Graph-based Feature Learning for Neuromorphic Vision Sensing

Yin Bi

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Electronic and Electical Engineering

University College London

September 1, 2020

I, Yin Bi, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

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, neuromorphic vision sensing comes with two key challenges: (i) the lack of large-scale annotated datasets to train advanced machine learning frameworks with; (ii) feature representation for NVS is far behind that of APS-based counterparts, resulting in lower accuracy for high-level computer vision tasks.

In this thesis, we attempt to bridge these gaps by firstly proposing an NVS emulation framework, termed as PIX2NVS, that converts frames from APS videos to emulated neuromorphic spike events so that we can generate large annotated NVS data from existing video frame collections (e.g., UCF101, YouTube-8M, YFCC 100m, etc.) used in machine learning research. We evaluate PIX2NVS with three proposed distance metrics and test the emulated data on two recognition applications.

Furthermore, given the sparse and asynchronous nature of NVS, we propose a compact graph representation for NVS, which allows for end-to-end learning with graph convolutional 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 our proposed residual-graph CNN (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 in order to efficiently model temporal dependencies over multiple graphs and allow for long temporal extent. We show that performance of this framework generalizes to object classification, action recognition, action similarity labeling and scene recognition, with state-of-the-art results. Importantly, our framework preserves the spatial and temporal coherence of spike events, while requiring less computation and memory.

Finally, to address the absence of large real-world NVS datasets for complex recognition tasks, we introduce, evaluate and make available a 100k dataset of NVS recordings of the American Sign Language letters (ASL-DVS) acquired with an iniLabs DAVIS240c device under real-world conditions, as well as three neuromorphic human action dataset (UCF101-DVS, HMDB51-DVS and ASLAN-DVS) and one scene recognition dataset (YUPENN-DVS) recorded by the DAVIS240c capturing their screen playback reflectance.

Impact Statement

This PhD thesis is directly related to an EPSRC project, The Internet of Silicon Retinas (IoSiRe, EP/P02243X/1). The work of Chapter 3 of this thesis had contributed to research developments of neuromorphic community. Namely, we proposed an NVS emulation framework, named as PIX2NVS, that can convert frames from APS videos to neuromorphic spike events. Therefore, PIX2NVS can efficiently address the need for larger and more diverse datasets in the neuromorphic neuromorphic community: researchers can leverage PIX2NVS to generate large annotated NVS data for their own academic research.

In addition, in Chapter 4 we proposed a graph based representation for neuromorphic events to keep their sparse and asynchronous nature, and we post-process the graph by using graph convolutional networks. This is the first time to process neuromorphic events as graphs via graph convolutional neural networks. Our results from the proposed models in Chapter 4 and Chapter 5 outperform traditional CNNs, setting the new benchmark, while require less computation and memory. Therefore, we provide an important direction of applying graph CNNs to event-based cameras to explore for the neuromorphic community.

Moreover, we address the lack of NVS data for training and inferencing complex recognition tasks by introducing a 100k dataset of NVS recordings of the American Sign Language letters under real-world conditions, as well as three human action datasets and one scene recognition dataset recorded by a NVS camera capturing their screen playback reflectance. To the best of our knowledge, they are the largest datasets in NVS community. All of these datasets are available and free to download for the research community, and these datasets can advance the field of neuromorphic signal processing and machine learning.

Acknowledgements

This thesis benefits tremendously from many people.

First and foremost, I would like to express my sincere thanks to my advisor Prof. Yiannis Andreopoulos. During the work with him, I am deeply inspired by his passion on exploring new domain, his vision and ability in dealing with challenging problems, his profound knowledge and respectful personality. In the constant discussion with him, he offered me a lot of valuable guidance and suggestion, which greatly contributed to my research. I highly appreciate his persistent support and generous encouragement. It is my great honour to work with him.

I would also like to express special thanks to my colleagues, Dr. Aaron Chadha, Dr. Alhabib Abbas and Dr. Eirina Bourtsoulatze. I am truly expressed with their concentration on work, rigorous academic attitude, enthusiasm and kindness. In the cooperation with them, they always gave me constructive feedback and insightful comments. I really owe many thanks to them for their constant and selfless assistance. It is a great pleasure to have frequent informal discussion with them.

I would like to extend my gratitude to the dissertation committee members: Dr. Toni Laura and Dr. Nishanth Sastry for their time and energy spent on the review of my thesis. I am very grateful for the research opportunity and environment provided by the Communications and Information Systems Group (CISG) in Dept. of EEE. I also want to express my sincere gratitude to UCL for awarding me the scholarship to support my PhD study.

