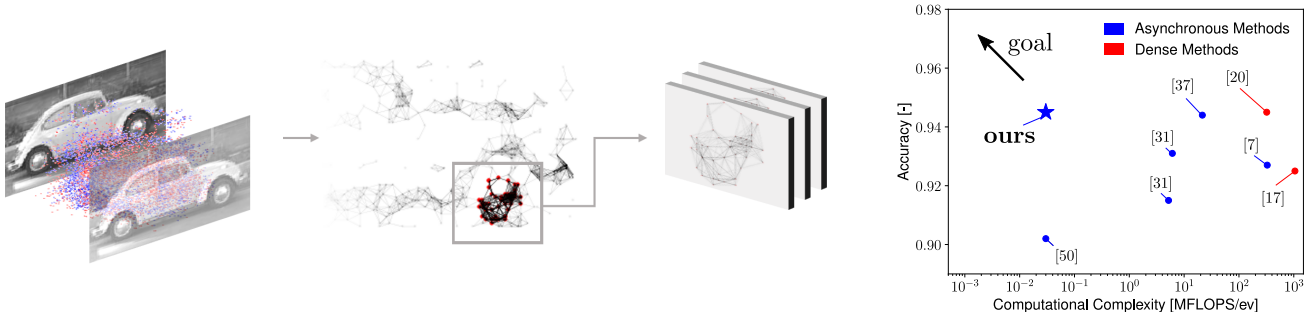
AEGNN: Asynchronous Event-based Graph Neural Networks

Simon Schaefer* Daniel Gehrig* Davide Scaramuzza

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



(a) Event Stream (b) Sparse Event Graph (c) GNN (d) Performance vs. Computation

Figure 1. Our method processes events (a) as spatio-temporal graphs (b), using a graph neural network (c). For each new event we only limit computation to a local subgraph of (b), corresponding to the receptive field of the network, thus significantly reducing per-event computation. Our method has a 200 times lower computational complexity than state-of-the-art asynchronous method [31], while achieving state-of-the-art results on object recognition and object detection (d).

Abstract

The best performing learning algorithms devised for event cameras work by first converting events into dense representations that are then processed using standard CNNs. However, these steps discard both the sparsity and high temporal resolution of events, leading to high computational burden and latency. For this reason, recent works have adopted Graph Neural Networks (GNNs), which process events as "static" spatio-temporal graphs, which are inherently "sparse". We take this trend one step further by introducing Asynchronous, Event-based Graph Neural Networks (AEGNNs), a novel event-processing paradigm that generalizes standard GNNs to process events as "evolving" spatio-temporal graphs. AEGNNs follow efficient update rules that restrict recomputation of network activations only to the nodes affected by each new event, thereby significantly reducing both computation and latency for event-by-event processing. AEGNNs are easily trained on synchronous inputs and can be converted to efficient, "asynchronous" networks at test time. We thoroughly validate our method on object classification and detection tasks, where we show an up to a 200-fold reduction in computational complexity (FLOPs), with similar or even better performance than state-of-the-art asynchronous methods. This reduction in computation directly translates to an 8-fold reduction in computational latency when compared to standard GNNs, which opens the door to low-latency event-based processing.

*these authors contributed equally

Multimedia Material

For videos, code and more, visit our project page https://uzh-rpg.github.io/aegnn/.

1. Introduction

Compared to standard frame-based cameras, which measure absolute intensity at a synchronous rate, event-cameras only measure *changes* in intensity, and do this independently for each pixel, resulting in an asynchronous and binary stream of *events* (Figure 1 (a)). These events measure a highly compressed representation of the visual signal and are characterized by microsecond-level latency and temporal resolution, a high dynamic range of up to 140

dB, low motion blur, and low power (milliwatts instead of watts). Due to these outstanding properties, event cameras are indispensable sensors in challenging application domains—such as robotics [9, 21, 47, 52], autonomous driving [18, 51, 56], and computational photography [3, 44, 53, 54]—characterized by frequent high-speed motions, low-light and high-dynamic-range scenes, or in always-on applications, where low power is needed, such as IoT video surveillance [23, 36]. A survey about applications and research in event-based vision can be found in [14].

The output of event cameras is inherently sparse and asynchronous, making them incompatible with traditional computer-vision algorithms designed for standard images. This prompts the development of novel algorithms that optimally leverage the sparse and asynchronous nature of events. In doing so, existing algorithms designed for event cameras have traded off latency and prediction performance. *Filtering-based* [39] [28] approaches process events sequentially, and, thus, can provide low-latency predictions and a high temporal resolution. However, these approaches usually rely on handcrafted filter equations, which do not scale to more complex tasks, such as object detection or classification. Spiking Neural Networks (SNNs) are one instance of filtering-based models, which seek to learn these rules in a data-driven fashion, but are still in their infancy, lacking general and robust learning rules [19, 29, 49]. As a result, SNNs typically fail to solve more complex high-level tasks [2, 39, 41, 51]. Many of the challenges above can be avoided by processing events as batches. In fact, recent progress has been made by converting batches of events into *dense*, image-like representations and processing them using methods designed for images, such as convolutional neural networks (CNNs). By adopting this paradigm, learning-based methods using CNNs have made significant strides in solving computer vision tasks with events [17, 22, 34, 42, 44, 53, 57, 58].

However, while easy to process, treating events as image-like representations discards their sparse and asynchronous nature and leads to wasteful computation. This wasteful computation directly translates to higher power consumption and latency [1, 23, 37]. A recent line of work [16] showed on an FPGA that by reducing the computational complexity by a factor of 5, they could reduce the latency by a factor of 5 while reducing the power consumption by a factor of 4. Therefore, by eliminating wasteful computation, we can expect significant decreases in the power consumption and latency of learning systems.

Currently, this wasteful computation is caused by two factors: On the one hand, due to the working principle of event cameras, they trigger predominantly at edges, while large texture-less or static regions remain without events. Image representations typically encode these regions as zeros, which are then unnecessarily processed by standard neural networks. On the other hand, for each new event, standard methods would need to recompute all network activations. However, events only measure single pixel changes and, thus, leave most of the activations unchanged, leading to unnecessary recomputation of activations.

