This paper has been accepted for publication at the
European Conference on Computer Vision (ECCV), Glasgow, 2020

Event-based Asynchronous Sparse Convolutional Networks

Nico Messikommer*, Daniel Gehrig*, Antonio Loquercio, and Davide Scaramuzza

Dept. Informatics, Univ. of Zurich and
Dept. of Neuroinformatics, Univ. of Zurich and ETH Zurich

Abstract. Event cameras are bio-inspired sensors that respond to per-pixel brightness changes in the form of asynchronous and sparse "events". Recently, pattern recognition algorithms, such as learning-based methods, have made significant progress with event cameras by converting events into *synchronous* dense, image-like representations and applying traditional machine learning methods developed for standard cameras. However, these approaches discard the spatial and temporal sparsity inherent in event data at the cost of higher computational complexity and latency. In this work, we present a general framework for converting models trained on synchronous image-like event representations into *asynchronous* models with identical output, thus directly leveraging the intrinsic asynchronous and sparse nature of the event data. We show both theoretically and experimentally that this drastically reduces the computational complexity and latency of high-capacity, synchronous neural networks without sacrificing accuracy. In addition, our framework has several desirable characteristics: (i) it exploits spatio-temporal sparsity of events explicitly, (ii) it is agnostic to the event representation, network architecture, and task, and (iii) it does not require any train-time change, since it is compatible with the standard neural networks' training process. We thoroughly validate the proposed framework on two computer vision tasks: object detection and object recognition. In these tasks, we reduce the computational complexity up to 20 times with respect to high-latency neural networks. At the same time, we outperform state-of-the-art *asynchronous* approaches up to 24% in prediction accuracy.

Keywords: Deep Learning: Applications, Methodology, and Theory, Low-level Vision

Multimedia Material

The code of this project is available at `https://github.com/uzh-rpg/rpg_asynet`. Additional qualitative results can be viewed in this video: `https://youtu.be/g_I5k_QFQJA`

* Equal contribution

1 Introduction

Event cameras are *asynchronous* sensors that operate radically differently from traditional cameras. Instead of capturing dense brightness images at a fixed rate, event cameras measure brightness *changes* (called *events*) for each pixel independently. Therefore, they sample light based on the scene dynamics, rather than on a clock with no relation to the viewed scene. By only measuring brightness changes, event cameras generate an asynchronous signal both sparse in space and time, usually encoding moving image edges [1] [1]. Consequently, they automatically discard redundant visual information and greatly reduce bandwidth. In addition, event cameras possess appealing properties, such as a very high dynamic range, high temporal resolution (in the order of microseconds), and low power consumption.

Due to the sparse and asynchronous nature of events, traditional computer vision algorithms cannot be applied, prompting the development of novel approaches. What remains a core challenge in developing these approaches is how to efficiently extract information from a stream of events. An ideal algorithm should maximize this information while exploiting the signal's spatio-temporal sparsity to allow for processing with minimal latency.

Existing works for processing event data have traded-off latency for prediction accuracy. One class of approaches leverage filtering-based techniques to process events in sequence and thus provide predictions with high temporal resolution and low latency [2,3,4,5]. However, these techniques require significant engineering: event features and measurement update functions need to be handcrafted. For this reason, they have difficulties in generalizing to many different tasks, especially high level ones as object recognition and detection. Similarly, other works aim at reducing latency by making inference through a dynamical system, *e.g.* a spiking neural network (SNN)[2]. Despite having low latency, both filtering methods and SNNs achieve limited accuracy in high levels tasks, mainly due to their sensitivity to tuning and their difficult training procedure, respectively. Recently, progress has been made by processing events in batches that are converted into intermediate input representations. Such representations have several advantages. Indeed, they have a regular, *synchronous* tensor-like structure that makes them compatible with conventional machine learning techniques for image-based data (e.g. CNN). This has accelerated the development of new algorithms [3,10,11,12,13]. In addition, it has been shown that many of these representations have statistics that overlap with those of natural images, enabling transfer learning with networks pretrained on image data [11,12,14,15]. This last class of approaches achieves remarkable results on several vision benchmarks but at the cost of discarding the asynchronous and sparse nature of event data.

[1] https://youtu.be/LauQ6LWTkxM?t=4

[2] Here we use the term SNN as in the neuromorphic literature [6], where it describes continuous-time neural networks. Other networks which are sometimes called SNNs are low precision networks, such as binary networks [7]. However, these are not well suited for asynchronous inputs [6,8,9].

By doing so they perform redundant computation at the cost of large inference times, thus losing the inherent low latency property of the event signal.

Contributions We introduce a general event-based processing framework that combines the advantages of low latency methods and high accuracy batch-wise approaches. Specifically, we allow a neural network to exploit the asynchronous and sparse nature of the input stream and associated representation, thus drastically reducing computation.We mathematically show that the resulting asynchronous network generates identical results to its synchronous variant, while performing strictly less computation. This gives our framework several desirable characteristics: (i) it is agnostic to the event representation, neural network architecture, and task; (ii) it does not require any change in the optimization or training process; (iii) it explicitly models the spatial and temporal sparsity in the events. In addition, we demonstrate both theoretically and experimentally that our approach fully exploits the spatio-temporal sparsity of the data. In order to do so, we relate our framework's computational complexity to the intrinsic dimensionality of the event signal, *i.e.* the events' stream fractal dimension [16]. To show the generality of our framework, we perform experiments on two challenging computer vision tasks: object recognition and object detection. In these tasks, we match the performance of high capacity neural networks but with up 20 times less computation. However, our framework is not limited to these problems and can be applied without any change to a wide range of tasks.

2 Related Work

The recent success of data-driven models in frame-based computer vision [17,18,19] has motivated the event-based vision community to adopt similar pattern recognition models. Indeed, traditional techniques based on handcrafted filtering methods [2,3,4,5] have been gradually replaced with data-driven approaches using deep neural networks [10,11,12,13,15]. However, due to their sparse and asynchronous nature, traditional deep models cannot be readily applied to event data, and this has sparked the development of several approaches to event-based learning. In one class of approaches, novel network architecture models directly tailored to the sparse and asynchronous nature of event-based data have been proposed [2,6,8,9,20,21,22]. These include spiking neural networks (SNNs) [2,6,8,9,20] which perform inference through a dynamical system by processing events as asynchronous spike trains. However, due to their novel design and sensitivity to tuning, SNNs are difficult to train and currently achieve limited accuracy on high level tasks. To circumvent this challenge, a second class of approaches has aimed at converting groups of events into image-like representations, which can be either hand-crafted [3,10,11,13] or learned with the task [12]. This makes the sparse and asynchronous event data compatible with conventional machine learning techniques for image data, *e.g.* CNNs, which can be trained efficiently using back-propagation techniques. Due to the higher signal-to-noise ratio of such representations with respect to raw event data, and the

high capacity of deep neural networks, these methods achieve state-of-the-art results on several low and high level vision tasks [11,12,13,23,24]. However, the high performance of these approaches comes at the cost of discarding the sparse and asynchronous property of event data and redunant computation, leading to higher latency and bandwidth. Recently, a solution was proposed that avoids this redundant computation by exploiting sparsity in input data [25]. Graham et al. proposed a technique to process spatially-sparse data efficiently, and used them to develop spatially-sparse convolutional networks. Such an approach brings significant computational advantages to sparse data, in particular when implemented on specific neural network accelerators [26]. Sekikawa et al. [27] showed similar computation gains when generalizing sparse operations to 3D convolutional networks. However, while these methods can address the spatial sparsity in event data, they operate on synchronous data and can therefore not exploit the temporal sparsity of events. This means that they must perform separate predictions for each new event, thereby processing the full representation at each time step. For this reason previous work has focused on finding efficient processing schemes for operations in neural networks to leverage the temporal sparsity of event data. Scheerlinck et al. [28] designed a method to tailor the application of a single convolutional kernel, an essential building block of CNNs, to asynchronous event data. Other work has focused on converting trained models into asynchronous networks by formulating efficient, recursive update rules for newly incoming events [29,30] or converting traditional neural networks into SNNs [31]. However, some of these conversion techniques are limited in the types of representations that can be processed [30,31] or lead to decreases in performance [31]. Other techniques rely on models that do not learn hierarchical features [29] limiting their performance on more complex tasks.

