On the Spiking neural networks

Comparison of performances and power consumption between ANNs and SNNs

Nicola Cassetta
nicola.cassetta@studenti.unipd.it

Andrea Di Trani
andrea.ditrani@studenti.unipd.it

Chiara Cavalagli
chiara.cavalagli@studenti.unipd.it

June 24, 2023

Abstract

From 2012 to 2021 the amount of computational power required to run state-of-the-art deep learning models has increased at a rate of 10x per year. The use of an ever increasing models size not only poses significant challenges in terms of computational resources and energy consumption, but it also raises concerns about its environmental sustainability, with some models requiring weeks or even months of training on multiple GPUs and TPUs, which results also in a alarming inference power consumption. A promising solution to these concernings are the Spiking Neural Networks (SNNs): biologically plausible nets that are notoriously energy-efficient due to the binary spike signals that convert traditional high-power multiply-accumulation (MAC) into a low-power accumulation (AC). In this project we will analyze the differences in terms of performances and energy consumption among classical ANNs and the modern SNNs on several benchmark data sets.

1 Introduction

Deep Neural Networks (DNNs) are a class of machine learning models that are inspired by the structure and function of the human brain. They are composed of interconnected nodes or neurons organized in layers capable of learning complex patterns and representations from large amounts of data. In the last decade they have gained significant attention and success in various fields of artificial intelligence with breakthroughs in different areas, from image and speech recognition [1] [2] to natural language processing [3] and many others.
These models have demonstrated state-of-the-art performance and have become the foundation of many cutting-edge applications in academia, industry, and various real-world use cases.

However, the increasing size and complexity of these models, driven by the availability of large datasets and advances in computing resources, have raised concerns about their computational requirements, energy consumption, and environmental sustainability: nowadays training large neural networks can require prohibitive computational and economic resources, including multiple GPUs or TPUs, and it can result in substantial emissions and energy usage [4].

Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Transformer (big)	192
w/ neural architecture search	626,155

Figure 1: From [4]: estimated CO2 emissions from training common NLP models, compared to familiar consumption.

Moreover, this trend has the potential to lead to a monopolized field, where only a few large companies with extensive resources can afford to develop and deploy AI models, further exacerbating inequalities in the field (it is sufficient to consider the third version of OpenAI's ChatGPT model, a LLM with over 175 billion parameters, which has been trained for 34 days on 1024 GPUs, for an estimated cost of $4.6M [5]). Additionally, as our world becomes increasingly reliant on embedded systems and battery-powered devices, there is a growing need for inference-efficient neural networks: by designing a less energy demanding net we can achieve longer battery life, reduce the need for frequent recharging, and contribute to a more sustainable and eco-friendly technology landscape.

In light of these challenges, there is an increasing urgency to explore sustainable approaches for the development of large-scale deep learning models, both in terms of training energy consumption and efficient inference. Many attempts have been made to reduce the power consumption of traditional deep learning models, for example: quantization [6], pruning [7] and knowledge-distillation [8]. Despite the remarkable results of these techniques, the problems highlighted in this section remain unsolved.
A promising alternative that has gained attention in recent years are the Spiking Neural Networks (SNNs), often referred to as the "third generation of neural networks."

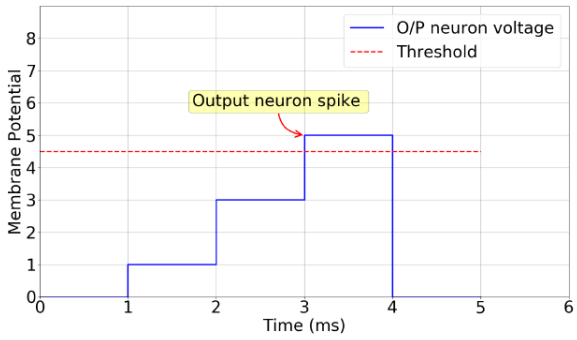

Figure 2: Example of a spiking neuron reaching its membrane potential threshold. In this specific case, after the spike is generated, the neuron voltage is reset to 0.

2 Spiking Neural Networks

Spiking Neural Networks are not only biologically inspired networks but also biologically plausible structures that replicate the fundamental communication and processing principles of neurons in the human brain. Unlike standard ANNs (Artificial Neural Networks), which use continuous activations and operate on real-valued inputs and weights, SNNs use discrete spikes or action potentials to represent and process information in a time-dependent manner, resulting in energy-efficient processing compared to traditional nets. Firstly proposed as a mathematical model to understand the basic functioning of biological neurons [9], the model remained theoretical until 1990, when research paved the way for the design and implementation of hardware platforms (known as "Neuromorphic" [10]) that could process spiking neural activity in real-time. The fields of neuromorphic hardware and SNNs have grown together, with advancements of one area brought often progress in the other. In a SNN neurons communicate with each other through discrete spikes, which are short electrical impulses occurring when the membrane potential of a neuron reaches a certain threshold. Once the neuron has exceeded its limit (Fig. 2), it fires, resetting the membrane voltage. This process can be divided into three main stages, that we call *Neuronal Charge*, *Neuronal Fire* and *Neuronal Reset*.

Those signals lead to a sparse activation behaviour, that, combined with the binary nature of the spike, result in large energy saving. Despite these remarkable results, these models are neither easy to use nor to implement, and the reasons are many:

- The training phase can be complex, and it requires specialized training algorithms.
- Implementing SNNs on conventional hardware platforms can be challenging and often results in negligible or even no energy saving.
- The research in this area is still at his early stages, which results in a lack of a unified framework for simulating these models.

Luckily, recalling the last point, nowadays there are some new frameworks that allow a good simulation of SNNs, together with an high-level interface. Among them we must cite "SpikingJelly" [11], an open-source deep learning framework for SNNs based on PyTorch, released in 2020. Due to the increasing popularity among researchers [12], [13], [14] and thanks to its simplicity, we chose this one for our tests and simulations.