A recent line of work seeks to address both of these challenges by reducing the computational complexity of learning-based approaches while maintaining the high temporal resolution of events. A key ingredient to keeping high performance in this setting was the adoption of geometric learning methods, such as recursive point-cloud processing [48] or Asynchronous Sparse Convolutions [35]. In both works, standard neural networks were trained using batches of events, leveraging well-established learning techniques such as backpropagation, and then deploying them in an *event-by-event* fashion at test time, thus minimizing computation. However, both of these methods suffer from limitations: While [48] does not perform hierarchical learning, limiting scalability to complex tasks, [35], relies on a specific type of input representation, which discards the temporal information of events.

In this work, we introduce Asynchronous, Event-based Graph Neural Networks (AEGNN), a neural network architecture geared toward processing events as graphs in a sequential manner (Fig. 1). For each new event, our method only performs local changes to the activations of the GNN, and propagates these asynchronously to lower layers. Similar to [35, 48], AEGNNs can be trained on batches of events—thus leveraging backpropagation—and can later be deployed in an asynchronous mode, generating the identical output. However, they address the key limitations of previous work: *(i)* They allow hierarchical learning using standard graph neural networks and *(ii)* model events as spatio-temporal graphs, thus retaining their temporal information, instead of discarding it. This leads to significant computational savings. We summarize our contributions as follows:

- We introduce AEGNN, a novel paradigm for processing events sparsely and asynchronously as temporally evolving graphs. This allows us to process events efficiently, without sacrificing their sparsity and high temporal resolution.
- (ii) We derive efficient update rules, which allow us to simply train AEGNNs on synchronous event-data, and then deploy them in an asynchronous mode during test-time. These rules are general and can be applied to most existing graph neural network architectures.
- (iii) We apply AEGNNs on object recognition and object detection benchmarks. For object detection, we show similar performance to state-of-the-art methods, while requiring up to 200 times less compute, while for object detection we show a 21-fold computation reduction with an up to 3.4% increase in terms of mAP.

2. Related Work

Since the advent of deep learning, event-based vision has adopted many of its models. Early models, relied on shallow learning techniques such as SVMs [51] or filtering-based techniques [15, 25, 28, 39], and have gradually shifted to deeper architectures such as CNN's [17, 34, 44, 57]. While achieving state-of-the-art performance, these types of models do not take into account the sparse and asynchronous nature of events, leading to redundant computation. This prompted the development of sparse network architectures such as SNNs, point cloud methods [48], Submanifold Sparse Convolutions [35] and graph neural networks [4, 5, 31]. Which all seek to reduce computation. While SNNs are traditionally harder to train, due to a lack of efficient learning rules, geometric learning methods such as [4, 5, 31, 35, 48] have gained popularity in recent years, since they are more suited to the asynchronous and sparse nature of events, and are easily trained and implemented thanks to the existence of well-maintained toolboxes.

In particular, graph-based methods such as [4, 5, 31, 46] show a significant reduction in computational complexity compared to dense methods that rely on standard CNNs. This is because, instead of processing events as dense image-like tensors, they only consider sparse connections between events, and confine message passing to these connections. Despite this sparsity, these methods still process events as batches and thus need to recompute all activations, whenever a new event arrives. However, each event only indicates a per-pixel change, and thus recomputing activations leads to the highly redundant computation. To counteract this, a recent line of work has focused on reusing network activations as much as possible between consecutive events, by applying efficient recursive update rules [48] and propagating these to lower layers [35].

These methods, however, do not allow for hierarchical learning [46, 48] or still rely on sparse but image-like input representations, which discard the temporal component of events. These factors either limit the scalability to more complex tasks in the case of [48], or degrade performance while incurring higher computation in the case of [35]. Most similar to our work, [46] learns on dynamic graphs, by performing learned updates each time node events are triggered. However, it also performs shallow learning, *i.e.* it only computes node embeddings, but does not use them for end-task learning.

In this work, we combine the advantages of graph-based methods with efficient recursive update rules, thus addressing these limitations: Asynchronous Event-based Graph Neural Networks are multi-layered, and can thus learn more complex tasks than [48], and leverage the spatio-temporal sparsity of events better than [35], leading to significant computation reduction.

3. Prerequisites

In this work, we model events as spatio-temporal graphs $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ with vertices $\mathcal{V}$ and (directed) edges $\mathcal{E}$. In this context, events are represented as nodes within the graph and connections are formed between neighboring events (Fig.2 (c)). We use a graph neural network to process this graph and generate a prediction y. It can be represented as a function $f(\mathcal{G}) = y$, which executes a set of operations on the graph level. Most common operations consist of graph convolutions and pooling steps, which operate on node features $\mathbf{x}_i$ attached to each node, and edge features e_{ij} attached to each edge.

Graph Convolutions: Graph convolutions generally consist of three distinct steps which are repeated for each node i in the graph: First the function ψ computes messages based on pairs of neighbors (i, j), where i is fixed and $j \in \mathcal{N}(i)$ is in the neighborhood of i. These messages depend on the node features at these nodes, the edge feature but also on the spatial arrangement of nodes i and j. Next, all messages are aggregated through summation[1], and followed by a function γ_Θ, which computes the new value for node i. These steps are summarized in the equations below:

$$\mathbf{z}_i = \sum_{j \in \mathbb{N}(i)} \psi_\Theta(\mathbf{x}_i, \mathbf{x}, \mathbf{e}_{ij}) \tag{1}$$

$$\hat{\mathbf{x}}_i = \gamma_\Theta(\mathbf{x}_i, \mathbf{z}_i) \tag{2}$$

Both ψ and γ denote differentiable functions such as a multi-layer perceptron, parametrized by $\Theta = \{\theta_\gamma, \theta_\psi\}$.