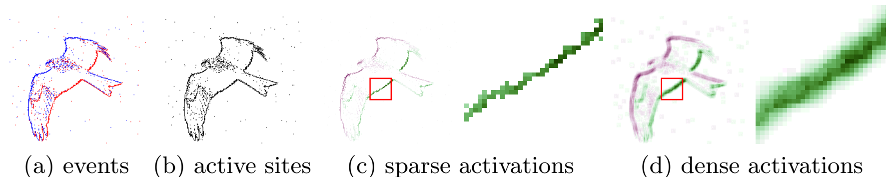
3 Method

In this section we show how to exploit the spatio-temporal sparsity of event data in classical convolutional architectures. In Sec. 3.1 we introduce the working principle of an event camera. Then, in Sec. 3.2 we show how sparse convolutional techniques, such as Submanifold Sparse Convolutional (SSC) Networks [25], can leverage this spatial sparsity. We then propose a novel technique for converting standard *synchronous networks*, into *asynchronous networks* which process events asynchronously and with low computation.

3.1 Event Data

Event cameras have independent pixels that respond to changes in the logarithmic brightness signal $L(\mathbf{u}_k, t_k) \doteq \log I(\mathbf{u}_k, t_k)$. An event is triggered at pixel $\mathbf{u}_k = (x_k, y_k)^T$ and at time t_k as soon as the brightness increment since the last event at the pixel reaches a threshold $\pm C$ (with $C > 0$):

$$L(\mathbf{u}_k, t_k) - L(\mathbf{u}_k, t_k - \Delta t_k) \geq p_k C \tag{1}$$



(a) events (b) active sites (c) sparse activations (d) dense activations

Fig. 1: Illustration of Submanifold Sparse Convolutions (SSC) [25]. A sparse event representation (a) is the input the network. SSCs work by only computing the convolution operation at active sites (b), *i.e.* sites that are non-zero, leading to sparse activation maps in the subsequent layers (c). Regular convolutions on the other hand generate blurry activation maps and therefore reduce sparsity (d).

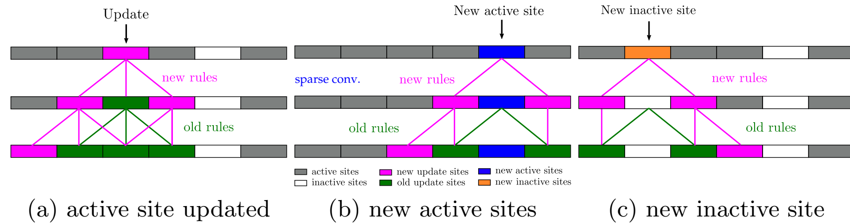
where $p_k \in \{-1, 1\}$ is the sign of the brightness change and Δt_k is the time since the last event at $\mathbf{u}$. Eq. (1) is the event generation model for an ideal sensor [4,32]. During a time interval $\Delta\tau$ an event camera produces a sequence of events, $\mathcal{E}(t_N) = \{e_k\}_{k=1}^N = \{(x_k, y_k, t_k, p_k)\}_{k=1}^N$ with microsecond resolution. Inspired by previous approaches [11,12,13,33] we generate image-like representations $H_{t_N}(\mathbf{u}, c)$ (c denotes the channel) from these sequences, that can be processed by standard CNNs. These representations retain the spatial sparsity of the events, since event cameras respond primarily to image edges, but discard their temporal sparsity. Therefore, previous works only processed them synchronously, reprocessing them from scratch every time a new event is received. This leads of course to redundant computation at the cost of latency. In our framework, we seek to recover the temporal sparsity of the event stream by focusing on the change in $H_{t_N}(\mathbf{u}, c)$ when a new event arrives:

$$H_{t_{N+1}}(\mathbf{u}, c) = H_{t_N}(\mathbf{u}, c) + \sum_i \Delta_i(c)\delta(\mathbf{u} - \mathbf{u}'_i). \tag{2}$$

This recursion can be formulated for arbitrary event representations, making our method compatible with general input representations. However, to maximize efficiency, in this work we focus on a specific class of representations which we term *sparse recursive representations* (SRR). SRRs have the property that they can be *sparsely updated* with each new event, leading to increments $\Delta_i(c)$ at only few positions $\mathbf{u}'_i$ in $H_{t_N}(\mathbf{u}, c)$. There are a number of representations which satisfy this criterion. In fact for the event histogram [11], event queue [33], and time image [34] only single pixels need to be updated for each new event.

3.2 Exploiting the Sparsity of the Event Signal

Event-cameras respond primarily to edges in the scene, which means that event representations are extremely sparse. Submanifold Sparse Convolutions (SSC) [25], illustrated in Fig. 1, leverage spatial sparsity in the data to drastically reduce computation. Hence, they are not equivalent to regular convolutions. Compared to regular convolutions, SSCs only compute the convolution at sites



(a) active site updated (b) new active sites (c) new inactive site

Fig. 2: Propagation of the rulebook $\mathcal{R}_{\mathbf{k},n}$ a 1D example. The input is composed of active (gray) and inactive (white) sites. (a) If the value of an active site changes (magenta), the update rules are incrementally added to the rulebook (lines) according to Eq. (5). (b) At newly active sites (blue) the sparse convolution is directly computed using Eq. (3) and repeated at each layer (here 1 to 3). (c) Similarly, new inactive sites (orange) are set to zero at each layer. Thus, new active sites (blue) and new inactive sites (orange) do not contribute to the rulebook propagation. Best viewed in color.

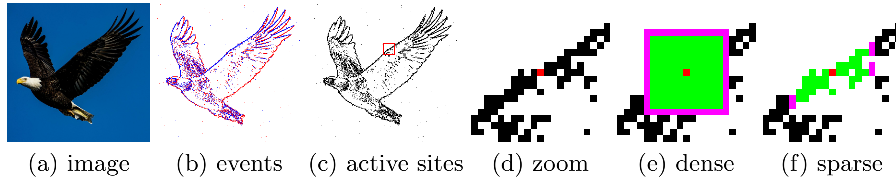
$\mathbf{u}$ with a non-zero feature vector, and ignore inputs in the receptive field of the convolution which are 0. These sites with non-zero feature vector are termed *active sites* $\mathcal{A}_t$ (Fig. 1 (b)). Fig. 1 illustrates the result of applying an SSC to sparse event data (a). The resulting activation map (c) has the same active sites as its input and therefore, by induction, all SSC layers with the same spatial resolution share the same active sites and level of sparsity. The sparse convolution operation can be written as

$$\tilde{y}^t_{n+1}(\mathbf{u}, c) = b_n(c) + \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_n} \sum_{(\mathbf{i},\mathbf{u})\in R_{t,\mathbf{k}}} W_n(\mathbf{k}, c', c) y^t_n(\mathbf{i}, c'), \quad \text{for } \mathbf{u} \in \mathcal{A}_t \tag{3}$$

$$y^t_{n+1}(\mathbf{u}, c) = \sigma(\tilde{y}^t_{n+1}(\mathbf{u}, c)). \tag{4}$$

Here $y^t_n(\mathbf{u}, c)$ is the activation of layer n at time t and is non-zero only for pixels $\mathbf{u}$, b_n denotes the bias term, W_n and $\mathbf{k} \in \mathcal{K}_n$ the parameters and indices of the convolution kernel, and σ a non-linearity. For the first layer $y^t_0(\mathbf{u}, c) = H_{t_N}(\mathbf{u}, c)$. We also make use of the rulebook $R_{t,\mathbf{k}}$ [25], a data structure which stores a list of correspondences $(\mathbf{i}, \mathbf{j})$ of input and output sites. In particular, a rule $(\mathbf{i}, \mathbf{j})$ is in $R_{t,\mathbf{k}}$ if both $\mathbf{i}, \mathbf{j} \in \mathcal{A}_t$ and $\mathbf{i} - \mathbf{j} = \mathbf{k}$, meaning that the output $\mathbf{j}$ is in the receptive field of the input $\mathbf{i}$ (Fig. 2 (a) lines). The activation at site $\mathbf{i}$ is multiplied with the weight at index $\mathbf{k}$ and added to the activation at output site $\mathbf{j}$. In (3), this summation is performed over rules which have the same output site $\mathbf{j} = \mathbf{u}$. When pooling operations such as max pooling or a strided convolution are encountered, the feature maps' spatial resolution changes and thus the rulebook needs to be recomputed. In this work we only consider max pooling. For sparse input it is the same as regular max pooling but over sites that are active. Importantly, after pooling, output sites become active when they have at least one active site in their receptive field. This operation increases the number of active sites.

