Graph-based Spatio-Temporal Feature Learning for Neuromorphic Vision Sensing

Yin Bi, Aaron Chadha, Alhabib Abbas, Eirina Bourtsoulatze and Yiannis Andreopoulos

***Abstract*—Neuromorphic vision sensing (NVS) devices represent visual information as sequences of asynchronous discrete events (a.k.a., "spikes") in response to changes in scene reflectance. Unlike conventional active pixel sensing (APS), NVS allows for significantly higher event sampling rates at substantially increased energy efficiency and robustness to illumination changes. However, feature representation for NVS is far behind its APS-based counterparts, resulting in lower performance in high-level computer vision tasks. To fully utilize its sparse and asynchronous nature, we propose a compact graph representation for NVS, which allows for end-to-end learning with graph convolution neural networks. We couple this with a novel end-to-end feature learning framework that accommodates both appearance-based and motion-based tasks. The core of our framework comprises a spatial feature learning module, which utilizes residual-graph convolutional neural networks (RG-CNN), for end-to-end learning of appearance-based features directly from graphs. We extend this with our proposed Graph2Grid block and temporal feature learning module for efficiently modelling temporal dependencies over multiple graphs and a long temporal extent. We show how our framework can be configured for object classification, action recognition and action similarity labeling. Importantly, our approach preserves the spatial and temporal coherence of spike events, while requiring less computation and memory. The experimental validation shows that our proposed framework outperforms all recent methods on standard datasets. Finally, to address the absence of large real-world NVS datasets for complex recognition tasks, we introduce, evaluate and make available the American Sign Language letters (ASL-DVS), as well as human action dataset (UCF101-DVS, HMDB51-DVS and ASLAN-DVS).**

***Index Terms*—Neuromorphic vision sensing, spatio-temporal feature learning, graph convolutional neural networks, object classification, human action recognition**

I. Introduction

With the prevalence and advances of CMOS active pixel sensing (APS) and deep learning, researchers have achieved good performance in APS-based computer vision tasks, such as object detection [1], [2], object recognition [3], [4] and action recognition [5], [6] . However, APS cameras suffer from limited frame rate, high redundancy between frames, blurriness due to slow shutter adjustment under varying illumination, and high power requirements [7] which limit the effectiveness of APS-based frameworks. To solve these problems, researchers have devised neuromorphic vision sensing (NVS) sensors such as the iniLabs DAVIS cameras [8] and the Pixium Vision ATIS cameras [9], which are inspired by the photoreceptor-bipolar-ganglion cell information flow in mammalian vision. NVS devices output events (i.e., spikes) asynchronously in response to a change in illumination. That is, when the transient change of illumination intensity in a scene exceeds a certain threshold, an event is generated. The output of the NVS camera is represented asynchronously as a collection of tuple sequences, referred to as an Address Event Representation (AER) [10] that is the standard interfacing protocol for neuromorphic engineering. Each tuple corresponds to one event and it comprises: the spatial coordinates, the timestamp and the polarity (i.e., ON or OFF) of the event. The polarity indicates an increase (ON) or decrease (OFF) in illumination intensity, where ON/OFF can be represented via +1/-1 values. The operation of an NVS camera is illustrated at the top part of Fig. 1, where impulses represent the generated events.

In contrast to APS devices (i.e., conventional cameras) that use a fixed-sampling rate in order to record entire frames at fixed frame rates, each CMOS array position (a.k.a., pixel) in an NVS sensor optimizes its own sampling rate independently, according to the change it detects in illumination. Therefore, the events produced from the entire NVS pixel array are sparse and asynchronous and can be represented as a space-time volume over a given time interval. This is illustrated at the bottom part of Fig. 1, where the neuromorphic event stream is overlaid with the corresponding RGB frames recorded at the video framerate; events are plotted according to their spatio-temporal coordinates and color coded as blue (OFF) and red (ON). Notably, there are many more intermediate events between the RGB frames, which indicates the substantially higher framerate achievable with an NVS camera and asynchronous outputs. Furthermore, the asynchronicity removes the data redundancy from the scene, which reduces the power requirement to 10mW, compared to several hundreds of mW for APS cameras. Remarkably, NVS devices achieve this with microsecond-level latency and robustness to uncontrolled lighting conditions, as no synchronous global shutter is used.

Beyond event sparsity and asynchronicity, neuromorphic event streams are naturally encoding spatio-temporal motion information [7]; as such, they are extremely adaptable to tasks related to moving objects such as action analysis/recognition, object tracking or high-speed moving scenes. We, therefore, look to perform feature learning directly on the raw neuro-

YB, AC, AA and YA are with the Electronic and Electrical Engineering Department, University College London, Roberts Building, Torrington Place, London, WC1E 7JE, UK (e-mail: {yin.bi.16, aaron.chadha.14, alhabib.abbas.13, i.andreopoulos}@ucl.ac.uk). EB is with the School of Computer Science and Electronic Engineering, University of Essex, Wiwenhoe Park, Colchester, CO4 3SQ, UK (e-mail: e.bourtsoulatze@essex.ac.uk). This work was supported by EPSRC, grants: EP/R025290/1 and EP/P02243X/1 and also by the EC H2020 programme, project ENVISION 750254. Parts of this work were presented at 2019 IEEE International Conference on Computer Vision, IEEE ICCV, Seoul, Korea.

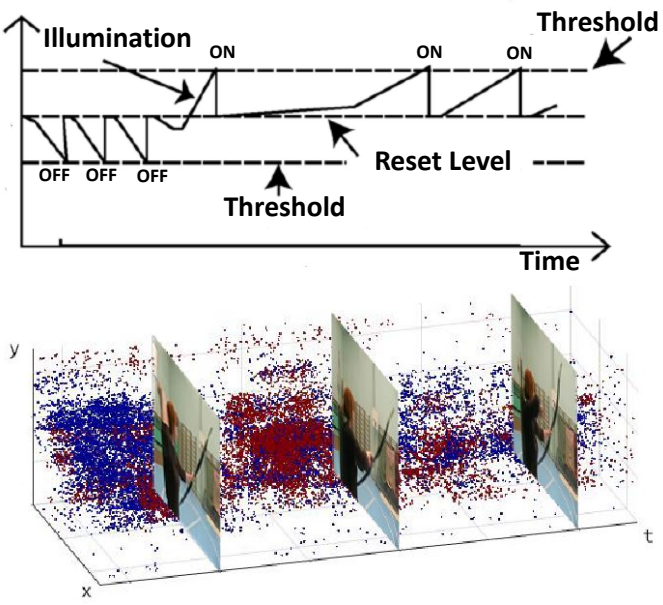


Fig. 1: (top): NVS camera operation. (bottom): Recording of archery action captured by APS and NVS cameras. APS cameras capture frames at a fixed rate, while NVS cameras output a stream of events. (Red:ON, Blue:OFF)

morphic events. Unfortunately, effective methods for representation learning on neuromorphic events to solve complex computer vision tasks are currently limited and outperformed by their APS-based counterparts. This is partly due to limited research in the NVS domain, as well as a lack of NVS data with reliable annotations to train and test on [7], [11]. Yet, more so, the sheer abundance of asynchronous and sparse events means that feature learning directly on events can be particularly cumbersome and unwieldy. Thus far, most approaches have attempted to solve this issue by either artificially grouping events into frame forms [12], [13] or deriving complex feature descriptors [14], [15], which do not always provide for good representations for complex tasks like object classification. Moreover, such approaches dilute the advantages of the asynchronicity of NVS streams by limiting the frame-rate, and may be sensitive to the noise and change of camera motion or viewpoint orientation. Finally, these methods fail to model long temporal event dependencies explicitly, thus rendering them less viable for motion-based tasks.

More recent methods on feature representation have employed end-to-end feature learning, where a convolutional neural network (CNN) [16], [17] or spiking neural network (SNN) [18], [19] is trained to learn directly from raw observations. While these methods show great promise, CNN-based learning methods require event grouping into frames. Therefore, they suffer from the same drawbacks as above. On the other hand, the biggest challenge of SNNs is that the activation functions of spiking neurons are asynchronous and non-differentiable in time. Hence, SNN-based methods cannot use well-established gradient-based learning rules. This makes SNN-based methods complex to train, resulting in lower performance compared to gradient-based alternatives. In addition, SNN inference requires bespoke hardware, which is less readily available than CPUs and GPUs. These difficulties are compounded by the fact that, from the sensing side, neuromorphic (spike) based sensors activate in an asynchronous manner in time, thereby producing data streams at irregular space-time coordinates, which depend on the scene activity. Graph-based processing is an ideal mechanism to deal with such asynchronous space-time data capture mechanisms. Therefore, instead of using CNNs or SNNs, we propose to leverage on graph-based learning, by training an end-to-end feature learning framework directly on neuromorphic events. By representing events as graphs, we are able to maintain event asynchronicity and sparsity, while performing training with traditional gradient-based backpropagation. To the best of our knowledge, this is the first attempt to represent neuromorphic spike events as graphs, which allows to use graph convolutional neural networks for end-to-end feature learning directly on neuromorphic events. Building partly on our previous work [20], our proposed graph based framework is able to accommodate both appearance and motion-based tasks; in this paper, we focus on object classification, action recognition and action similarity labelling as representative tasks. For object classification, we design a spatial feature learning module, comprising graph convolutional layers and graph pooling layers for processing a single input event graph. For action recognition and action similarity labeling, we extend this module with temporal feature learning, in order to learn a spatio-temporal representation over the entire input. Specifically, we introduce a Graph2Grid block for aggregating a sequence of graphs over a long temporal extent. Each event graph in the sequence is first processed by a spatial feature learning module; the mapped graphs are then converted to grid representation by the Graph2Grid block and the resulting frames are stacked, for processing with any conventional 2D or 3D CNNs. This is inspired by recent work in APS-based action recognition [21] that processes multiple RGB frames with 2D CNNs and aggregates the learned representations with a 3D convolution fusion and pooling.

In order to address the lack of NVS data for evaluation, we introduce the largest sourced NVS dataset for object classification, which we refer to as ASL-DVS. The task is to classify hand recordings as one of 24 letters from the American Sign Language (ASL). For action recognition and action similarity labeling, we leverage existing APS-based datasets such as UCF101 [22], HMDB51 [23] and ASLAN [24], and convert these to the NVS domain by recording a playback of each dataset captured from a display with a DAVIS240c NVS camera. The generated NVS datasets, UCF101-DVS, HMDB51-DVS and ASLAN-DVS, include more content than any previous NVS dataset in these action-based tasks.

