Asynchronous Spiking Neurons, the Natural Key to Exploit Temporal Sparsity

Amirreza Yousefzadeh, *Member, IEEE*, Mina A. Khoei, Sahar Hosseini, Priscila Holanda, Sam Leroux, Orlando Moreira, Jonathan Tapson, Bart Dhoedt, Pieter Simoens, Teresa Serrano-Gotarredona, *Senior Member, IEEE*, and Bernabe Linares-Barranco, *Fellow, IEEE*

***Abstract*— Inference of Deep Neural Networks for stream signal (Video/Audio) processing in edge devices is still challenging. Unlike the most state of the art inference engines which are efficient for static signals, our brain is optimized for real-time dynamic signal processing. We believe one important feature of the brain (asynchronous state-full processing) is the key to its excellence in this domain. In this work, we show how asynchronous processing with state-full neurons allows exploitation of the existing sparsity in natural signals. This paper explains three different types of sparsity and proposes an inference algorithm which exploits all types of sparsities in the execution of already trained networks. Our experiments in three different applications (Handwritten digit recognition, Autonomous Steering and Hand-Gesture recognition) show that this model of inference reduces the number of required operations for sparse input data by a factor of one to two orders of magnitudes. Additionally, due to fully asynchronous processing this type of inference can be run on fully distributed and scalable neuromorphic hardware platforms.**

***Index Terms*— Spiking Neural Network (SNN), Asynchronous Inference, Temporal sparsity, Deep Neural Network, Convolutional Neural Network (CNN), Bio-inspired processing, Neuromorphic Hardware.**

I. Introduction

DEEP Artificial Neural Networks (ANN) have flourished in the past decade as a promising technology. Despite significant breakthroughs in algorithmic procedures, the implementation of deep learning systems requires the availability of large amounts of data and computing power. Nowadays, these computations are mainly provided by GPU platforms with thousands of parallel processors.

However, a certain dedicated accelerator can dramatically optimize deep ANN workloads, for instance by data reuse strategies and more efficient implementations of the computational parallelism [1]. In some applications, training is not performed as frequently as inference and it makes sense to separately measure and optimize the required computation for each of them. Since low power and low latency inference is needed for many of today's practical edge devices, in this work we focus on a computationally optimized inference algorithm.

Even-though ANNs are inspired by biological brains, we are still far from understanding the function of the brain. Particularly we lack knowledge in three aspects: data collection (sensors), learning strategies and compute mechanisms. Performance of state of the art Neural Network processors is not close to the performance of the brain, concerning the trade-off between power consumption, accuracy, and speed. Scalable architecture, parallel processing, in-memory computation, communication with spikes, asynchronous execution, low precision computation, and fault tolerance are among some of the known features of our brain.

Most of the dedicated neural network processors are exploiting parallel processing to increase performance. However, the other features of the biological brain like asynchronous communication, sparse activity, and in-memory computation are only considered recently in cutting edge research. However, it is known that the efficient application of these features requires algorithm-hardware co-design. For example, an algorithm that can provide low-precision computation enables analog in-memory processing which will result in extreme power savings. In this work, we introduce an algorithm to enable asynchronous event-driven processing in hardware. Asynchronous communication and processing are one of the key features of biological brains. Asynchrony makes the brain architecture scalable and enables fully distributed neural computations. Additionally, as each neuron works individually without any synchronization overhead, the response latency is generally low.

We can categorize three different types of sparsities in neural network inference: 1) Structural sparsity, 2) Spatial sparsity, and 3) Temporal sparsity. As the dominant operation in DNN is multiplication-accumulation of synaptic weights and neuron's output (activation), an operation can be skipped

This work was supported in part by the EU H2020 NEURAM3 under Grant 687299 and HERMES under Grant 824164 and in part by the Spanish grant from the Ministry of Economy and Competitivity through the European Regional Development Fund, COGNET, under Grant TEC2015-63884-C2-1-P. This article was recommended by Guest Editor T. Delbruck. *(Amirreza Yousefzadeh and Mina A. Khoei contributed equally to this work.) (Corresponding author: Bernabe Linares-Barranco.)*

A. Yousefzadeh and O. Moreira are with GrAI Matter Labs (GML), 5656 AG Eindhoven, The Netherlands.

M. A. Khoei is with GrAI Matter Labs (GML), 75012 Paris, France.

S. Hosseini, T. Serrano-Gotarredona, and B. Linares-Barranco are with the Instituto de Microelectroónica de Seville (CSIC), Universidad de Seville, 41004 Seville, Spain (e-mail: bernabe@csic.es).

P. Holanda, S. Leroux, B. Dhoedt, and P. Simoens are with the IDLab, Ghent University-imec, 9052 Ghent, Belgium.

J. Tapson is with GrAI Matter Labs (GML), San Jose, CA 94043 USA.

Color versions of one or more of the figures in this article are available

when synaptic weight is zero (Structural sparsity) or neuron output is zero (Spatial sparsity). For example, weight pruning [2] is a technique to improve the efficiency of DNN inference by pruning the connections and neurons. Therefore, weight pruning may result in having structural sparsity. Another example is using $ReLU$ $(y = max(0, X))$ activation function which results in spatial sparsity as the output of neurons with $ReLU$ is zero when its input is negative. Zero-skipping in hardware imposes irregular execution which may reduce the performance. However, when sparsity is high enough, it pays the price [3]. Temporal sparsity is perhaps the most challenging to exploit. Temporary sparse data is a type of data that rarely change in time. For example, the output of a security camera in a quiet place mostly contains the background scene which doesn't change in time. Exploiting temporal sparsity means skipping re-calculation of a function when none of its input changed from the last update time. This means temporally sparse executions need remembering the results of past calculations and reusing them efficiently. Therefore it requires an extra amount of memory. It looks like that the preference of the biological brain is to trade power consumption with memory. Therefore even though the volume of the brain is high, it is very low power due to skipping redundant processing. While in the brain on average only between 1% to 10% of the neurons are active at each moment [4], [5], in conventional ANN inference, all the neurons in the neural network should be updated and communicate their output to the downstream neurons. Therefore, unlike brain, the amount of operations in the ANN is fixed, independent of temporal sparsity (amount of change over time) in the input stream.