Asynchronous processing While SSC networks leverage the spatial sparsity of the events, they still process event representations synchronously, performing



(a) image (b) events (c) active sites (d) zoom (e) dense (f) sparse

Fig. 3: The difference between asynchronous sparse and dense updates of events (b) is illustrated with an example image (a). The active sites, *i.e.* input sites that are non-zero, are visualized as black pixels (c). (d)-(f) show an asynchronous update (red pixel) processed with traditional convolutions (e) and our proposed asynchronous sparse convolutions (f). The receptive field of traditional convolution (e) grows quadratically with network depth leading to redundant computation. By contrast, our method (f) only updates active sites, which reduces the computational complexity. In both cases the growth frontier is indicated in magenta.

redundant computations. Indeed, for each new event, all layer activations need to be recomputed. Since events are triggered at individual pixels, activations should also be affected locally. We propose to take advantage of this fact by retaining the previous activations $\{y_n^t(\mathbf{u}, c)\}_{n=0}^N$ of the network and formulating novel, efficient update rules $y_n^t(\mathbf{u}, c) \rightarrow y_n^{t+1}(\mathbf{u}, c)$ for each new event. By employing SRRs each new event leads to sparse updates $\Delta_i(c)$ at locations $\mathbf{u}_i'$ in the input layer (Eq. (2)). We propagate these changes to deeper layers by incrementally building a *rulebook* $\mathcal{R}_{\mathbf{k},n}$ and *receptive field* $\mathcal{F}_n$ for each layer, visualized in Fig. 2. The rulebook (lines) are lists of correspondences $(\mathbf{i}, \mathbf{j})$ where $\mathbf{i}$ at the input is used to update the value at $\mathbf{j}$ in the output. The receptive field (colored sited) keeps track of the sites that have been updated by the change at the input. For sites that become newly active or inactive (Fig. 2 (b) and (c)) the active sites $\mathcal{A}_t$ are updated accordingly. At initialization (input layer) the rulebook is empty and the receptive field only comprises the updated pixel locations, caused by new events, *i.e.* $\mathcal{R}_{\mathbf{k},0} = \emptyset$ and $\mathcal{F}_0 = \{\mathbf{u}_i'\}_i$. Then, at each new layer $\mathcal{R}_{\mathbf{k},n}$ and $\mathcal{F}_n$ are expanded:

$$\mathcal{F}_n = \{\mathbf{i} - \mathbf{k} | \mathbf{i} \in \mathcal{F}_{n-1} \text{ and } \mathbf{k} \in \mathcal{K}_{n-1} \text{ if } \mathbf{i} - \mathbf{k} \in \mathcal{A}_t\} \tag{5}$$

$$\mathcal{R}_{\mathbf{k},n} = \{(\mathbf{i}, \mathbf{i} - \mathbf{k}) | \mathbf{i} \in \mathcal{F}_{n-1} \text{ if } \mathbf{i} - \mathbf{k} \in \mathcal{A}_t\}. \tag{6}$$

Rules that have a newly active or inactive site as output (Fig. 2 (b) and (c), blue or orange sites) are ignored.[3] We use Eq. (5) to formulate the update rules to layer activations from time t to $t+1$. At the input we set $y_0^t(\mathbf{u}, c) = H_{t_N}(\mathbf{u}, c)$

[3] In the supplement we present an efficient recursive method for computing $\mathcal{R}_{\mathbf{k},n}$ and $\mathcal{F}_n$ by reusing the rules and receptive field from the previous layers.

and then the update due to a single event can written as:

$$\Delta_n(\mathbf{u},c) = \sum_{\mathbf{k}\in\mathcal{K}_{n-1}} \sum_{(\mathbf{i},\mathbf{u})\in\mathcal{R}_{\mathbf{k},n}} \sum_{c'} W_{n-1}(\mathbf{k},c',c)(y^t_{n-1}(\mathbf{i},c') - y^{t-1}_{n-1}(\mathbf{i},c')) \tag{7}$$

$$\tilde{y}^t_n(\mathbf{u},c') = \tilde{y}^t_n(\mathbf{u},c') + \Delta_n(\mathbf{u},c) \tag{8}$$

$$y^t_n(\mathbf{u},c') = \sigma(\tilde{y}^t_n(\mathbf{u},c')). \tag{9}$$

Note, that these equations only consider sites which have not become active (due to a new event) or inactive. For newly active sites, we compute $y^t_n(\mathbf{u},c)$ according to Eq. (3). Finally, sites that are deactivated are set to 0, *i.e.* $y^t_n(\mathbf{u},c) = 0$. In both cases we update the active sites $\mathcal{A}_t$ before the update has been propagated. By iterating over Eqs. (7) and (5), all subsequent layers can be updated recursively. Fig. 3 illustrates the update rules above, applied to a single event update (red position) after six layers of both standard convolutions (e) and our approach (f). Note that (e) is a special case of (f) with all pixels being active sites. By using our local update rules we see that computation is confined to a small patch of the image (c). Moreover, it is visible that standard convolutions (e) process noisy or empty regions (green and magenta positions), while our approach (f) focuses computation on sites with events, leading to higher efficiency. Interestingly, we also observe that for traditional convolutions the size of the receptive field grows quadratically in depth while for our approach it grows more slowly, according on the *fractal dimension* of the underlying event data. This point will be explored further in Sec. 3.2.

Equivalence of Asynchronous and Synchronous Operation By alternating between Eqs. (5) and (7) asynchronous event-by-event updates can be propagated from the input layer to arbitrary network's depth. In the supplement we prove that processing N events by this method is equivalent to processing all events at once and present pseudocode for our method. It follows that a synchronous network, trained efficiently using back-propagation, can be deployed as an asynchronous network. Therefore, our framework does not require any change to the optimization and learning procedure. Indeed, any network architecture, after being trained, can be transformed in its asynchronous version, where it can leverage the high temporal resolution of the events at limited computational complexity. In the next section we explore this reduction in complexity in more detail.

Computational Complexity In this section we analyze the computational complexity of our approach in terms of floating point operations (FLOPs), and compare it against conventional convolution operations. In general, the number of FLOPs necessary to perform L consecutive convolutions (disregarding non-linearities for the sake of simplicity) is:

$$C_{\text{dense}} = \sum_{l=1}^{L} N(2k^2 c_{l-1} - 1)c_l \quad C_{\text{sparse}} = \sum_{l=1}^{L} N^l_r(2c_l + 1)c_{l-1} \tag{10}$$

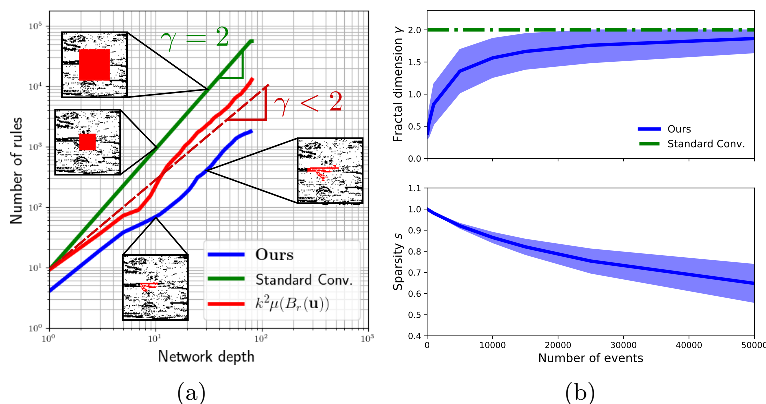


Fig. 4: To bound the complexity of our method, we use the empirical sparsity s and fractal dimension γ of event data (b). The fractal dimension characterizes the rate at which the number of update sites grows from one layer to the next. While for standard convolutions this number grows quadratically (as $(1+(k-1)n)^2$, k being the kernel size) with layer depth, with our method it grows more slowly, with an exponent $\gamma < 2$.