2.1 SNN architecture

There is a close connection between standard RNNs (Recurrent Neural Networks) and SNNs: as mentioned above, in order to fire a spike, a neuron

needs to be excited until the membrane potential threshold is reached (see Fig. 2). Similarly to RNNs, the spiking neuron is said to be stateful (or "has memory"), and the state corresponds to the membrane potential. The latter, defined as $\mathbf{V}[\mathbf{t}]$, is not determined only by the input $\mathbf{X}[\mathbf{t}]$, but also by its previous state $\mathbf{V}[\mathbf{t-1}]$, just like in RNNs.

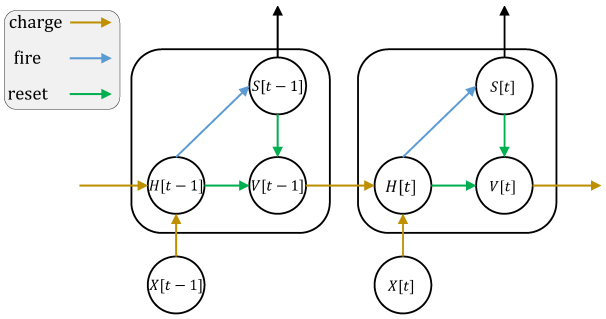

Figure 3: Neuronal dynamics: X is the input, H and V represent the membrane potential after neuronal dynamics and after the trigger of a spike at time-step t, respectively. S is the output spike

In the Fig. 3 the dynamic of a SNN is presented: given $\mathbf{X}[\mathbf{t}]$ the external input, $\mathbf{H}[\mathbf{t}]$ the hidden output, $\mathbf{V}[\mathbf{t}]$ the membrane potential with $\mathbf{V_{thr}}$ and $\mathbf{V_{reset}}$ respectively the threshold and reset potentials (constants) and $\mathbf{S}[\mathbf{t}]$ indicate the presence of a spike at that time-step. We can thus analyze the general equation of this dynamic:

$$H[t] = f(V[t-1], X[t]), \tag{1}$$

$$S[t] = \Theta(H_{t-1} - V_{thr}), \tag{2}$$

$$V[t] = H[t] \cdot (1 - S[t]) + V_{reset} \cdot S[t], \tag{3}$$

where $\Theta(x)$ is the Heaviside step function, and the equations are respectively named as: *Neuronal Charge* (1), *Neuronal Fire* (2), and *Neuronal Reset* (3).

2.2 I/O encoding-decoding

SNNs communicates and learn with spikes, for this reason we need to encode our input in a way that is meaningful to this type of net, and we need also to decode the result of the computation. We treat separately the input-output encoding methods, in particular:

- Input encoding: converts input data into spikes. Among the most famous techniques we have: the Latency encoding (or temporal), the Rate encoding, and the Delta modulation;

- Output decoding: learns how to treat the output of a network in a way that is informative. In this case we have again the Rate and the Latency encoding, together with the Population encoding.

Even if each method bring benefits and drawbacks, in literature the Rate encoding is undeniably one of most important due to its simplicity and its effectiveness, thus we used this technique in both input and output phase. The output decoding phase simply relies on the choice of the neuron with the highest firing rate (or spike count) as the predicted class (assuming a classification scenario), while in the input phase we need to encode the data into spikes (example in Fig. 4).

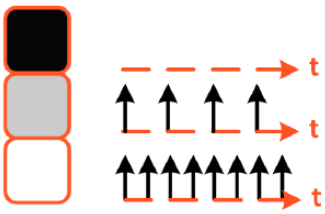

Figure 4: Example of Rate coded input pixel. An input pixel of greater intensity corresponds to a higher firing rate.

Following the work of [13], we used a Poisson encoder: this method generates spikes by converting the input data into a series of Poisson-distributed spike patterns (Fig. 5). This means that the spikes are generated stochastically and at random intervals, following a Poisson distribution.

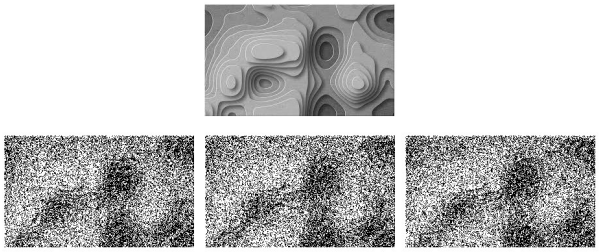

Figure 5: Example of grayscale image encoded three different times with a Poisson encoder. Even if they look similar it is still possible too see many differences.

2.3 Training of SNN

As for the classical ANNs, training a SNN involves adjusting the weights to learn a particular task. However, SNNs cannot be trained directly using the standard Backpropagation algorithm due to the non-differentiable nature of the spike function. To overcome this limitation, several methods are reported in literature, and among them, the techniques that empirical shows SoTA results are the "Surrogate gradient method" [15] and the "ANN2SNN" [16] (also called "shadow training"). Regardless from the method uses, when dealing with SNNs one of the most important hyperparameter to consider is the "simulation time" (in this work is referred as **T**): the duration specified by "T" allows the net to accumulate and propagate information over time, taking into account the temporal aspects of the input and the dynamics of the network's spiking activity. By specifying a large value for "T", the simulation can capture longer temporal patterns and provide more accurate representations of the network's behavior over time; however this comes at the cost of an higher computational time and cost. On the other hand, a small "T" value can lead to faster simulations but may sacrifice accuracy, with almost no spikes, thus finding a trade-off of this value is crucial in order to obtain an accurate and efficient model. When dealing with static input (ie. images) we can see this as feeding in input the same image T times, but each time the image is encoded in a slightly different way, and this allow the net to capture all the nuances of the input (interestingly this behaviour is really close to the human's saccades [17], quick and simultaneous movement of both eyes used for locating interesting parts in a scene).