Last but not least, I would express my deepest thanks to my parents. Their unconditional support and love is the biggest comfort in my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Neuromorphic Vision Sensing (NVS)

1.1.1 Neuromorphic Vision Sensor

Conventional active pixel sensing (APS) comprises frame-based vision sensing, which consists of a 2D array of synchronous pixels capturing light intensity at a fixed time interval, corresponding to the frame capture rate. However, APS-based sensing representations are known to be cumbersome for machine learning systems, due to: limited frame rate, too much redundancy between successive frames; calibration problems under irregular camera motion; blurriness due to shutter adjustment under varying illumination; and very high power requirements [9]. In biology, it is known that the mammalian retina converts raw light into electrical spikes in proportion to the relative change in light intensity over time or space; then the spikes are transmitted to the brain to stimulate high level perception and reaction. This process extracts all the essential feature of visual scenes rapidly and reliably even under dynamic lighting conditions, and is known to be extremely energy-efficient and bandwidth-efficient [10, 11]. Motivated by the biological vision systems that outperform conventional vision sensor in almost every aspect, the neuromorphic hardware community is developing a range of new sensing sensors that imitate the behaviour of biological retina and subsequent vision processing, collectively termed in this thesis as neuromorphic vision sensing (NVS).

The main principle behind the operation of neuromorphic vision sensing is il-

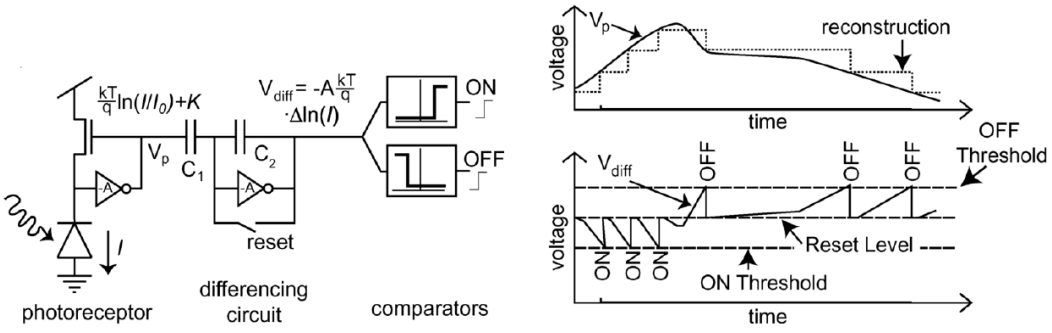


Figure 1.1: NVS pixel architecture and the principle of operation. Left: each pixel of sensor consists of a logarithmic photoreceptor, a differencing circuit, and two comparators. Right: illustration of operation of differencing circuit and comparators. Figure is repreduced from [1].

lustrated in the Fig. 1.1. As to the principle of operation, the events are computed asynchronously by each NVS pixel as illustrated. The continuous-time photoreceptor output, which encodes intensity logarithmically, is monitored for changes since the last event was emitted by the pixel. A detected change in *log* intensity, which crosses a threshold value, results in the emission of an ON or OFF event, e.g., the threshold for dynamic vision sensor (DVS) is typically set to about 10% contrast [1]. Communication of the event to the periphery resets the pixel, which causes the pixel to memorize the new *log* intensity value.

NVS sensor is a temporal difference device, each pixel only outputs events in response to the change of illumination [12]. Given a static scene where illumination intensity is constant over a period time, these pixels will not output any events. The output events are dependent on the activity of the scene. That is, when the logarithm of the intensity value of a CMOS sensor grid position changes, then a spike event is generated. Each pixel output events in the Address Event Representation (AER) protocol, a standard interfacing protocol for neuromorphic engineering, which contains the physical address of the pixel in the array, and generates a single bit to indicate whether the illumination on the pixel is increased or decreased. The following notation is commonly used to represent an event emitted by a pixel:

$$\{e_i\}_N = \{x_i, y_i, t_i, p_i\}_N \tag{1.1}$$

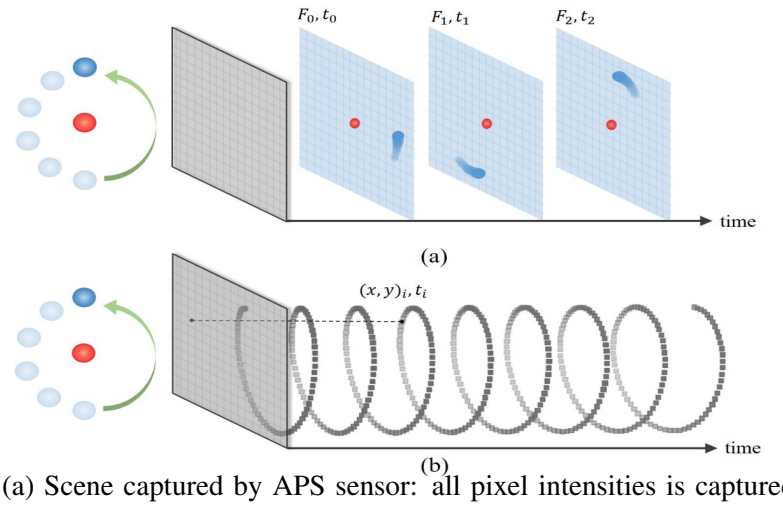


Figure 1.2: (a) Scene captured by APS sensor: all pixel intensities is captured at a fixed frame rate. (b) Scene captured by NVS sensor: only the pixel intensity change is asychrounously captured. Figure is repreduced from [2].

where N is the number of total events in a scene and i is the index of these events. The event e_i indicates that the pixel located at the address (x_i, y_i) on the camera sensor generates an event at time t_i. with the polarity of the change in illumination encoded as $p_i \in (-1, 1)$. Polarity $p_i = 1$ is referred to as an ON event, representing the increase in illumination, while $p_i = -1$ correspondingly an OFF event, representing the decrease in illumination occurred. In the Fig.1.2, we show the same scene captured by the APS and NVS camera. In contrast, an APS device captures all pixel intensities at a fixed frame rate. Therefore, it captures a clear shape of stationary object (red ball) and a blurred shape of a fast moving object (blue ball); while for NVS sensor, it only captures an asynchronous sequence of intensity changes caused by the fast moving blue ball while information of stationary objects (red ball and background) will not be recorded.