These formulae are explained in more detail in the supplement. For the sparse case, N_r^l counts the number of rules, *i.e.* input output pairs between layer $l-1$ and l, which corresponds exactly with the size of the rulebook in Eq. (3).

Our method minimizes the number of rules it uses at each layer by only using a subset of the rules used by SSCs and incrementally expanding it from layer to layer (Fig. 2). To characterize the computation from our approach we consider an update at a single pixel. At each layer the computation is proportional to the size of the rulebook in Eq. (5) for which we can find an upper bound. Let n_l be the number of active pixels within a patch of size $1+(k-1)l$ which is an upper bound for the number of updated sites in layer l. If we assume that each pixel can have at most k^2 rules, the number of rules at layer l is at most $n_l k^2$.

For a dense update, this number grows quadratically with the patch size $p = 1+(k-1)l$, however, for sparse updates, this number grows more slowly. To formalize this notion we define a measure $\mu(B(\mathbf{u}, r)))$ which counts the number of active sites within a patch of radius $r = \frac{p}{2}$. This measure can be used to define the *fractal dimension* of event data at pixel $\mathbf{u}$ according to [35]:

$$\gamma(\mathbf{u}) = \lim_{r\to 0} \frac{\log(\mu(B(\mathbf{u}, r)))}{\log 2r} \tag{11}$$

The fractal dimension describes an intrinsic property of the event event data, related to its dimensionality and has not been characterized for event data prior to this work. It measures the growth-rate of the number of active sites as the patch size is varied. In particular, it implies that this number grows approximately as $n_l \approx (1+(k-1)l)^\gamma$. To estimate the fractal dimension we consider the slope of $\mu(B(\mathbf{u}, r)))$ over $r = \frac{1+(k-1)l}{2}$ in the log-log domain which we visualize in Fig. 4.

Crucially, a slope $\gamma < 2$ indicates that the growth is slower than quadratic. This highlights the fact that event data exists in a submanifold of the image plane ($\gamma = 2$) which has a lower dimension than two. With this new insight we can find an upper bound for the computation using Eq. (10):

$$C_{\text{async. sparse}} \leq \sum_{l=1}^{L} c_{l-1}\,(2c_l+1)\,n_l k^2 \approx \sum_{l=1}^{L} c_{l-1}\,(2c_l+1)\,(1+(k-1)l)^{\gamma} k^2 \quad (12)$$

Where we have substituted the rulebook size at each layer. At each layer, our method performs at most $k^2 c_{l-1}(2c_l+1)\,(1+(k-1)l)^{\gamma}$ FLOPs. If we compare this with the per layer computation used by a dense network (Eq. (10)) we see that our method performs significantly less computation:

$$\frac{C^l_{\text{sparse. anync}}}{C^l_{\text{dense}}} \leq \frac{k^2(2c_l+1)c_{l-1}}{(2k^2c_{l-1}-1)c_l}\frac{(1+(k-1)l)^{\gamma}}{N} \approx \frac{(1+(k-1)l)^{\gamma}}{N} << 1 \quad (13)$$

Where we assume that $2c_l >> 1$ and $k^2 c_{l-1} >> 1$ which is the case in typical neural networks[4]. Moreover, as the fractal dimension decreases our method becomes exponentially more efficient. Through our novel asynchronous processing framework, the fractal dimension of event data can be exploited efficiently. It does so with sparse convolution, that can specifically process low-dimensional input data embedded in the image plane, such as points and lines.

4 Experiments

We validate our framework on two computer vision applications: object recognition (Sec. 4.1) and object detection (Sec. 4.2). On these tasks, we show that our framework achieves state-of-the-art results with a fraction of the computation of top-performing models. In addition, we demonstrate that our approach can be applied to different event-based representations. We select the event histogram [11] and the event queue [33] since they can be updated sparsely and asynchronously for each incoming event (see Sec. 3.2).

4.1 Object Recognition

We evaluate our method on two standard event camera datasets for object recognition: Neuromorphic-Caltech101 (N-Caltech101) [36], and N-Cars [10]. The N-Caltech101 dataset poses the task of event-based classification of 300 ms sequences of events. In total, N-Caltech101 contains 8,246 event samples, which are labelled into 101 classes. N-Cars [10] is a benchmark for car recognition. It contains 24,029 event sequences of 100 ms which contain a car or a random scene patch. To evaluate the computational efficiency and task performance we consider two metrics: prediction accuracy and number of floating point operations (FLOPs). While the first indicates the prediction quality, the second one

[4] In fact, for typical channel sizes $c_l \geq 16$ we incur a $< 3\%$ approximation error

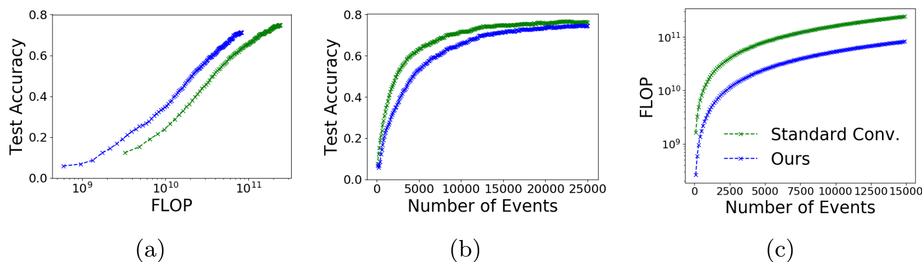


Fig. 5: Our approach requires less cumulative FLOPs w.r.t. the dense method (Standard Conv.) to produce similarly accurate predictions (a). Although dense processing requires fewer events to generate predictions of equal accuracy (b), it needs significantly more computation per event, thus having higher cumulative FLOPs (c).

measures the computational complexity of processing an input. The number of FLOPs is commonly used as a complexity metric [17,25,31], since it is independent of both the hardware and the implementation. Details of the FLOP computation are reported in the supplement. Analogously to previous work on sparse data processing [25], we use a VGG13 [37] architecture with 5 convolutional blocks and one final fully connected layer. Each block contains two convolution layers, followed by batch-norm [38], and a max pooling layer. We train the networks with the cross-entropy loss and the ADAM optimizer [39]. The initial learning rate of 10^{-4} is divided by 10 after 500 epochs.

Results In our first experiment, we compare our sparse and asynchronous processing scheme to the dense and synchronous one. For comparability, both methods share the same VGG13 architecture and the same input representation, which was generated with 25,000 events. This number was empirically found to yield a good trade-off between accuracy and computational efficiency (see supplement). We measure the approaches' computational complexity in terms of required FLOPs per single event update. Classification results shown in Tab. 1 demonstrate that our processing scheme has similar accuracy to the dense one but requires up to 19.5 times less computations per event. The low-latency of

		N-Caltech101		N-Cars	
	Representation	Accuracy ↑	MFLOP/ev ↓	Accuracy ↑	MFLOP/ev ↓
Standard Conv.	Event Histogram	**0.761**	1621	**0.945**	321
Ours	Event Histogram	0.745	**202**	0.944	**21.5**
Standard Conv.	Event Queue	0.755	2014	0.936	419
Ours	Event Queue	0.741	**202**	0.936	**21.5**

Table 1: Our approach matches the performance of the traditional dense and synchronous processing scheme at one order of magnitude less computations.

Method	1 event		100 events	
[MFLOP] ↓	Histogram	Queue	Histogram	Queue
Standard Conv.	1621	2014	1621	2014
Sparse Conv. [25]	892	967	892	967
Async. Conv. (Ours)	320	320	958	958
Async. Sparse Conv. (Ours)	**202**	**202**	**690**	**690**

Table 2: Exploiting both the sparse and asynchronous nature of the event signal provides significant reduction in computational complexity.