Graph Pooling Graph pooling operations transform a graph $\mathbb{G}$ to a more coarse graph $\mathbb{G}_c$. For an overview of the different types of graph pooling, we refer to [55]. Within this work, we will focus on cluster-based pooling methods, which aggregate the graph nodes into clusters $\mathcal{C}_k$ with cluster centers $k \in \mathcal{V}_c$ which form a subset of $\mathcal{V}$. The new features at these cluster centers are computed by aggregating features in each cluster:

$$\mathbf{x}_k = \max_{i \in \mathcal{C}_k} \mathbf{x}_i \tag{3}$$

Since clustering reduces the number of nodes, the original edges need to reconnected, and this is performed with the function π:

$$\mathbb{E}_c = \pi(\mathbb{E}, \mathbb{C}) \tag{4}$$

resulting in the final coarse graph.
Stacking these operations as layers enables rich, and high-level feature computation, making these models more powerful than the point cloud method in [48] or shallow features computed in [51].

[1] While summation is the most common form of aggregation, any function which is symmetric in its inputs can be used, such as $\max$ and $\min$ or $\sum$.

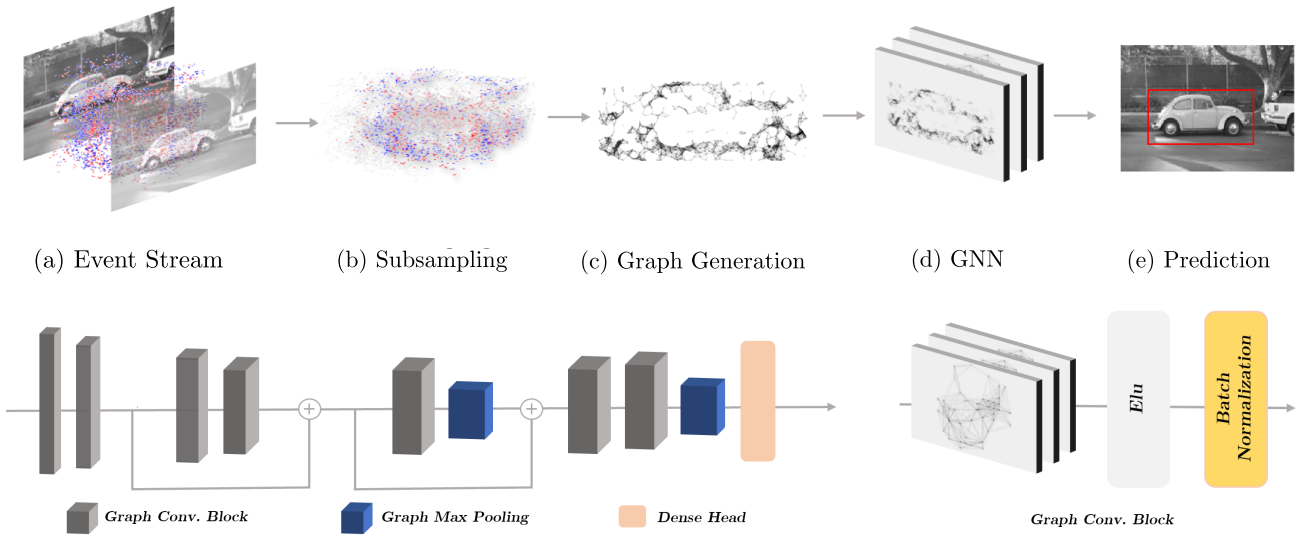


Figure 2. Overview of the processing steps in our method. The event stream (a) is first subsampled using uniform sampling (b). The subsampled events are used to generate a sparse spatio-temporal graph (c), which is processed by a graph neural network (GNN)(d), which generates a bounding-box prediction (d). Although our method works for any task, here we illustrate our method for the task of object detection. In the figure below, we show an overview of the used network architecture. It combines Graph Convolutions (here Spline Convolutions) with pooling layers, followed by a prediction head. Each graph convolution block consists of several graph convolutions followed by ELU and Batch Normalization.

4. Approach

Representing event data as spatio-temporal graphs allows us to efficiently process incoming events by performing sparse but complete graph updates. In the following, we show how a graph can be constructed from an event stream (Sec. 4.1), and we demonstrate how it can be used for efficient and asynchronous computations (Sec. 4.2). An overview of the full method is illustrated in Figure 2.

4.1. Graph Construction

Event cameras have independent pixels which each trigger events, whenever they perceive a brightness change. Each event encodes the pixel position (x_i, y_i), time t_i with microsecond level resolution and polarity (sign) $p_i \in \{-1, 1\}$ of the change. A group of event in a time window ΔT, can thus be represented as an ordered list of tuples

$$\{e_i\}_N = \{e_i\}_{i=1}^N \quad \text{with } e_i = (x_i, y_i, t_i, p_i) \qquad (5)$$

By embedding these events in a spatio-temporal space $\mathbb{R}^3$ we thus can see that they are inherently sparse and asynchronous (Fig.2 (a,b)).

For the sake of computational efficiency, we first subsample the events uniformly by a factor K (Fig. 2 (b)). In this work, we select $K = 10$. While this preprocessing step removes events, we found that it is critical to combat overfitting, since the network learns to consider larger contexts, focusing on more informative events. In contrast to other representations of event data such as event histograms [35] or event volumes [3, 42], the full temporal resolution of the event stream is preserved. This high temporal resolution is crucial in robotic applications like obstacle avoidance [9, 33, 47].

We use the remaining events to form an event graph $\mathcal{G}$, where each event is a node (Fig. 2 (c)). Inspired by [4] the event's temporal position is normalized by a factor β to map it to a similar range as the spatial coordinates. The position of each vertex is then denoted as $\mathbf{X}_i = (x_i, y_i, t_i^*)$ with $t_i^* = \beta t_i$.