Even though data compression techniques have been used for years to exploit the temporal sparsity (to save storage and transmission bandwidth), still the standard routine to process the same data includes decompression and applying heavy processing with a high amount of redundancy. For example, while a time-sampled voice needs around 64KB per second, a decent video call application [6] needs less than 12KB per second bandwidth to transfer this information. For video, the situation is more interesting, while the data rate of a raw High Definition (HD) video is more than 1.2Gb per second, the same application needs around 1.5Mb per second for an HD video call. Conventional video compression techniques seem to have similar features as the human eye. The human retina includes photo-receptors which fire spikes in response to change detection in light intensity. Therefore we can only see the objects if there is a relative movement between our retina and the object or if the object changes its brightness. To be able to see the non-moving objects [7], our eyes perform unintentional rapid ballistic jumps of amplitude up to one degree, called micro-saccades. This movement only happens once or twice per second.[1] In between micro-saccades, our vision system relies on motion events. On the other hand, the basis of video compression techniques [8] is to send the whole frames (I-Frame) once in a while and the data corresponding to the moving objects in between. Therefore, one can consider the eye as a compressor of visual stimulus and the brain as a processor of compressed data.

Fully synchronous processing is also a problem of conventional ANN inference in which each neuron must execute its internal process after all of its inputs are available. In practice, this means that a neuron can only fire on specific synchronous triggers together with other neurons and a centralized process should handle the synchronization. Scaling up a synchronous process increases the synchronization overhead and eventually saturates the parallel processing capability which is known as Amdahl's law [9]. However, an asynchronous neural network, in which each neuron can fire with a self-triggering mechanism (threshold based), doesn't need any synchronization. The brain is a proof for this statement. It can scale from the 10k neurons in a worm brain up to 86B neurons in a human brain while using the same basic building blocks and architecture. This level of scalability is achieved via asynchronous distributed processing and event-based communication [10]. Inspired from biology, event-based simulation techniques were used for simulations of spiking neuron networks for a long time [11] and hardware implementation of asynchronous communication and processing is implemented in many of the available neuromorphic processors [12]–[16]. The asynchronous activity allows separate processing units to work independently without sharing a clock signal or any synchronization overhead. Therefore, an asynchronous neuromorphic platform can be scaled up easily without losing performance. Additionally, asynchronous events can be communicated efficiently using a packet switching network on chip [10], [13], [17].

As Neural Networks are mostly used for natural signal processing (like audio/video streams), which is temporally sparse, enabling sparse processing in inference engines will have a great impact on power consumption. In this paper, we propose a modified version of the bio-inspired Spiking Neural Network (SNN) to carefully adapt its interesting asynchronous inference feature into a non-spiking time-triggered neural network without compromising the accuracy. We show our proposed hybrid network can naturally exploit spatial and temporal sparsity by avoiding redundant operations.

Fig 1 illustrates a simplified schematic of an SNN which is used in this paper. In an SNN with Integrate&Fire neurons, each neuron has a membrane potential which updates upon the integration of the inputs from other neurons over time. If a neuron's membrane potential reaches a predefined threshold, an output spike is fired (asynchronous self-triggering with threshold) and propagated downstream to all its connected neurons. Otherwise, the spiking neuron remains silent and does not provide any output. Similar to our brain, in this neural network only small fractions of neurons are active simultaneously and due to asynchronous processing, they almost do not consume any energy when they are silent.

This type of processing has two advantages over synchronous processing: First, unlike synchronous neurons which need to update and communicate their output continuously, our SNN neuron only updates its membrane potential (internal state) when there is an event (spike) in one of its inputs (event-driven process). Second, an input spike can be processed immediately

[1]A video of eye movement during micro-saccades: https://youtu.be/U_xphN6ubmc

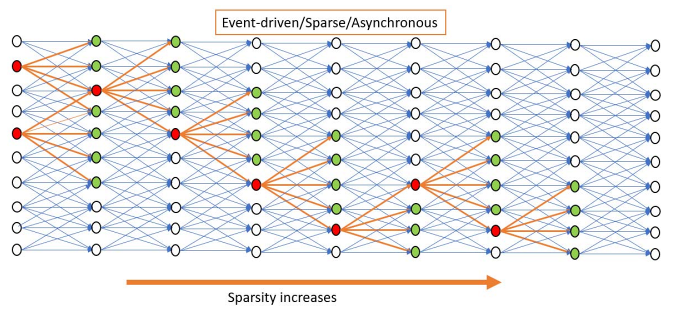

Fig. 1. A general feed-forward neural network. Small circles show individual neurons. The green neurons are the ones that received a spike and should be updated and the red neurons are the ones that fired after receiving a spike. The process starts from the left-hand side by input spikes coming from a sensor and ends in the right-hand side where output spikes can be easily related to a class of object or a decision which the network is trained to make. Typically, the amount of sparsity increases as we go further from the input layer to the output layer.

and the neuron can fire anytime without waiting for an external trigger signal (low latency).

Many researchers aim to train SNNs by using bio-inspired learning rules [18]–[21] and try to model the current understanding of the brain. Other researchers exploit the algorithmic advances in deep learning to train an ANN and study methods to convert a pre-trained ANN to an SNN architecture that is more suited for mapping to asynchronous neuromorphic platforms [22]–[26]. The third category tries to modify the available deep learning algorithms and directly train an SNN with error back-propagation [27]–[31]. As we do not want to touch the training algorithm and propose a one-to-one mapping of an already trained ANN neurons to SNN neurons (both architectures have the same number of neurons and synapses), this work mostly fits in the second category.

The most straightforward and used method to convert a pre-trained ANN to an SNN is to encode output values of ANN neurons in the firing rate of spiking neurons (rate coding) [32]. In this method, the output of an ANN unit is mapped to the average firing rate of a spiking neuron [24]. To reduce the amount of firing and synaptic operations in this coding scheme, some of the hyper-parameters of the SNN (like leak rate and threshold of neurons) can be optimized before [23], [24] or even during [26] inference. Even-though using frequency of firing as the neuron output makes the conversion mathematically exact, since the equations are based on the average firing rate of the neurons, many spikes per neuron are needed for the spiking network to become stable. This will increase latency, power and redundant processing of events in the hardware. This overhead is contradictory to the potential efficiency of SNN because low latency processing and low power consumption are the features that make SNN inference more interesting than their ANN counterparts [33]. Even though with this type of coding, we can benefit from spatial sparsity (since no spike will be fired for neurons with zero output), utilizing temporal sparsity is still a challenge.

