedded applications take advantage of the performance of Artificial Neural Networks (ANNs): smart cameras, mobile robotics, real-time autonomous driving, etc. However, the software implementation of neural networks on classical Von-Neuman architectures (CPU/GPU) involves high power consumption. It is therefore necessary to develop dedicated hardware accelerators to significantly change the computation paradigm and thus optimize the hardware implementation cost [1]. Neuromorphic accelerators are most often based on non-Von Neumann architectures, where memory and computation are parallelized and distributed like neurons, which allows the neuromorphic architectures to be much more efficient. The aim of this paper is to study and compare the performances of models of ANNs following two different approaches: formal models represented by the Multi-Layer Perceptron (MLP) derived from machine learning research, and impulsion-based models represented by the Spiking Neural Network (SNN) inspired from neuroscience, and that is closer to the behavior of biological neurons in the brain. Indeed, the biological brain is able to perform high performance cognitive tasks with a much higher efficiency than the most powerful computers with very low energy consumption. The two studied models rely on different computation and information coding principles, and there is no evidence in the literature to say which approach is more efficient in terms of accuracy, area, power, speed and other requirements of embedded systems, especially that very few studies have compared them.

This work is divided into three parts. First, we study the different implementations of ANNs in the state of the art and describe the characteristics of these two models. Then, we build and train both models using a neural network simulator supporting both approaches. Finally, we implement both MLP and SNN in VHDL for FPGA (Altera Cyclone V) and ASIC (CMOS 65 nm) synthesis, and we finally compare and discuss their performances in terms of accuracy and hardware implementation cost.

II. Artificial Neural Networks implementation

A. Neural Networks models

With the development of artificial intelligence and brain-inspired computing, ANNs algorithms come today from two largely separate domains: machine-learning and neuroscience. In this section, we describe both neural models and highlight the learning algorithms that are mostly used with each-one.

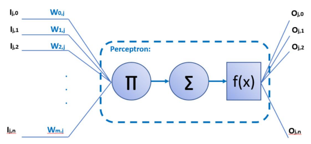

Fig. 1. Perceptron neuron

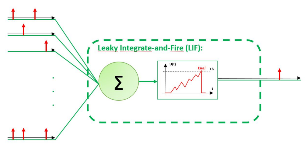

Fig. 2. IF neuron

1) Machine-Learning: The best-known ANN model for machine learning is the MLP, which consists of an input layer (generally, the input layer only consists of buffers with no function, i.e. the input neurons only transmit the information to the first hidden layer), one or more hidden layers (called Deep Neural Network, or DNN, for a large number of hidden layers), and an output layer [2]. Each layer has a number of neurons called perceptrons shown in Fig. 1 and the neurons between the layers are totally connected to each other. The neuron j in layer l performs the computation shown in Eq. 1.

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

Where y_j^l is the output of the neuron j in the layer l, N_l is the number of neurons in the layer l, w_{ji}^l is the synaptic weight between neuron i in layer $l-1$ and neuron j in layer l, and f is the non-linear activation function, e.g.: $f(x) = tanh(x)$ [3].

We notice that the sum goes from 0 to N_{l-1}, which means a total of $N_{l-1}+1$ neurons. The last neuron is the bias neuron that is connected to the next layer with its synaptic weight, and an output that is always equal to 1. The bias neuron is often helpful, as the bias value allows us to shift the activation function to the left or right, which may be critical for successful learning in some cases. When the neurons of the different layers are connected from inputs to outputs, the network is said to run in feed-forward [4].

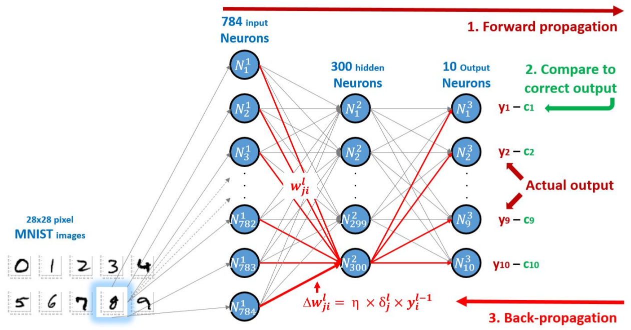

Fig. 3. Gradient back-propagation learning algorithm

The MLP learning algorithm is the gradient back-propagation (BP), a supervised off-line learning algorithm [2]. As shown in Fig. 3, the principle is to calculate the gradient of the error between the desired output and actual output and to back-propagate it to the neurons of the previous layers in order to adjust their synaptic weights. The process is repeated through all the learning data-base until either the number of learning iterations or the validation rate is reached. The synaptic weights are updated as shown in Eq. 2.

$$w_{ji}^l(t+1) = w_{ji}^l(t) + \eta \times \delta_j^l(t) \times y_i^{l-1}(t) \qquad (2)$$

Where t is the learning iteration, η is the learning rate and δ_j^l is the error gradient of the neuron j in the layer l, such that:

- At the output layer, $\delta_j^l(t) = f^{'}(s_j^l(t)) \times e_j^l(t)$ where e_j^l is the error, i.e.: the difference between the correct and the actual network output of the neuron j in the layer l.
- At the hidden layer, $\delta_j^l(t) = f^{'}(s_j^l(t)) \times \sum_{k=0}^{N_{l+1}} \delta_k^{l+1}(t) \times w_{kj}(t)$ where $f^{'}$ is the derivative of f.