2.3.1 Surrogate gradient

A "spike" can be seen as a discontinuous function that rapidly increases from a resting state to a fire state, where the neuron emits a signal. In its binary approximation this can be seen as the classical Heaviside step function:

$$\Theta(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{4}$$

which derivative results in the following:

$$\delta(x) = \begin{cases} +\infty, & x = 0 \\ 0, & x \neq 0 \end{cases} \tag{5}$$

To overcome the limitation of a non-differentiable function, a differentiable "surrogate" function that approximates the behavior of the spike is often used. This allows the gradients to be computed and used in backpropagation, enabling efficient training of SNNs.

2.3.2 ANN2SNN

This procedure aims to convert an already trained ANN to a SNN: this method gained a lot of attention in recent years since it has a way of preserving the already trained ANNs, and still having the opportunity to convert them in order to let them run in an embedded system with a little loss in terms of accuracy. Conventionally, SNNs can be obtained by converting a pre-trained ANNs by replacing the non-linear activation with spiking neurons. The theoretical basis upon which this process is based are described in detail in the work of [18], which highlight a strong correlation between the firing frequency of a neuron and the ReLU's activation function: in fact we have that the IF neuron (Integrateand-Fire) in SNN is an

unbiased estimator of ReLU activation function over time. This statement also highlight the main issue of this technique, that is the fact that in order to achieve a lossless convertion, it often requires a longer simulation time, which means an higher computational consumption if compared to the shallow version of SNNs (since this issue raises from deep nets i.e VGGish or Inception Nets). The work of [18] (partially) address this issue with a plethora of improvement, among which we cite:

- Parameter normalization: Weight normalization as a way to avoid approximation errors due to too low or too high firing rate;
- Batch Normalization Layer convertion;
- Spiking Max-Pooling Layers;

which allows now for the convertion of nearly arbitrary CNN architectures. During this work we will cite the 'Convertion Mode': with this term we want to describe the criterion used for the normalization of the activations. As stated in [16], to avoid the overflow of the spike rate it is needed to divide the activations of the original ANN by a certain amount, the way this value is chosen determines the 'Convertion Mode'. If we take the %99.99-th percentile, then we will refer to the '%99,99' mode, and in the same fashion we can have the modes: 'Max', '1/3' and so on.

2.4 Energy Consumption Estimation

While the number of Floating Point Operations (FLOPs) can be considered one of the main parameters to estimate energy consumption in DNNs, in SNNs we need to quantify instead the number of SyOPs, that are the number of spike-related operations involved in the computation of an SNN. Despite the name difference, the underlying structure of FLOPs and SyOPs is quite similar since both of them can be based on MAC and AC operations:

- FLOPs: in classical DNNs MAC operations refer to the multiplications of two operands followed by their accumulation (their sum), which is a common operation in matrix transactions, convolutions, and linear transformations. The AC operations include additions, subtractions, and activations like ReLU or sigmoid functions;
- SyOPs: a notation commonly adopted from the work of [19] and refers to operations like spike generation and spike propagation. Instead of propagating continuous values and performing numerous MAC operations, SNNs use event-driven processing, where spikes are passed along the network's neurons, triggering local computations only when necessary. This behavior enables SNNs to perform computations using mostly AC operations which are generally more computationally efficient.

In order to produce a fair and precise comparison between SNNs and ANNs in terms of power consumption, we should have access to the right hardware for both the architectures: standard GPUs and Neuromorphic chips. Unfortunately, the availability of the latters is not so obvious, thus we need to estimate the power consumption of SNNs. To overcome this shortcoming we used the SyOps library [20]: thanks to this tool we were able to count the number of synaptic operations in spiking neural networks.
Moreover, thanks to the work of [20] we also have a bound of the estimated consumption of SNNs: let F be the network, O_{mac} and O_{ac} be the number of MAC and AC operations, T the simulation time, E_{mac} and E_{ac} the energy for a single operation, we have the following:

$$E[F] \in [T \cdot E_{mac} \cdot O_{mac}, T \cdot (E_{ac} \cdot O_{ac} + E_{mac} \cdot O_{mac})]$$

where the values of E_{mac} and E_{ac} are strictly related to the hardware architecture. From the work of [20] and [4] we know that for 45nm architectures (like NVIDIA GeForce GTX 400 Series or AMD Radeon HD 5000 Series) the costs per operation is roughly 0.9pJ for ACs and 4.6pJ for MACs. Since our work is based on the NVIDIA Tesla T4 (provided by Colab), we are dealing with a 12nm architecture. From the white paper of the NVIDIA Turing architecture [21], and thanks to the work of [22] [23] we have that the cost per operation for E_{mac} and E_{ac} is roughly 1.6pJ and 0.58pJ respectively.

3 Experimental setup

In order to explore all the nuances of the two different paradigm, we planned several tests among different datasets. Regardless of the model and dataset, the structure of the tests is be summarized in the Fig. 6:

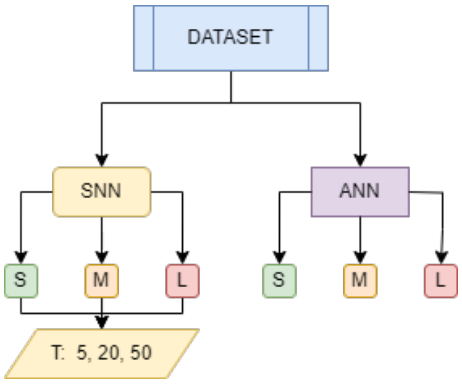

Figure 6: Setup of the project tests.