Several variants of neuromorphic vision sensors are in production today and can be used as prototypes for actual experiments. One of them is Dynamic Vision Sensor (DVS) from iniVation [1]. DVS has 128×128 resolution, 120 dB dynamic range and 15 microsecond latency, and communicates with a host computer using USB 2.0 [1]. More recently, Brandli developed the dynamic and active pixel vision sensor (DAVIS) [13] that can capture both events and synchronous

[1] iniVation web: https://inivation.com/ (accessed in Feb. 2020)

greyscale frames. The DAVIS240C camera has 240×180 resolution, 130 dB dynamic range (55dB for greyscale frames) and communicates with a host computer using USB 2.0. To design event-based visual-inertial methods, DAVIS240c is embedded a time-synchronised inertial measurement unit (IMU) on board that provides accelerometer and magnetometer data. Recently, several DVS devices have been proposed by further lowering power consumption, minimum temporal contrast sensitivity and fixed pattern noise level [14–16]. Another type of neuromorphic vision sensor is the Asynchronous Time-based Image Sensor (ATIS) camera [17], which provides absolute intensity values along with events. ATIS combines the DVS temporal contrast pixel with a new time-based intensity measurement pixel. Scene intensity change at each pixel causes three consecutive events: the first one event encodes the polarity as the DVS pixel does, the other two encode an absolute greyscale value in the inter-event time interval. The main progresses of ATIS, compared to DVS, are its higher resolution (304×240), higher dynamic range (143 dB) and lower latency (3 *us*). Recently, colour dynamic and active-pixel vision sensors have been proposed [18, 19], which can capture spatial details with color and track movements with high temporal resolution while keeping the data output sparse and fast.

1.1.2 Advantages of Neuromorphic Vision Sensor

Unlike conventional frame-based cameras that tend to blur the image due to slow shutter speed, silicon retinas capture the illumination changes caused by fast object motion. Remarkably, neuromorphic sensors achieve this with: (i) microsecond-level latency; (ii) low power requirements, e.g., 10mW versus 1–4W for APS-based video cameras; (iii) robustness to uncontrolled lighting conditions, as no synchronous global shutter is used [20, 21]. In practice, this means that neuromorphic vision sensing (NVS) data from hardware, like the iniLabs DAVIS240c and the Pixium Vision ATIS cameras, can be rendered to representations comprising up to 2000 frames-per-second (fps). On the other hand, a typical APS video camera only captures (up to) 60 fps at more than 100 times the active power consumption and with shutter induced blurriness when rapid illumination changes take place. Therefore,

the neuromorphic vision sensor will be a good alternative to the conventional APS.

As to advantages of Asynchronous Event-based Vision, as the above description of DVS output, neuromorphic vision sensor provides a revolutionary way for visual acquisition. It changes the way of how visual data is acquired, transmitted, encoded and processed. The advantages of event based acquisition over conventional frame-based imagers lie in following aspects [12, 20, 21]: (i) Event based acquisition is redundancy free: the output from neuromorphic vision sensor compressed data in the form of events responding to illuminance changes, only the dynamic information stimulated by a moving object or by the change of light intensity is recorded. (ii) Event based acquisition is asynchronous: each pixel can trigger an event independently from others once light intensity change is perceived. No global clock is used, unlike conventional frame intensity cameras that are sampled by a fixed interval. The timing of DVS events can be conveyed with a minimum temporal resolution of 1 us. Thus, the 'effective frame rate' is typically of several kHz. (iii) Event based acquisition has large dynamic range: DVS has dynamic range of 120dB. A large dynamic range is essential for neuromorphic vision sensor working in different environments both indoors and outdoors. When conventional cameras encounter over- or under-exposure problems, DVS reacts correctly with the light intensity of the scene, untouched by its extreme light conditions. Other advantages of event-based acquisition include low latency thanks to the AER protocol. Low latency is especially suitable for robotic applications that require high-speed sensory motor coupling. DVS achieves also a very low power consumption of only 23mW.

1.1.3 Applications of Neuromorphic Vision Sensing

As neuromorphic vision sensing has the advantages such as low power comsumption, low latency, and sparse and asynchoronous nature, it is much more suitable for developing real-time and efficient vision tasks for mobile devices. There exists many applications with the advance of the NVS devices, and the dominant applications include visual tracking, detection and recognition, reconstruction, localisation and mapping, and estimating motion of objects. In the following section, we will briefly describe the current work about these applications.