Another approach is to encode the neuron output values into the time of firing (Temporal coding) [32]. One famous type of coding information in spikes is called Time To First Spike (TTFS) coding [27]. In this coding, the output value for each neuron is coded in the time of its first firing. So each neuron can fire at maximum one time. Neurons don't have leakage and should be reset after processing each input frame. Using this type of coding is not trivial for converting an already trained ANN to SNN. Therefore in the previous works, researchers tried to modify the deep learning methods and train the network directly with spiking neurons [28], [34]. TTFS is not the only way to encode information in the time of spikes. Recently some less restrictive solutions are also proposed [29]–[31]. In these cases, the SNN neurons are leaky Integrate and fire and they are not limited to fire only once after reset. Generally, these methods show great improvement from the latency and number of spikes compare to rate coding approaches. However, we found the following disadvantages for them which forced us to not using them:

- These methods need to re-train the network as they are not compatible with ANN inference. Additionally, the introduced training algorithms for these methods are not mature yet and do not show competitive performance for more complex tasks.
- In the mentioned works [27]–[31], additional to neuron state, synapses are also dynamic and state-full (not instant). This means that the arrival of each spike equals a leaky current injection to the neuron which adds complexity during hardware implementation [35].
- Specifically, the TTFS method is optimized for image processing and cannot exploit temporal sparsity in streaming video/audio processing as it is required to reset all neurons after each frame.
- From the hardware implementation point of view, coding information in time may slow-down the throughput as "delay" is part of the algorithm and cannot be avoided. Accelerating the process would be possible only if the communication jitter remains reasonably small compared to the minimum time unit of the process. However, a packet switching network, in general, does not guarantee very low jitter communication.

Considering our needs for sparse inference, we decided to introduce a new neuron model which is somewhere in between "synchronous ANN nodes" and "leaky integrate and fire (LIF) spiking neurons". Our goal was to optimize the number of operations for spatio-temporal sparsity while performing asynchronous inference.

This paper is a continuation of our previous brief paper [36]. In the present paper, in addition to more detailed explanations, we extend our algorithm to use "valued_spikes" which results in a smaller number of required events. Additionally, since our previous paper only reported results from the MNIST dataset, we have added two more experiments to strengthen the paper. In the second experiment, we process a framed-based video for an autonomous steering application. In this experiment we showed that when there is temporal sparsity at the input, our proposed algorithm can exploit it by reducing the number of required operations per frames, on average. For the third experiment, we use event-based cameras for hand-gesture recognition to show that the proposed algorithm works well with these types of inputs. In the next Section, we will explain the proposed inference algorithm. Then in Section III we present the results from bench-marking this algorithm with the mentioned data-sets. Finally, a brief disscussion is provided at the end.

II. Proposed Algorithm

In a conventional neural network, output of a neuron is related to it's inputs and weights with the following equations:

$$x = W \times I + b \tag{1}$$

$$o = f(x) \tag{2}$$

W and b are the weight array[2] ($1 \times n$) and bias (trainable parameters), I is the input array ($n \times 1$), o is the neuron's output, $f(x)$ is a non-linear activation function and x is called 'neuron state'. In a conventional inference, for every new input array (for example, a new frame in a video), x (and o) need to be recalculated which require n^2 multiply and accumulation (MAC). For natural signal processing, where the input to each neuron is sparse in time ($\Delta I(t)$ is a sparse array),[3] we can reformulate the equations.

$$\Delta I(t) = I(t) - I(t-1) \tag{3}$$

$$x(t) = (W \times \Delta I(t)) + x(t-1) \tag{4}$$

$$\Delta o(t) = f(x(t)) - f(x(t-1)) \tag{5}$$

Eq. 4 shows it is possible to calculate the current state of neuron by adding the amount of change to the previous state. For a sparse $\Delta I(t)$, in which number of non-zero elements is m ($m << n$), number of required MAC operation to calculate the new state shrinks to $m \times n$.

Eq. 4 can be re-written as an asynchronous, self-triggered event-based procedure as described in Algorithm 1. Each event (or spike) comes from one of the input synapses (synapse i) with weight w_i and carries a non-zero value (δ_i). Composition of all the events in time-step t makes the $\Delta I(t)$ array in Eq. 4 (very similar to sigma-delta modulation [37], [38]).

Algorithm 1 Update State of Neuron When an Input Event Comes From Neuron i

```
$x = b$
for each incoming spike do
    $x' = x + (w_i \times \delta_i)$
    $\Delta o = f(x') - f(x)$
    if $\Delta o \neq 0$ then
        Fire a spike with value $\delta = \Delta o$
    end if
    $x = x'$
end for
```

To make the computation more efficient, we can quantize the Δo and introduce a threshold (T) on the minimum change. This will result in skipping events for negligible changes. To do so, we change the Δo equation in Algorithm 1 with the Eq. 6.

$$\Delta o = T \times \left(\left\lfloor \frac{f(x')}{T} \right\rfloor - \left\lfloor \frac{f(x)}{T} \right\rfloor \right) \tag{6}$$

where $\lfloor A \rfloor$ is the integer part of A.

[2] n is the number of inputs to the neuron.

[3] t is the algorithm time-step, for example can be the frame number in a frame-based system.

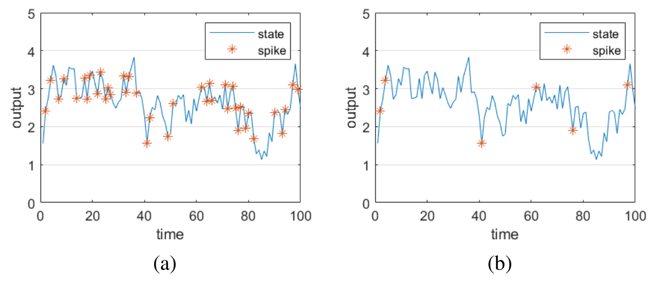

Fig. 2. Output of a neuron after receiving events with random values along with the generated output spikes when threshold is one (a)direct quantization (b)hysteresis quantization.

