FARSE-CNN: Fully Asynchronous, Recurrent and Sparse Event-Based CNN

Riccardo Santambrogio[1], Marco Cannici[2], and Matteo Matteucci[1]

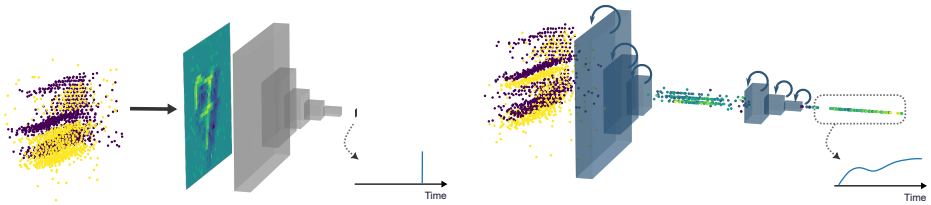
[1] Politecnico di Milano, Milan, Italy
[2] University of Zurich, Zurich, Switzerland

Abstract. Event cameras are neuromorphic image sensors that respond to per-pixel brightness changes, producing a stream of asynchronous and spatially sparse events. Currently, the most successful algorithms for event cameras convert batches of events into dense image-like representations that are synchronously processed by deep learning models of frame-based computer vision. These methods discard the inherent properties of events, leading to high latency and computational costs. Following a recent line of works, we propose a model for efficient asynchronous event processing that exploits sparsity. We design the Fully Asynchronous, Recurrent and Sparse Event-Based CNN (FARSE-CNN), a novel multi-layered architecture which combines the mechanisms of recurrent and convolutional neural networks. To build efficient deep networks, we propose compression modules that allow to learn hierarchical features both in space and time. We theoretically derive the complexity of all components in our architecture, and experimentally validate our method on tasks for object recognition, object detection and gesture recognition. FARSE-CNN achieves similar or better performance than the state-of-the-art among asynchronous methods, with low computational complexity and without relying on a fixed-length history of events. Our code is released at `https://github.com/AIRLab-POLIMI/farse-cnn`.

Keywords: Event cameras · Deep learning architectures

1 Introduction

Event cameras, or neuromorphic cameras, are image sensors inspired by the biological retina [22,31]. While conventional cameras capture snapshots at a predetermined frame rate, event cameras have independent pixels that asynchronously respond to changes of brightness in the scene. When the change crosses a certain threshold, the pixel triggers an event encoding the sign of the measured variation, as in brightness increase or brightness decrease. Thus, an event camera generates a stream of events in space and time. Compared to conventional cameras, event cameras have many advantages that bring them closer to biological vision. Since pixels monitor only brightness changes independently, and they do not have to wait for a global exposure time, events are generated with very high temporal resolution and transmitted by the event camera with low latency,



(a) Frame-based representation processed by a CNN

(b) FARSE-CNN

Fig. 1: Overview of FARSE-CNN in contrast to frame-based event processing. (a) A common approach builds dense event frames that are synchronously processed by a CNN. (b) Convolutional-recurrent modules in our architecture, instead, process events directly preserving their spatio-temporal sparsity, and the overall network produces an asynchronous stream of outputs.

both in the order of microseconds. Additionally, since the photoreceptors work in logarithmic scale, event cameras have a high dynamic range, which exceeds 120 dB. Because event cameras produce data only for pixels where a change is measured, and only at times in which the change occurs, they naturally suppress redundant visual information and the bandwidth is greatly reduced, leading to low power consumption. These properties make event cameras ideal for various applications, such as robotics, autonomous vehicles, IoT and edge applications.

To take advantage of these sensors, however, suitable processing techniques and algorithms must be identified. The deep learning methods that are commonly used in conventional computer vision, specifically Convolutional Neural Networks (CNN), cannot be directly applied to the output of event cameras, being it inherently sparse and asynchronous. For this reason, researchers have focused on ways to adapt the event data to the requirements of CNNs, by converting batches of events into synchronous and dense, image-like representations. Despite being effective, this approach discards the inherent properties of event data, and trades off latency and computational costs for task performance. Following a recent trend in event-based vision, we address the problem of developing deep learning techniques that exploit the asynchronous and sparse nature of event data, to allow for efficient event-by-event processing. Previous works have adopted solutions based on 3D point-cloud processing [37], graph neural networks [21, 36], sparse convolutions [25], and memory-augmented networks [17]. Inspired by techniques that combine the mechanism of 2D convolution with Recurrent Neural Networks (RNN) [4, 38], we propose the Fully Asynchronous, Recurrent and Sparse Event-Based CNN (FARSE-CNN), a novel deep architecture that uses recurrent units in a convolutional way to process events in a fully sparse and asynchronous fashion. In order to do this, we design an asynchronous convolutional RNN model based on the Long-Short-Term-Memory (LSTM) [15] and a submanifold convolutional RNN variant inspired by works on sparse convolutions [13], as well as modules for both spatial and temporal compression of event features. We show that our architecture can be applied to different tasks

in event-based vision, achieving performance and computational costs that are competitive to other state-of-the-art methods. Our contributions are summarized as follows:

- We propose FARSE-CNN, a convolutional RNN architecture tailored for asynchronous and sparse event processing
- We propose a novel temporal compression mechanism, in the form of a new Temporal Dropout module, as well as sparse implementations of pooling operations. Our Temporal Dropout introduces hierarchical temporal learning in our models. To the best of our knowledge, this is the first work to explore Hierarchical RNNs (HRNNs) in event vision.
- We derive theoretical bounds that allow to analyze the computational complexity of all elements in our architecture, in terms of floating point operations per event (FLOPs/ev).
- We experimentally demonstrate our method on publicly available datasets for object recognition, gesture recognition and object detection. Our method shows similar performance to the state of the art for asynchronous techniques in object recognition, and improves on it in object detection.