We evaluate our framework on object classification, action recognition and action similarity labelling, and show that our framework achieves state-of-the-art results on both tasks compared to recent work on conventional frame-based approaches. We summarize our contributions as follows:

1) We propose a novel graph based representation for neuromorphic events, not only maintaining asynchronicity and sparsity of events, but also allowing for fast end-to-end graph based training and inference. To the best of our knowledge, this paper and its corresponding conference

paper [20] are the first graph representations for NVS streams.
2) Apart from graph representation and object classification tasks that were also discussed in our recent work [20], in this paper, we introduce a novel Graph2Grid block and a temporal feature learning module for efficiently modelling coarse temporal dependencies over multiple graphs. We evaluate performance of the learning framework on action recognition and action similarity labeling.
3) We introduce new datasets for action recognition (UCF101-DVS and HMDB51-DVS) and action similarity labeling (ASLAN-DVS) to address the lack of NVS data for training and inference, and make these available to the research community. This extends the NVS datasets proposed in our corresponding conference paper [20] and provides a comprehensive set of benchmark datasets for evaluation of spatio-temporal learning with NVS representations.

In Section II we review related work. Section III details our method for graph-based spatio-temporal feature learning network. Three downstream applications (object classification, human action recognition and action similarity labeling) are presented in Section IV, where "downstream" denotes the dependency of the applications on the learned features.Section V concludes the paper.

II. Related Work

In the field of neuromorphic vision, recent literature focuses on two types of feature representation: handcrafted feature extraction and end-to-end trainable feature learning. Handcrafted feature descriptors are widely used by neuromorphic vision community. Some of the most common are corner detectors and line/edge extraction [25], [26]. While these efforts were promising early attempts for NVS-based object classification, their performance does not scale well when considering complex datasets. Inspired by their frame-based counterparts, optical flow methods have been proposed as feature descriptors for NVS [27], [28]. For a high-accuracy optical flow, these methods have very high computational requirements, which diminishes their usability in real-time applications. In addition, due to the inherent discontinuity and irregular sampling of NVS data, deriving compact optical flow representations with enough descriptive power for accurate classification and tracking still remains a challenge [27]. Lagorce *et al.* proposed event based spatio-temporal features called time-surfaces [29]. This is a time oriented approach to extract spatio-temporal features that are dependent on the direction and speed of motion of the objects. Inspired by time-surfaces, Sironi *et al.* proposed a higher-order representation for local memory time surfaces that emphasizes the importance of using the information carried by past events to obtain a robust representation [14].These descriptors are very sensitive to noise and strongly depend on the type of object motion in scene. Moreover, they fail to take temporal information into account and maintain a representation of dynamics over a long time. Thus, they can only be used for static object recognition, and not for long temporal applications such as action recognition evaluated in this work.

End-to-end feature learning for NVS-based tasks consists of two types of approaches: frame-based and event-based. The main idea of frame-based methods is to convert the neuromorphic events into synchronous frames of spike events, on which conventional computer vision techniques can be applied for the feature learning. Zhu *et al.* [12] introduced a four-channel image form with the same resolution as the neuromorphic vision sensor. Inspired by the functioning of spiking neural networks (SNNs) to maintain memory of past events, leaky frame integration has been used in recent work [13], [30], where the corresponding position of the frame is incremented by a fixed amount when a event occurs at the same event address. Amir *et al.* use a cascade of temporal filters to process the events, which is regarded as stacking frames, and then feed these frames into a CNN [17]. Similary, Ghosh *et al.* partitioned events into a three-dimensional grid of voxels where spatio-temporal filters are used to learn the features, and learnt features are fed as input to CNNs for action recognition [16]. Chadha *et al.* [31] generated frames by summing the polarity of events in each address as pixel, then fed them into a multi-modal teacher-student framework for action recognition. While useful for early-stage attempts, these frame-based methods are not well-suited for the neuromorphic event's sparse and asynchronous nature since the frame sizes that need to be processed are substantially larger than those of the original NVS streams. The advantages of event-based sensors are diluted if their event streams are cast back into synchronous frames for the benefit of conventional processors downstream, thus not providing efficient and power-saving learning systems.

The second type of end-to-end feature learning methods are event-based methods. The most commonly used architecture relies on spiking neural networks (SNNs) [18], [19] for inference. While SNNs are theoretically capable of learning complex representations, they still fail to achieve the performance of gradient-based methods due to the lack of suitable training algorithms. Essentially, since the activation functions of spiking neurons are not differentiable, SNNs are not able to leverage on popular training methods such as backpropagation. To address this, researchers currently follow a hybrid approach [32], [33]: a neural network is trained off-line using continuous/rate-based neuronal models with state-of-the-art supervised training algorithms; then, the trained architecture is mapped to an SNN. However, until now, despite their substantial implementation advantages at inference, the obtained solutions are complex to train and typically achieve lower performance than gradient-based CNNs. Thus, other directions for event-based feature learning for neuromorphic vision sensing have been also explored. Wang *et al.* interpreted an event sequence as a 3D point cloud in space and time [34], which is hierarchically fed into PointNet [35] to capture the spatio-temproal structure of motion. While providing useful insights, all these methods were tested on simple datasets (e.g., the DVS128 Gesture dataset [17] of gestures and postures) with a small number of classes and clean background. It is, therefore, unlikely that these methods can obtain such high accuracy for real-world scenarios, as they cannot capture long-term temporal dependencies. When applied to complex

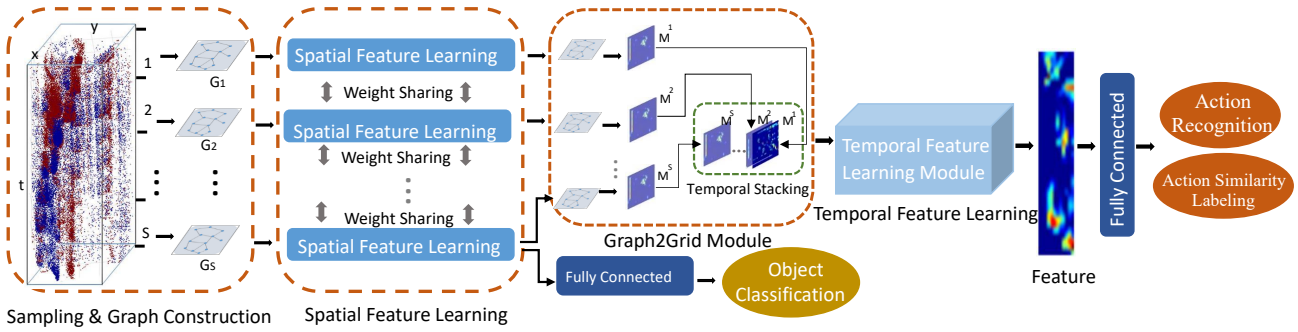


Fig. 2: Framework of graph-based spatio-temporal feature learning for neuromorphic vision sensing. Our framework is able to accommodate both object classification and action recognition/similarity labeling tasks. We first construct S graphs from the event stream (where $S = 1$ for object classification). Each graph is passed through a spatial feature learning module. For object classification, the output of this module is mapped to object classes directly by fully connected layers. For action recognition and action similarity labeling, we model coarse temporal dependencies over multiple graphs by converting to a grid representation via the Graph2Grid module and perform temporal feature learning with a 3D CNN.

datasets (e.g., UCF101_DVS) for human action recognition, the performance of these methods degrades significantly.

III. Methodology

The architecture of our graph-based spatio-temporal feature learning network is illustrated in Fig. 2 and comprises four parts: sampling and graph construction, a spatial feature learning module, a graph-to-frame mapping module and a temporal feature learning module. For object classification, a single graph is constructed, whereas for action-based tasks with longer temporal extent, multiple graphs are extracted over the event stream duration. Specifically, neuromorphic events are firstly sampled and represented by a sequence of graphs. Graphs are then individually processed by a spatial feature learning module, which consists of multiple graph convolution and pooling layers to map the input to a coarser graph encoding. For object classification, we obtain a single graph encoding that we pass to a single fully connected layer for prediction. Conversely, for action recognition and action similarity labeling, we obtain multiple graph encodings. As such, we convert the graphs to a grid representation with a graph-to-frame mapping module which we denote as Graph2Grid, and stack the resulting frames for temporal feature learning with a 3D CNN. In this way, we are able to effectively and efficiently learn spatio-temporal features for motion-based applications, such as action recognition. We provide more details on each component of the framework in the following sections.

A. Graph Construction

Given a NVS sensor with spatial address resolution of $H \times W$, we express a volume of events V produced by a NVS camera as a tuple sequence:

$$\{e_i\}_N = \{x_i, y_i, t_i, p_i\}_N \tag{1}$$

where $(x_i, y_i) \in \{1, 2, \dots H\} \times \{1, 2, \dots W\}$ is the spatial address at which the spike event occurred, t_i is the timestamp indicating when the event was generated and is presented in units of microseconds (μs), $p_i \in \{+1, -1\}$ is the event polarity (with +1/-1 signifying ON/OFF events respectively), and N is the total number of events.

To reduce the storage and computational cost, we use non-uniform grid sampling [36] to sample a subset of M representative events from the N total events in the sequence: $\{e_i\}_M \subset \{e_i\}_N$, where $M \ll N$. Specifically, we group k neighbouring events in the sequence into space-time volumes based on their spatio-temporal distance. Then from each space-time volume, we extract one event. In other words, if we consider $\mathbf{s}\{e_i\}_{i=1}^{k}$ to be a space-time volume containing k events, then only one event e_i ($i \in [1, k]$) is randomly sampled in this volume. We then define the M sampled events $\{e_i\}_{\{M\}}$ on a directed graph $\mathrm{G} = \{\nu, \varepsilon, \mathrm{U}\}$, with ν being the set of vertices, ε the set of the edges, and U the coordinates of the nodes that locally define the spatial relations of the nodes. The sampled events are independent and not linked, therefore, we regard each event $e_i : (x_i, y_i, t_i, p_i)$ as a node in the graph, such that $\nu_i : (x_i, y_i, t_i)$, with $\nu_i \in \nu$. We define the connectivity of nodes in the graph based on the radius-neighborhood-graph strategy, which is a commonly used term in graph theory [37]. The neighborhood construction and connectivity steps involved in the graph construction are illustrated in Fig.3 and are performed as follows. Nodes ν_i and ν_j are connected with an edge only if their weighted Euclidean distance $d_{i,j}$ is less than radius distance R. For two spike events e_i and e_j, the Euclidean distance between them is defined as the weighted spatio-temporal distance:

$$d_{i,j} = \sqrt{\alpha(|x_i - x_j|^2 + |y_i - y_j|^2) + \beta|t_i - t_j|^2} \leq \mathrm{R} \tag{2}$$

where α and β are weight parameters compensating for the difference in spatial and temporal grid resolution (timing accuracy is significantly higher in NVS cameras than spatial grid resolution). To limit the size of the graph, we constrain the maximum connectivity degree for each node by parameter $D_{\max}$. We subsequently define $u(i, j)$ for node i, with connected node j, as $u(i, j) = [|x_i - x_j|, |y_i - y_j|] \in \mathrm{U}$.