2) Neuroscience: Spiking Neural Networks (SNNs) are the brain-inspired family of ANNs, mostly used for large-scale simulations in neuroscience. There exist many models amongst SNNs [5], that can be separated in two sections: bio-mimetic models (Hodgkin-Huxley, Izhikevich, etc.) and bio-inspired models (Leaky Integrate-and-Fire (LIF), Integrate-and-Fire (IF)). We used the bio-inspired IF model, as it is the simplest and most used model for implementing learning in spiking neurons. The SNN has two main differences compared to the MLP. First, neuron inputs and outputs are not directly encoding an activity in a digital value, but in spikes, or pulses, that use only 1 bit in digital representation. Several types of spike coding can be implemented [6]: rate coding, time coding, rank coding, population coding, etc. In rate coding,

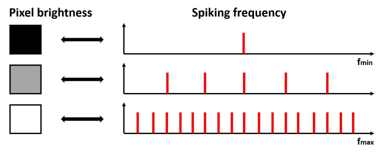

Fig. 4. Spike-based rate coding example

the frequency of spikes over the spike train is proportional to the brightness of the input pixel as shown in Fig. 4. We used the IF neuron shown in Fig. 2 with rate coding in order to easily transpose machine learning datasets in SNNs simulators [7]. Many topologies may be used in the SNN, but we chose the same feed-forward topology than the MLP, since we wanted to conduct a comparative study about the impact of information coding on hardware implementation cost, and using two different topologies could bias and unfairly influence the comparison. Second, the IF neuron computation is simpler than the perceptron's, as the neuron j in layer l performs the computation shown in Eq. 3. We may either generate a train of spikes proportional to the synaptic weight whenever there is an input spike in that synapse, or simply sum the synaptic weight whenever there is an input spike in that synapse. We used the second method.

$$\gamma_j^l(t) = \begin{cases} 0 & \text{if } s_j^l(t) \le \theta \\ 1 & \text{otherwise} \end{cases}, \quad p_j^l(t) = \begin{cases} s_j^l(t) & \text{if } s_j^l(t) \le \theta \\ 0 & \text{otherwise} \end{cases} \qquad (3)$$

Where $s_j^l(t) = p_j^l(t-1) + \sum_{i=0}^{N_{l-1}-1} w_{ij}^l \times \gamma_i^{l-1}(t)$, N_{l-1} is the number of neurons in the layer $l-1$, w_{ij} is the synaptic weight between the neuron i the in layer $l-1$ and the neuron j in the layer l, p_j^l is the potential of the neuron j in the layer l, θ is the threshold and γ_i^{l-1} is the spike state (0 or 1) of the neuron i in the layer $l-1$.

The bio-inspired learning algorithm adapted to SNNs is the Spike Time Dependent Plasticity (STDP), an on-line unsupervised learning algorithm, hence not requiring labeled datasets. STDP is a brain-inspired learning algorithm, where the idea is to detect the causality between the neurons for each input: if a neuron spikes soon (before the expiry of a time Δt) after receiving a spike from a given synapse, it suggests that synapse played an important role in the spike of the neuron, and therefore it reinforces that synapse by increasing its corresponding weight: it is called Long-Term Potentiation (LTP). In the other case, if a neuron spikes just before or a long time after receiving a spike from a given synapse, it suggests that synapse has no impact in the spike of the neuron, and therefore it decreases its corresponding weight: it is called Long-Time Depression (LTD) [8]. Here, the computation paradigm is no longer the same compared to BP, as we completely decentralize learning that becomes truly parallel and distributed among neurons, so that each neuron becomes responsible for its own learning [8].

B. Digital implementations

Neuromorphic architectures based on digital circuits benefit from the maturity of the associated manufacturing technologies and their reprogramming / reconfiguration facilities. Moreover, it is shown in [9] that below $22nm$, digital implementation becomes more attractive than analog implementation in terms of area and scalability for SNNs (LIF / IF model). Therefore, this study focuses on ANNs digital implementations. As described before, under embedded systems constraints, general programmable solutions on CPU/GPU are no longer suitable. However, some neuromorphic architectures are developed for large-scale neural networks simulations, mainly for neuroscience purposes. The most successful CPU-based neuromorphic architecture is the SpiNNaker chip designed for large-scale simulations of heterogeneous models of SNNs in biological real-time computing [10].

In another hand, digital solutions on Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) have been studied for several decades and offer a good compromise on all the criteria of low-power consumption, programmability / reconfigurability, performance, scalability, manufacturing maturity and biomimicry [1]. The new challenges for such digital circuits do no longer depend on the technology itself, but on the hardware architecture of the neural network [1], [11]. One example of digital architectures that gave a new breath to embedded machine-learning is the IBM TrueNorth chip [12]. Compared with an optimized neural network simulator running the same neural network on a general-purpose microprocessor, TrueNorth consumes 176,000 times less energy / synaptic event. Compared to SpiNNaker, TrueNorth consumes 769 times less energy / event. The results on TrueNorth clearly show that the digital implementation of SNNs is highly efficient in terms of hardware cost.