For each pair of nodes i and j, an edge e_{ij} between them is generated if they are within spatio-temporal distance R, *i.e.* $R \leq \|\mathbf{X}_i - \mathbf{X}_j\|$ from each other. To reduce computation and regularize the graph, we limit the maximal number of neighborhood nodes to D_{max}, i.e. $|\mathcal{N}(i)| \leq D_{max}$. Finally, we assign initial node features, $\mathbf{x}_i = p_i$ and edge features corresponding to the relative position between the connected vertices, normalized by R.

4.2. Asynchronous Processing

As we slide the time window ΔT, new events enter this window, and old events leave the window. While traditional methods would need to recompute all activations once this happens, here we present a recursive formulation that incorporates new events with minimal computation.

As a new event arrives, a new node is added to the graph, together with new edges connecting this node to existing vertices. The new connections are sparse, affecting only neighboring events. In fact, in the first layer, a new event only affects the state of its 1-hop subgraph (Fig.3, Layer 1), corresponding with the neighborhood of the new node i'. Therefore, activations in the next layer need to only be recomputed for this subgraph via Eq. (2).

$$\hat{\mathbf{z}}_i = \sum_{j \in \mathcal{N}(i)} \psi_\Theta(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{ij}) \tag{6}$$

$$\hat{\mathbf{x}}_i = \gamma_\Theta(\mathbf{x}_i, \mathbf{z}_i) \text{ for all } i \in \mathcal{N}(i') \tag{7}$$

As deeper layers are reached, this subgraph expands, hopping one node after each layer step, until at layer N the nodes in $\mathcal{H}_N(i')$ need to be updated. $\mathcal{H}_N(i')$ denotes the N-hop subgraph which contains all nodes j such that j could be reached from i' using N hops or fewer. We visualize this hopping behavior in Fig. 3. Instead of processing the whole graph, only this subgraph has to be processed to obtain the same resulting graph activations as Eq. 2. By iteratively applying this concept to each graph-convolution layer of a graph neural network, its forward pass can be formulated sparsely, which significantly reduces the computational effort. At each layer, the necessary computation is proportional to the number of nodes in the respective subgraph. This number is known in the graph-theory literature as *neighborhood function* [6], and is influenced by the average and variance of the connectivity of the graph, which together forms the *index of dispersion* [6].

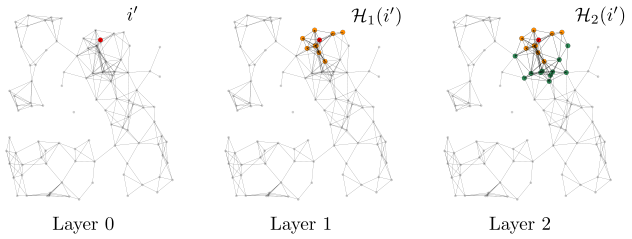


Figure 3. Message propagation in the event graph. A new event (red) is generated and added to the graph of precedent events (left). The added information is propagated to the k-hop neighborhood of the new event vertex, with $k = 1$ (middle) and $k = 2$ (right).

Graph Convolutions Our sparse update rules for graph convolutions are agnostic to the choice of functions ψ and γ (Eq. 2) and are therefore applicable to arbitrary types of graph convolution. It consists of two steps: During the *initialization* the convolution is applied to the full graph, while the resulting graph, i.e., the vertices and edges as well as their attributes, are stored. We perform this step at the beginning and whenever the camera is stationary and mostly noise events enter the sliding window. Thereafter, in the *processing* step, every time a new vertex is inserted into the graph, the graph only changes locally. Therefore, a full graph update is equivalent to updating the 1-hop subgraph starting from the new vertex, by applying Eq. 2 to its 1-hop subgraph only. Thereby, the subgraph can be efficiently obtained, as the graph's edges are known from the initialization and updated with every subsequent forward pass.
The same procedure can be applied to every subsequent convolutional layer. Hence, the update of the kth layer is limited to the k-hop subgraph of the new vertex. These steps lead to significant computational savings, as demonstrated in Sec. 5.

Graph Pooling Similar to sparse graph convolutions, sparse graph pooling operations are composed of an *initialization* and a *processing* step. During *initialization*, the procedure described in Sec. 3 is applied to the dense input graph $\mathcal{G}$, which results in the coarse output graph $\mathcal{G}_c$. Subsequently, in the *processing* stage, we assign events to the respective voxels where they are triggered, connecting them with nodes in the input graph, and then perform the max operation again for that specific voxel. If a node attribute is changed, we similarly perform the max operation again at the respective voxel. Finally, the output graph $\mathcal{G}_c$ can be efficiently computed by applying Eqs. 3 and 4 on $\mathcal{G}'_c$.

Other Layers Non-graph-based layers such as linear or batch normalization can be sparsely updated similarly, by storing the results of the dense update during initialization and only processing the subset of the input, which changes from the previous input, as described in [35]. However, since these layers are applied at the lowest level, most nodes need to be updated, leading to only small gains in computational efficiency.

4.3. Network Details

While the method described in Sec. 4 would allow to sparsely update any kind of graph convolution, we found that spline convolutions [13] find a balance between computational complexity and predictive accuracy. In contrast to the standard graph convolutions [27] used in [31], spline convolutions maintain spatial information in the encoding by using a B-spline-based kernel function in the positional vertex space. This means that spline convolutions also take the relative position of neighboring nodes into account, a feature which is ignored in standard GNN-based methods like [31]. We use voxel-grid-based max-pooling [50] due to its computational efficiency and simplicity. The method in [50] clusters the graph's vertices by mapping them to a uniformly spaced, spatio-temporal voxel grid, with all vertices in a voxel being assigned to one cluster. In this work we use voxels of size $12 \times 16 \times 16$. For each voxel, a node is sampled, resulting in the nodes of the coarse graph. Evaluating the effect of the clustering method on the overall