For each dataset we tested a Small (S), Medium (M) and Large(L) net, both SNN and ANN. With reference to the SNN version of the net, we have also three different simulation times for each model (T $\in \{5, 20, 50\}$).

The datasets used are the following:

- MNIST: a collection of $70,000$ handwritten digits, from 0 through 9, that has been widely used as a benchmark dataset for image classification tasks in deep learning. Each image has dimension 28×28 and it is in grayscale [24];
- CIFAR10: just like the MNIST, this is a widely used benchmark dataset, in the field of computer vision. This dataset consists of $60,000$ 32×32 colored images, classified into 10 different categories. The dataset is evenly distributed, with $6,000$ images per class, making it a balanced dataset for training and evaluation purposes [25];
- FLOWERS102: a dataset specifically focused on fine-grained image classification. It consists of 102 categories of flowers and offers a challenging task for classification algorithms due to its nature. The images have size 224×224 colored [26].

We now give a list of all the different models and their variants that we used per every dataset.

MNIST: For the MNIST dataset we have implemented a simple Multi Linear Perceptron (MLP) with the Dropout regularization technique (p=0.5) in the case of L and M nets.

MNIST		
Size	**Layers**	**Units**
S	1	256
M	3	512, 256, 64
L	4	2000, 500, 64, 16

CIFAR10: On Cifar we used classical CNNs, and for every net considered in this work we used a kernel of size 3 and the average pooling method after every convolutional layer.

CIFAR 10			
Size	**Conv**	**Fc**	**Units**
S	5	2	1024, 512
M	8		

In particular, the number of filters for each conv. layer are the following:

- **S**: 3, 16, 16, 32, 32
- **M**: 3, 32, 32, 64, 64, 128, 256, 256

FLOWER102: With reference to this dataset we must point out that the tests on it were aimed to highlight the behaviour of some famous ANN models being converted to SNN. The models used are the VGG19 and the MobileNetV2, both pretrained on ImageNet1k. The reason behind the choice of these models is twofold: on one side we wanted to test the consumption of a deep network with many parameters pre and post convertion (from here the choice of the VGG), on the other hand we also wanted to estimate the theoretical energy saving of a SNN compared to an ANN model designed to be efficient on

	MNIST										
	ANN					SNN					
Size	**Train Acc/Loss**		**Val Acc/Loss**		**Time (s)**	**Train Acc/Loss**		**Val Acc/Loss**		**T**	**Time (s)**
						0.977	1.477	0.961	1.491	5	291.97
S	**1**	**0**	**0.98**	**0.098**	**202.9**	**0.982**	**1.475**	**0.966**	**1.49**	**20**	**295.76**
						0.981	1.476	0.966	1.49	50	316.28
						0.979	1.477	0.973	1.484	5	325.05
M	**0.997**	**0.01**	**0.979**	**0.162**	**210.87**	0.981	1.476	0.971	1.486	20	327.04
						0.979	1.477	0.974	1.484	50	333.53
						0.866	1.512	0.881	1.512	5	338.23
L	0.273	1.713	0.285	1.68	269.2	0.877	1.51	0.885	1.512	20	359.65
						0.962	**1.487**	**0.97**	**1.488**	**50**	**387.74**

Table 1: MNIST results.

classical hardware and to work on embedded systems (from here the choice of the MobileNet).
In both cases we removed the standard classifiers of the models and used a custom one designed for the Flowers102 dataset (a simple MLP with dropout p= 0.5 and 512 - 256 units). For what concerns the feature extraction sections, in both cases we froze the backbone up to the last section and let train only the classifier and the last feature extraction part.

4 Results

Here we presents the results obtained through our tests. The tests have been performed with the following setup:

- MNIST and CIFAR: 30 epochs, Adam as optimizer, $1e^{-3}$ as learning rate and 128 as batch size;
- FLOWERS: since the models were pre-trained, in this case we lowered the learning rate to $1e^{-4}$, increased the number of epochs to 40 and we added a small weights decay ($1e^{-5}$).

4.1 Performances

In this section we will present the accuracy and loss of each model, then we will present the test results resulting from the best model for each dataset. The best model will be then used to estimate the power consumption and the emissions. It is important to remember that the train time reported in the following tables arise from the simulation of synaptic hardware performed on a classical GPU, hence for SNNs we have a much higher value mainly caused by an inefficient hardware platform.

4.1.1 MNIST

The results are displayed in the table 1. Considering only the S and M models we can see that there are no major differences in terms of accuracy between ANNs and SNNs (even tho in both cases the ANNs outperforms the SNNs by 1-2% in both the training and the validation set). An interesting point arises from the results of the L models: it seems that the ANN Large, mainly due to the number of units, is not able to achieve a good result (less than 29% on both the sets). The SNN Large, instead, outperform the classical model and seems less prone to overfit the training data (behavior that we would have expected). In this scenario the best model are: the ANN Small and the SNN Small with T=20. We can see that the ANN still achieve the best result but with a difference of 0.7-0.05% on the accuracy. The SNN is able to achieve a competitive result in this scenario, especially considering the fact that the ANN Small

	CIFAR										
	ANN					SNN					
Size	**Train Acc/Loss**		**Val Acc/Loss**		**Time (s)**	**Train Acc/Loss**		**Val Acc/Loss**		**T**	**Time (s)**
S	**0.939**	**0.181**	**0.732**	**1.242**	**492.02**	0.71	1.6669	0.6802	1.756	5	603.87
						0.529	**1.78**	**0.508**	**1.794**	**20**	**815.56**
						0.409	1.875	0.404	1.882	50	1882.53
M	**0.974**	**0.083**	**0.784**	**1.255**	**541.68**	0.1	2.303	0.098	2.303	5	692.43
						0.1	2.303	0.098	2.303	20	1025.97
						0.39	1.869	0.385	1.881	50	4807.68

Table 2: CIFAR results.

took 202s (3min) to train and the SNN Small took 291s (4.5min) being simulated on an unconventional hardware.