Algorithm 2 Update State of Neuron When an Input Event Comes From Neuron i, Hysteresis Quantization in Time Is Applied

```
$x = b$
$x_{LFT} = 0$
for each incoming spike do
    $x \leftarrow x + (w_i \times \delta_i)$
    $\Delta o = f(x) - f(x_{LFT})$
    if $|\Delta o| \geq T$ then
        Fire a spike with value $\delta = \Delta o$
        $x_{LFT} = x$
    end if
end for
```

Choosing the threshold is a trade-off as a coarser quantization results in higher quantization error but will generate fewer spikes (an extreme case is using an ANN with binary activation [39], [40]). Direct quantization, as explained in Eq. 6, results in an extra firing activity of spiking neurons. As it can be seen in Fig. 2(a) in direct quantization, when the output ($o = f(X)$) is somewhere near the transition point of two quantization levels, small variations/oscillations in output may result in extra firing of spikes which reduces sparsity in inference execution.

To increase sparsity even more, we decided to use hysteresis quantization. The output of the "Hysteresis Quantizer" depends not only on the current input value but also on the previous value of its output. Eq. 7 formulates the hysteresis quantization method where the quantization level (threshold) is '1'.

$$out_{new} \leftarrow \begin{cases} out_{old} + 1, & (input - out_{old}) > +1 \\ out_{old} - 1, & (input - out_{old}) < -1 \\ out_{old}, & otherwise \end{cases} \tag{7}$$

Algorithm 2 is proposing an asynchronous event-based inference with hysteresis quantization of neuron's output over time. As it can be seen, an extra persistent variable per neuron (x_{LFT}) is required to support hysteresis quantization.[4]

As mentioned before, unlike a neuronal unit in an ANN, the membrane potential of our spiking neuron dynamically changes over time by receiving asynchronous

[4] LFT stands for 'Last Firing Time'. We use x_{LFT} to recover the out_{old} in Eq.7, therefore it is possible to directly store the o_{LFT}.

pre-synaptic spikes. Compare to more conventional SNN neurons, in our current scheme a pre-synaptic spike charges the neuron immediately (stateless synapse) and the membrane potential leak is not needed (except for the first layer in case of using Dynamic Vision Sensor which is explained in section III).

III. Results

To be able to compare with state-of-the-art results and the performance indexes, we have applied the proposed method on three different datasets.

The first dataset is the well known handwritten digit dataset (MNIST [41]) used for image processing and contains static images (without temporal sparsity). As the content of MNIST frames is almost binary pixels (black and white) with a high amount of sparsity (on average, around 84% of pixels in each frame are zero), we decided to include this dataset for benchmarking.

The second dataset is autonomous steering dataset (PilotNet [42]) which contains a video stream recorded by an installed camera in-front of a car and a synchronous recording of the steering angle as the ground-truth label. This dataset reveals how our proposed algorithm can improve processing performance by exploiting temporal sparsity.

The third dataset is a set of hand gestures recorded by an event-based camera (DVS128 Gesture Dataset [43]). Event-based cameras or Dynamic Vision Sensors (DVSs) [44], [45], [45], [46] are bio-inspired vision sensors with independent asynchronous pixels which generate events when they detect a change in light intensity. Dynamic Vision Sensors generate and transmit information of the visual scene in a completely different manner than conventional frame-based cameras. Several advantages (like high-speed sensing, high dynamic range, and low power consumption) result in their growing use in neuromorphic engineering and low power mobile applications. The sparse event-based data provided by these sensors can be directly consumed by our network.[5]

A. Handwritten Digit Recognition (MNIST)

As the first experiment, we have applied the proposed method using the MNIST [41] dataset. For this experiment, we used binary output spikes (± 1) for all hidden layers. Initially, we have trained an ANN for the MNIST dataset using standard stochastic gradient descent. For conversion of the ANN network to SNN, the weights and biases are used in the equivalent SNN. To do the conversion we need to define the threshold (equivalent to quantization level) and also a method to convert input frames to spikes.

We converted the input frames to spikes with $Value = +1$ by sending the pixels with higher intensity earlier in time ("linear intensity to delay" coding) [18]. Additionally, pixels with an intensity equal or less than '0.2' (pixel values are normalized between '0' and '1') will not fire at all (the quantization threshold for input frame is '0.2'). Using this

[5]As it will be explained, we need to use a frame to event conversion for the other two datasets.

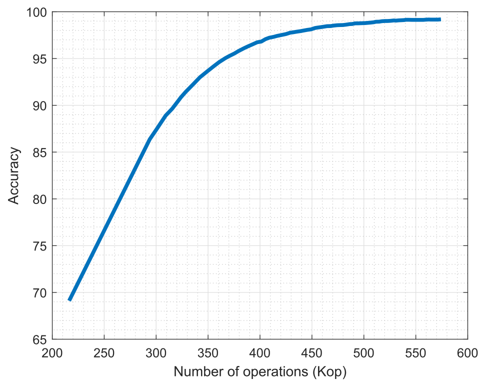

Fig. 3. Accuracy of SNN versus the average number of operations for all the neurons in the MNIST experiment network.

method, each MNIST frame is converted to a spike train with the duration of '0.8' TU (time Unit).[6]

We have trained a 4-layer convolutional neural network ($16C5 - 2AP2 - 8C5 - 2AP2 - 256FC[50\% Dropout] - 10FC$).[7] Table I shows the results of the proposed method for the MNIST dataset.

"Avg Num Spikes" in Table I, reports the average number of spikes per input frame per neuron in the network. To measure the accuracy of the MNIST network, we accumulated the number of output spikes (considering the sign of spikes) and the maximum number was assumed as the winner class.

In Table I, we also added the results from the other state-of-the-art works on supervised training of Spiking Neural Networks. Most of the previous works only reported the accuracy but not the number of spikes. In our conversion method simulation ends after one-time presentation of all the input spikes. However, in the case of "Rate Coding" where the intensity of the input pixels is converted to the firing rate, there is not a definite time for the end of the input presentation. In this case, by the longer presentation of the input, the accuracy increases until the network becomes completely stable. On the other hand, longer input presentation results in more number of spikes on average, resulting in a trade-off.