C. Neuromorphic architectures: MLP+BP vs. SNN+STDP

A comparative study between neuroscience and machine-learning approaches for neuromorphic accelerators was conducted in [13] where the objective was similar to ours, as they compared the MLP+BP and the SNN+STDP, with two main differences for the SNN: the topology is not a classical feed-forward but a map or one-layer topology and the neurons are not only connected through excitatory connections to the input neurons, but are also connected through lateral inhibitory connections to produce the so-called winner-takes-all (WTA) dynamics and create a form of recurrent network.

TABLE I
MLP+BP VS. SNN+STDP: ACCURACY COMPARISON

Neural network	MLP+BP	SNN+BP	SNN+STDP
Accuracy (%)	97.65	95.40	91.82

1) Accuracy comparison in [13]: Both MLP and SNN were trained on the MNIST database of handwritten digits (see section III-B). As shown in Table I, MLP+BP achieves a better recognition rate of 97.65%, compared with only 91.82% for the SNN+STDP (+5.83%). In order to reduce this loss, the authors implemented a hybrid version SNN+BP. The idea was to keep the SNN topology and information coding but to change the learning algorithm, going from STDP to BP. They found that the recognition rate of the SNN+BP reaches 95.40%, bridging the gap and reducing the difference of accuracy with MLP+BP to only 02.25%. The MLP+BP has therefore a better accuracy than SNN+STDP, but the difference between the two models comes mainly from the learning algorithm STDP, which gives less accuracy but allows on-line unsupervised learning.

2) Hardware cost comparison: We find in the literature two types of hardware digital implementation for ANNs [14] [1]:

- Spatially expanded: a completely parallel implementation where each neuron is physically represented, i.e. the design is similar to the conceptual representation of the neural network.
- Spatially folded: a time-multiplexed architecture where there is only ni physical neurons, each physical neuron representing N/ni logical neurons, where N is the total number of logical neurons.

TABLE II
MLP+BP VS. SNN+STDP: HARDWARE COST IN [13]

Neural network	**# inputs / operator**	**area** (mm^2)	**Energy** (uJ)	**Delay** (ns)	**# cycles / image**	**Energy / cycle** (mJ)
MLP+BP (784−100−10)	ni=01	01.05	000.38	2.24	882	000.43
	ni=16	06.36	000.29	2.25	057	005.08
	expanded	79.63	000.06	3.79	004	015.00
SNN+STDP (784−300)	ni=01	02.56	471.58	1.15	791×500	001.19
	ni=16	14.25	325.69	1.84	056×500	011.63
	expanded	38.89	214.70	2.61	500	429.40

As shown in Table II, and according to [13], the SNN is only better than the MLP in terms of delay. The SNN is less efficient in terms of total area (except for the spatially expanded implementation), and is not even comparable in terms of energy, as it is $1000\times$ more consuming on average for the spatially folded architectures and $3500\times$ for the spatially expanded ones.

We conclude from this comparison study that the MLP+BP has a better accuracy and a more efficient hardware implementation than the SNN+STDP in terms of total area (except for the expanded architecture) and energy. However, the neuroscience-inspired SNN+STDP remains intellectually attractive due to its closer relationship to the biological brain, as it allows the neural network to learn while it performs its task and it could become more attractive than machine-learning models for very large-scale implementations. Still, these results contradict our first intuition, since we showed that the SNN neuron model (LIF) is simpler than the MLP neuron model (perceptron), while it is here $2\times$ more consuming per cycle on average for the spatially folded and $28\times$ for the spatially expanded. We suppose that the SNN implementation in [13] was penalized by:

- The energy consumption of the embedded spike conversion of the MNIST input data.
- The STDP on-line unsupervised learning algorithm.
- The Leaky characteristic of the LIF neuron.

In order to understand the impact of information coding on the hardware implementation cost of ANNs, we implemented in this work both MLP and SNN with feed-forward topologies and BP, using a custom spike-based MNIST database and a simple Integrate-and-Fire (IF) neuron for the SNN, to have the fairest possible comparison.

III. Neural Networks building, training and test

The first step was the software simulation of both MLP and SNN. To avoid recoding the whole simulator for both formal and impulsion models, a comparative study of the available ANNs simulators was conducted.

A. *Neural Networks simulators comparison*

The comparison shows two families of ANNs simulators:

- Machine-learning simulators [1]: TensorFlow, Torch or Caffe that simulate DNN with BP learning for machine-learning oriented applications like image detection and recognition, mostly used with GPU accelerators.
- Neuroscience simulators [7]: Neuron, Nest or Brian that simulate large-scale bio-mimetic and bio-inspired neural networks with different neuron models (LIF, Izhikevich, Hodgkin-Huxley, etc.) and information codings (rate coding, time coding, etc.). Such simulators are designed to run on CPU/GPU, but also on specific neural networks simulating platforms like SpiNNaker. Some other simulators highlight the learning algorithm, like DeepSpike that implements a spike-based BP [15].