Methods	Representation	Async.	N-Caltech101		N-Cars	
			Accuracy ↑	MFLOP/ev ↓	Accuracy ↑	MFLOP/ev ↓
H-First [39]	Spike	✓	0.054	-	0.561	-
HOTS [28]	Time-Surface	✓	0.210	54.0	0.624	14.0
HATS [51]	Time-Surface	✓	0.642	4.3	0.902	0.03
DART [43]	Time-Surface	✓	0.664	-	-	-
YOLE [7]	Event-Histogram	✓	0.702	3659	0.927	328.16
EST [17]	Event-Histogram	✗	**0.817**	4150	0.925	1050
SSC [20]	Event-Histogram	✗	0.761	1621	0.945	321
AsyNet [35]	Event-Histogram	✓	0.745	202	0.944	21.5
NVS-S [31]	Graph	✓	0.670	7.8	0.915	5.2
EvS-S [31]	Graph	✓	0.761	11.5	0.931	6.1
Ours	Graph	✓	0.668	**0.369**	**0.945**	**0.03**

Table 1. Comparison with several asynchronous and dense methods for object recognition. Our graph-based method has the lowest computational complexity overall while achieving state-of-the-art performance. Especially, it obtains the best accuracy on N-Cars [51] with 20 times lower computational complexity, compared to the second-best asynchronous method.

network performance remains open for future work. Furthermore, we sub-sample the input event stream using uniform sampling to a fixed number of events. We found that other, more sophisticated sampling methods, such as non-uniform grid sampling [4], only marginally improved the performance, while being much more costly to compute.

Our model architecture is shown in Figure 2. It consists of 7 graph convolution blocks (see Figure 2, bottom right) and 2 pooling layers. For detailed information about our model architecture, we refer to the supplementary material.

5. Experiments

All experiments within this work have been conducted using the PyG library [12] in the Torch framework [40]. For training, we use the Lightning framework [10].

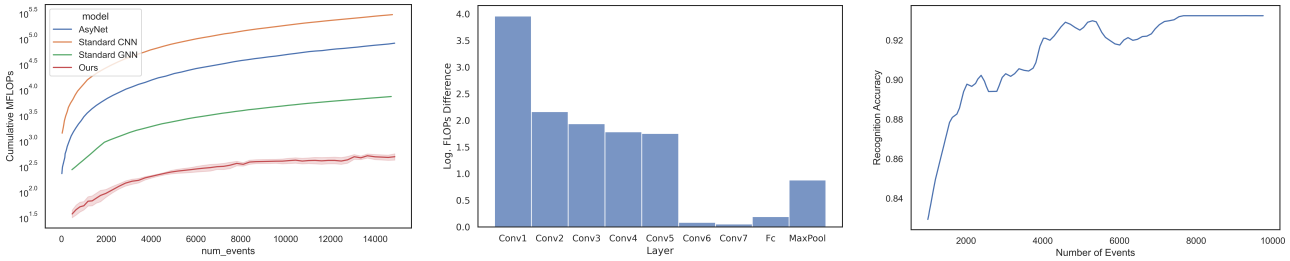
Implementation Details: We used Adam [26] with batch size 16 and an initial learning rate 10^{-3}, which decreases by a factor of 10 after 20 epochs. We apply AEGNN to the tasks of object recognition and object detection.

We have analytically deduced the computational complexity of a forward pass of our model by adding up the computational complexity of each layer. A detailed derivation can be found in the supplementary material.

5.1. Object Recognition

Event-based object recognition tackles the problem of predicting an object category from the event stream and is an important application of event cameras. Due to their high dynamic range and high temporal resolution, event cameras have the potential to detect objects, that would otherwise be undetectable by frame-based methods, especially in low-light conditions, or in conditions with severe motion blur. We demonstrate that our approach is capable of solving this task very efficiently while achieving state-of-the-art recognition performance. The model is evaluated on two diverse datasets: The Neuromorphic N-Caltech101 dataset [38] contains event streams recorded with a real event camera representing 101 object categories in $8,246$ event sequences. each 300 ms long, mirroring the well-known Caltech101 dataset [11] for images. The N-Cars dataset [51] has real events, assigned to either a car or the background. It has $24,029$ event sequences, each being 100 ms long. For training, we use the cross-entropy loss with batch-size 64 (N-Cars) and 16 (N-Caltech101).

Recognition Performance We compare AEGNN against several state-of-the-art methods, both asynchronous and synchronous, with different event representations (Tab. 1). We term methods as synchronous, if they require recomputation at each new event, and asynchronous otherwise. For quantitative comparison, we state the recognition accuracy on the test set. To assess the computational efficiency of each method, we process windows with $25,000$ events and measure the floating-point operations (FLOPs) required to update the prediction for each additional event. H-First [39], HOTS [28], HATS [51] and DART [43] propose hand-designed features for object recognition. Typically, they are computationally efficient, but widely outperformed by our data-driven method. EST [17] is a learnable and dense event representation that is jointly optimized with the downstream task. Although yielding very good recognition accuracy, it introduces additional data processing by using a learned representation and cannot be formulated asynchronously. Thus, our method is $3,000$ times more efficient while achieving a similar predictive performance on N-Cars. AsyNet [35] proposes an asynchronous, sparse network based on event-histograms. Hence, it does not explicitly account for the event's temporal component. Lastly, NVS-S and EvS-B [31] also use a graph-based event representation. In contrast



(a) MFLOPS over events (b) MFLOP savings per layer (c) Accuracy over events

Figure 4. Computational savings of our method compared to a dense CNN, GNN and the method in [35] on N-Cars [51]. We compare the cumulative FLOPS for processing events in sequence (a). Here it is visible that already using a GNN reduces the number of FLOPS by a factor of 10. By additionally using our asynchronous formulation, we further reduce this number by a factor of 30. Additionally, for our method, computation grows much more slowly than for other methods. We show in (b) the FLOPS saved per layer, compared to a dense GNN. We see that our method saves most of the computation in the early and middle layers, where high feature dimensions are used. Finally, we demonstrate the use of our method for early prediction (c). Although the model was trained with $10,000$ events, merely $2,500$ events are required to achieve over 90% accuracy.

to the standard graph convolutions used in EvS-B, the spline convolutions AEGNN encode spatial information. Consequently, our method is 21 times more efficient while achieving a similar accuracy, in comparison to [31].