We measured the pre-matured accuracy of our SNN when only part of the input spike is presented, which is plotted in Fig.3. This figure shows the average number of operations (synaptic update) which is needed to achieve a specific accuracy.[8]

B. Autonomous Steering (PilotNet)

PilotNet is a neural network introduced by NVIDIA [42] along with the dataset.[9] The dataset contains video recording (10 frames per second) by a camera placed in front of the car

[6]Time Unit (TU) is a unit of time and depends on the real-time hardware processing capability. For simulation purposes, a TU can be a second, millisecond, microsecond or so without losing generality.

[7]xCy is a convolutional layer with x filters and kernel size of $y \times y$ and xFC is a fully connected layer with x neurons. $xAPy$ is an Average-Pooling layer with kernel size of $x \times y$. For these convolutions, we keep the input and output size equal.

[8]In this experiment, the proposed Spiking Neural Network does not need multiplication as spike values are binary.

[9]https://github.com/lhzlhz/PilotNet

TABLE I

ACCURACY OF QUANTIZED ANN AND EQUIVALENT SNN FOR MNIST DATASET

	Model	ANN Accuracy	SNN Accuracy	Number of Neurons	Number of Parameters	Avg Num Spikes (per frame)
This work(thr=1.0)	**2xConv-2xFC**	**99.21%**	**99.19%**	**15k**	**100k**	**0.25**
This work(thr=2.0)	2xConv-2xFC	99.21%	97.26%	15k	100k	0.08
Rate Coding [24] (Reproduced, 25 time steps)	2xConv-2xFC	99.21%	99.18%	15k	100k	0.48
Rate Coding [24] (Reproduced, 15 time steps)	2xConv-2xFC	99.21%	98.80%	15k	100k	0.26
TTFS coding [27]	3xConv-2xFC	98.96%	98.57%	7.7k	1.2M	0.13
TTFS coding [47]	2xFC	98.50%	96.98%	1.4k	476k	0.10
Rate Coding [23]	2xConv-1xFC	99.14%	99.1%	5.1k	50k	19.25
SLAYER [29]	2xConv-1xFC	—	99.36%	22k	145k	—
DECOLLE [30]	3xConv-1xFC	99.09%	98.77%	19k	73k	—
Rate Coding [24]	—	99.44%	99.44%	8k	1.2M	—
Rate Coding [25]	1xConv-1xFC	98.3%	98.32%	9.5k	88k	—
Rate Coding [48]	2xConv-2xFC	—	99.42%	20.7k	0.6M	—
TTFS coding [28]	2xFC	—	97.55%	1.5k	635k	—

as well as the corresponding steering angels for each frame of the video.[10] To find out the amount of redundancy (especially temporal sparsity) in this dataset, we compressed the raw frames using "Motion JPEG 2000 lossless compression" algorithm which resulted in 18 times reduction in size. In our SNN, we exploit sparsity in all the neural layers (not only input layer) but sparsity in the input can result in sparsity in deeper layers.

PilotNet is a 10-layer feed-forward convolutional neural network (CNN) with input size of $3 \times 66 \times 200$ and configuration of $24C5-36C5-48C5-64C3-64C3-1164FC-100FC-50FC-10FC-1FC$. The output of the convolutions is only 'Valid' area (reduced size) with a stride of two in the first three layers. Activation functions for all the layers except the output neuron is $ReLU$. The output of this network is only one neuron with the activation function of $Tanh$ which predicts the steering angle for the input frame. This network contains 108K neurons and 1.6M parameters.

To run our asynchronous inference on the PilotNet, we used the provided trained network model. Input events to the network is generated with the Algorithm 3 which is similar to our spiking neuron model with a linear activation function.

Algorithm 3 Generating Input Events From Video Stream, Every Pixel Is Considered as One Neuron

```
$x_{LFT}(0) = 0$
for every frame do
    $x \leftarrow Pixel\ Value$
    $\Delta o = x - x_{LFT}$
    if $|\Delta o| \geq T$ then
        Fire a spike with value $\delta = \Delta o$
        $x_{LFT} \leftarrow x$
    end if
end for
```

We used the original trained model directly and only provided the neurons' threshold value to have reasonable accuracy. Fig. 4 shows the output of the PilotNet for the original CNN inference and the proposed SNN inference for part of the dataset.

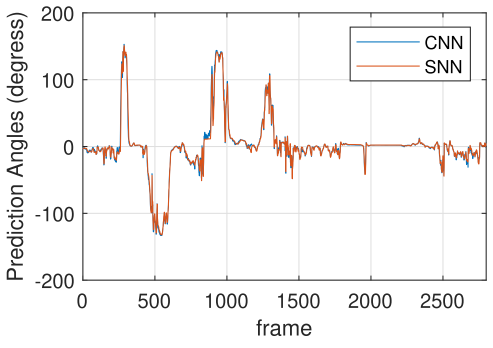

Fig. 4. Prediction angles of the first few frames for the original CNN inference and the proposed SNN inference.

Fig. 5 shows the number of required MAC operations during inference of PilotNet in the original CNN and also number of synaptic update (updates in Algorithm 2) in the SNN for the first 2800 frames.[11] For these frames, average number of operations in SNN is 3.1Mops and it decreases to 1.7 for the first 20,000 frames which is more than 16 times less than plain CNN inference. The current dataset has a very low frame rate (10 fps). To find out how are the numbers for higher frame-rates, we made a 15min recording with a 480fps camera[12] which resulted in 140 times reduction in number of operations compare to the plain CNN. As it is predictable, by increasing the frame-rate or increasing the resolution of the frames, the amount of reduction over the number of operation will increase due to increase in the redundancy.

C. Hand Gesture Recognition (DVS128 Gesture Dataset)

In the third experiment, we have evaluated the performance of our network by using the hand gesture dataset from IBM [43]. This dataset is recorded with a DVS camera for a gesture classification task and includes data recorded

[10]https://youtu.be/_N7nC-8YxzE

[11]https://youtu.be/OySJKJWGrjw

[12]https://youtu.be/LyPTj3Parp0

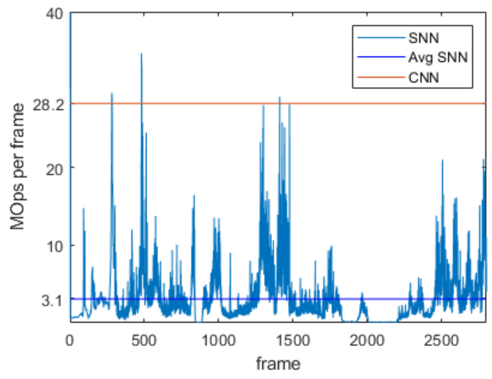

Fig. 5. The number of synaptic updates of the first few frames for the proposed SNN inference in comparison with the Number of MAC operations for the original CNN inference.

from 29 subjects, each performing 11 hand gestures under 3 different lighting conditions. Each recording is a stream of events with an average duration of 72 seconds.