MNIST Test Accuracy		
ANN	**SNN**	**ANN2SNN**
98.06	97.35	98.01

Table 3: Best Results.

4.1.2 CIFAR10

Results are displayed in the table 2. Interestingly in this situation the latency played a major role in the behavior of the model, highlighting that shallow nets can exploit very low latency while performing noticeably worse increasing T. Deeper nets, on the other hand, seems to need higher latency to work since for $T = 5, 20$ the resulting models was nothing more than a random predictor.

CIFAR Test Accuracy				
ANN	**SNN**	**ANN2SNN**		
		T=5	**T=20**	**T=50**
74.05	65.08	20.8	70.6	73.4

On the test set we can see some extremely interesting results on the converted model: there is an important trade-off between accuracy and latency. In order to achieve high precision as original ANNs, a longer simulation time is needed. Despite the fact that we were expecting this type of behaviour (since in the literature is a quite common result), seeing an increment of +53% on the accuracy by passing from T=5 to T=50 was astonishing (especially considering the fact that a simulation time of T=50 is not considered to be extremely long).

4.1.3 FLOWERS102:

As mentioned in the previous sections, training a full spiking deep neural network is still a challenge when using classical hardware. That's why we have that the ANN2SNN convertion is preferred. In the table 4 we present the results of the classical VGG19 and the MobileNetV2 and the converted versions. Here we can see some interesting results: the converted versions of the VGG19 achieves an higher accuracy with T equal to 40 and 100, but with a much higher time required, if compared to the ANN counterpart. Considering the MobileNet instead, we can see how the results are almost identical with the standard version and with the converted with T equal to 20 and 40. Despite that, we can see again (as for the VGG) that the standard model is able to achieve the same result but in half of the time even respect to the smallest simulation time used.

4.2 Consumes

Finally we can present the results on the consumes of the models. We highlight the fact that these results are not related to the training phase, but rather on

FLOWERS 102					
	ANN		ANN2SNN		
Name	**Test Acc**	**Time (s)**	**Train Acc**	**T**	**Time (s)**
VGG 19	44.77	**93.1**	40.	20	1057.50
			54.32	**40**	2060.52
			52.35	100	5123.18
MobileNetV2	75.52	**68.3**	**75.69**	**20**	151.97
			75.64	40	254.36
			71.65	100	550.7

Table 4: FLOWER results.

the inference performed on the entire dataset: this choice comes from the fact that after a model is trained, it will spend most of its time doing inference. Often even a computational and energy demanding training phase is acceptable if results in a more inference-efficient model. The following results are organized as follow: for each dataset we will present a table that summarize the number of AC and MAC operations, along with the estimated kWh, and after that we will also show a visualization of the consumes for each model. Because the number of AC and MAC operations often had reached the unit measurement of the million, each value is scaled in mega units i.e. divided by 10^6. We also point out that, for a matter of readability, all the graphical results related to the consumption are contained in a specif section named "Power consumption graphs".

4.2.1 MNIST

Here we show the results on the MNIST dataset, the second and third columns stand respectively for N°AC operations and N°MAC operations. We can see how a fully Spiking net is able to achieve almost the same results of its standard ANN counterpart, but with a value of kWh of two orders of magnitude smaller (taking into account the smallest SNN model with T=5, that we recall is able to achieve a 97% accuracy on the test set, compared to the 98.06% of its classical counterpart). If we instead consider the "best SNN model" (that is the one mentioned in the table 3, i.e the SNN Small with T=20) we can see that we have the same order of magnitude but for the SNN we have 1.7e-13 instead of 9e-13, almost an 8x reduction. We can see how this behaviour is constant in almost all the cases presented: the power consumed from a ANN is two orders of magnitude greater than the SNN with the smallest simulation time. If we consider the medium simulation time, then we have the same order of magnitude but with a smaller value (the value of the SNN is 1/3 of the ANN). Taking into account the largest simulation time, we have almost the same value of the standard ANN. Is it clear that the simulation time is one of the most important parameter to tune when working with this type of networks.

Name	AC	MAC	kWh
S_SNN_5	**1.3e3**	**1**	**1.0736e-14**
S_SNN_20	5.3e3	1	1.7151e-13
S_SNN_50	1.3e4	1	1.0716e-12
S_ANN	1	2e5	9.0453e-13
M_SNN_5	**4.2e3**	**1**	**3.3936e-14**
M_SNN_20	1.6e4	1	5.4271e-13
M_SNN_50	4.2e4	1	3.3916e-12
M_ANN	1	5.5e5	2.4459e-12
D_SNN_5	**1.3e4**	**1**	**1.0434e-13**
D_SNN_20	5.1e4	1	1.6691e-12
D_SNN_50	1.3e5	1	1.0432e-11
D_ANN	1	6.9e6	3.0747e-11

Table 6: MNIST consumes.

Name	AC	MAC	kWh
VGG	1	1.9e10	8.69212e-10
S_VGG	14e6	2.2e9	9.9851e-09
MobileNet	1	3.2e8	1.4245e-09
S_MobileNet	1256	3.1e8	1.3947e-09

Table 5: Consumes of Flower dataset.

From the graphs 12 and 13 we can appreciate a clear information regarding the MAC and AC operations: even the largest simulation time for the Small model has a number of AC operations that is much lower than the MACs for the ANN (less than 15k ACs vs +200k MACs). This gap becomes larger and larger by scaling the model, reaching a peak of almost 7 millions MACs for the D_ANN, a value that is 53 times larger than the total number of ACs operations for the SNN counterpart.