2 Related Work

Motivated by the success of deep learning architectures in frame-based computer vision, a number of works in event vision have focused on converting batches of events into dense representations which can be processed using convolutional models. These methods range from the construction of simple hand-crafted surfaces and volumes [33, 44, 45], to more complex data-driven techniques [4, 10]. This approach achieves state-of-the-art performance, but it discards the sparsity of events, introducing redundant computation. Furthermore, asynchronous event-by-event processing becomes impractical, since all activations of the network would have to be recomputed for each new event. Other architectures are better suited to deal with asynchronous and sparse inputs, like Spiking Neural Networks (SNNs) [24,27], which are also biologically inspired and hence naturally work with events. Nevertheless SNNs are at a relatively early stage, they still lack a general universal training procedure, and their performance on complex tasks remains limited [2, 20, 35].

For these reasons, researchers have recently developed methods that leverage the asynchronous and sparse nature of events using traditional deep learning techniques, to favour low latency and lightweight processing while seeking to retain the performance of deep models. Methods based on geometric deep learning have been proposed to model the spatio-temporal structure of events. In order to keep track of temporal context, a history of events is considered, and efficient update mechanisms are devised to avoid the pitfall of recomputation. EventNet [37] proposes a recursive variant of PointNet [32] for 3D point-cloud processing, which however does not perform hierarchical learning. Similarly, recursive rules that sparsely update the feature maps have been proposed for 2D regular convolutions [3] and sparse convolutions [25]. Graph neural network methods [21, 26, 36]

instead represent events as evolving graphs, that are inherently sparse and preserve the temporal structure. We adopt a geometric approach too, since our architecture is based on the convolutional mechanism. However, since it is an RNN model, our method is naturally recursive, instead of requiring to adapt an existing algorithm to work in this way. By making use of the implicit memory of our recurrent units, we do not keep track of a history of events, thus we are not bound to a sliding window of fixed length. In [17] a transformer architecture is proposed that can process batches of events while updating an associative memory. While this performs well in object recognition, we show that our convolutional architecture can be easily applied to dense prediction tasks like object detection, where we obtain the best results.

The combination of CNNs and RNNs has been introduced in ConvLSTM [38] to deal with spatio-temporal data, but it can only be applied to dense tensors. Matrix-LSTM [4] then used this concept and adapted it for event streams. While their method preserves sparsity in the computation, it uses a single recurrent layer to build end-to-end differentiable, synchronous and dense representations that are processed by CNNs. We take this approach further, and propose a convolutional RNN, based on the LSTM, that can process events asynchronously in multi-layered hierarchical networks, without the need of a dense CNN. It has been shown that RNNs share many similarities with SNNs [14, 42], hinting that this approach could emerge as the most suited for the domain of event data.

3 Method

3.1 Asynchronous Event Processing

Event cameras have independent pixels that measure the logarithmic light intensity signal and trigger an event as soon as the change exceeds a certain threshold. Each event encodes the coordinates of the originating pixel $u_i \doteq (x_i, y_i)^\top$, the timestamp of generation τ_i, and the polarity $p_i \in \{-1, +1\}$ that represents the sign of the variation. At any time T, the events produced by the camera since initialization are expressed as an ordered sequence of tuples containing these elements. More generally, we can consider a sequence where each tuple stores an arbitrary set of features v_i instead of the polarity only, *i.e.*, the sequence $\mathcal{E} = \{(u_i, \tau_i, v_i) \mid \tau_i \leq T\}$. By processing events asynchronously, our algorithm will produce, by the same time T, a sequence of predictions that are continuously updated. Thus, we wish to learn a mapping from a stream of inputs to a stream of output features, or target values (Fig. 1). We do it using a convolutional RNN, which processes the events in the input stream sequentially, capturing the spatial structure due to its convolutional nature, and integrating features in time due to its recurrent nature.

3.2 FARSE-Convolution

We now define the FARSE-Convolution, which acts as the main computational component in our architecture. Inspired by [4], we form a 2D grid of recurrent cells, where each cell operates on a spatial neighborhood in the underlying

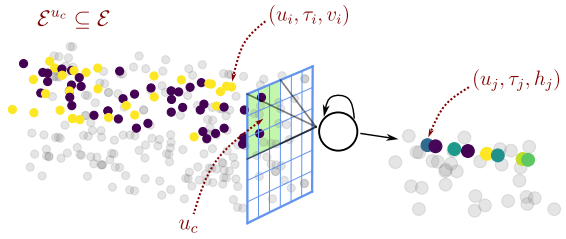


Fig. 2: FARSE-Convolution. Cells of FARSE-Convolution sequentially process the events in a spatial neighborhood, producing an asynchronous stream of output events.

structure of the event stream. This can be either a single pixel or a larger receptive field, and different strides can be used. That is, considering a receptive field centered at $u_c \doteq (x_c, y_c)^\top$, and defining $\mathcal{K}$ the set of positions inside the receptive field, the cell receives as input the event features v_i for the stream $\mathcal{E}^{u_c} = \{(u_i, \tau_i, v_i) \mid \exists k \in \mathcal{K}, u_c + k = u_i\} \subseteq \mathcal{E}$. As in the conventional convolution operation, the same parameters are shared across cells.