After connecting all nodes of the graph $\mathrm{G} = \{\nu, \varepsilon, \mathrm{U}\}$ via the above process, we consider the polarity of events as a signal that resides on the nodes of the graph G. In other words, we define the input feature for each node i, as $f^{(0)}(i) = p_i \in \{+1, -1\}$.

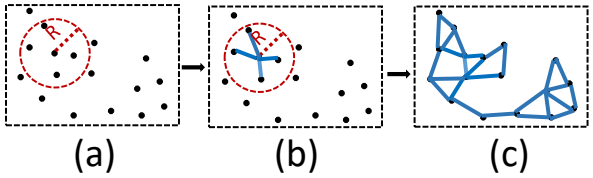


Fig. 3: Visualization of graph construction: (a) finding the neighboring events within radius R; (b) connecting an event to its neighbors within radius R; (c) a constructed graph from raw events.

We introduce the parameter S to represent the number of graphs constructed from one sequence of events. In other words, S partitions are extracted from an event sequence and a graph is constructed for each partition. Given that object classification is appearance-based and typically only requires a short temporal extent, we set $S = 1$. Specifically, we randomly extract T_{vol} length events over the entire event stream to construct a graph. Conversely, for action recognition and action similarity labeling, we divide the event stream into S volumes with the same time duration T/S, where T is the time duration of entire sequence of events.We then construct a graph for each volume in which $T_{\text{vol}} < T/S$ length events are randomly extracted to construct a graph, giving us a set of graphs $\mathcal{G} = \{\text{G}_n\}_{n=1}^{S}$. In this way, we efficiently model coarse temporal dependencies over the duration of the sample, without constructing a single large and substantially complex graph. The graphs can thus be processed individually by our spatial feature learning module before fusion with our Graph2Grid module and temporal feature learning. This is inspired by recent work on action recognition with RGB frames [21], which fuses representations over coarse temporal scales with 3D convolutions and pooling; indeed, our graph-based framework is substantially more lightweight and does not suffer from the limitations of active pixel sensing.

B. Spatial Feature Learning Module

The constructed graphs are first fed individually into a spatial feature learning module, where our framework learns appearance information. An illustration of the components of the spatial feature learning module is given in Fig.4. According to the common architectural pattern for feed-forward neural networks, these graph convolutional neural networks are built by interlacing graph convolution layers and graph pooling layers, where the graph convolution layer performs a non-linear mapping and the pooling layer reduces the size of the graph.

Graph convolution generalizes the convolutionl operator to the graph domain. Similar to frame-based convolution, graph convolution can be categorized into two types: spectral and spatial. Spectral convolution [38], [39] defines the convolution operator by decomposing a graph in the spectral domain and then applying a spectral filter on the spectral components. However, this operation requires identical graph input and handles the whole graph simultaneously, so it is not suitable for the variable and large graphs constructed from NVS. On the other hand, spatial convolution [40], [41] aggregates a new feature vector for each vertex, using its neighborhood information weighted by a trainable kernel function. Because of this property, we consider spatial convolution operation as a better choice when dealing with graphs from NVS.

Similar to conventional frame-based convolution, spatial convolution operations on graphs are also a one-to-one mapping between kernel function and neighbors at relative positions w.r.t. the central node of the convolution. Let i denote a node of the graph with feature $f(i)$, $\mathcal{N}(i)$ denote the set of neighbors of node i and $g(u(i,j))$ denote the weight parameter constructed from the kernel function $g(\cdot)$. The graph convolution operator $\otimes$ for this node can then be written in the following general form:

$$(f \otimes g)(i) = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} f(j) \cdot g(u(i,j)) \tag{3}$$

where $|\mathcal{N}(i)|$ is the cardinality of $\mathcal{N}(i)$. We can generalize (3) to multiple input features per node. Given the kernel function $\mathbf{g} = (g_1, ..., g_l, ..., g_{M_{in}})$ and input node feature vector $\mathbf{f}_l$, with M_{in} feature maps indexed by l, the spatial convolution operation $\otimes$ for the node i with M_{in} feature maps is defined as:

$$(\mathbf{f} \otimes \mathbf{g})(i) = \frac{1}{|\mathcal{N}(i)|} \sum_{l=1}^{M_{in}} \sum_{j \in \mathcal{N}(i)} f_l(j) \cdot g_l(u(i,j)) \tag{4}$$

The kernel function $\mathbf{g}$ defines how to model the coordinates U. The content of U is used to determine how the features are aggregated and the content of $f_l(j)$ defines what is aggregated. As such, several spatial convolution operations [40]–[42] on graphs were proposed by using different choice of kernel functions. Among them, SplineCNN [40] achieves state-of-the-art results in several applications, so in our work we use the same kernel function as in SplineCNN. In this way, we leverage properties of B-spline bases to efficiently filter NVS graph inputs of arbitrary dimensionality. Let $((N_{1,i}^m)_{1 \le i \le k_1}, ..., (N_{d,i}^m)_{1 \le i \le k_d})$ denote d open **B**-spline bases of degree m with $\mathbf{k} = (k_1, ..., k_d)$ defining d-dimensional kernel size [43]. Let $w_{\boldsymbol{z},l} \in \mathbf{W}$ denote a trainable parameter for each element $\boldsymbol{z}$ from the Cartesian product $\mathcal{Z} = (N_{1,i}^m)_i \times \cdots \times (N_{d,i}^m)_i$ of the B-spline bases and each of the M_{in} input feature maps indexed by l. Then the kernel function $g_l : [a_1, b_1] \times \cdots \times [a_d, b_d] \to \mathbb{R}$ is defined as

$$g_l(\mathbf{u}) = \sum_{\boldsymbol{z} \in \mathcal{Z}} w_{\boldsymbol{z},l} \cdot \prod_{s=1}^{d} N_{s,z_s}(u_s) \tag{5}$$

We denote a graph convolution layer as $\text{Conv}(M_{\text{in}}, M_{\text{out}})$, where M_{in} is the number of input feature maps and M_{out} is the number of output feature maps indexed by $l^{'}$. Then, a graph convolution layer with bias b_l and activation function $\xi(t)$, can be written as:

$$\begin{aligned} \text{Conv}_{l'} \quad = \quad & \xi\Big(\frac{1}{|\mathcal{N}(i)|} \sum_{l=1}^{M_{\text{in}}} \sum_{j \in \mathcal{N}(i)} f_l(j) \cdot \sum_{\boldsymbol{z} \in \mathcal{Z}} w_{z,l} \\ & \cdot \quad \prod_{s=1}^{d} N_{s,z_s}(u_s) + b_{l'}\Big) \end{aligned} \tag{6}$$

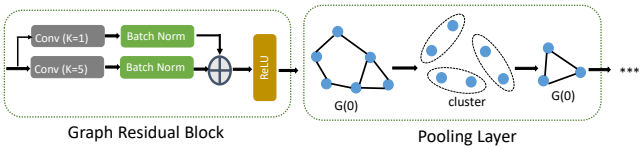


Fig. 4: Visualization of spatial feature learning module that is stacked by graph residual block and graph pooling layer.

where $l^{'} = 1, .., M_{\text{out}}$, indicates the $l^{'}$th output feature map. This defines a single graph convolutional layer. For C consecutive graph convolutional layers, $(\text{Conv}^{(c)})_{c\in[0,C]}$, the c-th layer has a corresponding input feature map $\mathbf{f}^{(c)}$ over all nodes, with the input feature for node i of the first layer $\text{Conv}^{(0)}$, $f^{(0)}(i) = p_i \in \{+1, -1\}$.

To accelerate deep network training, we use batch normalization [44] before the activation function. That is, the whole node feature $f_{l'}$ over the $l^{'}$-th channel map is normalized individually via

$$f^{'}_{l'} = \frac{f_l - E(f_{l'})}{\sqrt{\text{Var}(f_{l'}) + \epsilon}} \cdot \gamma + \beta \quad l^{'} = 1, .., M_{\text{out}} \tag{7}$$

where $E(f_{l'})$ and $\text{Var}(f_{l'})$ denote mean and variance of $f_{l'}$ respectively, ϵ is used to ensure normalization does not overflow when the variance is near zero, and γ and β represent trainable parameters.

Residual Graph CNNs: Inspired by the ResNet architecture [45], we propose residual graph CNNs for our spatial feature learning module, in order to resolve the well-known degradation problem inherent with increasing number of layers (depth) in graph CNNs [46]. Our residual graph CNN (RG-CNN) is effectively composed of a series of residual blocks and pooling layers. Considering equations (6) and (7) denote a single graph convolutional layer with batch normalization [44] that accelerates the convergence of the learning process, we apply residual connections in spatial feature learning module by summing element-wise the outputs of graph convolutions. Our "shortcut" connection comprises a graph convolution layer with kernel size $K = 1$ for mapping the feature dimension to the correct size, and is also followed by batch normalization. We denote the resulting graph residual block as $\text{Res}_{\text{g}}(\text{c}_{\text{in}}, \text{c}_{\text{out}})$, with c_{in} input feature maps and c_{out} output feature maps.

A residual block is followed by max pooling over clusters of nodes; given a graph representation, let us denote the spatial coordinates for node i as $(x'_i, y'_i) \in \{1, 2, \dots H'\} \times \{1, 2, \dots W'\}$ and resolution as $H' \times W'$. We define the cluster size as $s_h \times s_w$, which corresponds to the downscaling factor in the pooling layer of $\left\lceil \frac{H'}{s_h} \right\rceil \times \left\lceil \frac{W'}{s_w} \right\rceil$. For each cluster, we generate a single node, with feature set to the maximum over node features $\mathbf{f}$ in the cluster, and coordinates set to the average of node coordinates (x'_i, y'_i) in the cluster. Importantly, if there are connected nodes between two clusters, we assume the new generated nodes in these two clusters are connected with an edge.

For object classification, where the entire event stream can be modelled by a single graph, we can directly map the output of the spatial feature learning module to the classes with a fully connected layer. Given M_{in} feature maps $\mathbf{f} \in \mathbb{R}^{I \times M_{\text{in}}}$ from a graph with I nodes, similar to CNNs, a fully connected layer in a graph convolutional network is a weighted linear combination linking all input features to outputs. Let us denote $f_l^{\text{spatial}}(i)$ as the lth output feature map of the ith node of the spatial feature learning module, then we can derive a fully connected layer in the graph as:

$$f_q^{\text{FC}} = \xi\Big(\sum_{i=1}^{I}\sum_{l=1}^{M_{\text{in}}} F_{i,l,q} f_l^{\text{spatial}}(i)\Big) \quad q = 1, ..., Q \tag{8}$$

where Q is the number of output channels indexed by q, F is an array of trainable weights with size $I \times M_{\text{in}} \times Q$, $\xi(t)$ is the non-linear activation function, e.g. ReLU: $\xi(t) = \max(0, t)$. For the remainder of the paper, we use $\text{FC}(Q)$ to indicate a fully connected layer with Q output dimensions.