4.2.2 CIFAR10

Again, here it's clear that the simulation time plays a major role when considering the energy consumption of a spiking model, in fact it determines whenever the spiking model is more efficient than it's ANN counterpart. However we can see that for bigger models (compared with the ones used for MNIST) the T value can be set higher retaining the efficiency advantage of SNN (see Fig. 14 and Fig. 15).

Name	AC	MAC	kWh
M_SNN_5	**2.6e5**	**1**	**2.1555e-12**
M_SNN_20	1e6	1	3.4487e-11
M_SNN_50	2.6e6	1	2.1554e-10
M_CNN	1	8.7e6	3.9011e-11
D_SNN_5	**7.1e5**	**1**	**5.7953e-12**
D_SNN_20	2.8e6	1	9.2724e-11
D_SNN_50	7.2e6	1	5.7955e-10
D_CNN	1	9e7	4.0333e-10

Table 7: CIFAR consumes.

4.2.3 FLOWERS102

As we would have expected, in the table 5 are shown much less prominent differences: we can clearly see that converting a trained model, rather than training a full spiking one, results in a much less power saving. Only with the VGG we can appreciate a reduction of 8x on the number of MACs with the Spiking version, resulting in a reduction on the kHw of one order of magnitude. Instead, for the MobileNet, almost nothing changes: neither in term of accuracy, or number of operation or on the power consumption. Their corresponding consumes plots are shown in the Fig. 16 and Fig. 17.

4.3 Robustness

Another appealing aspect of the SNNs is that they tends to exhibit inherent advantages in robustness compared to classical ANNs, particularly in the presence of noisy data and adversarial attacks. These advantages can be attributed to a key factor intrinsic to the nature of SNNs: the sparsity in the training. SNNs exhibit sparsity in spike patterns during training, meaning that only a subset of neurons fires spikes in response to a given input. This sparsity enables SNNs to focus on relevant features and suppress irrelevant or noisy information. As a result, SNNs can robustly extract and represent essential features from the input, enhancing their resistance to noisy data and maliciously perturbed data. Of course these advantages are present only with nets that have been trained in that way, thus converted model does not acquire any benefits.

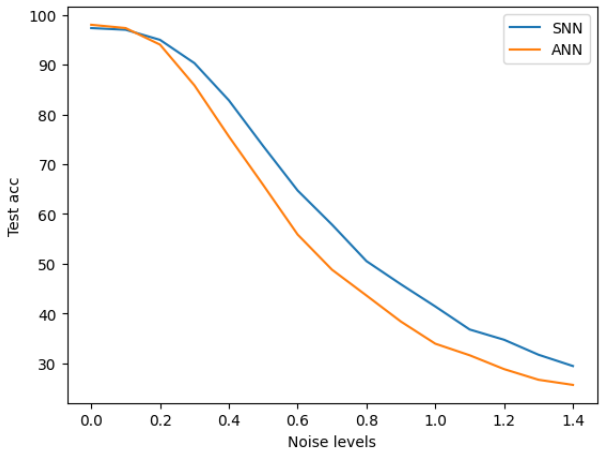

Figure 7: Robustness on noisy input: SNN vs ANN on the MNIST dataset

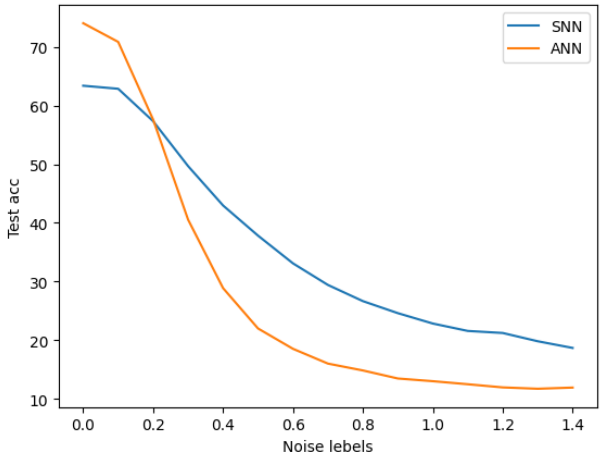

Figure 8: Robustness on noisy input: SNN vs ANN on the CIFAR10 dataset

4.3.1 Noisy data

The robustness to noise is an important aspect to consider when evaluating the performance of machine learning models. In this section, we investigate the ability of the SNNs to handle noisy data compared to ANNs on both the MNIST and the CIFAR10 datasets. We will analyze the performance of the two best models (one for the ANNs and one for the SNNs) in the presence of various levels of noise and evaluate their robustness to noisy inputs. The final scores are displayed in the Fig. 7 and Fig. 8.

4.3.2 Adversarial attacks

With the term "adversarial attack" we refer to the process of modifying in a malicious way an input for a machine learning model. These perturbations are carefully crafted to deceive the model into making incorrect predictions. Szegedy et al. [27] first pointed out the existence of adversarial examples in the image classification domain: given an input x which belong to a class t, it is possible to find a new, maliciously perturbed, input x', that is similar to x but classified as t'. Based on the target class we can consider "targeted" and "untargeted" (i.e. any class that is not t) attacks, while based on the knowledge of the target model, we can consider "white box" and "black box" attacks where in the first one the attackers has totally access to the model and its gradient, while in the second case, it has minimum information about it. In this project we focus on the simplest case: white-box untargeted scenario.

$$\boldsymbol{x}^{adv} = \boldsymbol{x}^{real} + \epsilon \cdot \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{x}^{real}, y))$$

Figure 9: FGSM adversarial update of the samples

Build a model that is able to resist to adversarial attack is a crucial aspect nowadays, especially when it comes to the image classification tasks. One of the most common adversarial attack is the Fast Gradient Sign Method (FGSM) which involves calculating the gradient of the loss function with respect to the input data and then adding a small perturbation to the input data in the direction of the sign of the gradient.