Object visual tracking: Visual tracking aims to locate a moving object over time and finds a number of applications in computer vision field including video surveillance, robotics, human-computer interface and vehicle navigation. NVS devices are extremely adaptable to object tracking because of its spatial temporal encoding of movement. To simplify the problem, initial work focus on event camera motion tracking with artificially pre-created or known scenes. For example, a particle filter based tracking algorithm was proposed by Weikersdorfer and Conradt [22]. This system tracks relatively slow 3-DoF planar robot motion, and the event-based sensor observes planar scenes parallel to the plane of motion. Mueggler proposed an on-board 6-DoF localisation flying robot system equipped with a DVS that is able to track very rapid motion such as flips of a quad-rotor [23] . In this system, a black and white line-based known target is required and the current 3D position and orientation is estimated by minimising the point-to-line projection error. Recently, Gallego proposed a more general 6-DoF motion tracking algorithm, which is able to tracking fast motions in realistic and natural scenes with a single event camera [24].

To bridge the gap that NVS cameras do not capture synchronous reference information (i.e. reference image frames or 2D/3D maps), Censi combined a NVS camera with a standard grey-scale camera to estimate the small relative motion from the previous frame of a standard camera for every incoming event [25]. However, this method is subject to high latency, motion blur and low dynamic range because of their reliance on a standard camera, thus discarding the advantages of using NVS cameras and introducing extra complications including synchronisation and calibration problems.

As described in Section 1.1.1, DAVIS240c is embedded an IMU unit that has a high potential for the complementary nature of event and inertial data. Therefore, Yuan included the IMU of the DAVIS camera to keep the camera upright in order to support their vertical line-based tracking [26] . Zhu fused a purely event-based tracking algorithm with an IMU, to provide accurate metric tracking of a camera's full 6-DoF pose [27]. Recently, Rebecq also proposed an accurate key frame-based, tightly-coupled visual-inertial odometry algorithm based on nonlinear optimisation,

showing their system works well even in challenging conditions by fully utilising the NVS cameras' outstanding properties [28]. Recently, Renner proposed a fully event-driven vision and processing system for selective attention and tracking [29]. This application is realized on a neuromorphic processor Loihi [30] interfaced to an event-based Dynamic Vision Sensor DAVIS, where attention mechanism is realized as a recurrent spiking neural network. This attention mechanism is created by configuring a population of neurons with a winner-take-all connectivity of dynamic neural fields, which creates sustained activation that supports object tracking.

Detection and Recognition: NVS cameras only capture the motion and represent it sparsely as a stream of events, which are naturally encoding spatio-temporal motion information; therefore, they are extremely adaptable to detection and recognition tasks as the background information is already subtracted on the sensor. A joint project between ETHZ and Samsung conducted a human gesture recognition interface based on a stereo pair of DVS cameras. The motion trajectory of a moving hand is detected spatio-temporally by the output events of DVS cameras. Neuromorphic approach is used to track correlated events and the stereo vision is used to strengthen robustness. Finally, gesture patterns are classified by applying a hidden Markov model (HMM) based method [31]. Recently, researchers from IBM Research developed a low power, fully event-based gesture recognition system [32]. This work implements deep convolutional neural networks end-to-end on event-based neuro-synaptic processor TrueNorth to recognize hand gestures in real-time.

Most methods follow the principle that feature descriptors are firstly extracted from NVS data streams and then a classifier is applied for the recognition. Examples are time-surfaces feature (e.g., HFirst [33], HOTS [34] and HATS [35]), a time oriented approach to extract spatio-temporal features that are dependent on the direction and speed of motion of the objects. Ramesh introduced a generic visual descriptor, termed as distribution aware retinal transform (DART), that encodes the structural context using log-polar grids for event cameras [36]. Importantly, DART represents a significant step forward in computing a structural descriptor for event-based data and can be used in four applications including object classifica-

tion, tracking, detection and feature matching. Apart form descriptors, there are end-to-end methods for detection and recognition. They are neither pair an event stream with standard frame-based deep convolutional neural network (CNN) or recursive architectures [37–40] or use spike neural network (SNN) to directly learn patterns [41–44]. All of these methods progress the development of recognition method for neuromorphic events.

As to detection, a limited number of work addresses object detection. Liu *et al.* focused on combining a frame-based CNN detector to facilitate the event-based module [45]. It is true that using deep neural networks for event-based object detection may achieve good performance with lots of training data and computing power, but they go against the idea of low-latency, low-power event-based vision. In contrast, Lenz *et al.* presented a practical event-based approach to face detection by looking for pairs of blinking eyes [46]. While this is applicable to human faces in the presence of activity, it is not a general purpose for event-based object detection method in case of different shapes or occluded objects.