C. Graph2Grid: From Graphs to Grid Snippet

For motion-based tasks, we need to model temporal dependencies over the entire event stream. As discussed in Section III-A, given a long sample duration, it is not feasible to construct a single graph over the entire event stream, due to the sheer number of events. It is more computationally feasible to generate multiple graphs for time blocks of duration T_{vol}. These are processed individually by the spatial feature learning module. However, to model coarse temporal dependencies over multiple graphs, we must fuse the spatial feature representations. We propose a new Graph2Grid module that transforms the learned graphs from our spatial feature learning module to a grid representation and performs stacking over temporal dimension, as illustrated in Fig. 2. In this way, we are effectively able to create pseudo frames from the graphs, with M_{in} channels and timestamp $(n-1)T_{\text{vol}}$, corresponding to the n-th graph.

Again, denoting the output spatial feature learning map as $f_l^{\text{spatial}}(i)$ for the lth output feature map of the ith node with coordinates $(x'_i, y'_i) \in \{1, 2, \dots, H_{\text{spatial}}\} \times \{1, 2, \dots, W_{\text{spatial}}\}$, we define a grid representation $\mathbf{f}^{\text{grid}}$ of spatial size $H_{\text{spatial}} \times W_{\text{spatial}}$ as follows:

$$f^{\text{grid}}_{a,b,l} = \begin{cases} f_l^{\text{spatial}}(i), & \text{when } a = x'_i, b = y'_i \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

where $(a, b) \in \{1, 2, \dots, H_{\text{spatial}}\} \times \{1, 2, \dots, W_{\text{spatial}}\}$. The resulting grid feature representation $\mathbf{f}^{\text{grid}} \in \mathbb{R}^{H_{\text{spatial}} \times W_{\text{spatial}} \times M_{in}}$ is for a single graph; for S graphs over the temporal sequence, we simply concatenate over a fourth temporal dimension. We denote the resulting grid feature over S graphs as $\mathbf{F}^{\text{grid}} = \mathbf{f}^{\text{grid},1} || \mathbf{f}^{\text{grid},2} || \dots || \mathbf{f}^{\text{grid},S}$, where $||$ denotes concatenation over the temporal axis. Thus, the dimensions of $\mathbf{F}^{\text{grid}}$ are $H_{\text{spatial}} \times W_{\text{spatial}} \times M_{\text{in}} \times S$. This grid feature matrix can therefore be fed to a conventional 3D convolutional neural network in our temporal feature learning module, in order to learn both the coarse temporal dependencies, but also a full spatio-temporal representation of the input.

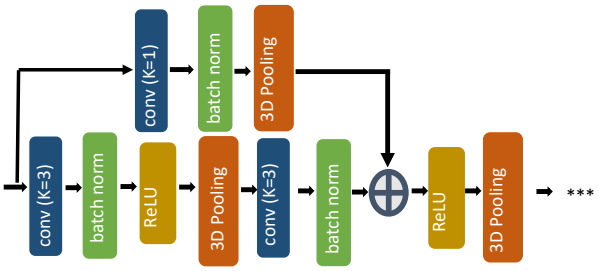


Fig. 5: Visualization of 3D residual block as an example for temporal feature learning module.

D. Temporal Feature Learning Module

The output feature matrix $\mathbf{F}^{\text{grid}}$ contains both spatial and temporal information over the entire sample duration, which can be effectively encoded with a conventional 3D CNN [5] in order to generate a final spatio-temporal representation of the video input for action recognition. In this paper, we consider three network architectures for the 3D CNN; a plain architecture with interlaced 3D convolutional and pooling layers, an I3D-based architecture comprising multiple I3D blocks as configured in [6], and a 3D residual block design. To illustrate temporal feature learning module, we visualize an example of our 3D residual block design in Fig. 5; essentially for C consecutive convolutional layers, every $(c-2)$-th layer is connected to the c-th layer via a non-linear residual connection, for all $c \in \{3, 5 \dots C-2, C\}$, and every layer is followed by batch normalization. For all architectures, we aggregate the features in the final layer of the CNN with global average pooling and pass to a fully connected layer for classification. We provide further experimental details in Section IV, describing the number of input and output channels per layer.

It is worth noting that while 3D CNNs are notorious for being computationally heavy, typical NVS cameras like the iniLabs DAVIS240c has spatial resolutions of the order of 240×180; in conjunction with the use of pooling in our spatial feature learning module, this means that the spatial size of $\mathbf{F}^{\text{grid}}$ is at most 30×30. This is substantially lower input resolution than APS-based counterparts ingesting RGB frames, where the spatial resolution to the 3D CNN is typically 224×224 or higher.

IV. Experimental Details and Evaluation

In this section, we demonstrate the potential of our framework as a method of representation learning for high-level computer vision tasks with NVS inputs. In Section IV-A, we focus on object classification as an appearance-based application. Then in Sections IV-B and IV-C, we present results for large-scale multi-class human action recognition and action similarity labeling as motion-based applications. In all our comparisons, we benchmark our results against state-of-the-art methods that can be applied to NVS data, i.e., we do not extend the comparison to methods that need APS data and generate optical flow and other modalities from APS, as they go beyond the realm of NVS-only sensing. Beyond evaluation on standard

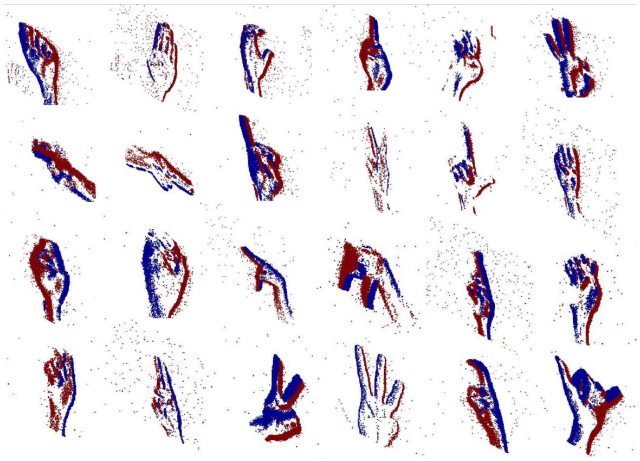


Fig. 6: Examples of the ASL-DVS dataset (the visualizations correspond to letters A-Y, excluding J, since letters J and Z involve motion rather than static shape). Events are grouped to image form for visualization (Red/Blue: ON/OFF events).

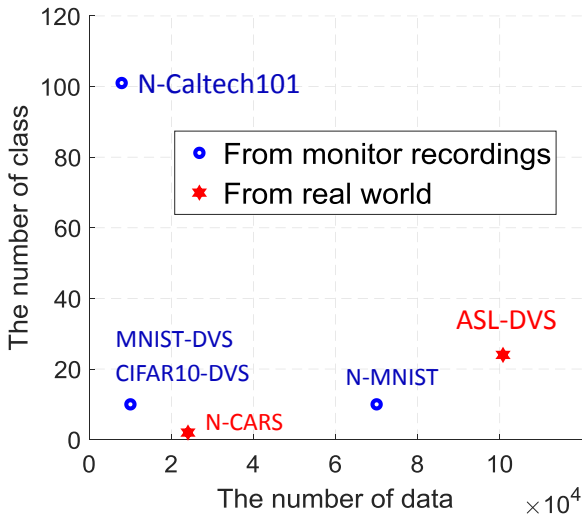


Fig. 7: Comparison of NVS datasets w.r.t. the number of classes and the total size.

datasets, we introduce our newly proposed ASL-DVS dataset in Section IV-A, which is the largest-source dataset for object classification. We additionally generate the largest NVS-based action recognition and action similarity labelling datasets by converting standard APS datasets, UCF101, HMDB51 and ASLAN, to the NVS domain and explain the recording process prior to evaluation in Sections IV-B and IV-C respectively.

A. Object Classification

Object classification finds numerous applications in visual surveillance, human-machine interfaces, image retrieval and visual content analysis systems. We first introduce the datasets we evaluate on, including our new ASL-DVS dataset, before discussing implementation details and presenting results. We compare with recent state-of-the-art methods and perform complexity analysis.

Datasets: Many neuromorphic datasets for object classification are converted from standard frame-based datasets,

such as N-MNIST [47], N-Caltech101 [47], MNIST-DVS [48] and CIFAR10-DVS [49]. N-MNIST and N-Caltech101 were acquired by an ATIS sensor [9] moving in front of an LCD monitor while the monitor is displaying each sample image. Similarly, MNIST-DVS and CIFAR10-DVS datasets were created by displaying a moving image on a monitor and recording with a fixed DAVIS sensor [50]. Emulator software has also been proposed in order to generate neuromorphic events from pixel-domain video formats using the change of pixel intensities of successively rendered images [26], [51]. While useful for early-stage evaluation, these datasets cannot capture the real dynamics of an NVS device due to the limited frame rate of the utilized content, as well as the limitations and artificial noise imposed by the recording or emulation environment. To overcome these limitations, N-CARS dataset [14] was created by directly recording objects in urban environments with an ATIS sensor. Despite its size, given that it only corresponds to a binary classification problem, N-CARS cannot represent the behaviour of object classification algorithms on more complex NVS-based tasks.

We present a large 24-class dataset of handshape recordings under realistic conditions. Its 24 classes correspond to 24 letters (A-Y, excluding J) from the American Sign Language (ASL), which we call ASL-DVS. Examples of recordings are shown in Fig 6. The ASL-DVS was recorded with an iniLabs DAVIS240c NVS camera set up in an office environment with low ambient noise and constant illumination. For all recordings, the camera was at the same position and orientation to the persons carrying out the handshapes. Five subjects were asked to pose the different static handshapes relative to the camera in order to introduce natural variance into the dataset. For each letter, we collected 4,200 samples (total of 100,800 samples) and each sample lasts for approximately 100 milliseconds. Fig. 7 shows a comparison of existing NVS datasets w.r.t. the number of classes and the total size. Within the landscape of existing datasets, our ASL-DVS is a comparably complex dataset with the largest number of labelled examples. We, therefore, hope that this will make it a useful resource for researchers to build comprehensive models for NVS-based object recognition, especially given the fact that it comprises real-world recordings. ASL-DVS and related code are available online [1].