This results in a modified input that is similar to the original input but can cause the model to make incorrect predictions. In order to compare the behaviour of our models, we measured the accuracy on a ever-increasing level of ϵ (ie. the size of the step to take during the FGSM), starting from 0.0 up to 1 with a step of 0.05

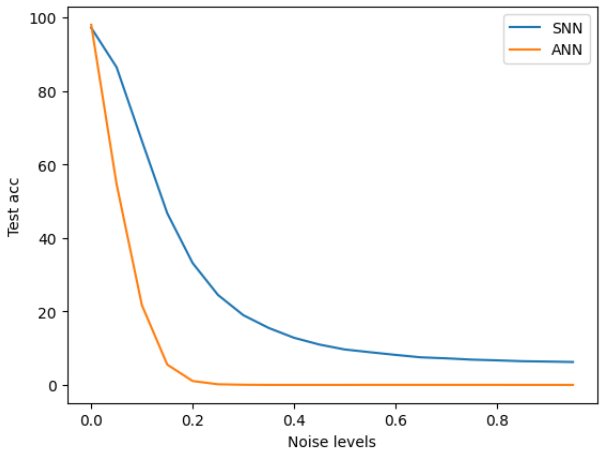

Figure 10: Robustness on FGSM attack: SNN vs ANN on the MNIST dataset

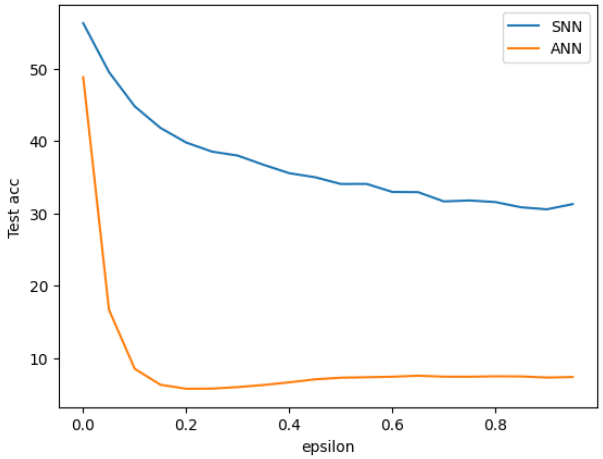

Figure 11: Robustness on FGSM attack: SNN vs ANN on the CIFAR10 dataset

5 Conclusions

AI is occupying an increasingly significant space in our lives and simultaneously becoming the core business for millions of companies. The need to reconcile these models, which are active 24/7, with sustainable development will be ever-present in the coming years. In the literature, there are hundreds of studies on spiking neural networks, and one of the key "green points" is always the efficiency in terms of power consumption of this "new" network paradigm. However, it is extremely challenging to find quantitative information on this aspect. The combination of specialized hardware, specific datasets, and emerging frameworks makes quantifying these consumptions highly complex. For this reason, our work has primarily focused on estimating power consumption in multiple scenarios.

We have shown that classical nets tends to consumes a value for kHw that is of two order of magnitude larger than the Spiking counterpart, with a gap in performance that is almost irrelevant in the case of simple tasks (MNIST) and a gap in more complex task (CIFAR) that can be reduced with a proper architecture search (different neurons, different surrogate functions or with a potential initialization). This result highlight how as the research and development in SNNs and neuromorphic hardware continue to advance, we can expect SNNs to play a pivotal role in enabling energy-efficient and high-performing embedded systems of the future, making them an ideal choice for battery-conscious devices, knowing that this type of structures allows for hundreds of times of inference cycles. Moreover we can confirm two points that are recurrent in the literature:

- Training a full deep spiking net is still a complex task that requires some tuning on both hyperparameters and architecture;
- The convertion of a ANN to a SNN is still one of the easiest (and more effective) way to achieve almost the same performances as the starting model, but with a energy consumption reduction of one order of magnitude.

Additionally we have demonstrate how a fully SNN is natively extremely more robust to adversarial attacks and to noisy data if compared to the classical counterpart.The research in this field is still in an early stage, with major improvement happened only in 2017 and 2020. Despite that, many aspects of these type of nets are appealing, and they can effectively represents a valid alternative to classical ANN in some specialized field.

6 Power consumption graphs

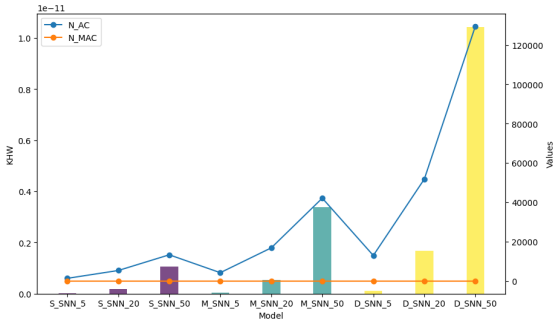

Figure 12: ACs, MACs and estimated consumption of SNN models on MNIST. In particular, [S_SNN, M_SNN, D_SNN] are displayed.

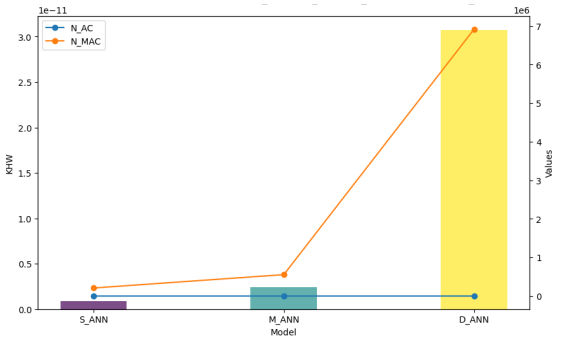

Figure 13: Comparison of ACs, MACs and estimated consumption of ANN models on MNIST. In particular, [S_ANN, M_ANN, D_ANN] are displayed.