Events from DVS contain a binary value (± 1 as the sign of spike), which means an increase in light intensity results in a positive spike and decrease in light intensity results in a negative spike. In an ideal case (ignoring the analog circuit artifacts and noise), a DVS pixel behaves very similar to what is described in Algorithm 3 except the fact that there is no frame-period. Therefore DVS pixels reacts almost instantaneously and asynchronously to the change in light intensity.

In practical scenarios, due to noise in analog circuits, each pixel may have a little bit of unbalance in their positive and negative thresholds. Additionally, each pixel may have spontaneous positive or negative firing without any change in light intensity. If we change the light intensity from α to β and then back to α again, it is highly probable that the accumulation of the output events from an analog DVS may have an offset from zero. If we directly connect an analog DVS to our network, this artifact results in the saturation of the neurons' state in the next layers. To mitigate this issue, we took the following actions: 1) We do not use the sign of events as the value of the event and apply $value = +1$ for all the DVS events. Instead, we accumulate events with different signs in different neurons (which means we consider two input channels for our network).[13] 2) We introduce an intermediate layer between DVS and our neural network. This layer is built-up with the SNN neurons in Algorithm 2 and a linear activation function. However, as input events are always positive, to avoid saturation and to forget the past events, we apply a periodic exponential leakage in this layer as described in Algorithm 4. This process happens in parallel with normal event processing which is described in Algorithm 2.

As it can be seen in Algorithm 4, leakage also can cause spike firing. These spikes propagate in the network and result in the removal of the past event from neurons' state. Please note that leakage process in Algorithm 4 only applies to the intermediate layer between DVS and our SNN.

[13]It is possible to completely ignore the sign of the events and only have one input channel. In [49] it is explained that ignoring the sign of events may result in a small accuracy loss.

Algorithm 4 Leakage of State for Intermediate Layer Between DVS and Our SNN. This Algorithm Runs Periodically to Apply Leak. τ Is the Leak Rate

```
for every leak period do
    $x \leftarrow x \times e^{(leak\ period/\tau)}$
    $\Delta o = x - x_{LFT}$
    if $|\Delta o| \geq T$ then
        Fire a spike with value $\delta = \Delta o$
        $x_{LFT} \leftarrow x$
    end if
end for
```

To use standard back-propagation of error and GPU accelerated training, we train our networks in Tensorflow with a frame-based method. However, for this experiment, input data is event-based. To use our frame-based platform for training, we used the fact that neuron states in our SNN network are equivalent to neuron states in the frame-based network (when ignoring the quantization error). To build a dataset for training, we take periodic snapshots from the state of neurons in our intermediate leaky layer. These snapshots are used as the input frames for our ANN training platform. The snapshot rate should be high enough to capture the temporal dynamics of the scene. In this experiment, we used the sampling period of 64 ms. Using higher snapshot rate results in more accurate SNN conversion but also slows down the training.

Similarly, the leak rate (τ) in Algorithm 4 is a parameter that depends on the dynamics of the scene. By looking at the snapshots from the intermediate layer, it is possible to manually choose a proper leak rate. High leak rate results in empty snapshots while a low leak rate results in blurred snapshots. In this experiment leak rate is $32ms$. It is also possible to systemically optimize the leak rate by looking at the network accuracy. Leak Period is another hyper-parameter in Algorithm 4. Ideally, leak period should be as small as possible. However, to reduce the number of operations, we need to compromise. In this experiment leak period is $6.4ms$.

We have trained a 4-layer all convolutional neural network (CNN) [50] with input size of $2 \times 64 \times 64$ and configuration of $64C7 - 2AP2 - 128C7 - 2AP2 - 128C7 - 2AP2 - 11GAP$(Global Average Pooling). The activation function for all the hidden layers is $ReLU$.

To report the accuracy, we used the state of output neurons and reported the neuron with maximum output as winner.[14] As in this dataset, the presentation of each class takes 6.5 seconds on average, one may choose different inference time windows for classification. We perform each inference by resetting the state of all neurons and then presenting a part of the recording with the length of inference time window. For example, when the inference time window is $640ms$, a recording of 6.4 seconds is divided into 10 individual recordings and results in 10 separate classifications.

[14]This is equivalent to choose $T = 0$ for the last layer and accumulate the values of output spikes for each output neurons.

TABLE II

COMPARISON OF CLASSIFICATION RESULTS FOR DVS-GESTURE DATASET

	Model	ANN Accuracy	SNN Accuracy	inference window	Number of neurons	Number of parameter
This work(thr=64)	**4xConv**	**90.5%**	**95.24%**	**896ms**	**445K**	**1.2M**
IBM EEDN [43]	16xConv	93.06%	94.59	80ms	286K	19M
SLayer [29]	8-layers	—	93.64%	1500ms	—	—
DECOLLE [30]	3xConv-1xFC	—	94.18%	1800ms	95K	1.4M

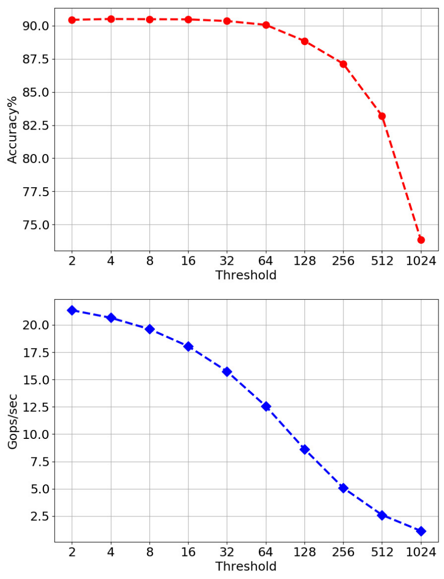

Fig. 6. Measured accuracy and number of synaptic update in SNN inference system for classification of DVS gesture dataset, over a range of spiking threshold for hidden layers and inference time window of 64 *ms*.