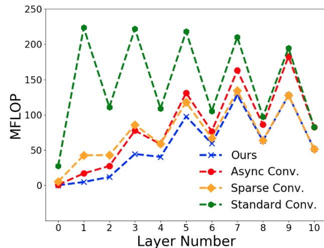


Fig. 6: Layer-wise FLOPs

the event signal allows us to make fast predictions. For this reason we compare the maximal prediction accuracy that can be achieved given a fixed computation budget for our method and a standard CNN. This experiment imitates the scenario where an object suddenly appears in the field of view, *e.g.* a pedestrian crossing the street. To do this we use samples from N-Caltech101 [36] and report the multi-class prediction accuracy (Fig. 5 (b)) and total number of FLOPs used (Fig. 5 (c)) as a function of number of events observed for our method and a standard CNN with the same architecture. For each new event we compute the FLOPs needed to generate a new prediction and its accuracy, taking into account all previously seen events. It can be seen that both standard CNN and our method have a higher prediction accuracy as the number of events increases (Fig. 5 (b)). However, compared to standard networks our method performs far less computation (Fig. 5 (c)). This is because for standard networks all activations need to be recomputed for each new event, while our method retains the state and therefore only needs to perform sparse updates. As a result, our method achieves up to 14.8% better accuracy at the same FLOP budget (Fig. 5 (a)), thus improving the prediction latency. Moreover, to show the flexibility of our approach to different update rates, we initialize a representation with $25,000$ events and update it either for each new event or for a batch of 100 new events. For comparability, all methods share the same VGG13 architecture and input representation. The results of this experiment are reported in Tab. 2. For both single and batch event update, exploiting the asynchronous and sparse nature of the signal offers significant computational benefits. Our approach (*Async. Sparse Conv.*) performs on average 8.9 times less computation than standard convolution for one event update, and of 2.60 times for 100 events update. We additionally evaluate the computation per layer of each of previous models in the case of one event update. Fig. 6, which presents the results of this evaluation, shows that sparse and asynchronous processing saves the majority of computations in the initial layers of the network. Indeed, the input representations of the event stream are spatially very sparse (see Fig. 4(b)), and are only locally updated for each new event.

Comparison with State-of-the-Art Methods We finally compare our approach with state-of-the-art methods for event-based object recognition. We

Methods	Async.	N-Caltech101 Accuracy ↑	N-Caltech101 MFLOP/ev ↓	N-Cars Accuracy ↑	N-Cars MFLOP/ev ↓
H-First [2]	✔	0.054	-	0.561	-
HOTS [21]	✔	0.210	54.0	0.624	14.0
HATS [10]	✔	0.642	**4.3**	0.902	**0.03**
DART [40]	✔	0.664	-	-	-
YOLE [30]	✔	0.702	3659	0.927	328.16
EST [12]	✗	**0.817**	4150	0.925	1050
SSC [25]	✗	0.761	1621	**0.945**	321
Ours	✔	**0.745**	202	**0.944**	21.5

Table 3: Comparison with asynchronous and dense methods for object recognition.

consider models that, like ours, perform efficient per-event updates due to a light-weight computational model (HATS [10], HOTS [21], DART [40]), a spiking neural network (H-First [2]), or asynchronous processing (YOLE [30]). The results for this evaluation are presented in Tab. 3. Our method outperforms the state-of-the-art (YOLE) by 4.3% in accuracy on N-Caltech101 and 1.7% on N-Cars at only 6% (on average over datasets) of its computational cost. Finally, we compare against the synchronous state-of-the-art method [12]. Our method achieves a slightly higher accuracy on N-Cars using only 21.5 MFLOPs (47 times reduction). Similarly, our method using the asynchronous processing requires 20 times fewer FLOPs on N-Caltech101 but at the cost of lower accuracy. In addition to the performance evaluation, we timed our experiments conducted on N-Caltech101 by measuring the processing time for a single event on a i7-6900K CPU. In our framework implemented in C++ and Python, our method requires 80.4 ms, while the standard dense CNN needs 202 ms. Therefore, our approach becomes roughly 2.75 times faster by leveraging the sparsity. However, in the highly-optimized framework PyTorch [41], a dense inference takes only 23.4 ms. Given its lower number of FLOPs, we expect that our method will experience a significant run-time reduction with further optimizations and specific hardware.

4.2 Object Detection

Object detection is the task of regressing a bounding box and class probabilities for each object in the image. We evaluate our method on two standard benchmarks for event-based object detection: N-Caltech101 [36] and Gen1 Automotive [42]. While the former contains the N-Caltech101 samples each with a single bounding box, the latter contains 228,123 bounding boxes for cars and 27,658 for pedestrians collected in an automotive scenario. For this task we combine the first convolutional blocks of the object recognition task with the YOLO output layer [43]. The resulting feature maps are processed by the YOLO output layer to generate bounding-boxes and class predictions. As is standard, we report the performance using the mean average precision (mAP) metric [44] using the implementation of [45]. As for the recognition task, we measure computational complexity with FLOPs.

	Representation	N-Caltech101 mAP ↑	N-Caltech101 MFLOP ↓	Gen1 Automotive mAP ↑	Gen1 Automotive MFLOP ↓
YOLE [30]	Leaky Surface	0.398	3682	-	-
Standard Conv.	Event Queue	0.619	1977	**0.149**	2614
Ours	Event Queue	0.615	**200**	0.119	**205**
Standard Conv.	Event Histogram	0.623	1584	0.145	2098
Ours	Event Histogram	**0.643**	**200**	0.129	**205**

Table 4: Accuracies for object detection.

Results and Comparison with State-of-the-Art Tab. 4 shows quantitative results on object detection while Fig. 1 in the supplement illustrates qualitative results. Our approach achieves comparable or superior performance with respect to standard networks at a fraction of the computational cost. Specifically, for the histogram representation our method outperforms dense methods by 2.0% on N-Caltech101. On the Gen1 Automotive dataset, we experience a slight performance drop of 1.8%. The slight performance improvement in N-Caltech101 is probably thanks to the sparse convolution, which give less weight to noisy events. In terms of computation, our method reduces the number of FLOPs per event by a factor of 10.6 with respect to the dense approach, averaged over all datasets and representations. Tab. 4 also compares our approach to the state-of-the-art method for event-based detection, YOLE [30]. Compared to this baseline, we achieve 24.5% higher accuracy at 5% of the computational costs.

5 Discussion

In the quest of high prediction accuracy, event-based vision algorithms have relied heavily on processing events in synchronous batches using deep neural networks. However, this trend has disregarded the sparse and asynchronous nature of event-data. Our work brings the genuine spatio-temporal sparsity of events back into high-performance CNNs, by significantly decreasing their computational complexity (up to 20 times). By doing this, we achieve up to 15% better accuracy than state-of-the-art synchronous models at the same computational (FLOP) budget. In addition, we outperform existing asynchronous approaches by up to 24.5% in accuracy. Our work highlights the importance for researchers to take into account the intrinsic properties of event data in the pursuit of low-latency and high-accuracy event vision algorithms. Such considerations will open the door to unlock the advantages of event cameras on all tasks that rely on low-latency visual information processing.

6 Acknowledgements

This work was supported by the Swiss National Center of Competence Research Robotics (NCCR), through the Swiss National Science Foundation, and the SNSF-ERC starting grant.