Scalability While previously assuming a constant number of input events, in the following, we analyze the impact the number of events has on both the computational complex and the recognition accuracy to determine the viability of our method for low-latency prediction. To do this, we compare our model's test set accuracy on N-Cars for different numbers of events, and plot the accuracy and required cumulative computation in Figs. 4 (a) and (c). To highlight the efficiency of our method, we also plot the required number of FLOPs for the dense GNN, the asynchronous method [35] and its dense, synchronous variant. Our proposed method outperforms [35] in terms of accuracy (Tab.1) and in terms of FLOPs (Fig. 4 (a)), showing a computation reduction by a factor of 300. The computational savings come from the comparably flat architecture and sparse graph representation. Notably, our model does not require the full event stream, that it was trained on, for a correct prediction. As demonstrated in Fig 4 (c), only $5,000$ events are required to achieve state-of-the-art recognition accuracy, further improving the computational efficiency of our method. Moreover, our method takes 30 ± 4.8 kFLOPS/ev for 25'000 events, averaged over all sequences. The low variance indicates a high level of stability.

Model	2000	4000	6000	8000
Standard GNN	3.81	6.69	9.38	11.83
Ours	0.11	0.10	0.23	0.32

Figure 5. Computational effort in MFLOPs per event of our sparse method compared to its dense equivalent, evaluated on NCaltech101. With a higher number of events, and thus increasing complexity of the event graph, the computational gap becomes larger.

5.2. Object Detection

Event-based object detection seeks to classify and detect object bounding boxes from an event stream and is an emerging topic in event-based vision. Especially in nighttime scenarios or when objects travel at high speeds, frame-based object detection degrades due to image degradation, caused by underexposure or severe motion blur. Event cameras by contrast do not suffer from these issues and are thus viable alternatives in these cases. We apply our framework to this task and validate our approach on two challenging datasets: the N-Caltech101 dataset [38], see Sec. 5.1, and the Gen1 dataset [8]. While N-Caltech101 contains only one bounding box per sample, it contains 101 classes, making it a difficult classification task. By contrast, Gen1 targets an automotive scenario in an urban environment with annotated pedestrians and cars. With $228,123$ bounding boxes for cars and $27,658$ for pedestrians, the Gen1 dataset is much larger. To avoid the well-known over-smoothing problem of GNNs [30], we adopt the same backbone as for the recognition task but use a YOLO-based object detection head [45], as illustrated in Fig. 2. Similar to [45] we use a weighted sum of class, bounding box offset and shape as well as prediction confidence losses.

Detection Performance To evaluate the performance of our model, we use the eleven-point mean average precision (mAP) [32] score as well as the computational complexity per event, as described in Sec. 5.1. We compare with synchronous and asynchronous state-of-the-art methods and present the results in Tab.2. Qualitative results of our object detector on N-Caltech101 and the Gen1 dataset are shown in Fig.6 We reimplement NVS-S [31], as open-source code is not available.

Our method outperforms NVS-S [31] by 7.7%, while using 21 times less computation. This is because NVS-S uses standard graph convolutions, and thus have a re-

Methods	Representation	Async.	N-Caltech101		Gen1	
			mAP ↑	MFLOP/ev ↓	mAP ↑	MFLOP/ev ↓
YOLE [7]	Event-Histogram	✓	0.398	3682	-	-
Asynet [35]	Event-Histogram	✓	**0.643**	200	0.129	205
RED [42]	Event-Volume	✗	-	-	**0.40**	4712
NVS-S [31]	Graph	✓	0.346*	7.8	0.086*	7.8
Ours	Graph	✓	0.595	**0.37**	0.163	**0.39**

Table 2. Comparison with several asynchronous and dense methods for object detection. The method in [31] was re-implemented and trained by us, as [31] only reports results for the object recognition task.

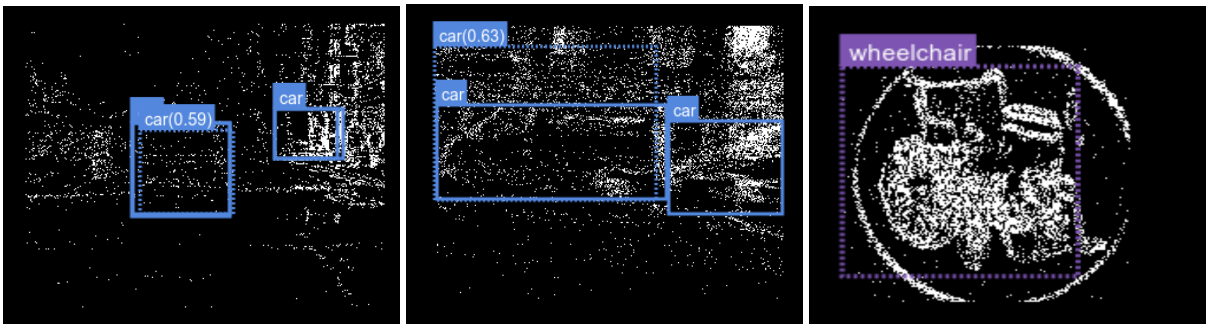


Figure 6. Qualitative results of the object detection performed by our model on Gen1 [8] and N-Caltech101 [38] dataset. Our predictions are shown as a dashed line, the labels as solid line.

ceptive field that is limited to their direct neighborhood, which deteriorates detection performance. Compared to RED [42], we achieve a lower accuracy but outperform the method by a significant margin: While our method uses 0.39 MFLOPs/ev, [42] uses 4712 MFLOPs/ev. This is because [42] uses a dense, synchronous recurrent network, and it is thus not capable of event-by-event processing. Finally, AsyNet [35] outperforms AEGNN on N-Caltech101 by 4.8 mAP, but we show a 3.4 mAP higher performance on Gen1. While performances are comparable, we achieve this with 520-540 times fewer MFLOPs per event.