The recurrent cells are based on the common equations of the LSTM [15], but to express spatial kernels in the input-to-state transition, we define different input weight matrices for each position in the receptive field, *i.e.*, $W_{ii}^k, W_{if}^k, W_{ig}^k, W_{io}^k, \ \forall k \in \mathcal{K}$, and sum over all positions that receive simultaneous inputs. Hence, the cell is defined by the set of equations:

$$
\begin{aligned}
i_t &= \sigma\Big(\sum_{k\in\mathcal{K}} W_{ii}^k v_t^k + b_{ii} + W_{hi} h_{t-1} + b_{hi}\Big) & f_t &= \sigma\Big(\sum_{k\in\mathcal{K}} W_{if}^k v_t^k + b_{if} + W_{hf} h_{t-1} + b_{hf}\Big) \\
g_t &= tanh\Big(\sum_{k\in\mathcal{K}} W_{ig}^k v_t^k + b_{ig} + W_{hg} h_{t-1} + b_{hg}\Big) & o_t &= \sigma\Big(\sum_{k\in\mathcal{K}} W_{io}^k v_t^k + b_{io} + W_{ho} h_{t-1} + b_{ho}\Big) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t & h_t &= o_t \odot tanh(c_t)
\end{aligned}
\tag{1}
$$

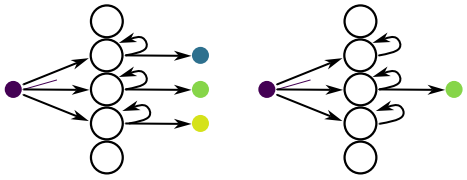
where i_t, f_t, o_t are the input, forget and output gates respectively, g_t is the candidate cell, and c_t, h_t are the cell state and hidden state, all at time t. Gates use the sigmoid activation σ, and $\odot$ is the Hadamard product, or element-wise product. Multiple event features v_t^k associated to the same timestamp are processed in the same time step t, distinguished by their position k.

In contrast to ConvLSTM [38], which applies an LSTM on dense 3D tensors using convolutional connections, the independent cells of the FARSE-Convolution are activated sparsely only when inputs are received. Inside the cell, the position k of each input is used to select the corresponding set of weight matrices, and the summation is performed sparsely only for positions where an event is present. ConvLSTM also implements the state-to-state transition as a convolutional filter with the same kernel and stride as the input-to-state transition. In the FARSE-Convolution, the state-to-state transition is convolutional too, since cells do not share a common state (hence the transition is not fully-connected across the grid), but the kernel is effectively restricted to a size of 1. Because cells in FARSE-Convolution are independent, they also asynchronously

propagate outputs as soon as these are computed. A stream of output events is generated, each storing the output event features h_t, which are computed from the input features v_t using the spatio-temporal context of previous events inside the receptive field. This output stream of events is also spatially structured, since each output is associated to a location in the grid of cells. Thus, we perform a mapping $\mathcal{M} : \mathcal{E}^{\text{in}} \rightarrow \mathcal{E}^{\text{out}}$, as shown in Fig. 2, and we can chain multiple layers of FARSE-Convolution together, having the output stream of one layer become the input stream of the successive one.

3.3 Submanifold FARSE-Convolution

When an input event falls into the receptive field of multiple FARSE-Convolution cells, they all trigger an output event simultaneously. After a few consecutive layers, this can result in almost completely dense feature maps generated by a single event in the original input stream. Indeed, FARSE-Convolution suffers from the same problem of input densification that was observed for regular 2D convolution [11, 12]. To address this issue, we define a variant of FARSE-Convolution inspired by Submanifold Sparse Convolutions (SSC) [13], which we name Submanifold FARSE-Convolution. SSCs are not mathematically equivalent to regular convolutions, as they only compute an output if the corresponding site in input is active (different from zero). To make this possible, the convolution must be applied with a stride of 1, and the input must be padded so to preserve the spatial dimensions in the output, as in the so-called *same* padding scheme. Submanifold FARSE-Convolution follows these same rules. In our case, positions that are padded are simply inactive, and cells that have them in their receptive fields will never receive any events in those positions. Then, when an input event is received, all cells that are affected by it update their internal state as usual, but *only* the cell in the corresponding output site triggers an output event and propagates the features forward, as shown in Fig. 3.



(a) FARSE-Conv. **(b)** Submanifold FARSE-Conv.

Fig. 3: FARSE-Convolution versus Submanifold FARSE-Convolution. With FARSE-Convolution, all cells that receive the event update their state and trigger an output. With Submanifold FARSE-Convolution instead, all cells update their state, but only the one in site corresponding to the input's triggers an output.

A single layer of Submanifold FARSE-Convolution performs exactly the same computation as a regular FARSE-Convolution, since all cells that receive an input are updated as it normally happens. This is necessary to deal with the asynchronicity of the event stream. If we updated, at time step t, only the cells whose corresponding site in input is active then we would have single active cells for most time steps, and information from neighbouring positions would not be

incorporated. Conversely, since in Submanifold FARSE-Convolution only cells whose corresponding input site is active propagate the events forward, sparsity is maintained in the output and we are guaranteed that the number of events in $\mathcal{E}^{\text{out}}$ is the same as in $\mathcal{E}^{\text{in}}$. This largely affects the computational cost as it prevents the number of events from exploding through the execution of the network.

3.4 Sparse Pooling Operations

Pooling operations are important building blocks in convolutional networks, and they are necessary to downscale the inputs when using Submanifold FARSE-Convolution. Hence, we define sparse variants for pooling operations that can work with event streams. Sparse Pooling operations form a 2D grid of stateless cells. A pooling cell receives input events that fall in its receptive field, and propagates them asynchronously. If multiple events are received simultaneously, their features are aggregated using an aggregation function. If a single event is received, its features are unchanged. Sparse Pooling operations can therefore be interpreted as a special type of FARSE-Convolution where cells are defined by the single non-recurrent equation

$$h_t = \underset{k \in \mathcal{K}}{\alpha}(v_t^k) \tag{2}$$

where $\alpha_{k \in \mathcal{K}}$ is an aggregation function applied over all positions in the receptive field. We consider two pooling operations: Sparse Max Pooling which aggregates features by the element-wise max operation, and Sparse Average Pooling which performs the element-wise average.