3D Reconstruction: It is difficult to realise 3D reconstruction purely based on a stream of events; therefore researchers used multiple event cameras to tackle this problem. Carneiro *et al.* presented a N-camera 3D reconstruction algorithm and applied it to multi-cameras systems of event-based sensors [47]. By using geometrical and temporal constraints, this method can achieve highly accurate reconstructions despite the low spatial resolution of the NVS cameras. Belbachir utilised a special event-based depth camera, which consists of a pair of bio-inspired dynamic vision line sensors and creates real-time 3D 360° panoramas [48], to realise real-time 3D panoramic reconstruction of natural scenes with dense vertical resolution and high dynamic range properties. This system also potentially has a low processing cost and low power consumption. Matsuda *et al.* combined a NVS camera with an active projector to develop a new structured light scanning system called the Motion Contrast 3D Laser Scanner (MC3D), achieving high quality 3D object reconstruction that is better than laser scanners or RGB-D cameras [49]. By replacing a standard camera with a NVS camera, they managed to avoid performance trade-offs

in acquisition speed, resolution, and efficiency that traditional APS devices struggle. Recently, Rebecq proposed a more lightweight event-based 3D reconstruction method, which only requires a single moving event camera and its trajectory provided by an external pose estimator [50]. In contrast to traditional multi-view systems that estimate dense 3D structure from a set of known viewpoints by solving the data association problem, this method estimates semi-dense 3D structure without the need for explicit data association or pixel intensity values, and accumulates the number of rays from an event pixel for every incoming event, then finds 3D structure information from it.

Localisation and mapping: Performing localisation and mapping simultaneously is a difficult problem, especially when only a stream of NVS events is available. Therefore, researchers simplify this problem by imposing restrictions on motion and scenes, or combining an NVS camera with other sensors. For instance, Weikersdorfer *et al.* proposed a 2D SLAM method to track a ground robot's pose while reconstructing a planar ceiling map with an upward DVS camera [22]. Later, they proposed an event-based 3D SLAM method with a DVS camera combined with a RGB-D sensor to overcome the limitations of their previous work [51]. The multi-camera system is able to produce a stream of sparse events with depth information by finding pixel correspondences between two sensors through off-line calibration, which enables the estimation of semi-dense 3D structure of the scene. Similarly, Kueng *et al.* developed event-based low latency visual odometry algorithm by utilising DAVIS cameras that can capture both events and frames simultaneously [52]. They used image frames to update visual features, and the feature is used for mapping by probabilistic depth filtering.

The first event-based 2D and 3D SLAM method was proposed in [53] and [54], they realised joint estimation of general 6-DoF camera motion, scene intensity and scene 3D depth from pure event data. Later, Milford *et al.* presented a simple visual odometry system by using a DVS camera with loop closure and accumulating frames from events, to investigate a large scale visual SLAM problem [55]. However, relying on artificial event frames created by accumulating events within a time

interval introduces unnecessary bounds on the update rate and latency. To overcome this problem, Kim proposed a method that performs real-time 3D reconstruction from a single hand-held event camera with no additional sensing and works in unstructured scenes where there is no prior knowledge [54]. This method is based on three decoupled probabilistic filters, each estimating 6-DoF camera motion, scene logarithmic (log) intensity gradient and scene inverse depth, on which a real-time graph is built to track and model over an extended local work space.

Estimating Visual Motion: Accurate and fast measurement of motion, also termed as optical flow, is a necessary requirement for using this flow in vision tasks such as detecting moving obstacles, visual navigation, acquiring structure from motion information or motion-based recognition. Currently in spite of the large number of optical flow algorithms, the majority of these methods resemble the original formulation of Horn-Schunck method [56] and Lucas-Kanade method [57]. Their high accuracy requires massive and complex computation and diminishes their usability in real-time applications. For instance, the highest-ranking algorithm on the Middlebury benchmark [58] takes 11 minutes for two frames. However, the development of the asynchronous NVS devices make it possible to propose promising new approaches to visual signal processing.

As described in Section 1.1.1, in contrast to conventional image sensors, the NVS cameras do not produce frames but asynchronous address-events as output. They indicate positive and negative changes in log intensity at each pixel address, generating ON and OFF events respectively. This approach has several advantages and optical flow algorithms operating on the NVS output can benefit from these characteristics. Frame-based methods suffer the large displacements problem that occurs in fast motion, while event-based method provides solution to this problem. For instance, Benosman [59] made use of the high temporal precision of NVS data by computing gradients on the surface consisting of most recent events. Another problem of conventional optical flow methods is motion discontinuities at object boundaries, where at least two distinct motions overlap, while in event-based datasets, they do not have motion discontinuities, because the motion is generated

by camera rotation. In case of NVS cameras, contrast edges at motion discontinuities would generate events exactly at the discontinuities where Barranco [60] extract the location and motion of contours. Furthermore, they combined NVS events with intensity frames to reduce computational cost and increase performance and stability. Barranco [61] developed a phase-based method to improve estimation in textured regions. Brosch [62] provided a comprehensive analysis of event-based visual motion estimation. Recently, they suggested an event-based flow computation method using biologically inspired filter-banks that detect the orientation of an edge. Orchard and Etienne-Cummings proposed a spiking neural network architecture, which uses synaptic delay to create receptive field that is sensitive to motion [63]. All these developments are indicative of the potential to resolve major problems of conventional frame-based flow estimation by using NVS devices.