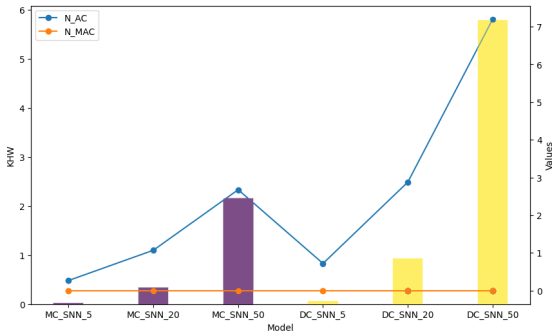

Figure 14: Comparison of ACs, MACs and estimated consumption of SNN models on CIFAR

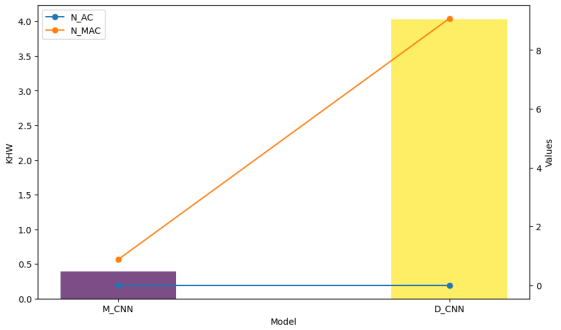

Figure 15: Comparison of ACs, MACs and estimated consumption of ANN models on CIFAR

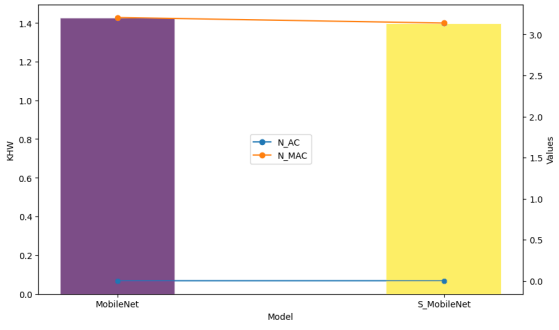

Figure 16: Comparison of ACs, MACs and estimated consumption before and after the convertion of the MobileNet. Flowers102

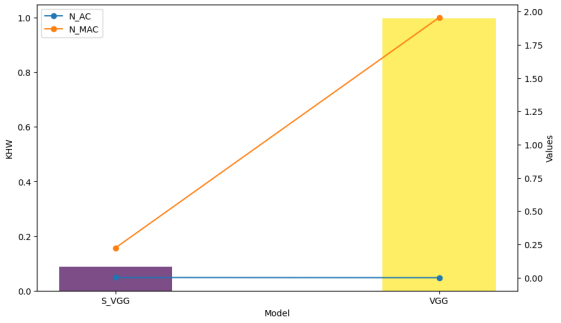

Figure 17: Comparison of ACs, MACs and estimated consumption before and after the convertion of the VGG19

References

[1] S. Kriman, S. Beliaev, B. Ginsburg, J. Huang, O. Kuchaiev, V. Lavrukhin, R. Leary, J. Li, and Y. Zhang, "Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions," 2019.

[2] A. Mehrish, N. Majumder, R. Bhardwaj, R. Mihalcea, and S. Poria, "A review of deep learning techniques for speech processing," 2023.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[4] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," vol. 57, pp. 10–14, 02 2014.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[6] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021.

[7] L. Liebenwein, C. Baykal, B. Carter, D. Gifford, and D. Rus, "Lost in pruning: The effects of pruning neural networks beyond test accuracy," 2021.

[8] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[9] . H. A. F. HODGKIN, A. L., "Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo," 1952.

[10] Z. Yu, A. M. Abdulghani, A. Zahid, H. Heidari, M. A. Imran, and Q. H. Abbasi, "An overview of neuromorphic computing for artificial intelligence enabled hardware-based hopfield neural network," *IEEE Access*, vol. 8, pp. 67085–67099, 2020.

[11] W. Fang, Y. Chen, J. Ding, D. Chen, Z. Yu, H. Zhou, T. Masquelier, Y. Tian, and other contributors, "Spikingjelly." https://github.com/fangwei123456/spikingjelly, 2020. Accessed: YYYY-MM-DD.

[12] Z. Zhou, Y. Zhu, C. He, Y. Wang, S. YAN, Y. Tian, and L. Yuan, "Spikformer: When spiking neural network meets transformer," in *The Eleventh International Conference on Learning Representations*, 2023.

[13] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," 2021.

[14] Z. Hao, J. Ding, T. Bu, T. Huang, and Z. Yu, "Bridging the gap between anns and snns by calibrating offset spikes," 2023.

[15] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," 2019.

[16] J. Tang, J. Lai, X. Xie, L. Yang, and W.-S. Zheng, "Snn2ann: A fast and memory-efficient training framework for spiking neural networks," 2022.

[17] C. Pierrot-Deseilligny, S. Rivaud, B. Gaymard, R. Müri, and A.-I. Vermersch, "Cortical control of saccades," *Annals of Neurology: Official Journal of the American Neurological Association and the Child Neurology Society*, vol. 37, no. 5, pp. 557–567, 1995.

[18] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in Neuroscience*, vol. 11, 2017.

[19] P. A. Merolla and J. V. Arthur, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[20] "Syops." https://github.com/iCGY96/syops-counter.

[21] "Nvidia turing gpu architecture."

[22] Y. Sun, N. B. Agostini, S. Dong, and D. Kaeli, "Summarizing cpu and gpu design trends with product data," 2020.

[23] S. Shankar and A. Reuther, "Trends in energy estimates for computing in AI/machine learning accelerators, supercomputers, and compute-intensive applications," sep 2022.

[24] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[25] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[26] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[27] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.