References

1. G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *arXiv e-prints*, vol. abs/1904.08405, 2019. [Online]. Available: http://arxiv.org/abs/1904.08405
2. G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, "HFirst: A temporal approach to object recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 10, pp. 2028–2040, 2015.
3. X. Lagorce, G. Orchard, F. Gallupi, B. E. Shi, and R. Benosman, "HOTS: A hierarchy of event-based time-surfaces for pattern recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 7, pp. 1346–1359, Jul. 2017.
4. G. Gallego, J. E. A. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza, "Event-based, 6-DOF camera tracking from photometric depth maps," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 10, pp. 2402–2412, Oct. 2018.
5. H. Kim, S. Leutenegger, and A. J. Davison, "Real-time 3D reconstruction and 6-DoF tracking with an event camera," in *Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 349–364.
6. J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Front. Neurosci.*, vol. 10, p. 508, 2016.
7. M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 525–542.
8. J. A. Perez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward ConvNets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2706–2719, Nov. 2013.
9. A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. D. Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, J. Kusnitz, M. Debole, S. Esser, T. Delbruck, M. Flickner, and D. Modha, "A low power, fully event-based gesture recognition system," in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2017, pp. 7388–7397.
10. A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, "HATS: Histograms of averaged time surfaces for robust event-based object classification," in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2018, pp. 1731–1740.
11. A. I. Maqueda, A. Loquercio, G. Gallego, N. García, and D. Scaramuzza, "Event-based vision meets deep learning on steering prediction for self-driving cars," in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2018, pp. 5419–5427.
12. D. Gehrig, A. Loquercio, K. G. Derpanis, and D. Scaramuzza, "End-to-end learning of representations for asynchronous event-based data," in *Int. Conf. Comput. Vis. (ICCV)*, 2019.
13. A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "EV-FlowNet: Self-supervised optical flow estimation for event-based cameras," in *Robotics: Science and Systems (RSS)*, 2018.
14. H. Rebecq, T. Horstschäfer, G. Gallego, and D. Scaramuzza, "EVO: A geometric approach to event-based 6-DOF parallel tracking and mapping in real-time," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 593–600, 2017.

15. H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, “Events-to-video: Bringing modern computer vision to event cameras,” in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2019.
16. Y. Xu, H. Ji, and C. Fermüller, “Viewpoint invariant texture description using fractal analysis,” *International Journal of Computer Vision*, vol. 83, no. 1, pp. 85–100, 2009.
17. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2016, pp. 770–778.
18. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2016.
19. R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, “NetVLAD: CNN architecture for weakly supervised place recognition,” in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2016, pp. 5297–5307.
20. G. Orchard, R. Benosman, R. Etienne-Cummings, and N. V. Thakor, “A spiking neural network architecture for visual motion estimation,” in *IEEE Biomed. Circuits Syst. Conf. (BioCAS)*, 2013, pp. 298–301.
21. X. Lagorce, G. Orchard, F. Gallupi, B. E. Shi, and R. Benosman, “HOTS: A hierarchy of event-based time-surfaces for pattern recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 7, pp. 1346–1359, Jul. 2017.
22. D. Neil, M. Pfeiffer, and S.-C. Liu, “Phased LSTM: Accelerating recurrent network training for long or event-based sequences,” in *Conf. Neural Inf. Process. Syst. (NIPS)*, 2016.
23. A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, “Unsupervised event-based learning of optical flow, depth, and egomotion,” in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2019.
24. H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, “High speed and high dynamic range video with an event camera,” *IEEE Trans. Pattern Anal. Mach. Intell.*, 2019.
25. B. Graham, M. Engelcke, and L. van der Maaten, “3D semantic segmentation with submanifold sparse convolutional networks,” in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2018, pp. 9224–9232.
26. A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S. Liu, and T. Delbruck, “Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 644–656, 2019.
27. Y. Sekikawa, K. Ishikawa, K. Hara, Y. Yoshida, K. Suzuki, I. Sato, and H. Saito, “Constant velocity 3d convolution,” in *International Conference on 3D Vision (3DV)*, 2018, pp. 343–351.
28. C. Scheerlinck, N. Barnes, and R. Mahony, “Asynchronous spatial image convolutions for event cameras,” *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 816–822, Apr. 2019.
29. Y. Sekikawa, K. Hara, and H. Saito, “EventNet: Asynchronous recursive event processing,” in *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2019.
30. M. Cannici, M. Ciccone, A. Romanoni, and M. Matteucci, “Asynchronous convolutional networks for object detection in neuromorphic cameras,” in *IEEE Conf. Comput. Vis. Pattern Recog. Workshops (CVPRW)*, 2019.
31. 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,” *Front. Neurosci.*, vol. 11, p. 682, 2017.

32. G. Gallego, C. Forster, E. Mueggler, and D. Scaramuzza, "Event-based camera pose tracking using a generative event model," 2015, arXiv:1510.01972.
33. S. Tulyakov, F. Fleuret, M. Kiefel, P. Gehler, and M. Hirsch, "Learning an event sequence embedding for dense event-based deep stereo," in *Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019.
34. A. Mitrokhin, C. Fermuller, C. Parameshwara, and Y. Aloimonos, "Event-based moving object detection and tracking," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.
35. Y. Xu, H. Ji, and C. Fermüller, "Viewpoint invariant texture description using fractal analysis," *Int. J. Comput. Vis.*, vol. 83, no. 1, pp. 85–100, 2009.
36. G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Front. Neurosci.*, vol. 9, p. 437, 2015.
37. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Int. Conf. Learn. Representations (ICLR)*, 2015.
38. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learning (ICML)*, 2015.
39. D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *Int. Conf. Learn. Representations (ICLR)*, 2015.
40. B. Ramesh, H. Yang, G. Orchard, N. A. L. Thi, and C. Xiang, "DART: distribution aware retinal transform for event-based cameras," *arXiv e-prints*, Oct. 2017. [Online]. Available: http://arxiv.org/abs/1710.10800
41. A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Workshops*, 2017.
42. P. de Tournemire, D. Nitti, E. Perot, D. Migliore, and A. Sironi, "A large scale event-based detection dataset for automotive," *ArXiv*, vol. abs/2001.08499, 2020.
43. J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.
44. M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–338, 06 2010.
45. R. Padilla, S. L. Netto, and E. A. B. da Silva, "Survey on performance metrics for object-detection algorithms," *International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020.

7 Supplementary Material

In the supplementary material, references which point to the main manuscript will be referenced with a leading "M-". In Sec. 7.1 we describe an efficient recursive method for computing the rulebook $\mathcal{R}_{\mathbf{k},n}$ in Eq. (M-7) and present the asynchronous propagation of changes through events in algorithmic form in Tab. 1. In Sec. 7.2 we present a proof of the equivalence of network outputs using asynchronous and synchronous networks. In Sec. 7.3 we present additional details about the input representations and FLOP calculations used in the experiments in Sec. M-4. In Sec. 7.4 we present a sensitivity analysis where we vary the number of events used for training and justify our choice of 25'000 events for all experiments. Finally, in Sec. 7.5 we show additional qualitative object detection results.

7.1 Efficient Rulebook Update

At each layer the rulebook $\mathcal{R}_{\mathbf{k},n}$ and receptive field $\mathcal{F}_n$ are

$$\begin{aligned}\mathcal{F}_n &= \{\mathbf{i}-\mathbf{k} | \mathbf{i} \in \mathcal{F}_{n-1} \text{ and } \mathbf{k} \in \mathcal{K}_{n-1} \text{ if } \mathbf{i}-\mathbf{k} \in \mathcal{A}_t\} \\ \mathcal{R}_{\mathbf{k},n} &= \{(\mathbf{i}, \mathbf{i}-\mathbf{k}) | \mathbf{i} \in \mathcal{F}_{n-1} \text{ if } \mathbf{i}-\mathbf{k} \in \mathcal{A}_t\}.\end{aligned}$$

At the input these are initialized as $R_{\mathbf{k},0} = \emptyset$ and $\mathcal{F}_0 = \{\mathbf{u}'_i\}$. The propagation of these two data structures is illustrated in Fig. M-2. We observe that at each layer the rulebook and receptive field can be computed by reusing the data from the previous layer. We can do this by decomposing the receptive field into a frontier set f_n (Fig. M-2 (a) magenta sites)and visited state set F_n (Fig. M-2 (a) green sites). At each layer $\mathcal{F}_n = f_n \cup F_n$. To efficiently update both $\mathcal{F}_n$ and $\mathcal{R}_{\mathbf{k},n}$ at each layer we only consider the rules that are added due to inputs in the frontier set (Fig. M-2 (a), magenta lines). These can be appended to the existing rulebook. In addition, the receptive field $\mathcal{F}_n$ can be updated similarly, by adding new update sites reached from the frontier set. This greatly reduces the sites that need to be considered in the computation of $\mathcal{R}_{\mathbf{k},n}$ and $\mathcal{F}_n$ in Eqs. (M-6) and (M-7).