Among the different neural network simulators, N2D2 [16] gives the possibility to simulate both machine-learning and neuroscience models, typically the MLP+BP and SNN+BP for our work, and to compare their performances in terms of recognition rate and computational cost.

B. *Learning on MNIST database with N2D2*

Both MLP and SNN were trained on the MNIST database of handwritten digits, a training set of 60,000 examples, and then tested on the test set of 10,000 examples. To build our networks, two main parameters have to be determined: the number of hidden layers, and the number of neurons in each layer. This is a common problem in any neural network design, as there is no proved mathematical rule on how to fix these parameters. However, experience in machine-learning has brought some rules of thumb in this domain.

First, we know that our data are not linearly separable. Thus, we need to use at least one hidden layer, and it is sufficient for the large majority of problems, as the situations in which performance improves with a second or third hidden layer are very few [3]. Therefore, we use only one hidden layer.

Second, MNIST handwritten digits are 28 pixels x 28 pixels images for a total of 784 pixels / image. Hence, we need 784 input neurons and 10 output neurons, since the classification result is an integer between 0 and 9. Furthermore, from the study in [3], error on the MNIST test set using a one-hidden layer MLP is 4.7% for a network with 300 hidden neurons, and 4.5% for a network with 1000 hidden neurons. It shows that it is a much better compromise to use 300 hidden neurons, which is sufficient for more complex image classification tasks.

In conclusion, the topology of our neural networks is 784-300-10. We use the same topology for both MLP and SNN so that we could have a fair comparison on accuracy and hardware implementation cost, the only difference being the neuron computation and information coding model.

1) MLP learning: We built a 784-300-10 MLP in N2D2 with fully connected layers and a bias neuron for the input and hidden layers. The MNIST input data were first normalized, as the convergence is usually faster if the average of each input variable over the training set is close to zero [17]. The network was trained with the hyperparameters shown in Table III, that we fixed following the work of LeCun [3] and the experimental results we obtained using N2D2. After the learning process,

TABLE III
MLP TRAINING HYPERPARAMETERS

Hyperparameter	Value
Validation	0.2
BatchSize	1
Momentum	0
Learning rate	0.001
Learning rate decay	0.01

we extracted the learning synaptic weights from N2D2 output files to prepare the hardware implementation. After performing the test over the 10000 MNIST test images, we obtained a recognition rate of 95.73% which is slightly better than the result from [3] on the same topology.

2) SNN learning: We built a 784-300-10 SNN in N2D2 with the same hyperparameters as for the MLP, except for the spike-based computing, where we use specific hyperparameters[1]. We used the Poissonian distribution in order to have the same as the custom spike-based MNIST database that we used as input for the FPGA simulation to validate the behavior of the SNN. N2D2 uses BP for the training of both MLP and SNN, but since we cannot use classical BP on a SNN, mainly

[1]StimulusType=Poissonian; Threshold=1; BipolarThreshold=0

because some modules within the SNN are not differentiable [15], N2D2 is performing the training on the formal (MLP) version of the SNN, and then converting the learning weights to adapt to the new computation model of the SNN [18].

IV. Neuromorphic Architectures

When we obtained the learning synaptic weights for both MLP and SNN, we moved to the hardware design (VHDL) of their neuromorphic architectures. We worked under Intel Quartus 17.0 for FPGA prototyping, and ModelSim for simulation in order to validate the behavior of our designs.

A. MLP design

We first describe the architecture of the perceptron neuron, that has three computational parts as shown in Fig. 5: product, sum and activation. We implemented a spatially expanded version of the MLP where each neuron is physically implemented, except for input neurons that are temporally multiplexed in one physical input neuron. If the MLP reads all the 784 image pixels at once, the case where we need 784 physical input neurons, each hidden neuron will receive all its 784 inputs simultaneously, and will need 784 multipliers to perform its computation. That would be too expensive in terms of hardware cost, especially that each multiplier implies a DSP block in the FPGA device. Therefore, we used temporally multiplexed neuron inputs, where each neuron performs its computation synapse per synapse. Since the input neurons are only buffers and receive pixels one by one, we only used one hardware neuron.

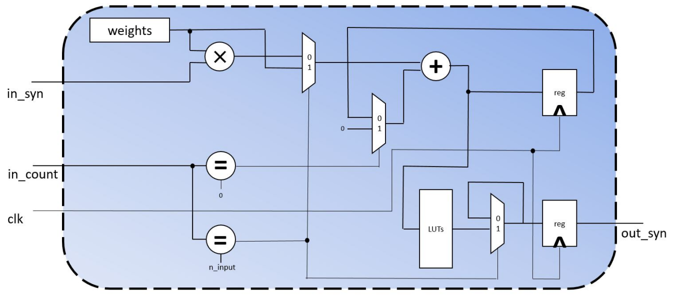

Fig. 5. Perceptron neuron hardware architecture

The simplified hardware architecture of the MLP is shown in Fig. 6. The MLP uses one input neuron, n hidden neurons and m output neurons that are generated during compilation, n and m being generic parameters that we can change depending on the size of our MLP. The neurons interconnections are coded in signed fixed-point $Q3.7$. The learning synaptic weights are stored in each neuron during synthesis. We used a local memory and assigned each weight to the corresponding synapse of the corresponding neuron. The input neuron is a buffer that outputs the image pixel per pixel. The hidden and output neurons do the same computation, they only differ in the number of inputs. The product is coded in $Q3.7$, while the sum is coded in $Q11.7$ to prevent overflow. As shown in Fig. 5, the activation function is implemented with LUTs