Timing Experiments We timed our method, implemented in Python and CUDA, on an Nvidia Quadro RTX. To construct the graph we implemented the radius search algorithm in [31] in CUDA, which takes 2 ms to generate a graph with 2,500 nodes. For processing one event in an event graph of $4,000$ from N-Caltech101, the dense update requires $167ms$, our sparse method $92ms$. For $25,000$ events, the dense GNN needs $1014ms$, our sparse method $129ms$, an improvement by a factor of 8. A dense CNN with the same input requires $202ms$. While our method is only 1.5 times faster than a CNN, we point out here that CNNs have highly optimized implementations in the PyTorch Library [40]. However, we expect that if implemented on suitable hardware, such as FPGA or IPU [24] processors, the reported computation reduction will lead to significant reductions in latency and power consumption, as was already demonstrated in [16].

6. Conclusion

While event-based vision has made significant strides by adopting standard learning-based methods based on CNNs, these discard the spatio-temporal sparsity of events, which leads to wasteful computation. For this reason, geometric-learning approaches for event-based vision have gained in popularity. In this work, we introduced AEGNNs, which model events as evolving spatio-temporal graphs and formulate efficient update rules for each new event that restrict recomputation of network activations only to a few nodes, which are propagated to lower layers. We applied AEGNNs to the tasks of object recognition and detection. While in object recognition we achieved an up to a 200-fold reduction in computational complexity (FLOPs), for object detection we achieved an up to 21-fold reduction, while outperforming asynchronous methods by 3.4% mAP. We showed that this computation reduction speeds up processing latency by a factor of 8 compared to dense GNNs. We believe that, if our method is implemented on specialized hardware such as FPGA or IPUs [24], we will see additional reductions in latency and a significant reduction in power consumption.

7. Acknowledgment

This work was supported Huawei, and as a part of NCCR Robotics, a National Centre of Competence in Research, funded by the Swiss National Science Foundation (grant number 51NF40_185543).

References

[1] Alessandro Aimar, Hesham Mostafa, Enrico Calabrese, Antonio Rios-Navarro, Ricardo Tapiador-Morales, Iulia-Alexandra Lungu, Moritz B. Milde, Federico Corradi, Alejandro Linares-Barranco, Shih-Chii Liu, and Tobi Delbrück. NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE Trans. Neural Netw. Learn. Syst.*, 30(3):644–656, Mar. 2019. 2

[2] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A low power, fully event-based gesture recognition system. In *Conference of Computer Vision and Pattern Recognition (CVPR)*, pages 7388–7397, 2017. 2

[3] Patrick Bardow, Andrew J. Davison, and Stefan Leutenegger. Simultaneous optical flow and intensity estimation from an event camera. In *Conference of Computer Vision and Pattern Recognition (CVPR)*, pages 884–892, 2016. 2, 4

[4] Yin Bi, Aaron Chadha, Alhabib Abbas, Eirina Bourtsoulatze, and Yiannis Andreopoulos. Graph-based object classification for neuromorphic vision sensing. *IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019. 3, 4, 6

[5] Yin Bi, Aaron Chadha, Alhabib Abbas, Eirina Bourtsoulatze, and Yiannis Andreopoulos. Graph-based spatio-temporal feature learning for neuromorphic vision sensing. *IEEE Transactions on Image Processing*, 29:9084–9098, 2020. 3

[6] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. Hyperanf: Approximating the neighbourhood function of very large graphs on a budget. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 625–634, New York, NY, USA, 2011. Association for Computing Machinery. 5

[7] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Asynchronous convolutional networks for object detection in neuromorphic cameras. In *CVPRW*, 2019. 6, 8

[8] Pierre de Tournemire, Davide Nitti, Etienne Perot, Davide Migliore, and Amos Sironi. A large scale event-based detection dataset for automotive. *arXiv e-prints*, abs/2001.08499, 2020. 7, 8

[9] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5(40):eaaz9712, 2020. 2, 4

[10] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 2019. 6

[11] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *CVPRW*, 2004. 6

[12] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 6

[13] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. *Conference of Computer Vision and Pattern Recognition (CVPR)*, 2018. 5

[14] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE T-PAMI.*, 2020. 2

[15] Guillermo Gallego, Jon E. A. Lund, Elias Mueggler, Henri Rebecq, Tobi Delbruck, and Davide Scaramuzza. Event-based, 6-DOF camera tracking from photometric depth maps. *IEEE T-PAMI.*, 40(10):2402–2412, Oct. 2018. 3

[16] Chang Gao, Antonio Rios-Navarro, Xi Chen, Shih-Chii Liu, and Tobi Delbruck. Edgedrnn: Recurrent neural network accelerator for edge inference. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 10(4):419–432, 2020. 2, 8

[17] Daniel Gehrig, Antonio Loquercio, Konstantinos G. Derpanis, and Davide Scaramuzza. End-to-end learning of representations for asynchronous event-based data. In *IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019. 2, 3, 6

[18] Mathias Gehrig, Willem Aarents, Daniel Gehrig, and Davide Scaramuzza. Dsec: A stereo event camera dataset for driving scenarios. In *IEEE RA-L*, December 2021. 2

[19] Mathias Gehrig, Sumit Bam Shrestha, Daniel Mouritzen, and Davide Scaramuzza. Event-based angular velocity regression with spiking networks. *IEEE International Conference on Robotics and Automation (ICRA)*, 2020. 2

[20] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. In *Conference of Computer Vision and Pattern Recognition (CVPR)*, pages 9224–9232, 2018. 6

[21] Jesse J. Hagenaars, Federico Paredes-Vallés, Sander M. Bohté, and Guido C. H. E. de Croon. Evolved neuromorphic control for high speed divergence-based landings of mavs. *IEEE Robotics and Automation Letters*, 5(4):6239–6246, 2020. 2

[22] Javier Hidalgo-Carrió, Daniel Gehrig, and Davide Scaramuzza. Learning monocular dense depth from events. *IEEE Int. Conf. 3D Vis. (3DV)*, 2020. 2