For the experiments in this section, we have quantized all of the synaptic weights to 8 bits and stored them as integer numbers between -128 to $+127$. We applied several spiking thresholds to neurons of hidden layers to illustrate the trade-off between accuracy and number of operations.[15] The results are summarized in Fig.6 where accuracy is reported with the inference time window of 64 ms. The number of synaptic operations per second for the proposed SNN inference has also been shown in Fig.6. In an equivalent CNN with frame-based input and frame rate of 60 Hz, the number of required MAC operations is around 146 $Gops$ per second. Fig.6 shows for SNN with $T = 256$ number of synaptic updates is around 5 $Gops$.

Fig.7 shows the accuracy of the SNN inference for several inference time windows. As it is expected, by increasing the inference time window which equivalent of increasing the amount of information for each classification, the classification accuracy (and latency) increases. We have compared the results of this experiment with some other recent works in Table II. In this table, SNN accuracy is the accuracy of the system after presenting input for a time equal to "inference window", while CNN accuracy refers to the accuracy of the frame-based system. For example in our work, each frame period is 64ms and therefore, CNN accuracy is possible to be less than the SNN accuracy when SNN inference window is 896ms. As it is illustrated in Fig.7, SNN accuracy when

[15]In each test, all the neurons have a similar threshold.

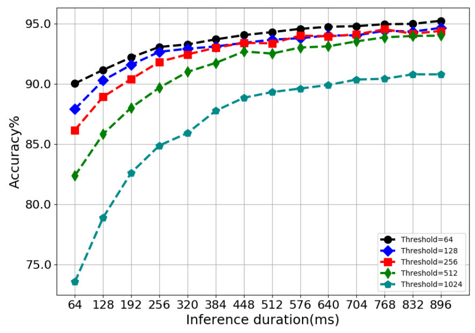

Fig. 7. Accuracy at different inference time windows and for 5 different thresholds.

inference window is 64ms, is always smaller or equal to the CNN accuracy.

IV. DISSCUSSION

In this work, we presented a method to convert a synchronous ANN to an asynchronous SNN and efficiently exploit spatio-temporal sparsity. Even though our goal was not trying to mimic biological neural networks, we used the sparsity and asynchronous execution features of bio-inspired implementations. Our motivation was to offer a better implementation of synchronous ANN since a direct ANN implementation is not easily scalable and won't easily exploit the sparsity of the network for efficient power consumption. We have shown our proposed method reduces the number of operations dramatically for different datasets while its impact on accuracy is negligible.

In this paper, we compared the number of operations in an ANN with the number of operations in the proposed SNN. However, the type of operations on these two platforms are not exactly similar. Even though both operations include reading weights and inputs from memories, performing a MAC operation and writing back the results, SNN neurons should perform extra comparison and subtraction. However, in recent silicon technology, energy consumption is dominant by memory accesses. For example, a 32 bit read and write into an on-chip SRAM memory consumes approximately 6 times than a 32b floating-point addition and 12 times than a 32b fixed-point addition.[16] Read and write into an off-chip DRAM memory will be approximately 100 times more expensive than the on-chip SRAM.

As it is mentioned before, even though using this algorithm reduces the amount of memory accesses, having stateful neurons increase the footprint of memories. For example, for the MNIST/PilotNet/Hang-Gesture networks which have 15k/108k/445k neurons, we need to allocate

[16]This numbers are approximate and may change based on the technology, temperature, and layout of the chip.

60KB/432KB/1.78MB of state memory (if states of a neuron consume 32b) beside the 100KB/1.6MB/1.2MB of weight memory (for 8b weights). For the hardware platforms that are using expensive on-chip SRAM to reduce power consumption, this might be a big problem (due to silicon area). Additionally, using event-driven process requires true random access to the memory which disrupts burst data read/write from/to off-chip memory and reduces its effective bandwidth. Nevertheless, new memory technologies can be a good solution for this problem [51] due to their compact bit size and lower static power consumption. An option would be to stop thinking about the integration of memory into our processors and think of integrating our processing units into a memory chip [52].

Other optimization methods like weight quantization, weight pruning or activation quantization during training can be done in parallel to this work for further optimizing the networks. For future works, we will take a deeper look into advanced video compression algorithms to see how we can adapt their techniques to more efficiently remove redundancies in space, time and frequency domains.

Acknowledgment

The authors would like to thank all GrAI Matter Labs employees, specially Ingolf Held, Gokturk Cinserin, Peng Qiao, Louis Rouillard and Georgi Valkov for their kind support and interesting discussions. The authors would like to thanks the reviewers for the valuable comments and their contribution in improvement of this paper.

References

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

[2] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *Proc. Int. Conf. Learn. Represent.*, 2019. [Online]. Available: https://openreview.net/forum?id=rJlnB3C5Ym

[3] A. Aimar *et al.*, "NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 3, pp. 644–656, Mar. 2019.

[4] R. Q. Quiroga and G. Kreiman, "Measuring sparseness in the brain: Comment on bowers (2009)," *Psychol. Rev.*, vol. 117, no. 1, pp. 291–297, 2010.

[5] S. Shoham, D. H. O'Connor, and R. Segev, "How silent is the brain: Is there a 'dark matter' problem in neuroscience?" *J. Comparative Physiol. A*, vol. 192, no. 8, pp. 777–784, 2006.

[6] *How Much Bandwidth Does Skype Need?* Accessed: Oct. 2019. [Online]. Available: https://support.skype.com/en/faq/FA1417/how-much-bandwidth-does-skype-need

[7] T. Masquelier, G. Portelli, and P. Kornprobst, "Microsaccades enable efficient synchrony-based coding in the retina: A simulation study," *Sci. Rep.*, vol. 6, Apr. 2016, Art. no. 24086.

[8] Telecommunication Standardization Sector of ITU. (2019). *H.264: Advanced Video Coding for Generic Audiovisual Services*. [Online]. Available: https://www.itu.int/rec/T-REC-H.264

[9] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. Spring Joint Comput. Conf. AFIPS*, Apr. 1967, pp. 483–485.

[10] A. Yousefzadeh *et al.*, "On multiple AER handshaking channels over high-speed bit-serial bidirectional LVDS links with flow-control and clock-correction on commercial FPGAs for scalable neuromorphic systems," *IEEE Trans. Biomed. Circuits Syst.*, vol. 11, no. 5, pp. 1133–1147, Oct. 2017.