Implementation Details: For simple datasets N-MNIST and MNIST-DVS, our spatial feature learning module is only comprised of two graph residual blocks. Graph residual blocks are described in Section III-B, and we fix the kernel size $K = 5$ for all convolutional layers outside of the skip connection. We denote a graph convolutional layer as $\mathrm{Conv_g}(c_{\mathrm{in}}, c_{\mathrm{out}})$, fully connected layer as $\mathrm{FC}(c_{\mathrm{in}}, c_{\mathrm{out}})$ and graph residual block as $\mathrm{Res_g}(\mathrm{c_{in}}, \mathrm{c_{out}})$, where c_{in} and c_{out} are the input and output channels respectively. Additionally, we denote max graph pooling layers as $\mathrm{MaxP_g}(\mathrm{s_h}, \mathrm{s_w})$, where s_{h} and s_{w} represent the cluster size. With this notation, the architecture of our network for these can be written as $\mathrm{Conv_g}(1,32) \longrightarrow \mathrm{MaxP_g}(2,2) \longrightarrow \mathrm{Res_g}(32,64) \longrightarrow \mathrm{MaxP_g}(4,4) \longrightarrow \mathrm{Res_g}(64,128) \longrightarrow \mathrm{MaxP_g}(7,7) \longrightarrow \mathrm{FC}(128,128) \longrightarrow \mathrm{FC}(128,Q)$, where Q is the number of classes of each dataset. For the remaining datasets, three residual graph blocks are used, and the utilized network architecture is $\mathrm{Conv_g}(1,64) \longrightarrow \mathrm{MaxP_g}(\mathrm{s_h}, \mathrm{s_w}) \longrightarrow \mathrm{Res_g}(64,128) \longrightarrow \mathrm{MaxP_g}(\mathrm{s_h}, \mathrm{s_w}) \longrightarrow \mathrm{Res_g}(128,256) \longrightarrow \mathrm{MaxP_g}(\mathrm{s_h}, \mathrm{s_w}) \longrightarrow \mathrm{Res_g}(256,512) \longrightarrow \mathrm{MaxP_g}(\mathrm{s_h}, \mathrm{s_w}) \longrightarrow \mathrm{FC}(512,1024) \longrightarrow \mathrm{FC}(1024,Q)$. Since the datasets are recorded from different sensors, the spatial resolution of each sensor is different (i.e., DAVIS240c: 240×180, DAVIS128 & ATIS: 128×128), leading to various maximum coordinates for the graph. We, therefore, set the cluster size in pooling layers to: *(i)* 4×3, 16×12, 30×23 and 60×45 for N-Caltech101 and ASL-DVS datasets; *(ii)* 4×4, 6×6, 20×20 and 32×32 for CIFAR10-DVS and N-CARS datasets. We also compare the proposed residual graph networks (RG-CNNs) with their corresponding plain graph networks (G-CNNs), which utilize the same number of graph convolutional and pooling layers but without the residual connections. The degree of B-spline bases m of all convolutions in this work is set to 1.

For the N-MNIST, MNIST-DVS and N-CARS datasets, we use the predefined training and testing splits, while for N-Caltech101, CIFAR10-DVS and ASL-DVS, we follow the experiment setup of Sironi [14]: 20% of the data is randomly selected for testing and the remaining is used for training. During the non-uniform sampling, the maximal number of events k in each space-time volume is set to 8. When constructing graphs, the radius R is 3, weight parameters α and β are set to 1 and 0.5×10^{-5}, respectively, the maximal connectivity degree $D_{\max}$ for each node is 32, and $T_{\mathrm{vol}} = 30$ millisecond length events are randomly extracted to form the graph. In particular, α and β are selected so that the spatial and temporal components (μs) of (2) are balanced in magnitude as best as possible. Despite NVS cameras being frameless, NVS events can be grouped and visualized as sparse "frames" with frame rate being as high as 2000 fps (i.e., 1 frame containing the grouping of multiple NVS events appearing every 0.5ms as reported for the Samsung Dynamic Vision Sensor). This observation shows that a window of events corresponding to 30ms is sufficient for graph construction in our applications. In order to reduce overfitting, we add dropout with probability 0.5 after the first fully connected layer and also perform data augmentation. In particular, we spatially scale node positions by a randomly sampled factor within $[0.95, 1)$, perform mirroring (randomly flip node positions along 0 and 1 axis with 0.5 probability) and rotate node positions around a specific axis by a randomly sampled factor within $[0, 10]$ in each dimension. Networks are trained with the Adam optimizer and the cross-entropy loss between softmax output and the one-hot label distribution for 150 epochs with batch size 64 and learning rate 0.001 step-wise decreasing by 0.1 after 60 and 110 epochs.

Results: We compare Top-1 classification accuracy obtained from our model with that from HOTS [15], H-First [52], SNN [19], [53] and HATS [14]. We report results from Sironi *et al.* [14], since we use the same training and testing methodology. The results are shown in Table I. For the simple N-MNIST and MNIST-DVS datasets, whose accuracy is already close to near-perfect classification, our models achieve comparable results.

[1] https://github.com/PIX2NVS/NVS2Graph

TABLE I: Top-1 accuracy of our graph CNNs w.r.t. the state-of-the-art, other graph convolution networks and deep CNNs.

Model	N-MNIST	MNIST-DVS	N-Caltech101	CIFAR10-DVS	N-CARS	ASL-DVS
H-First [52]	0.712	0.595	0.054	0.077	0.561	-
HOTS [15]	0.808	0.803	0.210	0.271	0.624	-
Gabor-SNN [19], [53]	0.837	0.824	0.196	0.245	0.789	-
HATS [14]	**0.991**	0.984	0.642	0.524	0.902	-
GIN [54]	0.754	0.719	0.476	0.423	0.846	0.514
ChebConv [38]	0.949	0.935	0.524	0.452	0.855	0.317
GCN [55]	0.781	0.737	0.530	0.418	0.827	0.811
MoNet [42]	0.965	0.976	0.571	0.476	0.854	0.867
VGG_19 [56]	0.972	0.983	0.549	0.334	0.728	0.806
Inception_V4 [57]	0.973	0.985	0.578	0.379	0.864	0.832
ResNet_50 [45]	0.984	0.982	0.637	**0.558**	0.903	0.886
G-CNNs	0.985	0.974	0.630	0.515	0.902	0.875
RG-CNNs (proposed)	0.990	**0.986**	**0.657**	0.540	**0.914**	**0.901**

For the other datasets, our proposed RG-CNNs consistently set the new state-of-the-art on these datasets.

Table I also includes the classification results stemming from other graph convolutional networks; namely, GIN [54], ChebConv [54], GCN [55] and MoNet [42]. The architectures of these networks are the same as our plain graph networks (G-CNNs) introduced in this section, with the only difference being the graph convolutional operation. The training details and data augmentation methods are the same as described before. The Top-1 classification accuracy stemming from all networks of Table I indicates that our proposed RG-CNN and G-CNN outperform all the other graph convolutional networks.

To further validate our proposal, we compare our results with conventional deep convolutional networks. There are no conventional CNNs specifically designed for NVS events, so we train/evaluate on three well-established CNNs, namely VGG_19 [56], Inception_V4 [57] and ResNet_50 [45]. The format of the required input for these CNNs is frame-based, so we convert neuromorphic spike events to frame form similarly to the grouping of Zhu *et al.* [12]. We thereby introduce a two-channel event image form with the same resolution as the NVS sensor: the two channels encode the number of positive and negative events that have occurred at each position. In addition, each frame grouping corresponds to a random time segment of 30 ms of spike events. To avoid overfitting, we supplement the training with heavy data augmentation: first, we resize the input images such that the smaller dimension is 256 and keep the aspect ratio; then, we use a random cropping of 224×224 spatial samples of the resized frame; finally, the cropped volume is randomly flipped and normalized according to its mean and standard deviation. We train all CNNs from scratch using stochastic gradient descent with momentum set to 0.9 and L_2 regularization set to 0.1×10^{-4}. The learning rate is initialized at 10^{-3} and decayed by a factor of 0.1 every 10k iterations. As shown in Table I, despite performing comprehensive data augmentation and L_2 regularization to avoid overfitting, the results acquired from conventional CNNs are still below the-state-of-the-art since event images contain far less information (see Fig. 1). Thus, except for the CIFAR10-DVS dataset, the accuracy of our proposals surpasses that of conventional frame-based deep CNNs.

Complexity Analysis: We now turn our attention to the complexity of our proposals and compare the number of floating-point operations (FLOPs) and the number of parameters of each model. In conventional CNNs, we compute FLOPs for convolution layers as [58]:

TABLE II: Complexity (GFLOPs) and size (MB) of networks.

Model	GFLOPs	Size (MB)
VGG_19 [56]	19.63	143.65
Inception_V4 [57]	12.25	42.62
ResNet_50 [45]	3.87	25.61
G-CNNs	0.39	18.81
RG-CNNs	0.79	19.46

$$\text{FLOPs} = 2HW(C_{\text{in}}K^2 + 1)C_{\text{out}} \tag{10}$$

where H, W and C_{in} are height, width and the number of channels of the input feature map, K is the kernel size, and C_{out} is the number of output channels. For graph convolution layers, FLOPs stem from 3 parts [40]; *(i)* for computation of B-spline bases, there are $N_{\text{edge}}(m+1)^d$ threads each performing $7d$ FLOPs (4 additions and 3 multiplications), where N_{edge} is the number of edges, m the B-spline basis degree and d the dimension of graph coordinates; *(ii)* for convolutional operations, the FLOPs count is $3N_{\text{edge}}C_{\text{in}}C_{\text{out}}(m+1)^d$, with factor 3 stemming from 1 addition and 2 multiplications in the inner loop of each kernel and C_{in} and C_{out} is the number of input and output channels, respectively; *(iii)* for scatter operations and the bias term, the FLOPs count is $(N_{\text{edge}} + N_{\text{node}})C_{\text{out}}$, where N_{node} is the number of nodes. In total, we have

$$\begin{aligned} \text{FLOPs} &= N_{\text{edge}}(m+1)^d(3C_{\text{in}}C_{\text{out}} + 7d) \\ &+ (N_{\text{edge}} + N_{\text{node}})C_{\text{out}} \end{aligned} \tag{11}$$

For fully connected layers, in conventional CNNs, G-CNNs and RG-CNNs, we compute FLOPs as [58] $\text{FLOPs} = (2I-1)O$, where I is the input dimensionality and O is the output dimensionality. As to the number of parameters, for each convolution layer in CNNs, G-CNNs and RG-CNNs, it is $(C_{\text{in}}K^2+1)C_{\text{out}}$, while in fully connected layers, it is $(C_{\text{in}}+1)C_{\text{out}}$. As shown by (11), FLOPs of graph convolution depend on the number of edges and nodes. Since the size of input graph varies per dataset, we opt to report representative results from N-Caltech101 in Table II. G-CNNS and RG-CNNs have the smaller number of weights and require the less computation compared to deep CNNs. The main reason is that the graph representation is compact, which in turn reduces the amount of data that needs to be processed. For N-Caltech101, the average number of nodes of each graph is 1000, while

grouping events to 2-channel image makes the input size equal to 86,400.