[23] Giacomo Indiveri, Bernabe Linares-Barranco, Tara Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain SAÏGHI, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, and Kwabena Boahen. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5:73, 2011. 2

[24] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. Dissecting the graphcore ipu architecture via microbenchmarking. *ArXiv*, abs/1912.03413, 2019. 8

[25] Hanme Kim, Stefan Leutenegger, and Andrew J. Davison. Real-time 3D reconstruction and 6-DoF tracking with an event camera. In *European Conference of Computer Vision (ECCV)*, pages 349–364, 2016. 3

[26] Diederik P. Kingma and Jimmy L. Ba. Adam: A method for stochastic optimization. *(ICLR)*, 2015. 6

[27] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *(ICLR)*, 2017. 5

[28] Xavier Lagorce, Garrick Orchard, Francesco Gallupi, Bertram E. Shi, and Ryad Benosman. HOTS: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE T-PAMI.*, 39(7):1346–1359, July 2017. 2, 3, 6

[29] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Front. Neurosci.*, 10:508, 2016. 2

[30] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *AAAI*, abs/1801.07606, 2018. 7

[31] Yijin Li, Han Zhou, Bangbang Yang, Ye Zhang, Zhaopeng Cui, Hujun Bao, and Guofeng Zhang. Graph-based asynchronous event processing for rapid object recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 934–943, October 2021. 1, 3, 5, 6, 7, 8

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference of Computer Vision (ECCV)*, pages 740–755. 2014. 7

[33] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. In *Science Robotics*, October 2021. 4

[34] Ana I. Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Conference of Computer Vision and Pattern Recognition (CVPR)*, pages 5419–5427, 2018. 2, 3

[35] Nico A. Messikommer, Daniel Gehrig, Antonio Loquercio, and Davide Scaramuzza. Event-based asynchronous sparse convolutional networks. In *European Conference of Computer Vision (ECCV)*, 2020. 2, 3, 4, 5, 6, 7, 8

[36] Srinjoy Mitra, Stefano Fusi, and Giacomo Indiveri. Real-time classification of complex patterns using spike-based learning in neuromorphic vlsi. *IEEE Transactions on Biomedical Circuits and Systems*, 3(1):32–42, 2009. 2

[37] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE Transactions on Biomedical Circuits and Systems*, 12(1):106–122, 2018. 2

[38] Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Front. Neurosci.*, 9:437, 2015. 6, 7, 8

[39] Garrick Orchard, Cedric Meyer, Ralph Etienne-Cummings, Christoph Posch, Nitish Thakor, and Ryad Benosman. HFirst: A temporal approach to object recognition. *IEEE T-PAMI.*, 37(10):2028–2040, 2015. 2, 3, 6

[40] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Workshops*, volume 8, 2017. 6, 8

[41] José A. Perez-Carrasco, Bo Zhao, Carmen Serrano, Begoña Acha, Teresa Serrano-Gotarredona, Shouchun Chen, and Bernabé 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 T-PAMI.*, 35(11):2706–2719, Nov. 2013. 2

[42] Etienne Perot, Pierre de Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi. Learning to detect objects with a 1 megapixel event camera. *Adv. Neural Inf. Process. Syst.*, 2020. 2, 4, 8

[43] Bharath Ramesh, Hong Yang, Garrick Orchard, Ngoc Anh Le Thi, and Cheng Xiang. DART: distribution aware retinal transform for event-based cameras. *arXiv e-prints*, Oct. 2017. 6

[44] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. High speed and high dynamic range video with an event camera. *IEEE T-PAMI.*, 2019. 2, 3

[45] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Conference of Computer Vision and Pattern Recognition (CVPR)*, 2016. 7

[46] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. In *Proc. Int. Conf. Mach. Learning Workshops (ICMLW)*, 2020. 3

[47] Nitin Sanket, Chethan M. Parameshwara, Chahat Singh, Ashwin Varghese Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, and Yiannis Aloimonos. 05 2020. 2, 4

[48] Yusuke Sekikawa, Kosuke Hara, and Hideo Saito. EventNet: Asynchronous recursive event processing. In *Conference of Computer Vision and Pattern Recognition (CVPR)*, 2019. 2, 3

[49] Sumit Bam Shrestha and Garrick Orchard. SLAYER: Spike layer error reassignment in time. In *Adv. Neural Inf. Process. Syst.*, 2018. 2

[50] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. *Conference of Computer Vision and Pattern Recognition (CVPR)*, 2017. 5

[51] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. HATS: Histograms of averaged time surfaces for robust event-based object classification. In *Conference of Computer Vision and Pattern Recognition (CVPR)*, pages 1731–1740, 2018. 2, 3, 6, 7

[52] S. Sun, G. Cioffi, C. de Visser, and D. Scaramuzza. Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events. *IEEE Robotics and Automation Letters*, 6(2):580–587, 2021. 2

[53] Stepan Tulyakov, Daniel Gehrig, Stamatios Georgoulis, Julius Erbach, Mathias Gehrig, Yuanyou Li, and Davide Scaramuzza. Time lens: Event-based video frame interpolation. In *Conference of Computer Vision and Pattern Recognition (CVPR)*, pages 16155–16164, 2021. 2

[54] Xiang Zhang, Wei Liao, Lei Yu, Wen Yang, and Gui-Song Xia. Event-based synthetic aperture imaging with a hybrid

network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14235–14244, June 2021. 2

[55] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 2020. 3

[56] Alex Zihao Zhu, Dinesh Thakur, Tolga Ozaslan, Bernd Pfrommer, Vijay Kumar, and Kostas Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3D perception. *IEEE RA-L*, 3(3):2032–2039, July 2018. 2

[57] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. EV-FlowNet: Self-supervised optical flow estimation for event-based cameras. In *Robotics: Science and Systems (RSS)*, 2018. 2, 3

[58] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. Unsupervised event-based learning of optical flow, depth, and egomotion. In *Conference of Computer Vision and Pattern Recognition (CVPR)*, 2019. 2