3.5 Temporal Dropout

Pooling operations compress the input spatially. In standard convolutional networks, this allows to learn hierarchically in space and to reduce computation at deeper layers. Event data, however, live in a spatio-temporal domain, and since events are asynchronous, spatial pooling alone is not effective in compressing the stream of events. For these reasons, we define the Temporal Dropout module to compress the event data along the temporal dimension. Temporal Dropout is applied independently on each coordinate of the input stream. This forms once again a 2D grid of asynchronous cells, with receptive fields restricted to single positions. Temporal Dropout is controlled by a window size parameter l, and when cells receive input events, they propagate only the last in every window of l events. That is, each cell keeps a counter which is incremented at every new event as follows

$$c_t = (c_{t-1} + 1) \bmod l \tag{3}$$

and triggers an output event only if $c_t = l - 1$. If an output is triggered at time step t, the input event at t is propagated without modifying any of its attributes. If an output is not triggered instead, the input event is simply dropped.

Adding one or multiple Temporal Dropout layers in a multi-layered network realizes a network that learns hierarchically in time, *i.e.*, an HRNN [40]. Layers following a Temporal Dropout learn to focus on longer-term dependencies that span across many time steps. Layers preceding it, focus on shorter-term dependencies, and during training learn to produce optimal summarized representations of events in the same window. Since windows are defined in terms of number of input events, and not time intervals, the output rate of the Temporal Dropout is activity-driven, and the high temporal resolution of events is in principle preserved. Thanks to Temporal Dropout we can effectively compress the stream of events propagated through the network, which majorly impacts the computational complexity of our model, as demonstrated in Sec. 4.

3.6 Computational Complexity

We analyze the computational complexity of our architecture in terms of number of floating point operations that are performed for each input event (FLOPs/ev). This metric is independent from the software and hardware implementation, and allows us to compare our method with others in the literature. FARSE-CNN exploits sparsity because, at any layer, only the cells that are affected by an event in input perform some computation. In the 1D case, and when all layers behave as regular convolutions, the maximum number of sites a_n that can be activated at layer n as a consequence of a single event at the input layer is upper bounded as follows

$$a_n \leq 1 + \sum_{i=1}^{n} \frac{k_i - 1}{\prod_{j=i}^{n} s_j} \tag{4}$$

with k_i denoting here the kernel size and s_i the stride of layer i. To extend this formula to submanifold convolutions, we consider $k_i = 1$ always if layer i is a Submanifold FARSE-Convolution layer. A proof of this upper bound is provided in the Supplementary Materials. In the following, we use a_n to denote the number of active sites in the usual 2D case; For square kernels, this means that the derived upper bound is simply squared.

Upon receiving an input, a cell of FARSE-Convolution performs the same computation as a standard LSTM cell, except that different weight matrices $W_{ii}^k, W_{if}^k, W_{ig}^k, W_{io}^k$ are used for each position, and the results of matrix-vector multiplications for simultaneous inputs are summed together. A cell can receive at most $\min(k^2, a_{n-1})$ inputs, so each of the 4 gates has to aggregate this same amount of $|h|$ dimensional vectors. For all cells that can be active, we obtain the following bound for the FLOPs/ev performed by a FARSE-Convolution at layer n:

$$\text{FLOP}_{\text{FARSE-Conv}}^{n} \leq a_n \Big(8|h| \cdot \big(|h| + |v| + \frac{1}{2}\big) + 4|h| \cdot \big(\min(k^2, a_{n-1}) - 1\big) \Big) \tag{5}$$

Equation (5) also applies to Submanifold FARSE-Convolution because the amount of computation performed by each cell is exactly the same as for FARSE-Convolution, as explained in Sec. 3.3. Pooling cells, instead, only incur the cost

of aggregating the features of simultaneous inputs. If the Sparse Pooling layer is used with $k = s$, there is no overlap between receptive fields, and over all cells each input is aggregated at most once. The number of operations performed by a Sparse Pooling at layer n is therefore upper bounded as

$$\text{FLOP}_{\text{SP}}^{n} \leq \begin{cases} a_n \min(k^2, a_{n-1})\text{FLOP}_{\alpha} & \text{if } k > s \\ a_{n-1}\text{FLOP}_{\alpha} & \text{if } k = s \end{cases} \tag{6}$$

where FLOP_{α} is the amount of FLOPs performed by the aggregation function α on a pair of inputs. We assume $\text{FLOP}_{\max} \leq |v|$ for the max operation, and $\text{FLOP}_{\text{avg}} \leq 2|v|$ for the average operation.

Finally, the temporal compression performed by Temporal Dropout has a large impact on the cost of all layers following it, since it only propagates a portion of its inputs forward. In our experiments in Sec. 4 we compute the FLOPS/ev counting the effective number of inputs and aggregations performed at each layer. However, in the Supplementary Material we show that it is possible to analyze theoretically the effect of temporal compression in our networks.

4 Experiments

4.1 Implementation and Experimental Setup

All the modules described in the previous section have been implemented using PyTorch [30]. To implement the FARSE-Convolution we adopt a variation, that restricts its parameter space, to make it compatible with PyTorch's highly optimized implementation of the LSTM. Instead of using Eq. (1) directly, we define the positional weight matrices $W^k \in \mathbb{R}^{|h| \times |v|}\ \forall k \in \mathcal{K}$, and pass as input to the LSTM the vector $\tilde{v_t} \in \mathbb{R}^{|h|}$ computed as

$$\tilde{v_t} = \sum_{k \in \mathcal{K}} W^k v_t^k. \tag{7}$$

Thus, we pre-multiply the event features with the corresponding weight matrix, and sum together for simultaneous events. The LSTM cell then multiplies this by gate-specific weight matrices, *e.g.*, for the input gate $W_{ii}\tilde{v_t} = W_{ii} \sum_{k \in \mathcal{K}} W^k v_t^k$, which is equivalent to $\sum_{k \in \mathcal{K}} W_{ii} W^k v_t^k$ due to distributive property. Therefore, this formulation follows Eq. (1), but we are imposing a factorization of W_{ii}^k in $W_{ii}W^k$, and similarly for other gates. Notice that after training we can just multiply these weight matrices together once to obtain an implementation equivalent to our original formulation. At the input layer, we represent events with the polarity and the delay from the previous one at the same site, normalized between 0 and a maximum value d_{max} to avoid numerical issues. When a delay larger than d_{max} is computed, the normalized delay is clamped to 1, encoding the fact that an arbitrarily large amount of time might have passed since the last event at the same site. In our case, d_{max} is set to 100 ms, and the first event received at every site is also assigned this delay.