7.2 Equivalence of Synchronous and Asynchronous Updates

We start with Eq. (M-4), which we repeat here:

$$\begin{aligned}\tilde{y}^t_{n+1}(\mathbf{u}, c) &= \begin{cases} b_n(c) + \sum_{c'} \sum_{\mathbf{k} \in \mathcal{K}_n} \sum_{(\mathbf{i},\mathbf{u}) \in R_{t,\mathbf{k}}} W_n(\mathbf{k}, c', c) y^t_n(\mathbf{i}, c'), & \text{for } \mathbf{u} \in \mathcal{A}_t \\ 0 & \text{else} \end{cases} \\ y^t_{n+1} &= \sigma(\tilde{y}^t_{n+1})\end{aligned}$$

Here the input layer is $y^t_0(\mathbf{u}, c) = H_{t_N}(\mathbf{u}, c)$. In a next step we assume changes to the input layer as in Eq. (M-3). These changes occur at sites $\mathbf{u}_i \in \mathcal{F}_0$ with

magnitude $\Delta_i(c) = y_0^{t+1}(\mathbf{u}_i, c) - y_0^t(\mathbf{u}_i, c)$. The sites $\mathbf{u}_i$ can be categorized into three groups: sites that are and remain active, sites that become inactive (feature becomes 0) and sites that become active. We will now consider how $y_n^{t+1}(\mathbf{u}, c)$ evolves:

$$\tilde{y}_1^{t+1}(\mathbf{u}, c) = b_0(c) + \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in R_{t+1,\mathbf{k}}} W_n(\mathbf{k}, c', c) y_0^{t+1}(\mathbf{i}, c') \tag{14}$$

$$= b_0(c) + \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in R_{t+1,\mathbf{k}}} W_0(\mathbf{k}, c', c) \left(y_0^t(\mathbf{i}, c') + \Delta(\mathbf{i}, c')\right) \tag{15}$$

$$\tag{16}$$

Here we define the increment $\Delta_0(\mathbf{u}, c)$. This increment is only non-zero for sites at which the input $y_0^t(\mathbf{i}, c)$ changed, so for $(\mathbf{i}, \mathbf{u}) \in R_{t+1,\mathbf{k}}$ such that $\mathbf{i} \in \mathcal{F}_0$. At time $t + 1$ the rulebook $R_{t+1,\mathbf{k}}$ is modified for every site $\mathbf{u}_j$ that becomes newly active:

$$R_{t+1,\mathbf{k}} = R_{t,\mathbf{k}} \cup \bigcup_j \{(\mathbf{u}_j + \mathbf{k}, \mathbf{u}_j) | \mathbf{u}_j + \mathbf{k} \in \mathcal{A}_{t+1}\} \cup \{(\mathbf{u}_j, \mathbf{u}_j - \mathbf{k}) | \mathbf{u}_j - \mathbf{k} \in \mathcal{A}_t\}$$

and every site u_l that becomes inactive

$$R_{t+1,\mathbf{k}} = R_{t,\mathbf{k}} \backslash \bigcup_l \{(\mathbf{u}_l + \mathbf{k}, \mathbf{u}_l) | \mathbf{u}_l + \mathbf{k} \in \mathcal{A}_t\} \cup \{(\mathbf{u}_l, \mathbf{u}_l - \mathbf{k}) | \mathbf{u}_l - \mathbf{k} \in \mathcal{A}_{t+1}\}$$

For both newly active and newly inactive site we may ignore the first term in the union since these correspond to rules that influence the output sites $\mathbf{u}_j$ and $\mathbf{u}_l$. In Fig. M-2 (b) and (c) these rules would correspond to lines leading from input sites (top layer) to the newly active (blue) or newly inactive (white) sites. However, the outputs at these sites can be computed using Eq. (M-4) for newly active sites and simply set to 0 for newly inactive sites, and so we ignore them in updating the next layer. What remains are the contributions of the second term in the union which correspond to the magenta lines in the top layer of Fig. M-2 (b) and (c), which we define as $r_{\mathbf{k},\text{act}}$ and $r_{\mathbf{k},\text{inact}}$ respectively.

If we restrict the output sites $\mathbf{u}$ to be sites that remain active, we may expand Eq. (14) as:

$$\begin{aligned}
\tilde{y}_1^{t+1}(\mathbf{u}, c) =& b_0(c) + \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in R_{t+1,\mathbf{k}}} W_0(\mathbf{k}, c', c) \left(y_0^t(\mathbf{i}, c') + \Delta(\mathbf{i}, c')\right) \\
=& b_0(c) + \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in R_{t,\mathbf{k}}} W_0(\mathbf{k}, c', c) \left(y_0^t(\mathbf{i}, c') + \Delta(\mathbf{i}, c')\right) \\
&- \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in r_{\mathbf{k},\text{inact}}} W_0(\mathbf{k}, c', c) \underbrace{\left(y_0^t(\mathbf{i}, c') + \Delta(\mathbf{i}, c')\right)}_{=0 \text{ for } \mathbf{i}=\mathbf{u}_l} \\
&+ \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in r_{\mathbf{k},\text{act}}} W_0(\mathbf{k}, c', c) (\underbrace{y_0^t(\mathbf{i}, c')}_{=0 \text{ for } \mathbf{i}=\mathbf{u}_j} + \Delta(\mathbf{i}, c'))
\end{aligned}$$

Where we have used the fact that at newly inactive sites $y_0^t(\mathbf{i}, c') + \Delta(\mathbf{i}, c') = 0$ and at newly active sites $y_0^t(\mathbf{i}, c') = 0$. This can be simplified as:

$$\begin{aligned}
\tilde{y}_1^{t+1}(\mathbf{u}, c) =& b_0(c) + \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in R_{t,\mathbf{k}}} W_0(\mathbf{k}, c', c) \left(y_0^t(\mathbf{i}, c') + \Delta(\mathbf{i}, c')\right) \\
&+ \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in r_{\mathbf{k},\text{act}}} W_0(\mathbf{k}, c', c) \Delta(\mathbf{i}, c') \\
=& \underbrace{b_0(c) + \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in R_{t,\mathbf{k}}} W_0(\mathbf{k}, c', c) y_0^t(\mathbf{i}, c')}_{y_1^t(\mathbf{u}, c)} \\
&+ \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in R_{t,\mathbf{k}}} W_0(\mathbf{k}, c', c) \Delta_0(\mathbf{i}, c') \\
&+ \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in r_{\mathbf{k},\text{act}}} W_0(\mathbf{k}, c', c) \Delta_0(\mathbf{i}, c') \\
=& y_1^t(\mathbf{u}, c) + \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in R_{t,\mathbf{k}} \cup r_{\mathbf{k},\text{act}}} W_0(\mathbf{k}, c', c) \Delta_0(\mathbf{i}, c')
\end{aligned}$$

It remains to find the rules $(\mathbf{i}, \mathbf{u})$ in $R_{t,\mathbf{k}} \cup r_{\mathbf{k},\text{act}}$ which have a non-zero contribution to the sum, *i.e.* for which $\Delta_0(\mathbf{i}, c')$ is non-zero. The increment is exactly non-zero for $\mathbf{i} \in \mathcal{F}_0$, corresponding to the input site affected by the new event. Note that this site could either *(i)* remain active (input for rule in $R_{t,\mathbf{k}}$), *(ii)* become inactive (input for rule in $R_{t,\mathbf{k}}$) or *(iii)* become active (input for rule in $r_{\mathbf{k},\text{act}}$)). Therefore, the rules that have a non-zero contribution are the ones drawn as magenta lines in the top row of Fig. M-2 (a), (b) and (c) respectively, where we ignore rules with newly active or inactive sites output sites. These rules also correspond exactly to $\mathcal{R}_{\mathbf{k},1}$ defined in Eq. M-7. The resulting update equation becomes:

$$\tilde{y}_1^{t+1}(\mathbf{u}, c) = y_1^t(\mathbf{u}, c) + \sum_{c'} \sum_{\mathbf{k}\in\mathcal{K}_0} \sum_{(\mathbf{i},\mathbf{u})\in\mathcal{R}_{\mathbf{k},1}} W_0(\mathbf{k}, c', c) \Delta_0(\mathbf{i}, c')$$

This is exactly Eq. (M-9). By applying the non-linearity we arrive at Eq. (M-10).