[11] R. Brette *et al.*, "Simulation of networks of spiking neurons: A review of tools and strategies," *J. Comput. Neurosci.*, vol. 23, no. 3, pp. 349–398, Dec. 2007.

[12] G. Indiveri *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers Neurosci.*, vol. 5, p. 73, May 2011.

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

[14] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proc. IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[15] P. A. Merolla and *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, pp. 668–673, Aug. 2014.

[16] A. Yousefzadeh, G. Orchard, E. Stromatias, T. Serrano-Gotarredona, and B. Linares-Barranco, "Hybrid neural network, an efficient low-power digital hardware implementation of event-based artificial neural network," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[17] L. A. Plana, J. Bainbridge, S. Furber, S. Salisbury, Y. Shi, and J. Wu, "An on-chip and inter-chip communications network for the spinnaker massively-parallel neural net simulator," in *Proc. IEEE Int. Symp. Networks-on-Chip (NOCS)*, Apr. 2008, pp. 215–216.

[18] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," 2017, *arXiv:1611.01421*. [Online]. Available: https://arxiv.org/abs/1611.01421

[19] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, "Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks," *Pattern Recognit.*, vol. 94, pp. 87–95, Oct. 2019.

[20] A. Yousefzadeh, T. Masquelier, T. Serrano-Gotarredona, and B. Linares-Barranco, "Hardware implementation of convolutional stdp for on-line visual feature learning," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[21] A. Yousefzadeh, E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, "On practical issues for stochastic STDP hardware with 1-bit synaptic weights," *Frontiers Neurosci.*, vol. 12, p. 665, Oct. 2018.

[22] J. A. Pérez-Carrasco and *et al.*, "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. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2706–2719, Nov. 2013.

[23] 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 *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8.

[24] 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 Neurosci.*, vol. 11, p. 682, Dec. 2017.

[25] E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, "An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data," *Frontiers Neurosci.*, vol. 11, p. 350, Jun. 2017.

[26] D. Zambrano, R. Nusselder, H. S. Scholte, and S. M. Bohté, "Sparse computation in adaptive spiking neural networks," *Frontiers Neurosci.*, vol. 12, p. 987, Jan. 2019.

[27] B. Rueckauer and S.-C. Liu, "Conversion of analog to spiking neural networks using sparse temporal coding," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[28] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 7, pp. 3227–3235, Jul. 2018.

[29] S. B. Shrestha and G. Orchard, "SLAYER: Spike layer error reassignment in time," in *Proc. Adv. Neural Inf. Process. Syst.*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 1412–1421.

[30] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning," 2018, *arXiv:1811.10766*. [Online]. Available: https://arxiv.org/abs/1811.10766

[31] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," 2019, *arXiv:1901.09948*. [Online]. Available: https://arxiv.org/abs/1901.09948

[32] R. Rullen and S. Thorpe, "Rate coding versus temporal order coding: What the retinal ganglion cells tell the visual cortex," *Neural Comput.*, vol. 13, no. 6, pp. 1255–1283, Jun. 2001.

[33] C. Farabet *et al.*, "Comparison between frame-constrained fix-pixel-value and frame-free spiking-dynamic-pixel convnets for visual processing," *Frontiers Neurosci.*, vol. 6, p. 32, Apr. 2012.

[34] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," 2016, *arXiv:1608.08782*. [Online]. Available: https://arxiv.org/abs/1608.08782

[35] T. S. Gotarredona and B. L. Barranco, "Method and system for converting pulsed-processing neural network with instantaneous integration synapses into dynamic integration synapses," U.S. Patent 2015 0 120 631 A1, Apr. 30, 2015.

[36] A. Yousefzadeh *et al.*, "Conversion of synchronous artificial neural network to asynchronous spiking neural network using sigma-delta quantization," in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Mar. 2019, pp. 81–85.

[37] B. E. Boser and B. A. Wooley, "The design of sigma-delta modulation analog-to-digital converters," *IEEE J. Solid-State Circuits*, vol. 23, no. 6, pp. 1298–1308, Dec. 1988.

[38] Y. C. Yoon, "LIF and simplified SRM neurons encode signals into spikes via a form of asynchronous pulse sigma–delta modulation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 5, pp. 1192–1205, May 2017.

[39] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Computer Vision—ECCV*. Cham, Switzerland: Springer, 2016.

[40] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1," 2016, *arXiv:1602.02830*. [Online]. Available: https://arxiv.org/abs/1602.02830

[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[42] M. Bojarski *et al.*, "Explaining how a deep neural network trained with end-to-end learning steers a car," 2017, *arXiv:1704.07911*. [Online]. Available: https://arxiv.org/abs/1704.07911

[43] A. Amir *et al.*, "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7388–7397.

[44] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128 $\times$ 128 1.5% contrast sensitivity 0.9% FPN 3μs latency 4mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers," *IEEE J. Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, Mar. 2013.

[45] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output," *Proc. IEEE*, vol. 102, no. 10, pp. 1470–1484, Oct. 2014.

[46] A. Yousefzadeh, "Digital design for neuromorphic bio-inspired vision processing," Ph.D. dissertation, Dept. Electron. Electromagn., Univ. Sevilla, Seville, Spain, 2018.

[47] H. Mostafa, B. U. Pedroni, S. Sheik, and G. Cauwenberghs, "Fast classification using sparsely active spiking networks," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[48] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers Neurosci.*, vol. 12, p. 331, May 2018.

[49] A. Yousefzadeh, G. Orchard, T. Serrano-Gotarredona, and B. Linares-Barranco, "Active perception with dynamic vision sensors. minimum saccades with optimum recognition," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 4, pp. 927–939, Aug. 2018.

[50] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," 2014, *arXiv:1412.6806*. [Online]. Available: https://arxiv.org/abs/1412.6806

[51] B. Linares-Barranco, "Memristors fire away," *Nature Electron.*, vol. 1, pp. 100–101, Feb. 2018.

[52] D. Fick. (2018). *Mythic Hot Chips 2018*. [Online]. Available: https://medium.com/mythic-ai/mythic-hot-chips-2018-637dfb9e38b7