Table 1: Comparison with asynchronous methods for object recognition.

	N-Caltech101		N-Cars	
	Accuracy ↑	MFLOP/ev ↓	Accuracy ↑	MFLOP/ev ↓
HOTS [19]	0.210	54.0	0.624	14.0
HATS [39]	0.642	4.3	0.902	0.03
YOLE [3]	0.702	3659	0.927	328.16
AsyNet [25]	0.745	202	0.944	21.5
NVS-S [21]	0.670	7.8	0.915	5.2
EVS-S [21]	0.761	11.5	0.931	6.1
AEGNN [36]	0.668	7.31	0.945	0.47
EventFormer [17]	**0.848**	**0.048**	0.943	**0.013**
FARSE-CNN (Ours)				
TD window 2		-	**0.949**	2.056
TD window 4	0.687	0.339	0.939	0.495
TD window 6	0.678	0.190	0.913	0.335
TD window 8	0.660	0.148		-

To train our networks, we use the Lightning framework [7]. Training is done using the Adam optimizer [18], and the gradients of multiple batches are accumulated to obtain an effective batch size of 256. The learning rate starts at 10^{-3} and follows a cosine annealing schedule with warm restarts every 10 epochs [23]. In each setting, we run the training once with a hold-out validation set to find the optimal number of epochs, and we use this number to train on the full data.

4.2 Object Recognition

We evaluate our method on two common datasets. N-Cars [39] targets urban environment applications, and contains recordings of real world objects. It is composed of 24,029 samples, each one is 100 ms long and can either contain a car or be a background scene. N-Caltech101 [28] is the neuromorphic version of Caltech101 [8], generated by moving an event camera in front of still pictures displayed on a monitor. It contains 8,246 samples, each recorded over 300 ms, which are divided into 101 classes.

Our network configuration, selected by hyper-parameters search on the N-Cars dataset, is composed of 11 Submanifold FARSE-Conv layers alternated by 4 blocks of a Sparse Max Pooling and a Temporal Dropout, after which the stream is pooled into a single coordinate and passed to an LSTM and linear classifier. More details on our configurations are in Supplementary Materials. On both datasets we ablate the impact of different Temporal Dropout windows. On N-Cars we use window size up to 6, after which the effect of temporal compression mostly saturates due to the short samples in this dataset. All networks are trained for 130 epochs. On N-Caltech101 we use windows from 4 to 8, and train on sequences of 100 ms for 130 epochs before fine-tuning on the full sequences for 30 epochs. We found that this training scheme offers a better trade-off between performance and training complexity on this dataset. On both datasets we use

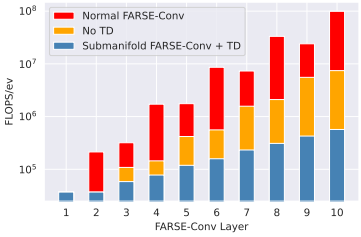


Fig. 4: FLOPS/ev per layer on the N-Cars dataset.

Table 2: Accuracy on DVS128 Gesture. Compared to two synchronous methods that represent the state of the art for frame-based and point-based approaches.

	Async.	Acc. ↑	MFLOP/ev ↓
TBR + I3D [16]	✗	0.996	38820
VMV-GCN [43]	✗	0.975	330
FARSE-CNN (Ours)			
w/o state init	✓	0.807	1.905
w/ state init	✓	0.966	1.905

data augmentations: event coordinates are flipped horizontally at random and translated to random positions inside a maximum frame size.

In Tab. 1 we report our results, compared to other asynchronous methods. FARSE-CNN outperforms previous networks on N-Cars when using a window size of 2, while it performs slightly worse, on par with other methods, when the window size is increased to 4, in favor of a 4-times lower complexity. On N-Caltech101, FARSE-CNN achieves among the best accuracy-complexity trade-offs, and with higher window sizes it reduces its complexity by more than a half while compromising less than 3% of accuracy. While EventFormer surpasses FARSE-CNN on this benchmark, its architecture is limited to high-level tasks, while our solution can easily be applied to dense prediction, as shown in Sec. 4.4. We also emphasize that N-Caltech101 offers a semi-realistic scenario, with static 2D images and identical motion across samples, thus not challenging networks' abilities in exploiting motion-rich information from events. FARSE-CNN on the other hand, excels on real data.

Compared to others, FARSE-CNN is the *only* method whose complexity does not increase when moving from the N-Cars to the N-Caltech101 dataset. As evident from the analysis of Sec. 3.6, the computation performed by FARSE-CNN does not depend on the spatio-temporal density of events, differently from other methods. In [25] this dependence is formalized through their *fractal dimension*, in graph-based methods [21, 36] the sparsity of the data affects the number of edges in the graph, and in [17] the size of the sequences has an impact on the complexity.

Figure 4 shows how the FLOPS/ev are distributed among the layers of FARSE-CNN, considering only Submanifold FARSE-Convolutions with 3×3 kernel, where most of the computation is performed. In the same image we also ablate the complexity of a network that does not use Temporal Dropout and one that uses regular FARSE-Convolutions (with Temporal Dropout), where complexity grows by 1 and 2 orders of magnitude respectively.

4.3 Gesture Recognition

We use the DVS128 Gesture dataset [1], which contains 1,342 samples divided in 11 categories of hand and arm gestures. This dataset allows us to study the

behavior of our method on long event recordings, in the order of seconds. The network configuration is the same that is used for object recognition, with window size 2. Training data is augmented using random coordinates translations.