Related Demonstrations: Here we review some interesting demos implemented by using NVS devices. The following demos were conducted at Neuroinformatics Institute of ETHZ, Switzerland, where a series of NVS cameras were invented. (i) A ping-pong player has been built to demonstrate the sensor's high dynamic characteristics and the potential of coupling NVS cameras with high-speed sensory motor systems [64]. In this demo, the robot arm hits the balls immediately after they have arrived, in which a small latency of 2.8 ± 0.5ms is measured. (ii) A rod balancing robot is developed for the same demonstration purpose. The robot is used to stabilize a pencil by applying event based Hough line transform [65]. The 3D locations of the pencil are calculated by using a pair of DVS cameras mounted on different angles. The lightweight tracking and control algorithms are computed on an ARM7 microprocessor, which shows the practicability and the advantages of integrating DVS into embedded platforms with limited resources. (iii) Another demo is the development of an object fall detector by tracking vertically falling point clouds, dedicated to elderly people home caring [66].

1.2 Challenges and Contribution

As described in Section 1.1.2, NVS has many advantages, such as low latency, low power requirements, robustness to uncontrolled light, high dynamic range, *etc.*; therefore, NVS devices and processors have great potential to solve the limitations that APS devices suffer and replace the APS devices in many applications. However, when working with neuromorphic vision sensing, there are two challenges: (i) lack of large annotated datasets; (ii) machine learning algorithms for NVS are inferior to APS-based counterparts. We will illustrate these challenges that motivate our work in details, and also illustrate our corresponding solutions and contributions in following sections.

Challenge 1: One crucial issue is that current work in computer vision with NVS-based hardware focuses on relatively simple datasets like MNIST digit classification, or basic motion or geometric shape recognition. To the best of our knowledge, there is currently no NVS-based framework for large-scale multi-class human action recognition problems corresponding to datasets like UCF101 and HMDB. Therefore it is uncertain whether current algorithms can also perform well in complex scenarios in real world. It is necessary to build new benchmark datasets in the NVS domain, in order to advance the state-of-the-art in data-driven learnable NVS-based computer vision systems. In terms of this challenge, we provide two solutions to solve the lack of large labeled datasets in following.

1. We propose a NVS emulation, termed PIX2NVS, that converts frames from APS videos to emulated neuromorphic spike events so that we can generate large annotated NVS data from existing video frame collections (e.g., UCF101, YouTube-8M, YFCC 100m, etc.) used in data-driven learnable computer vision systems. Moreover, we propose three distance metrics to quantify the accuracy of the model-generated events against ground-truth event streams streams. These distance metrics can also be used to search for better parameter settings for PIX2NVS. Finally, we evaluate the efficacy of emulated NVS datasets by elaborating on the training and testing aspects for two separate applications human action recognition and sign language

recognition. Our results show that training with emulated spike events leads to marginal loss of accuracy for both applications.

2. We address the lack of NVS data for training and inferencing complex recognition tasks by introducing a 100k dataset of NVS recordings of the American Sign Language letters (ASL-DVS) acquired with an iniLabs DAVIS240c device under real-world conditions, as well as three neuromorphic human action datasets (UCF101-DVS, HMDB51-DVS and ASLAN-DVS) and one scene recognition dataset (YUPENN-DVS) recorded by the DAVIS240c capturing their screen playback reflectance. To our best of knowledge, they are amongst the largest dataset in NVS community. All of these datasets are available to the research community.

Challenge 2: Even though NVS devices have many advantages, designing algorithms that respond to asynchronous, sparse, yet accurately timed information is still not typical in computer vision. Crucially, the output of NVS device is fundamentally different from the frame-based data that is captured at a fixed global sampling rate. The image processing community is accustomed to processing images by looping over the entire image pixels with one frame as a basic processing unit. In the case of NVS, the major challenge is that an event should be the basic processing unit with its timestamp information encapsulating the most crucial information. Therefore, current algorithms for conventional computer vision cannot be directly applied to NVS data, and new ways of processing are needed for such data streams. In terms of this challenge, we provide two solutions to address the problems in NVS data representation and processing in following.

1. We propose a graph based representation for neuromorphic events, allowing for fast end-to-end graph based training and inference. We design a new graph-based spatial feature learning module, and we evaluate performance of this module on object classification. Our results show that this approach requires less computation and memory in comparison to conventional CNNs, while achieving superior results to the state-of-the-art in various datasets.

2. We extend our spatial feature learning module to spatial-temporal feature learning module with our Graph2Grid block and temporal feature learning module for efficiently modelling coarse temporal dependencies over multiple graphs. We evaluate performance of the learning framework on action recognition, action similarity labeling and scene recognition. Results show that our framework sets the new benchmark in all tasks.

Although we provide potential solutions to the existing challenges, there are some shortcomings of our proposed methods. As to the PIX2NVS, the timestamp of generated events are largely restricted by the frame of rate of video collections and generated events are much cleaner that my decrease the robustness of developed algorithms. As to the graph-based feature learning, a key limitation is that graphs are constructed from a fixed-time window of events, which may not adaptable for various applications. One potential solution is constructing graph dynamically as we will discussed in the future wrok.