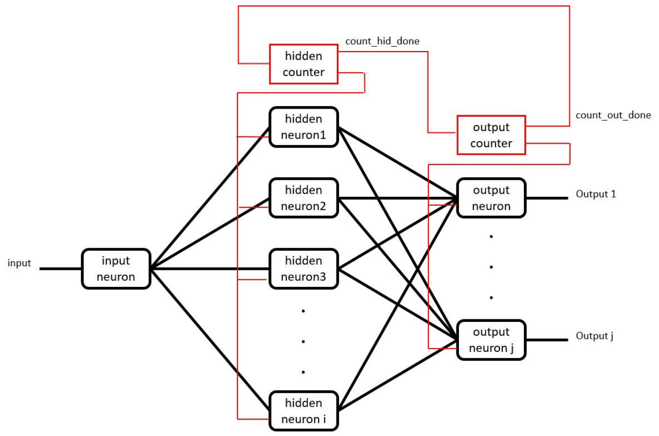

Fig. 6. Input-time-multiplexed neuromorphic architectures

for more efficiency. Finally, we used a hidden counter and an output counter to synchronize the neurons computations with the inputs reading, as the inputs are time-multiplexed.

B. SNN design

The architecture of the IF neuron has only one computational part for the sum, a comparator and multiplexers. We multiplexed the output of the sum such that it is equal to the sum of the previous potential and the synaptic weight when the synaptic input is equal to 1 (i.e. the previous layer's neuron connected to that synapse emitted a spike) and only equal to the previous potential if the synaptic input is equal to 0, as shown in Fig. 7. The neuron sums all the synaptic weights for which there is an input spike along its synaptic inputs, and when the sum is complete, if it is equal or exceeds the threshold then the neuron emits a spike and the sum is reseted.

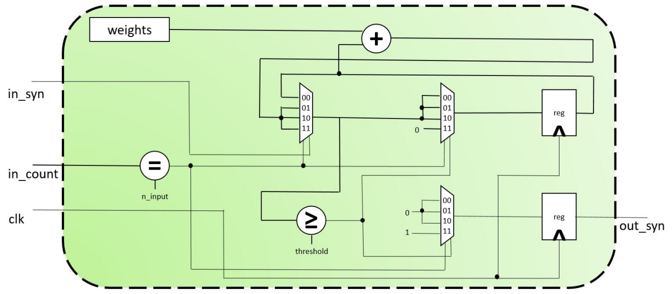

Fig. 7. Integrate-and-Fire neuron hardware architecture

An important remark is that the IF neuron only does the comparison to the threshold once the sum is complete over all the synaptic inputs, which means that even if the sum (potential) exceeds the threshold before its end, the neuron waits until the end to spike if the sum is still above the threshold (which is not guaranteed, since we have negative weights). The reason is that the spikes generated from the previous layer neurons are supposed to arrive at the same time, but since we used the same spatially expanded architecture than the MLP where the inputs are temporally multiplexed, the spikes are received one by one.

The time window and the frequency of spikes depend on the conversion we make to go from the formal to an event-based MNIST dataset. For the validation of our design, we generated a custom spike-based version of the original MNIST database. Therefore, for each image, we have a matrix of 784 $\times$ 100 logical variables (true = spike event; false = no event). As described in [18], the brighter the pixel, the more spikes are generated. The SNN has the same feed-forward topology and input-time-multiplexed architecture than the MLP shown in Fig. 6, the only difference being the neurons interconnections that are coded on 1 bit, as the neurons only receive and transmit spikes.

V. MLP+BP vs. SNN+BP: Comparison results

We compare in this section the MLP and SNN over two main criteria: accuracy and hardware cost for FPGA and ASIC implementations. We implemented four topologies with 10, 50, 100 and 300 hidden neurons. The results on the reference topology (784-300-10) are summarized in Table IV.

TABLE IV
MLP+BP vs. SNN+BP: comparison study summary

Neural Network			MLP 784-300-10	SNN 784-300-10	SNN gain (%)
Accuracy on MNIST database		Accuracy (%)	95.73	95.37	-00.38
		Read events / virtual synapse / pattern	1	0.268	73.20
Hardware cost	**FPGA: Altera Cyclone V**	Pins	113	14	87.62
		Logic (ALMs)	61809	34950	43.46
		Registers	6274	34950	43.56
		Max number of neurons	0342	1007	194.45
		DSP blocks	310	0	100.00
		Dynamic power (mW)	012.03	004.95	58.86
		Total power (mW)	538.78	530.21	01.60
	ASIC: CMOS 65nm	Ports	113	014	87.62
		Total cell area (mm^2)	01.888	00.869	53.98
		Cell internal power (mW)	00.740	00.400	45.91
		Net switching power (mW)	00.141	00.026	81.62
		Dynamic power (mW)	00.881	00.426	51.63
		Cell leakage power (mW)	51.224	28.040	45.26
		Total power (mW)	52.106	28.467	45.37

A. Image recognition accuracy

In terms of accuracy, the SNN has almost the same performances as the MLP (-00.38%), as shown in Fig. 8. We conclude that the spike-based coding does not penalize the network in terms of accuracy. Hence in [13], it is mainly the learning algorithm that reduces the recognition rate, while it offers the advantage of on-line unsupervised learning.