Since using the full samples is computationally impractical and detrimental to the training itself, we split samples into windows of 100 ms with a stride of 50 ms, and train on those. At inference time, however, FARSE-CNN processes incoming events continuously without re-initialization, and since the model is optimized for sequences of 100 ms, when we feed it the entire recordings its performance degrades quickly. To generalize the model to sequences of indefinite lengths, we adopt the following training strategy. At the start of each epoch we compute and store the state of the entire network after it has processed randomly selected samples for each class (in our case, we store 2 states for each class). Then, during the forward pass, the network is initialized with randomly selected states, or it is left uninitialized with a 20% probability. In this way, the model learns to perform correct inference long after a clean initialization, thus operating flexibly on different time horizons. In both cases, the network is trained for 30 epochs. For a robust evaluation over long sequences, we take the most common prediction (*i.e.*, the statistical mode) over time on the full samples. We increase the accuracy from 80.7% obtained with the simple training to a 96.6% obtained with state initialization. As this is the first asynchronous method that is tested on DVS128 Gesture, we report for reference the accuracy of other *synchronous* methods in comparison with ours in Tab. 2.

4.4 Object Detection

Event-based object detection meets particular interest in applications where the advantages of event cameras would allow for real time detection in challenging scenarios, such as automotive applications. We use a dataset that targets precisely this field, the Gen1 Automotive dataset [41]. Gen1 Automotive is collected from almost 40 hours of real world recordings, and contains bounding boxes for 228,123 cars and 27,658 pedestrians. To perform detection with FARSE-CNN, we add to it a fully connected YOLO [34] output layer which detects bounding boxes over a 5×4 grid. The network comprises 8 layers, instead of the 13 that were used for object recognition. We find that making the network deeper in this case does not improve performance, while increasing complexity. This could be explained by the fact that we do not use here any data augmentation, which deeper networks have shown to take more advantage of in our experiments. We make this choice because the detection targets follow a specific distribution in the 2D image space, as explained in [41], and tampering this distribution with spatial augmentations could be counterproductive.

We optimize a loss that is based on the original YOLO loss, with the small tweaks described here. We use sigmoid activation for all bounding box values, and softmax activation for class scores. For bounding box coordinates we use the mean squared error loss, while for objectness and class prediction we use the cross entropy loss. We set the hyperparameters $\lambda_{\text{coord}} = 20$, $\lambda_{\text{noobj}} = 1$, and use an additional weight for the objectness error of boxes containing an object

Table 3: Comparison with asynchronous methods on Gen1 Automotive. For NVS-S we report the result obtained by [36],

	mAP ↑	MFLOP/ev ↓
AsyNet [25]	0.129	205
NVS-S [21]	0.086	7.8
AEGNN [36]	0.163	5.26
FARSE-CNN (Ours)	0.288	**0.137**
FARSE-CNN + NMS (Ours)	**0.300**	**0.137**

$\lambda_{\text{obj}} = 0.1$, in order to balance the loss with the new activations. The network is trained and tested on windows of 100 ms preceding each label, and training takes 40 epochs.

We evaluate our model using the eleven-point mean average precision (mAP) metric [6], implemented in the tool provided by [29], as well as the computational complexity. In Tab. 3 we report our results in comparison to other asynchronous methods. FARSE-CNN achieves a 28.8% mAP, which is further improved when we use the non-maximum suppression algorithm [9] to remove multiple overlapping detections, reaching 30.0% mAP, and outperforming the previous best method, AEGNN, by 13.7%. Considering the individual categories, FARSE-CNN scores 40.5% AP on cars and 19.5% AP on pedestrians, which are under-represented in the dataset. Since this performance is obtained using a relatively shallow network, the complexity is also extremely low, about 38 times lower than the number of operations performed by AEGNN. Thus, our method improves on the state of the art for asynchronous methods on both mAP and complexity for object detection on this dataset. Qualitative results of our network are in the Supplementary Materials.

4.5 Latency

We measure the execution time of the FARSE-CNN YOLO configuration on the Gen1 Automotive dataset. Processing a single event through the entire network takes on average 16.7 ms on an Intel Xeon CPU. Due to the compression along the time dimension of Temporal Dropout, however, not all stages of the network will be executed in response to every input event. In view of this, a breakdown of the latency for the individual layers is shown in Fig. 5a. As mentioned in Sec. 4.1, our implementation is based on the LSTM provided by PyTorch and it is optimized for parallel processing of multiple sequences on GPU. This allows much faster training times, but it is not as efficient on single step processing. Indeed, processing a single event on an Nvidia Quadro GPU takes 41.6 ms, but the latency scales favorably over longer sequences of events, as shown in Fig. 5b.

We also measure the latency of our method on object recognition datasets following the procedure adopted in [17], *i.e.*, measuring the average run time per sample and dividing by the average number of events per sample, on an Nvidia Quadro GPU. We find a latency of 0.0009 ms on N-Caltech101, and

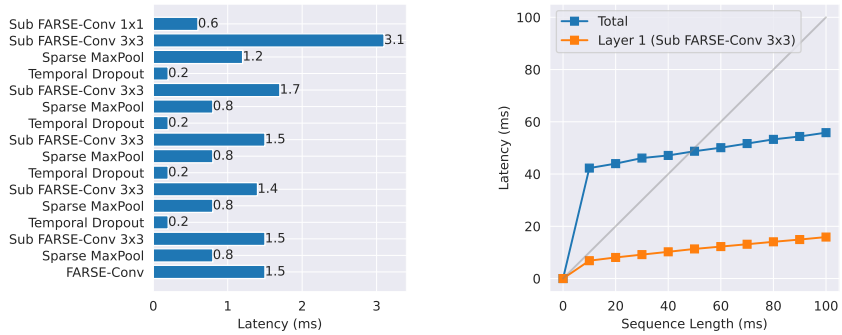


Fig. 5: Execution latency on Gen1 Automotive. (a) Latency of individual layers when processing a single event on CPU. (b) Latency over sequences of different time length on GPU. The latency of the slowest layer is also shown separately.