We will list our research outcomes here. The work completed during this PhD has resulted in four conference publications. There is also a journal paper submitted to IEEE transaction on image processing which is under review. We also note that we only present a part of our PhD work in this thesis, and more is included in our papers. The following are our publication.

1. **Yin Bi**, Aaron Chadha, Alhabib Abbas, Eirina Bourtsoulatze, Yiannis Andreopoulos, " Graph-based Spatial-temporal Feature Learning for Neuromorphic Vision Sensing[J] ", IEEE Transactions on Image Processing

2. **Yin Bi**, Aaron Chadha, Alhabib Abbas, Eirina Bourtsoulatze, Yiannis Andreopoulos, " Graph-Based Object Classification for Neuromorphic Vision Sensing[C] ", IEEE International Conference on Computer Vision, 2019

3. **Yin Bi**, Yiannis Andreopoulos, " PIX2NVS: Parameterized Conversion of Pixel-domain Video Frames to Neuromorphic Vision Streams[C] ", IEEE International Conference on Image Processing, 2017

4. Aaron Chadha, **Yin Bi**, Alhabib Abbas, Yiannis Andreopoulos, " Neuromorphic Vision Sensing for CNN-based Action Recognition[C] ", IEEE International Conference on Acoustics, Speech and Signal Processing, pages 7968-7972, 2019

5. Martini Maria, Khan Nabeel, **Yin Bi**, Yiannis Andreopoulos *et al.*, "Challenges and Perspectives in Neuromorphic-based Visual IoT Systems and Networks." IEEE International Conference on Acoustics, Speech and Signal Processing, 2020.

Besides the publications, this thesis led to four public datasets for community and one open-source software.

1. The largest object classification dataset (ASL-DVS) and three multi-classes and complex human action datasets (UCF101-DVS, HMDB-DVS and ASLAN-DVS) are available online[2][3].

2. An open-source simulator PIX2NVS is available online [4].

1.3 Thesis Structure

Here we provide an overview of the remaining chapters of this thesis. Chapter 2 reviews recent work published in the field of feature extraction for neuromorphic vision sensing streams and graph-based deep learning.

Chapter 3 firstly introduces framework of PIX2NVS. We also propose and evaluate three distance metrics to quantify the accuracy of the model-generated events against ground-truth, and describe the method of optimizing the parameter of PIX2NVS with random search. Finally, we test on two applications including human action recognition and American sign language recognition to evaluate the effectiveness of emulated NVS data.

Chapter 4 proposes a graph based representation for neuromorphic events to keep their sparse and asynchronous nature and we couple this representation with

[2]https://github.com/PIX2NVS/NVS2Graph
[3]https://github.com/PIX2NVS/NVS_FeatureLearning
[4]https://github.com/PIX2NVS/PIX2NVS

novel residual graph CNN architectures that efficiently preserve the spatial and temporal coherence of spike events for the object classification. Also, we introduce and make available our ASL-DVS datasets in this chapter.

Chapter 5 proposes a framework for spatio-temporal feature learning by extending spatial feature learning framework in Chapter 4 with our proposed Graph2Grid block and temporal feature learning module for efficiently modelling temporal dependencies over multiple graphs and a long temporal extent. Then we accommodate this end-to-end feature learning framework to both appearance-based and motion-based tasks. Specifically, we apply this model into three different applications including action recognition, action similarity labeling and scene recognition, and followed by experimental validation.

In the last chapter, we conclude the thesis by briefly summarizing the main content of this thesis and laying out potential prospects of future event-based research based on the insights we gained from the body of work in this thesis.

Chapter 2

Literature Review

In this chapter, we review recent work published in the field of feature extraction for neuromorphic vision sensing streams and graph-based deep learning. When reviewing the feature extraction for neuromorhic vision sensing, we start by describing the feature descriptors extraction for NVS streams, followed by the end-to-end feature learning methods. Specifically, we mainly divide end-to-end feature learning methods into frame-based and event-based methods. As our feature learning framework is based on graph, therefore we also simply review the development of graph-based deep learning in this chapter.

2.1 Feature extraction for NVS streams

There are mainly two types of feature representation: handcrafted feature extraction and end-to-end trainable feature learning in neuromorphic vision sensing area. Moreover we divide end-to-end trainable feature learning methods into two categories: frame-based and event-based methods. We will simply review these methods in the following sections.