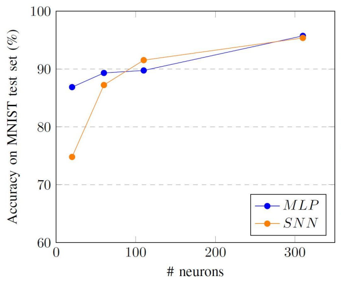

Fig. 8. Accuracy

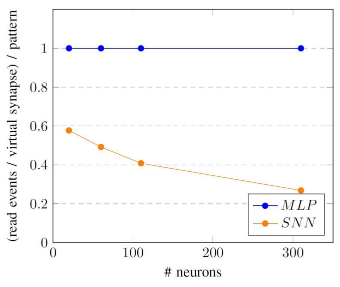

Fig. 9. Operations efficiency

Another important result obtained with N2D2 is the measurement of "read events per virtual synapse per pattern (average)". This statistic reports the average number of accumulations per synapse per input stimulus (image) in the network. For all the four tested topologies, this number is always below 1.0, which means that the spiking versions of the neural networks are more efficient than their formal counterpart in terms of total number of operations [16]. Furthermore, this number is decreasing when the number of neurons increases as shown in Fig. 9, which means that the more neurons we use, the higher the operations efficiency of the SNN compared to the MLP. In our case, this statistic only impacts the spike emission frequency between neurons, since the IF neuron always performs its computation independently from the input spike, decreasing its effect on the total energy consumption of the SNN. It means that we can achieve better performances for the SNN in our future works if the IF neuron is only activated when there is an input spike.

B. Hardware cost

1) FPGA implementation: For the FPGA implementation, we used Altera FPGA Cyclone V: 5CGXFC9E7F35C8 device that gives a good flexibility in terms of resources (ALMs, memory bits and DSP blocks). The FPGA implementation shows important gains for the SNN in terms of logic utilization (43.46%) due to the extra ALMs used for the activation function in the MLP, total pins (87.62%) due to the information coding and DSP blocks (100.00%) due to the multipliers that are used for the MLP only, with a small loss in the total registers (-08.40%), due to the stored spike signal in the SNN. Another important point is that we have a gain of 194,45% in the maximum number of neurons we can implement on this particular FPGA device, as we can use a maximum of 342 neurons for the MLP (limited by DSP blocks), while we can use a maximum of 1007 neurons for the SNN (limited by ALMs). In terms of clock frequency, the SNN goes up to 80MHz and has a gain of about 33%, but we chose to work with the same clock frequency for both neural networks. In terms of power consumption, the SNN is more efficient in dynamic power (58.86%), even if this gain is irrelevant in the total consumption of the FPGA device (01.60%).

2) ASIC implementation: We targeted a CMOS 65nm technology to implement our neural networks on ASIC and we compiled our designs using Synopsys Design Compiler. The

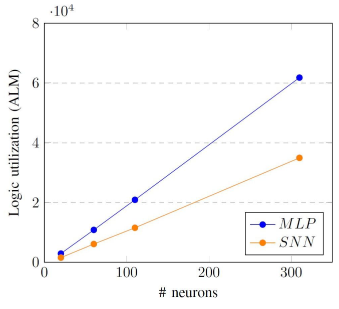

Fig. 10. FPGA logic

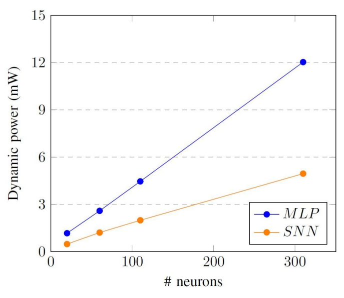

Fig. 11. FPGA dynamic power

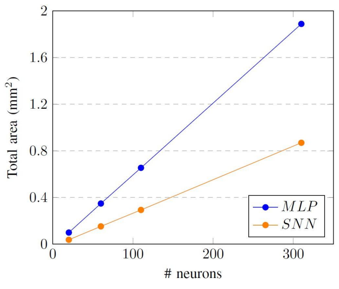

Fig. 12. ASIC total area

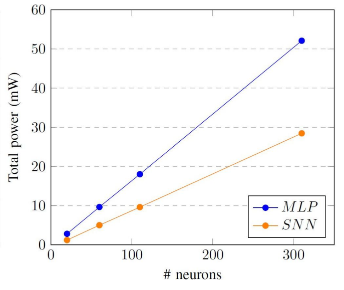

Fig. 13. ASIC total power

ASIC implementation shows a clear advantage for the SNN, as it is more efficient in terms of number of ports (87.62%) and total cell area (53.98%). These results are coherent since they reflect the SNN gains in FPGA resources. In terms of power consumption, the SNN is more efficient in total dynamic power (51.63%), that is approximately the same result found in FPGA. However, the SNN is also more efficient in cell leakage power (45.26%) as it is proportional to the used area, and therefore more efficient in terms of total power (45,37%).