B. Action Recognition

Action recognition has numerous applications in intelligent surveillance, human behavior analysis, and other motion-based tasks [5], [6], [59]. Unlike recognition in static scenes that focuses on visual appearance, one crucial factor of action recognition is the motion dynamics. The performance of action recognition system largely depends on whether the dynamics of motion can be effectively represented and utilized [60].We firstly introduce datasets for evaluation, then proceed to discussing implementation details for framework, detailing the architectures of the spatial feature learning module, Graph2Grid block and temporal feature learning module. Finally, we present results and complexity analysis on datasets for variants of our framework and other recent state-of-the-art methods.

Datasets: Previous work on neuromorphic vision sensing for action recognition evaluates on the DVS128 Gestures Dataset [17] and posture dataset [61]. DVS128 Gesture Dataset comprises 1,342 instances of 11 hand and arm gestures, while the posture dataset includes only three human actions, namely, "bend", "sit/stand" and "walk". Both datasets were collected from an experimental setting environment with a monotonous background, and relative to equivalent datasets for APS-based evaluation datasets, both are modest in their size and class count; as such, they cannot represent complex real-life scenarios and are not robust to evaluation for advanced algorithms. Moreover, previous work [17], [34], [61], [62] already achieves high accuracies on them. This is why, it is necessary to establish larger and more complex datasets for the evaluation of our proposal and for future proposals on NVS-based action recognition.

We provision two new neuromorphic event datasets, namely UCF101-DVS and HMDB51-DVS. Both datasets were respectively captured from playbacks of the UCF101 [22] and HMBD [23] datasets, which are well established datasets for the evaluation of action recognition in the APS domain. UCF101 comprises 13,320 videos of 101 different human actions, while HMDB51 includes 6,766 videos with 51 human action categories. Of relevance is the work of Hu *et al.* [63] which previously recorded UCF50 by displaying existing benchmark videos to stationary neuromorphic vision sensors under controlled lighting conditions. We follow a recording procedure similar to that of [63] to wholly capture *remaining* of UCF101 and HMDB51. Displayed videos are recorded by a neuromorphic vision sensor DAVIS240c that is adjusted to cover the region of interest on the monitor. Our captured datasets are the largest neuromorphic datasets for action recognition, and recorded UCF101-DVS and HMDB51-DVS can be found Online [2].

Implementation Details: We present our results on action recognition in Table III and Table IV, where the total number of graphs constructed from each event stream S is set to either 8 or 16. Events within $T_{\text{vol}} = 1/30$ seconds are constructed into one spatial graph, where individual nodes are connected to their nearest neighbor. Given that we construct multiple graphs from events sampled from multiple consecutive 30ms time windows, our representation can cover a sufficiently long temporal extent. Spatial features are learned using our proposed residual graph CNNs (RG-CNN) where two residual blocks are stacked, each followed by a graph max-pooling layer. Specifically, for DVS128 Gesture Dataset [17] we use the architecture: $\text{Res}_\text{g}(1,64) \longrightarrow \text{MaxP}_\text{g}(2,2) \longrightarrow \text{Res}_\text{g}(64,128) \longrightarrow \text{MaxP}_\text{g}(4,4)$. Similarly, for UCF101-DVS and HMDB51-DVS we use three residual blocks, and the architecture is: $\text{Res}_\text{g}(1,32) \longrightarrow \text{MaxP}_\text{g}(2,2) \longrightarrow \text{Res}_\text{g}(32,64) \longrightarrow \text{MaxP}_\text{g}(4,3) \longrightarrow \text{Res}_\text{g}(64,128) \longrightarrow \text{MaxP}_\text{g}(8,6)$. For the temporal feature learning module, we explore three types of architectures as described in Section III-D:

1) Plain 3D: We first consider a series of consecutive 3D convolutional and pooling layers, where each intermediate convolution layer is followed by batch normalization layer and a ReLU activation function. We use $\text{Conv}_\text{3D}(c_\text{in}, c_\text{out})$ to denote traditional 3D convolutional layers with batch normalization and activation functions, where c_in and c_out are the number of input and output channels respectively. 3D max pooling and global average pooling are denoted as Pool3D and GlobAvgP respectively, fully connected layers as FC and task classes as Q. Plain 3D convolution architectures are thus represented as follows: $\text{Conv}_\text{3D}(128,128) \longrightarrow \text{Pool3D} \longrightarrow \text{Conv}_\text{3D}(128,256) \longrightarrow \text{Pool3D} \longrightarrow \text{Conv}_\text{3D}(256,512) \longrightarrow \text{Pool3D} \longrightarrow \text{Conv}_\text{3D}(512,512) \longrightarrow \text{Pool3D} \longrightarrow \text{GlobAvgP} \longrightarrow \text{FC}(Q)$. With the notation (h, w, t) denoting height, width and time dimensions, we note that the kernel size and stride in every convolution layer is $(3,3,3)$ and $(1,1,1)$ respectively, and the window size and stride of all 3D max pooling layers is $(2,2,2)$, expect for the first pooling layer, where the stride is $(2,2,1)$ to ensure that temporal downscaling is not aggressive early on.

2) Inception-3D(4): We next consider an Inception-3D architecture, comprising a series of four consecutive I3D blocks. In order to ensure that temporal feature learning is not bottlenecked, we restrict the number of I3D blocks to four. Similar to [6], our implementation of the I3D block is a concatenation of four streams of convolutional layers with varying kernel sizes. Where we use the shorthand $\text{Inc}_\text{b}(\text{c}_\text{in}, \text{c}_\text{out})$ to denote each b-th I3D block, we setup our architecture as: $\text{Inc}_1(128,480) \longrightarrow \text{Pool3D} \longrightarrow \text{Inc}_2(480,512) \longrightarrow \text{Pool3D} \longrightarrow \text{Inc}_3(512,512) \longrightarrow \text{Pool3D} \longrightarrow \text{Inc}_4(512,512) \longrightarrow \text{Pool3D} \longrightarrow \text{GlobAvgP} \longrightarrow \text{FC}(Q)$. The number of output channels of the n-th convolutional layer for the s-th stream is labelled as $c_\text{out}[s][n]$, and the number of output channels per convolutional layer for each I3D block is: $[[128],[128,192],[32,96],64]]$, $[[192],[96,208],[16,48],64]]$, $[[160],[112,224],[24,64],64]]$ and $[[128],[128,256],[24,64],64]]$.

3) Residual 3D: Finally, we consider 3D residual CNNs, where we effectively replace the I3D block with a 3D residual block. The 3D residual block design for temporal feature learning is illustrated in Fig. 5; essentially, there are two 3D convolutional layers in the base stream of the block, with a

[2]https://github.com/PIX2NVS/NVS_FeatureLearning

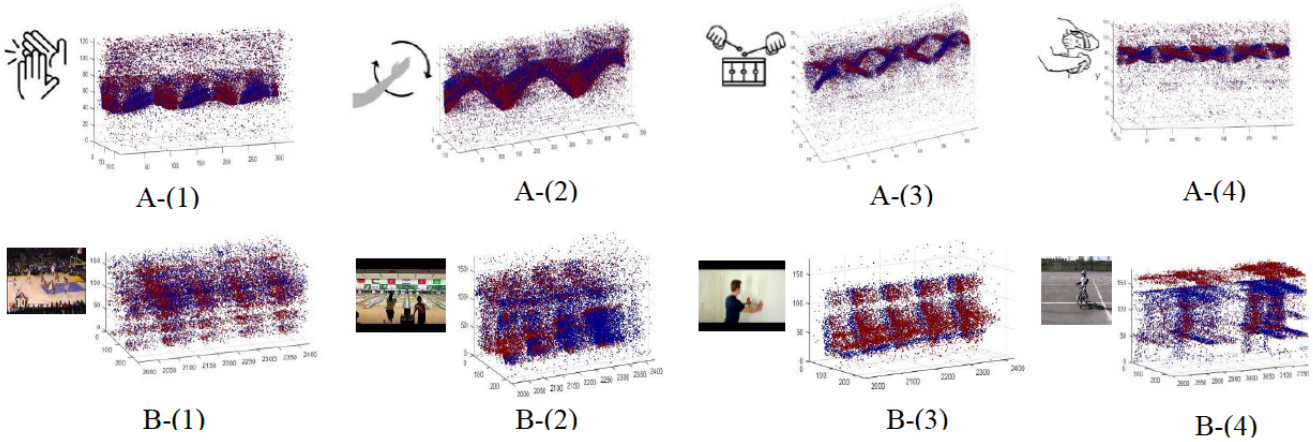


Fig. 8: Visualization of samples from DVS128 Gesture Dataest [17] and UCF101-DVS [22]. (A) DVS128 Gesture Dataset : A-1: hand clap; A-2: right hand rotation clockwise; A-3: air drums; A-4: forearm roll. (B) UCF101-DVS: B-1: basketball dunk; B-2: bowling; B-3: wall pushups; B-4: biking

non-linear residual connection from the input of the first to the output of the second layer. We can define a 3D residual block as $\mathrm{Res}(c_{\mathrm{in}}, c_{\mathrm{inter}}, c_{\mathrm{out}})$, where c_{inter} represents the number of input channels to the second convolutional layer in the base stream and c_{in} and c_{out} are the respective number of input and output channels to the residual block. The 3D residual CNN is defined as follows: $\mathrm{Res}(128, 256, 512) \longrightarrow \mathrm{Pool3D} \longrightarrow \mathrm{Res}(512, 512, 1024) \longrightarrow \mathrm{Pool3D} \longrightarrow \mathrm{GlobAvgP} \longrightarrow \mathrm{FC}(Q)$. Again, denoting (h, w, t) as the height, width and time dimensions, the kernel size is $(3, 3, 3)$ and stride is $(1, 1, 1)$ for all convolutional layers in the base stream.

In all of our tests, sampled graphs are spatially scaled by random sampling factors within $[0.8, 1]$, and are randomly left-right flipped with a probability of 0.5. For all of our reported results, we train using the Adam optimizer for 150 epochs, with batch sizes respectively set to 32 and 16 for $S = 8$ and $S = 16$. The learning rate is set to 0.001, with stepwise decay by a factor of 0.1 after 60 epochs.

Reference Networks: We compare action recognition results of our proposed RG-CNN + Plain 3D, RG-CNN + Incep. 3D(4) and RG-CNN + Res. 3D models with previous proposals for the APS domain, where we repurpose their use to the NVS domain by maintaining the spatial coherence of events to pass them as input frames. As external benchmarks, we include C3D [5], I3D [6], 3D ResNet with 34 layers [45], P3D with 63 layers [64], R2+1D [65] and 3D ResNext with 50 layers [66]. In contrast to our framework, these aforementioned proposals are entirely grid-based, and we construct independent frames for their use by summing events within a $1/30$ seconds duration at each spatial position of the NVS sensor. In this way, resulting event frames are represented by two channels, where ON and OFF events are grouped independently, and in order to align event maps with the number of input graphs utilized in our framework, we produce $S = 8$ or $S = 16$ sampled frames for each input volume of events. To avoid over-fitting during training, we supplement training with data augmentation, where we normalize the input and re-size the input frames such that the smaller side is 128 (178 for P3D, 256 for I3D) and keep the aspect ratio, and use a random cropping to acquire appropriately sized inputs, and cropped volumes are randomly left-right flipped with a probability of 0.5. We randomly initialize the parameters of all models and use stochastic gradient descent with momentum set to 0.9, and learning rate initialized at 0.01 with a decay factor of 0.1 every 50 epochs.