0.0140 ms on N-Cars, when Temporal Dropout has window size of 4. For comparison, [17] obtained 0.0007 ms on N-Caltech101 and 0.0005 ms on N-Cars using an Nvidia RTX3090. Although their method reports a complexity lower by one order of magnitude on the same datasets, our approach remains competitive in this benchmark on N-Caltech101, where the longer sequences can take advantage of our current parallel implementation.

We believe that large improvements in latency will be obtained through hardware and software implementations optimized for inference, and aiming towards the pipelined execution of the architecture.

5 Conclusions and Future Improvements

We proposed FARSE-CNN, a novel asynchronous method that leverages sparsity of events in all its components. We showed results of our method on object recognition, and on object detection, where we improved on the state of the art for Gen1 Automotive in both performance and complexity. On gesture recognition we provided results which may be useful as reference for future works.

Our Temporal Dropout compresses features inside windows of events, imposing a simple hierarchy which may not be able to capture the real temporal structure of event data. This could be replaced with a learned latent structure, as in the Multiscale HRNN [5]. Different RNN models could be employed, like the GRU, to potentially decrease computational complexity even further. Moving towards the neuromorphic approach, more biologically inspired RNN models could be adopted, such as the Spiking Neural Unit proposed in [42] and compared by the authors to the LSTM. Given the connection between RNNs and SNNs, discussed also in [14], we believe that our method goes in the direction of bridging the gap between the two lines of research, and hopefully can prove useful also in the study of SNNs applied to event-based vision.

Acknowledgements

This paper is supported by the FAIR (Future Artificial Intelligence Research) project, funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, investment 1.3, line on Artificial Intelligence).

References

1. Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., Debole, M., Esser, S., Delbruck, T., Flickner, M., Modha, D.: A low power, fully event-based gesture recognition system. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7388–7397 (2017). https://doi.org/10.1109/CVPR.2017.781
2. Barbier, T., Teulière, C., Triesch, J.: Spike timing-based unsupervised learning of orientation, disparity, and motion representations in a spiking neural network. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 1377–1386 (2021). https://doi.org/10.1109/CVPRW53098.2021.00152
3. Cannici, M., Ciccone, M., Romanoni, A., Matteucci, M.: Asynchronous convolutional networks for object detection in neuromorphic cameras. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). pp. 1656–1665 (2019). https://doi.org/10.1109/CVPRW.2019.00209
4. Cannici, M., Ciccone, M., Romanoni, A., Matteucci, M.: Matrix-lstm: a differentiable recurrent surface for asynchronous event-based data. In: Proceedings of the European Conference on Computer Vision (ECCV), Glasgow, UK. pp. 23–28 (2020)
5. Chung, J., Ahn, S., Bengio, Y.: Hierarchical multiscale recurrent neural networks. In: 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings (2017), www.scopus.com
6. Everingham, M., Gool, L.V., Williams, C.K.I., Winn, J.M., Zisserman, A.: The pascal visual object classes (voc) challenge. International Journal of Computer Vision **88**(2), 303–338 (2010), https://doi.org/10.1007/s11263-009-0275-4
7. Falcon, W., The PyTorch Lightning team: PyTorch Lightning (3 2019). https://doi.org/10.5281/zenodo.3828935, https://github.com/Lightning-AI/lightning
8. Fei-Fei, L., Fergus, R., Perona, P.: One-shot learning of object categories. IEEE Transactions on Pattern Analysis and Machine Intelligence **28**, 594–611 (2006)
9. Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. IEEE Transactions on Pattern Analysis and Machine Intelligence **32**(9), 1627–1645 (2010). https://doi.org/10.1109/TPAMI.2009.167
10. Gehrig, D., Loquercio, A., Derpanis, K., Scaramuzza, D.: End-to-end learning of representations for asynchronous event-based data. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 5632–5642 (2019). https://doi.org/10.1109/ICCV.2019.00573
11. Graham, B.: Sparse 3d convolutional neural networks. British Machine Vision Conference (2015)

12. Graham, B.: Spatially-sparse convolutional neural networks. arXiv preprint arXiv:1409.6070 (2014)
13. Graham, B., Engelcke, M., Maaten, L.v.d.: 3d semantic segmentation with submanifold sparse convolutional networks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9224–9232 (2018). https://doi.org/10.1109/CVPR.2018.00961
14. He, W., Wu, Y., Deng, L., Li, G., Wang, H., Tian, Y., Ding, W., Wang, W., Xie, Y.: Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences. Neural networks : the official journal of the International Neural Network Society **132**, 108–120 (2020)
15. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (11 1997). https://doi.org/10.1162/neco.1997.9.8.1735, https://doi.org/10.1162/neco.1997.9.8.1735
16. Innocenti, S.U., Becattini, F., Pernici, F., Del Bimbo, A.: Temporal binary representation for event-based action recognition. In: 2020 25th International Conference on Pattern Recognition (ICPR). pp. 10426–10432 (2021). https://doi.org/10.1109/ICPR48806.2021.9412991
17. Kamal, U., Dash, S., Mukhopadhyay, S.: Associative memory augmented asynchronous spatiotemporal representation learning for event-based perception. In: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net (2023), https://openreview.net/pdf?id=ZCStthyW-TD
18. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (12 2014)
19. Lagorce, X., Orchard, G., Galluppi, F., Shi, B.E., Benosman, R.B.: HOTS: A hierarchy of event-based time-surfaces for pattern recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence **39**(7), 1346–1359 (2017). https://doi.org/10.1109/TPAMI.2016.2574707
20. Lee, J.H., Delbruck, T., Pfeiffer, M.: Training deep spiking neural networks using backpropagation. Frontiers in Neuroscience **10** (2016). https://doi.org/10.3389/fnins.2016.00508, https://www.frontiersin.org/articles/10.3389/fnins.2016.00508
21. Li, Y., Zhou, H., Yang, B., Zhang, Y., Cui, Z., Bao, H., Zhang, G.: Graph-based asynchronous event processing for rapid object recognition. In: 2021 IEEE/CVF International Conference on Computer Vision (ICCV). pp. 914–923 (2021). https://doi.org/10.1109/ICCV48922.2021.00097
22. Lichtsteiner, P., Posch, C., Delbruck, T.: A 128× 128 120 db 15 µs latency asynchronous temporal contrast vision sensor. Solid-State Circuits, IEEE Journal of **43**, 566 – 576 (03 2008). https://doi.org/10.1109/JSSC.2007.914337
23. Loshchilov, I., Hutter, F.: SGDR: stochastic gradient descent with warm restarts. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net (2017), https://openreview.net/forum?id=Skq89Scxx
24. Merolla, P.A., Arthur, J.V., Alvarez-Icaza, R., Cassidy, A.S., Sawada, J., Akopyan, F., Jackson, B.L., Imam, N., Guo, C., Nakamura, Y., Brezzo, B., Vo, I., Esser, S.K., Appuswamy, R., Taba, B., Amir, A., Flickner, M.D., Risk, W.P., Manohar, R., Modha, D.S.: A million spiking-neuron integrated circuit with a scalable communication network and interface. Science **345**(6197), 668–673 (2014). https://doi.org/10.1126/science.1254642, https://www.science.org/doi/abs/10.1126/science.1254642