C. Discussion

In this paper, it is shown that the SNN achieves practically the same performances than the MLP in terms of accuracy, while it is about 50% more efficient in hardware resources and power consumption. Here, we assume that both MLP and SNN are embedded in continuously operating cameras. Therefore, the power consumption gain of the SNN is equivalent to its energy consumption gain in the same amount of time. Consequently, the main advantage of the MLP is the speed (processed images per second), as the SNN takes $N\times$ more clock periods to process an image, where N is the maximum number of spikes that can be generated for one pixel. This number N is an important parameter influencing both the accuracy and the image processing time of the SNN, such that increasing N results in higher accuracy and longer processing time, i.e. more input spikes as shown in Fig. 14. We illustrate in Fig. 14 the evolution of the number of input spikes versus the image resolution for two cases: the frame-driven based systems with different N and the Dynamic Vision Sensor (DVS) based system. The approximate plot based on the work in [18] (poker cards) shows that systems with DVS cameras are more efficient in terms of number of input spikes to process, as

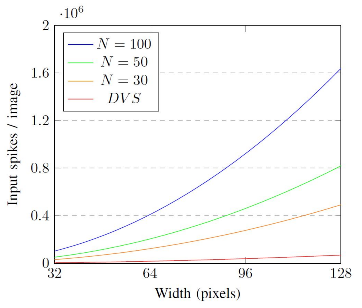

Fig. 14. Frame-driven and frame-free (DVS) input spikes efficiency

DVS sensors only send the difference between two consecutive frames, and that directly impacts the SNN processing time and energy consumption.

Now, the obtained results are compared to those of the specially expanded versions of both networks in the comparative study conducted in [13]. The area gain for the same precision is approximately the same (50%). However, there is a very large difference in terms of SNN energy gain. In [13], the MLP has a gain of 96.38%. In our study, the SNN has a total energy gain of 45.37%, which fits to the theoretical study on the computational complexity of the two models, where the IF neuron performs a simpler computation based on spike-events. Therefore, it is not the spike-based coding that penalizes the SNN in the study conducted by Z. Du *et al.* [13], but some of the three factors: embedded spike-conversion, STDP learning algorithm and the Leaky part of the LIF neuron.

When implementing the MLP and the SNN on the same target technology (CMOS 65nm) with Synopsys Design Compiler, we have an area gain of about 97% compared to the one in [13], as shown in Table V. In one hand, the comparison between SNN+BP (our study) and SNN+STDP [13] may be unfair as the SNN+STDP implements on-line unsupervised learning, but in another hand, the comparison between both MLP+BP clearly shows that we have a more efficient architecture, as we have 97.62% less area with 200 extra neurons. The reason is that even if both studies used spatially expanded architectures, we used time-multiplexed neuron inputs, whereas they used parallel inputs, which means that each neuron has as much multipliers as inputs. Thus, the main advantage of time-multiplexing neuron inputs is the hardware area efficiency. Nevertheless, our time-multiplexed inputs architecture is limited for the SNN in terms of integration to

TABLE V
OUR AREA GAINS COMPARED TO THE STUDY IN [13]

Neural Network	**MultiLayer Perceptron**		**Spiking Neural Network**	
	Our study	[13]	Our study	[13]
	MLP+BP 784-300-10	MLP+BP 784-100-10	SNN+BP 784-300-10	SNN+STDP 784-300
Area (mm^2)	01.89	79.63	00.87	38.89
Our area gain	97.62%		97.76%	

DVS cameras, as we have to gather their output data into frames before processing them.

To overcome this limitation, the study in [18] presents an interesting method for mapping a formal neural network to a SNN. This method is illustrated in a DNN processing data from a DVS camera. The system is trained in its formal version then mapped to its spiking version. The work in [19] shows another method for training SNNs with DVS versions of MNIST data-base. We can notice two aspects of the SNN architecture developed in [18]:

- The input data come directly from a bio-inspired asynchronous (DVS) camera.
- The recognition occurs while the camera is providing events-data (spikes).

These aspects are not present in our implementation: in the SNN, the input data must be gathered into frames before processing them. Thus, the input and computation are not simultaneous. Integrating this approach into our system is therefore very promising. However, we cannot apply this method to a system trained from a standard data-base. In [18], the DVS data-base is built following these steps:

1) Record event streams of 80ms from the DVS camera.
2) Generate images from these recordings by collecting events during 30ms.

To get advantage from this method without recording the whole data-set with a DVS sensor, the standard training data-base has to be adapted. This leads us to ask the question of the transformation of standard data-bases (frame-driven) in a way to use them with frame-free spike coding. In addition, recognizing images while the camera is transmitting events is ensured in [18] thanks to a parallel (non-multiplexed) architecture that is hardly compatible with FPGA/ASIC-based implementations for embedded systems.