Results: We first evaluate our method on the DVS128 Gesture Dataset, and compare with both recent state-of-theart methods and reference networks. The results are shown in Table III, and for all recent methods, considered event recording durations are set to 0.25 and 0.5 seconds. We follow the same set up to set the number of graphs, enabling a fair comparison. Examining the results, we find LSTM-based methods [67] to be outperformed by others, and we attribute this to the fact that LSTMs regard event streams as pure temporal sequences and only learn temporal features from events, without encoding spatial dependencies. In contrast, PointNet-based methods [34], [35], [68] are more accurate, and consider inputs as point clouds to learn to summarize their geometric features. With regards to reference networks, although I3D [6] and 3D ResNet-34 [45] perform spatio-temporal feature learning, there is no explicit modelling of event dependencies as events are directly grouped into frames. As such, our proposal outperforms all existing works and reference networks on this dataset and sets a new benchmark. We attribute this to the combination of our graph representation, spatial feature learning and temporal feature learning over multiple graphs, which results in learning a more informative spatio-temporal representation of the input.

TABLE III: Top-1 classification accuracies on the DVS128G gestures dataset. Performance is reported for input duration with temporal depths of 0.25 and 0.5 seconds.

Method	Duration(0.25s)	Duration(0.5s)
LSTM [67]	0.882	0.865
PointNet [35]	0.887	0.902
PointNet++ [68]	0.923	0.941
Amir CVPR2017 [17]	-	0.945
Wang WACV2019 [34]	0.940	0.953
ResNet_34 [45]	0.943	0.955
I3D [6]	0.951	0.965
RG-CNN + Plain 3D	0.954	0.968
RG-CNN + Incep. 3D	0.957	0.968
RG-CNN + Res. 3D	**0.961**	**0.972**

As shown in Fig. 8, DVS128 Gesture Dataset contain salient

pattern differences, while UCF101-DVS comprises more complex event volumes, and as shown in Table III, results of the best performing models on DVS128 Gesture Dataset are close to achieving complete accuracy. Therefore, we further evaluate our algorithms on our newly introduced datasets, UCF101-DVS and HMDB51-DVS, which contain more classes and overall present a more challenging task for action recognition. We note that when evaluating current NVS-based methods for action recognition on UCF101-DVS and HMDB51-DVS, the accuracy obtainable is only around 5%-7%, since these methods only perform spatial (PointNet, PointNet++) or temporal (LSTM) feature learning, and thus leaning to degenerate solutions. Therefore, we focus our comparison on reference networks for these datasets.

The Top-1 recognition accuracy of all models is reported in Table IV for UCF101-DVS and HMDB51-DVS, where it shows that all variants of our architecture outperform tested benchmarks. Specifically, the highest performance obtained from reference models is from I3D, while our base model (RG-CNN + Plain 3D) outperforms I3D by 3.3% on UCF101-DVS and by 6.1% on HMDB51-DVS when $S = 8$. The accuracy of our models is further increased when considering the Inception-3D and Residual-3D variants, where our model performance increases slightly due to the higher capacity of these architectures.

TABLE IV: Top-1 classification accuracy of UCF101-DVS and HMDB51-DVS w.r.t. various model.

Model	UCF101-DVS		HMDB51-DVS	
	$S = 8$	$S = 16$	$S = 8$	$S = 16$
C3D [5]	0.382	0.472	0.342	0.417
ResNet-34 [45]	0.513	0.579	0.350	0.438
P3D-63 [64]	0.484	0.534	0.343	0.404
R2+1D-36 [65]	0.496	0.628	0.312	0.419
ResNext-50 [66]	0.515	0.602	0.317	0.394
I3D [6]	0.596	0.635	0.386	0.466
RG-CNN + Plain 3D	0.629	0.663	0.447	0.494
RG-CNN + Incep. 3D	**0.632**	**0.678**	0.452	**0.515**
RG-CNN + Res. 3D	0.627	0.673	**0.455**	0.497

Complexity Analysis: We compare the complexity of tested models, and do so with respect to the number of floating-point operations (FLOPs) and required parameter counts. For graph-based convolutional and fully-connected layers, FLOPs and parameter counts are calculated as detailed in Section IV-B. For conventional 3D convolutional layers, we compute FLOPs as $2HWT(C_{\text{in}}K^3 + 1)C_{\text{out}}$ multi-add operations, where H, W, and T are the height, width, and temporal length of input maps, C_{in} is the number of input feature channels, K is the kernel size, and C_{out} is the number of output channels. Using similar notation, parameter counts of conventional 3D convolutional layers are calculated as $(C_{\text{in}}K^3{+}1)C_{\text{out}}$. FLOPs of graph convolutions depend on edge and node counts (see Section IV-B), and we specifically report results for UCF101-DVS in Table V. For each sample, 16 graphs are sampled as inputs to the spatial feature learning module, and FLOPs in respective modules are the averages over the whole of UCF101-DVS. Our results show how graph convolutions can manage with smaller or comparably sized input volumes relative to all reference models. As for complexity, though our models require more floating-point operations when compared to P3D-63 and ResNext-50, they achieve better performance on all three datasets. On the other hand, accuracies of I3D are close to ours while requiring complexities which are two to three times higher.

TABLE V: Comparison of models w.r.t. complexity (GFLOPs) and size of architecture parameters.

Model	GFLOPs	Parameters($\times 10^6$)
C3D [5]	39.69	78.41
ResNet-34 [45]	11.64	63.70
P3D-63 [64]	8.30	25.74
R2+1D-36 [65]	41.77	33.22
ResNext-50 [66]	6.46	26.05
I3D [6]	30.11	12.37
RG-CNN + Plain 3D	12.46	6.95
RG-CNN + Incep. 3D	12.39	3.86
RG-CNN + Res. 3D	13.72	12.43

C. Action Similarity Labeling

Action similarity labeling is a binary classification task wherein alignments of action pairs are predicted. In other words, models are required to learn to evaluate the similarity of actions rather than recognize particular actions. The challenge of action similarity labeling lies in that the actions of test sets belong to separate classes and are not available during training [24]. That is to say, training does not provide an opportunity to learn actions presented at test time. To the best of our knowledge, as of yet there is no work on similarity detection in the neuromorphic domain, and no existing dataset can be used for evaluation. We use the ASLAN [24] dataset which comprises 3,697 samples from 432 different action classes. Using a similar setting to the one described in Section IV-B, we captured an equivalent neuromorphic dataset ASLAN-DVS to be publicly provisioned for relevant research. Our captured ASLAN-DVS can be found online [3].

Training Details: We use the "View-2" method as detailed in [24] to split samples into 10 mutually exclusive subsets, where each subset contains 600 video pairs, with 300 to be classified as "similar" and 300 to be classified as "not similar". We report our results by averaging scores of 10 separate experiments in a leave-one-out cross validation scheme. In this application, we used models trained for action recognition as feature extractors, and extracted L_2-normalised output features from the last $\mathrm{GlobalAvgP}$ and $\mathrm{Pool3D}$ layers to acquire two distinct types of representation. Similar to [24], we independently compute 12 different distances for said features and for every pair of actions. Finally, a support vector machine with a radial basis kernel is trained to classify whether action pairs are of similar or different activities. As baselines, we consider the performance of reference architectures detailed in Sec. IV-B, where features are extracted as the outputs of the last two layers, and classifications are performed by support vector machines. The complexity of our proposed spatio-temporal feature learning and other reference models remain the same as in Section IV-B.

In Table VI we report the performance of different models as measured accuracies and areas under ROC curves (AUC). Our

[3]https://github.com/PIX2NVS/NVS_FeatureLearning

RG-CNN + Incep. 3D framework outperforms state-of-the-art results acquired from I3D by 2.6% on accuracy and 3.1% on AUC, which clearly indicates that graph-based models are better suited for feature learning for the purposes of action similarity labeling.

TABLE VI: Action similarity labeling performance

Model	Acc.	AUC
ResNet-34 [45]	0.605	0.643
P3D-63 [64]	0.598	0.638
R2+1D-36 [65]	0.615	0.652
ResNext-50 [66]	0.605	0.643
I3D [6]	0.623	0.659
RG-CNN + Plain 3D	0.635	0.674
RG-CNN + Incep. 3D(4)	**0.649**	**0.690**
RG-CNN + Res. 3D	0.641	0.684

V. Conclusion

In this work we develop an end-to-end trainable graph-based feature learning framework for neuromorphic vision sensing. We first represent neuromorphic events as graphs, which are explicitly aligned with the compact and non-uniform sampling of NVS hardware. We couple this with an efficient end-to-end learning framework, comprising graph convolutional networks for spatial feature learning directly from graph inputs. We extend our framework with our Graph2Grid module that converts the graphs to grid representations for coarse temporal feature learning with conventional 3D CNNs. We demonstrate how this framework can be employed for object classification, action recognition and action similarity labeling, and evaluate our framework on all tasks with standard datasets. We additionally propose and make available three large-scale neuromorphic datasets in order to motivate further progress in the field. Finally, our results on all datasets show that we outperform all recent NVS-based proposals while maintaining lower complexity.

Potential proposals of this work can be extended in future work. Firstly, we observed that the size of graphs tends to be large even though we apply non-uniform sampling over events. One interesting direction is to construct graphs dynamically and adaptively based on various scenes instead of using events within a fixed time window, thus making them more compact and representative. In addition, instead of propagating graphs into grids and using 3D CNNs for temporal feature learning, it may be more efficient to propose a graph convolution that can directly aggregate features over multiple graphs, as graph convolution will be over sparse nodes, requiring less memory and computation. Finally, due to the limited availability of NVS data, an important direction is to use the newly-released datasets to develop robust few-shot learning methods that can learn to make reliable predictions from small datasets.

References

[1] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 1440–1448.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[4] J. Deng, W. Dong *et al.*, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2009, pp. 248–255.

[5] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 4489–4497.

[6] J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the kinetics dataset," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2017, pp. 6299–6308.

[7] T. Delbrück, "Neuromorphic vision sensing and processing," in *Eur. Solid-State Dev. Res. Conf.*, 2016, pp. 7–14.

[8] T. Delbrück, B. Linares-Barranco, E. Culurciello, and C. Posch, "Activity-driven, event-based vision sensors," in *Proc. IEEE Int. Symp. on Circuits and Syst.*, 2010, pp. 2426–2429.