Now let us consider how the update propagates to the next layer. For this we need to find $\mathcal{F}_1$, *i.e.* the input sites of layer 1 that change. These are exactly the updated output sites of layer 0. Every $\mathbf{i} \in \mathcal{F}_0$ affects the output site $\mathbf{u}$ for which $(\mathbf{i}, \mathbf{u}) \in \mathcal{R}_{\mathbf{k},1}$. To be part of this rulebook $\mathbf{u} = \mathbf{i} - \mathbf{k}$ and so we see that $\mathbf{i} - \mathbf{k} \in \mathcal{F}_1$ for all $\mathbf{k} \in \mathcal{K}_0$, which is exactly mirrored by Eq. (M-7).

To propagate updates through layer 1 we thus repeat the steps up until now, but only consider changes at sites $\mathcal{F}_1$ instead of $\mathcal{F}_0$. By iterating this procedure, all layers of the network can be updated. This concludes the proof.

7.3 Representations and FLOP computation

Representations As the proposed asynchronous framework does not require any specific input representation, we evaluate two event embeddings, which are

sparse in time and space. The two event representations tested for both tasks are the event histogram [11] and the event queue [33]. The former creates a two-channel image by binning events with positive polarity to the first channel and events with negative polarity to the second channel. This histogram is created for a sliding window containing a constant number of events. Therefore, if an event enters or leaves the sliding window, an update site is created and propagated through the network. The second representation called event queue [33] is applied in a sliding window fashion as well. The event queue stores the timestamps and polarities of the incoming events in a queue at the corresponding image locations. The queues have a fixed length of 15 entries and are initialised with zeros. If a queue is full, the oldest event is discarded. The four dimensional tensor containing the timestamps and polarities of up to 15 events is reshaped to a three dimensional tensor by concatenating the timestamps and polarities along the 15 entries. The resulting three dimensional tensor has two spatial dimensions and a channel dimension with 30 entries.

FLOP computation Tab. 5 shows the number of FLOPs to perform different operation in the network for standard networks and our method. The FLOPs

	Dense Layer	Sparse Layer
Convolution	$H_{\text{out}}W_{\text{out}}c_{\text{out}}(2k^2c_{\text{in}}-1)$	$N_rc_{\text{in}}(2c_{\text{out}}+1)$
Max Pooling	$H_{\text{out}}W_{\text{out}}c_{\text{out}}k^2$	$N_ac_{\text{out}}k^2$
Fully Connected	$2c_{\text{in}}c_{\text{out}}$	$2c_{\text{in}}c_{\text{out}}$
ReLU	$H_{\text{out}}W_{\text{out}}c_{\text{in}}$	N_ac_{in}

Table 5: FLOPs computation at each layer. Here N_r are the number of rules at that layer and N_a are the number of active sites.

needed for a standard convolution is $H_{\text{out}}W_{\text{out}}c_{\text{out}}(2k^2c_{\text{in}}-1)$ excluding bias. This is the result of performing k^2c_{in} multiplications and $k^2c_{\text{in}}-1$ additions for each pixel and each output channel resulting in the term found in the table. For our asynchronous sparse formulation we compute the number of operations by following Eqs. (M-8) and (M-9). Computing the differences in Eq. (M-8) results in N_rc_{in} operations, where N_r are the number of rules at that layer. The convolution itself uses N_rc_{in} multiplications and $N_r(c_{\text{in}}-1)$ additions for each output channel, resulting in a total of $c_{\text{out}}N_r(2c_{\text{in}}-1)$ operations. Finally, from Eq. (M-9) additional N_rc_{out} operations need to be expended to add these increments to the previous state. In total, this results in $N_rc_{\text{in}}(2c_{\text{out}}+1)$ operations.

7.4 Sensitivity on the Number of Events

Tab. 6 shows the computational complexity in MFLOPS and test accuracy on N-Caltech101[36] for both sparse and dense VGG13. The table shows that the

test accuracy is maximized at 25’000 events for the sparse network and reaches a plateau for the dense network. At this number of events amount of computation in the sparse network is 46% lower than for the dense network. For this reason we selected 25’000 events for all our further experiments in the main manuscript.

	64		256		5000		25000		50000	
	Accuracy	MFLOP	Accuracy	MFLOP	Accuracy	MFLOP	Accuracy	MFLOP	Accuracy	MFLOP
Dense VGG13	0.257	1621.2	0.456	1621.2	0.745	1621.2	0.761	1621.2	0.766	1621.2
Sparse VGG13	0.247	224.2	0.435	381.0	0.734	697.5	0.745	884.2	0.730	959.2

Table 6: Computational complexity and test accuracy on N-Caltech101[36] for sparse and dense VGG13 and a varying number of events.

7.5 Qualitative Results on Object Detection

Here we show qualitative results of our method applied to the task of event-based object detection (Sec. M-4.2 and Tab. 4 in the main manuscript). Failure cases of our approach include very similar classes, such as ”rooster” and ”pigeon” in the third column. In the Gen1 Automotive dataset it can be seen that our approach works well for cars that are close and have a high relative motion. However, some cars are missed, especially if they have small relative speed and thereby only trigger few events (Fig. 7, bottom right).

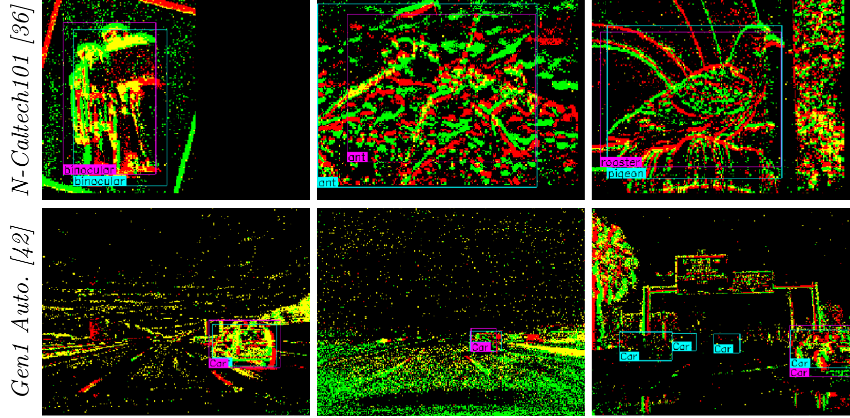


Fig. 7: Qualitative results of object detection (best viewed in color). Our predictions are shown in magenta, and labels in cyan. The first two columns present success cases, while the last column a failure case. On the first dataset, our method is mainly fooled by similar classes, such as "pigeon" and "rooster". In the second dataset, our approach detects cars generally well, but fails to detect the ones moving at similar speed due to the low event rate (bottom right).

Algorithm 1 Asynchronous Sparse Convolution at layer n

Update Active Sites
if the first layer ($n = 1$) **then**
- Update the active set $\mathcal{A}_t$ with all new active and new inactive sites
- Initialize the rulebook $\mathcal{R}_{\mathbf{k},0} = \emptyset$ and receptive field $\mathcal{F}_0 = \{\mathbf{u}_i\}_i$.

end if

Update rulebook and receptive field
- compute $\mathcal{R}_{\mathbf{k},n}$ using Eq. (M-7) with $\mathcal{F}_{n-1}$
- compute $\mathcal{F}_n$ using Eq. (M-6) with $\mathcal{F}_{n-1}$.

Layer update
for all $\mathbf{u}$ in $\mathcal{F}_n$ **do**
  if $\mathbf{u}$ remains an active site **then**
    - compute increment Δ_n using Eq. (M-8) with y^t_{n-1} and y^{t-1}_{n-1}
    - compute activation $\tilde{y}^t_n$ using Eq. (M-9) with Δ_n and $\tilde{y}^{t-1}_n$

  end if
  if $\mathbf{u}$ is newly active **then**
    - compute activation $\tilde{y}^t_n$ using Eq. (M-4) with y^t_{n-1}

  end if
  if $\mathbf{u}$ is newly inactive **then**
    - set activation $\tilde{y}^t_n$ to 0

  end if
  - compute y^t_n by applying a non-linearity as in Eq. (M-10)

end for