25. Messikommer, N., Gehrig, D., Loquercio, A., Scaramuzza, D.: Event-based asynchronous sparse convolutional networks. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.M. (eds.) Computer Vision – ECCV 2020. pp. 415–431. Springer International Publishing, Cham (2020)
26. Mitrokhin, A., Hua, Z., Fermüller, C., Aloimonos, Y.: Learning visual motion segmentation using event surfaces. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 14402–14411 (2020). https://doi.org/10.1109/CVPR42600.2020.01442
27. O'Connor, P., Neil, D., Liu, S.C., Delbruck, T., Pfeiffer, M.: Real-time classification and sensor fusion with a spiking deep belief network. Frontiers in Neuroscience **7** (2013). https://doi.org/10.3389/fnins.2013.00178, https://www.frontiersin.org/articles/10.3389/fnins.2013.00178
28. Orchard, G., Jayawant, A., Cohen, G., Thakor, N.: Converting static image datasets to spiking neuromorphic datasets using saccades. Frontiers in Neuroscience **9** (07 2015). https://doi.org/10.3389/fnins.2015.00437
29. Padilla, R., Passos, W.L., Dias, T.L.B., Netto, S.L., da Silva, E.A.B.: A comparative analysis of object detection metrics with a companion open-source toolkit. Electronics **10**(3) (2021). https://doi.org/10.3390/electronics10030279, https://www.mdpi.com/2079-9292/10/3/279
30. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
31. Posch, C., Serrano-Gotarredona, T., Linares-Barranco, B., Delbruck, T.: Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output. Proceedings of the IEEE **102**, 1470–1484 (10 2014). https://doi.org/10.1109/JPROC.2014.2346153
32. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017)
33. Rebecq, H., Ranftl, R., Koltun, V., Scaramuzza, D.: Events-to-video: Bringing modern computer vision to event cameras. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)
34. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 779–788 (2016). https://doi.org/10.1109/CVPR.2016.91
35. Rueckauer, B., Liu, S.C.: Conversion of analog to spiking neural networks using sparse temporal coding. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS). pp. 1–5 (2018). https://doi.org/10.1109/ISCAS.2018.8351295
36. Schaefer, S., Gehrig, D., Scaramuzza, D.: AEGNN: Asynchronous event-based graph neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 12371–12381 (June 2022)
37. Sekikawa, Y., Hara, K., Saito, H.: Eventnet: Asynchronous recursive event processing. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019)

38. Shi, X., Chen, Z., Wang, H., Yeung, D.Y., Wong, W.k., Woo, W.c.: Convolutional lstm network: A machine learning approach for precipitation nowcasting. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 28. Curran Associates, Inc. (2015), https://proceedings.neurips.cc/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf
39. Sironi, A., Brambilla, M., Bourdis, N., Lagorce, X., Benosman, R.B.: HATS: Histograms of averaged time surfaces for robust event-based object classification. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition pp. 1731–1740 (2018)
40. Sordoni, A., Bengio, Y., Vahabi, H., Lioma, C., Grue Simonsen, J., Nie, J.Y.: A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. p. 553–562. CIKM '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2806416.2806493, https://doi.org/10.1145/2806416.2806493
41. de Tournemire, P., Nitti, D., Perot, E., Migliore, D., Sironi, A.: A large scale event-based detection dataset for automotive. arXiv preprint arXiv:2001.08499 (2020)
42. Wozniak, S., Pantazi, A., Bohnstingl, T., Eleftheriou, E.: Deep learning incorporating biologically inspired neural dynamics and in-memory computing. Nature Machine Intelligence **2**, 325–336 (06 2020). https://doi.org/10.1038/s42256-020-0187-0
43. Xie, B., Deng, Y., Shao, Z., Liu, H., Li, Y.: Vmv-gcn: Volumetric multi-view based graph cnn for event stream classification. IEEE Robotics and Automation Letters **7**(2), 1976–1983 (2022). https://doi.org/10.1109/LRA.2022.3140819
44. Zhu, A., Yuan, L., Chaney, K., Daniilidis, K.: Ev-flownet: Self-supervised optical flow estimation for event-based cameras. In: Proceedings of Robotics: Science and Systems. Pittsburgh, Pennsylvania (June 2018). https://doi.org/10.15607/RSS.2018.XIV.062
45. Zhu, A.Z., Yuan, L., Chaney, K., Daniilidis, K.: Unsupervised event-based learning of optical flow, depth, and egomotion. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 989–997 (2019)