[9] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, Jan. 2010.

[10] R. Berner, T. Delbrück, A. Civit-Balcells, and A. Linares-Barranco, "A 5 Meps $100 USB2.0 address-event monitor-sequencer interface," in *Proc. IEEE Int. Symp. on Circuits and Syst.*, 2007, pp. 2451–2454.

[11] C. Tan, S. Lallee, and G. Orchard, "Benchmarking neuromorphic vision: Lessons learnt from computer vision," *Frontiers in Neuroscience*, vol. 9, p. 374, 2015.

[12] A. Z. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "EV-FlowNet: Self-supervised optical flow estimation for event-based cameras," *arXiv:1802.06898*, 2018.

[13] M. Cannici, M. Ciccone, A. Romanoni, and M. Matteucci, "Event-based convolutional networks for object detection in neuromorphic cameras," *arXiv:1805.07931*, 2018.

[14] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, "HATS: Histograms of averaged time surfaces for robust event-based object classification," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2018, pp. 1731–1740.

[15] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "HOTS: A hierarchy of event-based time-surfaces for pattern recognition," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 39, no. 7, pp. 1346–1359, 2017.

[16] R. Ghosh, A. Gupta, A. Nakagawa, A. Soares, and N. Thakor, "Spatiotemporal filtering for event-based action recognition," *arXiv:1903.07067*, 2019.

[17] A. Amir, B. Taba *et al.*, "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2017, pp. 7243–7252.

[18] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in Computational Neuroscience*, vol. 9, p. 99, 2015.

[19] J. H. Lee, T. Delbrück, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in Neuroscience*, vol. 10, p. 508, 2016.

[20] Y. Bi, A. Chadha, A. Abbas, E. Bourtsoulatze, and Y. Andreopoulos, "Graph-based object classification for neuromorphic vision sensing," in *Proc. IEEE Int. Conf. Comput. Vision*, 2019, pp. 491–501.

[21] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2016, pp. 1933–1941.

[22] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A dataset of 101 human actions classes from videos in the wild," *arXiv:1212.0402*, 2012.

[23] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "HMDB: A large video database for human motion recognition," in *Proc. IEEE Int. Conf. Comput. Vision*, 2011, pp. 2556–2563.

[24] O. Kliper-Gross, T. Hassner, and L. Wolf, "The action similarity labeling challenge," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 34, no. 3, pp. 615–621, 2011.

[25] E. Mueggler, C. Bartolozzi, and D. Scaramuzza, "Fast event-based corner detection," in *Proc. Brit. Machine Vision Conf.*, vol. 1, 2017.

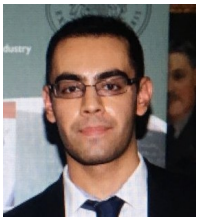
[26] E. Mueggler, H. Rebecq, G. Gallego, T. Delbrück, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM," *The Int. Journal of Robotics Research*, vol. 36, no. 2, pp. 142–149, 2017.

[27] X. Clady, J.-M. Maro, S. Barré, and R. B. Benosman, "A motion-based feature for event-based pattern recognition," *Frontiers in Neuroscience*, vol. 10, p. 594, 2017.

[28] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow," *IEEE Trans. Neural Networks and Learning Syst.*, vol. 25, no. 2, pp. 407–417, 2014.

[29] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman, "HOTS: A hierarchy of event-based time-surfaces for pattern recognition," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 39, no. 7, pp. 1346–1359, 2016.
[30] M. Cannici, M. Ciccone, A. Romanoni, and M. Matteucci, "Attention mechanisms for object recognition with event-based cameras," *arXiv:1807.09480*, 2018.
[31] A. Chadha, Y. Bi, A. Abbas, and Y. Andreopoulos, "Neuromorphic vision sensing for CNN-based action recognition," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Process.*, 2019, pp. 7968–7972.
[32] P. U. Diehl, D. Neil *et al.*, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, 2015, pp. 1–8.
[33] E. Stromatias, D. Neil *et al.*, "Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on SpiNNaker," in *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, 2015, pp. 1–8.
[34] Q. Wang, Y. Zhang, J. Yuan, and Y. Lu, "Space-time event clouds for gesture recognition: From RGB cameras to event cameras," in *Proc. IEEE Winter Conf. Appl. of Comput. Vision*, 2019, pp. 1826–1835.
[35] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2017, pp. 652–660.
[36] K. Lee, H. Woo, and T. Suk, "Point data reduction using 3D grids," *The Int. Journal of Adv. Manuf. Technol.*, vol. 18, no. 3, pp. 201–210, 2001.
[37] A. A.-K. Jeng and R.-H. Jan, "The r-neighborhood graph: An adjustable structure for topology control in wireless ad hoc networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 4, pp. 536–549, 2007.
[38] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.
[39] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv:1312.6203*, 2013.
[40] M. Fey, J. Eric Lenssen, F. Weichert, and H. Müller, "SplineCNN: Fast geometric deep learning with continuous B-spline kernels," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2018, pp. 869–877.
[41] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on Riemannian manifolds," in *Proc. IEEE Int. Conf. Comput. Vision Workshops*, 2015, pp. 37–45.
[42] F. Monti, D. Boscaini *et al.*, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2017, pp. 5115–5124.
[43] L. Piegl and W. Tiller, *The NURBS book*. Springer Science & Business Media, 2012.
[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML*, 2015.
[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2016, pp. 770–778.
[46] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. 32nd AAAI Conf. Artificial Intell.*, 2018.
[47] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, vol. 9, p. 437, 2015.
[48] T. Serrano-Gotarredona and B. Linares-Barranco, "Poker-DVS and MNIST-DVS. Their history, how they were made, and other details," *Frontiers in Neuroscience*, vol. 9, p. 481, 2015.
[49] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "CIFAR10-DVS: An event-stream dataset for object classification," *Frontiers in Neuroscience*, vol. 11, p. 309, 2017.
[50] P. Lichtsteiner, C. Posch, and T. Delbrück, "A 128x128, 120 dB 30mW latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.
[51] Y. Bi and Y. Andreopoulos, "PIX2NVS: Parameterized conversion of pixel-domain video frames to neuromorphic vision streams," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, 2017, pp. 1990–1994.
[52] G. Orchard, C. Meyer *et al.*, "HFirst: A temporal approach to object recognition," *IEEE Trans. Pattern Analysis and Machine Intell.*, vol. 37, no. 10, pp. 2028–2040, 2015.
[53] D. Neil, M. Pfeiffer, and S.-C. Liu, "Phased LSTM: Accelerating recurrent network training for long or event-based sequences," in *Advances in Neural Inf. Process. Syst.*, 2016, pp. 3882–3890.
[54] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. ICLR*, 2019.
[55] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2017.
[56] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.
[57] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," in *Proc. 31st AAAI Conf. Artificial Intell.*, 2017.
[58] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *Proc. ICLR*, 2017.
[59] Y. Yuan, Y. Zhao, and Q. Wang, "Action recognition using spatial-optical data organization and sequential learning framework," *Neurocomputing*, vol. 315, pp. 221–233, 2018.
[60] Y. Yuan, D. Wang, and Q. Wang, "Memory-augmented temporal dynamic learning for action recognition," in *Proc. AAAI Conf. on Artificial Intell.*, vol. 33, 2019, pp. 9167–9175.
[61] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, "Feedforward categorization on AER motion events using cortex-like features in a spiking neural network," *IEEE Trans. Neural Networks and Learning Syst.*, vol. 26, no. 9, pp. 1963–1978, 2014.
[62] X. Peng, B. Zhao, R. Yan, H. Tang, and Z. Yi, "Bag of events: An efficient probability-based feature extraction method for AER image sensors," *IEEE Trans. Neural Networks and Learning Syst.*, vol. 28, no. 4, pp. 791–803, 2016.
[63] Y. Hu, H. Liu, M. Pfeiffer, and T. Delbrück, "DVS benchmark datasets for object tracking, action recognition, and object recognition," *Frontiers in Neuroscience*, vol. 10, p. 405, 2016.
[64] Z. Qiu, T. Yao, and T. Mei, "Learning spatio-temporal representation with pseudo-3D residual networks," in *Proc. IEEE Int. Conf. Comput. Vision*, 2017, pp. 5533–5541.
[65] D. Tran, H. Wang *et al.*, "A closer look at spatiotemporal convolutions for action recognition," in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2018, pp. 6450–6459.
[66] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3D CNNs retrace the history of 2D CNNs and ImageNet?" in *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, 2018, pp. 6546–6555.
[67] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Process.*, 2015, pp. 4580–4584.
[68] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Inf. Process. Syst.*, 2017, pp. 5099–5108.

Yin Bi received the BSc degree in electrical engineering and the MSc degree in signal and information processing from Northeastern University, China. He obtained his PhD in Electrical and Electronic Engineering from University College London, under the supervision of Professor Yiannis Andreopoulos. His research interests are in machine learning, signal processing for neuromorphic vision sensing cameras and multimedia systems.



Aaron Chadha received the M.A. and M.Eng. degrees (Hons.) in Information and Computer Engineering from the University of Cambridge in 2014 and the Ph.D. degree in Computer Vision and Machine Learning from University College London in 2018. He is currently working as a Senior Machine Learning Engineer at iSize Technologies. His research interests and expertise are in machine learning and computer vision, including image/video recognition, compression and domain adaptation. He has experience in working with multiple modalities and event-based neuromorphic vision sensing.

Alhabib Abbas obtained his BSc in Electrical and Electronic Engineering from the University of Bahrain, and his MSc and PhD from University College London, U.K. He is currently a post-doctoral researcher in machine learning and video analysis under the supervision of Professor Yiannis Andreopoulos. His research interests are in machine learning, image and video semantic analysis, and error-tolerant computing in multimedia systems.

Eirina Bourtsoulatze [S'11 - M'14] received the Diploma in Electrical and Computer Engineering from the Aristotle University of Thessaloniki, Greece, in 2008 and the Ph.D. degree in Electrical Engineering from the Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland, in 2013. She is currently a Lecturer at the School of Computer Science and Electronic Engineering at the University of Essex, Colchester, UK. Her research interests include machine learning for communications, image/video processing, information-centric networks, network coding and multimedia communications.

Yiannis Andreopoulos obtained the Electrical Engineering Diploma and an MSc from the University of Patras, Greece, and the PhD in Applied Sciences from the Vrije Universiteit Brussel, Belgium. He is Professor of Data and Signal Processing Systems at the Electronic and Electrical Engineering Department of University College London, London, U.K. His research interests are in machine learning and multimedia systems. In his academic work, he has made major contributions to image/vision computing and large-scale multidimensional data analysis. In collaboration with industry, elements of his research in video processing systems have been integrated into commercial products. Author of 150+ papers, 3 patents, 10+ contributions to standards and 3M+ in research grants as Principal Investigator at UCL.