VI. Conclusion and further works

In the context of embedded image classification, we showed that the neuroscience approach (SNN) reaches the same performances than machine-learning (MLP) in terms of accuracy, while it is about twice as efficient in terms of hardware implementation cost (area, power). The MLP, though, is faster in terms of number of clock cycles for processing one image. The presented neuromorphic architectures are spatially expanded and input time-multiplexed, giving the required hardware efficiency of embedded systems. Nevertheless, the integration of the SNN with an asynchronous camera is still in progress. The obtained results give two main perspectives: Firstly, the neuroscience approach gives the possibility to implement Deep SNNs for more complex applications, with low-energy consumption for embedded systems. Secondly, SNNs give the possibility to implement brain-inspired learning algorithms with on-line unsupervised learning, and take a step forward going from *Deep Machine Learning* that solves a specific task on labeled data to *Machine Intelligence* that continuously learn from unlabeled data to enhance the performances of autonomous and flexible systems, and thus take a step forward in the design of embedded artificial intelligence.

Acknowledgment

This material is based upon work supported by the French research agency through project ANR-17-CE24-0036, by PACA region and by CIM PACA association.

References

[1] C. D. Schuman, T. E. Potok, R. M. Patton, J. Douglas Birdwell, M. E. Dean, G. S. Rose and J. S. Plank, ”A Survey of Neuromorphic Computing and Neural Networks in Hardware”, Proceedings of CoRR journal, vol. abs/1705.06963, 2017.
[2] V. Sze, Y.H. Chen, T.J. Yang and J. Emer, ”Efficient Processing of Deep Neural Networks: A Tutorial and Survey”, Proceedings of CoRR journal, vol. abs/1703.09039, 2017.
[3] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, ”Gradient-Based Learning Applied to Document Recognition”, Proceedings of the IEEE journal, vol. 86, no. 11, 1998.
[4] G. Bebis and M. Georgiopoulos, ”Feed-forward neural networks”, IEEE Potentials, vol. 13, no. 4, pp. 27-31, 1994.
[5] E. M. Izhikevich, "Which model to use for cortical spiking neurons?", IEEE Trans. on Neural Networks, vol. 15, no. 5, pp. 1063-1070, 2004.
[6] R. Brette, ”Philosophy Of the Spike: Rate-Based vs. Spike-Based Theories of the Brain”, Frontiers in Neuroscience Journal Series, 2015.
[7] R. Brette et al., ”Simulation of networks of spiking neurons: A review of tools and strategies”, Journal of Computational Neuroscience, vol. 23, no. 3, pp. 349-398, 2007.
[8] S. Narayanan, A. Shafiee and R. Balasubramonian, ”INXS: Bridging the throughput and energy gap for spiking neural networks”, International Joint Conference on Neural Networks (IJCNN), pp. 2451-2459, 2017.
[9] A. Joubert, B. Belhadj, O. Temam and R. Héliot, ”Hardware spiking neurons design: Analog or digital?”, Proceedings of The International Joint Conference on Neural Networks (IJCNN), 2012.
[10] S. B. Furber, F. Galluppi, S. Temple and L. A. Plana, ”The SpiNNaker Project”, Proceedings of the IEEE, vol. 102, no. 5, 2014.
[11] L. Rodriguez, B. Miramond, I. Kalbousi and B. Granado, ”Embodied Computing: Self-adaptation in Bio-inspired Reconfigurable Architectures”, International Parallel and Distributed Processing Symposium Workshops, 2012.
[12] P.A. Merolla, J.V. Arthur, R. Alvarez-Icaza, A.S. Cassidy, J. Sawada2, F. Akopyan, B.L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S.K. Esser, R. Appuswamy, B. Taba, A. Amir, M.D. Flickner, W.P. Risk, R. Manohar and D.S. Modha, ”A million spiking-neuron integrated circuit with a scalable communication network and interface”, Science, 2014.
[13] Z. Du, D.D.B.D. Rubin, Y. Chen, L. Hel, T. Chen, L. Zhang, C. Wu and O. Temam, ”Neuromorphic Accelerators: A Comparison Between Neuroscience and Machine-Learning Approaches”, Proceedings of 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2015.
[14] L. Fiack, L. Rodriguez and B. Miramond, ”Hardware design of a neural processing unit for bio-inspired computing”, Proceedings of IEEE 13th International Conference on New Circuits And Systems, 2015.
[15] P. O’Connor and M. Welling, ”Deep Spiking Networks”, Proceedings of CoRR journal, vol. abs/1602.08323, 2016.
[16] V. Lorrain et al., N2D2: open source CAD framework for Deep Neural Network simulation, https://github.com/CEA-LIST/N2D2, 2017.
[17] Y. LeCun, L. Bottou, G.B.Orr and K.R. Mller, ”Efficient BackProp”, Neural Networks: Tricks of the Trade, vol. 1524, 1998.
[18] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen and B. Linares-Barranc, ”Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing–Application to Feedforward ConvNets”, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 35, no. 11, 2013.
[19] E. Stromatias, M. Soto, T. Serrano-Gotarredona and B. Linares-Barranc, ”An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data”, Frontiers in Neuroscience, vol. 11, 2017.