2.1.1 Handcrafted feature descriptors

Handcrafted feature descriptors are widely used by neuromorphic vision community. Some of the most common descriptors are corner detectors and line/edge extraction [67–70]. Clady proposed a method that relies on the use of space–time properties of moving edges [67]. Specifically, Corner events are defined as the spatio-temporal locations, which is constrained by the motion attributes of the edges

with respect to their spatio-temporal locations using local geometric properties of visual events. Vasco proposed method that adapts the commonly used Harris corner detector to the event-based data, since an event-based camera naturally enhances edges in the scene, which simplifies the detection of corner features [68]. Muggler proposed a method to reduce an event stream to a corner event stream [69]. This method extracts relevant tracking information (corners do not suffer from the aperture problem) and decreases the event rate for later processing stages, which can processes event by event with very low latency. Later Muggler leveraged the continuous-time representation to perform visual-inertial odometry since a continuous-time representation of the event camera pose can deal with the high temporal resolution and asynchronous nature in a principled way [71]. This representation allows direct integration of the asynchronous events with micro-second accuracy and the inertial measurements at high frequency. 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 [60–63, 72, 73]. Barranco compared image motion estimation with asynchronous event-based cameras to Computer Vision approaches using as input frame-based video sequences, which takes image motion as "contour motion" [60]. Algorithm presented that accurate contour motion is estimated from local spatio–temporal information for two camera models: the dynamic vision sensor (DVS), which asynchronously records temporal changes of the luminance, and a family of new sensors which combine DVS data with intensity signals. Later, a new paradigm based on asynchronous event-based data provides an interesting alternative and has shown to provide good estimation at high contrast contours by estimating motion based on very accurate timing. Inspired by this paradigm, Barranco presentd a simple method for locating those regions, and a novel phase-based method for event sensors that estimates more accurately these regions [61]. Brosch commented that a gradient-based motion detection and integration scheme, using the scheme of Lucas and Kanade, can be utilized to numerically estimate

second-order spatio-temporal derivatives on a function that represents the temporal derivative of the luminance distribution. This requires to employ proper numerical difference schemes which also demonstrates the disadvantage of increased noise sensitivity [62]. He also pointed out that methods exploiting the local structure of the cloud of events are more robust in general. However, the goodness of fit depends on the size of the spatio-temporal neighborhood. if we consider a neighborhood that is too small then the plane fit may eventually become arbitrary and thus instable. If the neighborhood is too large then the chances increase that the event cloud contains structure that is not well approximated by a local plane [62]. Therefore, he suggested a novel filter that samples the event-cloud along different spatio-temporal orientations, which relies upon the superposition of space-time separable filters with out-of-phase temporal modulation filter-responses [62]. 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 [62, 63, 72, 73]. Moreover, these approaches have multiple issues regarding to real implementation. The most important observation is that when a luminance edge passes a pixel's receptive field of the DVS sensor, the amount of events is in the range of about 10 events (often even less), thus huge approximation errors occur. Another issue is that in many real-time applications temporal windows are small enough such that the motion edge has not already passed through the receptive field, which largely limits the number of events to even less and leads to magnifying the outlined problems [62].

Later, Orchard *et al.* introduced HFirst descriptors that used spike timing to encode the strength of events and implemented a max operation to output a number representing the strength of input [33]. Specifically, HFirst is structured in a similar manner to hierarchical neural models which consist of four layers named Simple 1, Complex 1, Simple 2 and Complex 2. In these frame base architectures, cells in simple layers densely cover the scene and respond linearly to their inputs, while

cells in complex layers have a non-linear response and only sparsely cover the scene. The Simple layers in HFirst are in fact non-linear due to the use of a spike threshold and binary spike output.

Lagorce *et al.* proposed event based spatio-temporal features called time-surfaces [34]. This is a time oriented approach to extract spatio-temporal features that are dependent on the direction and speed of motion of the objects. Output events are firstly built as a time-surface that describes the recent time history of events in the spatial neighborhood of an event, since the structure of these events contain information about the object and movement. These time-surfaces are then clustered into prototypes represented as surfaces. When an input event arrives at a bank of time-surface prototypes, the time-surface associated to the incoming event is calculated and compared to the time-surface of each prototype. The prototype with the time-surface most closely matching the surface of the input event will then generate an output event, resulting in the activation and constituting the output of next layer. In this way, such architecture consists in a hierarchy Of time-surfaces which is building and extracting a set of features (the prototypes from the final layer) out of a stream of input events. The time-surface prototypes be used as time-surface features descriptors.

Inspired by time-surfaces, Sironi 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 [35]. This method introduces a new event-based scalable machine learning architecture, relying on a low-level operator called Local Memory Time Surface. A time surface is a spatiotemporal representation of activities around an event relying on the arrival time of events from neighboring pixels. However, the direct use of this information is sensitive to noise and non-idealities of the sensors. By contrast, This method emphasizes the importance of using the information carried by past events to obtain a robust representation. Moreover, This method shows how to efficiently store and access this past information by defining a new architecture based on local memory units, where neighboring pixels share the same memory block. In this way, the Local Memory

Time Surfaces can be efficiently combined into a higher-order representation, which is called Histograms of Averaged Time Surfaces. This results in an event-based architecture which is significantly faster and more accurate than existing time surface feature descriptors [33, 34]. Driven by brain-like asynchronous event based computations, this new architecture offers the perspective of a new class of machine learning algorithms that focus the computational effort only on active parts of the network.

Recently, Ramesh *et al.* introduced a generic visual descriptor termed as DART that encodes the structural context using log-polar grids for events [36]. A log-polar grid that simulates the distribution of cones in the primate fovea is centered at the latest event and the past events, whose space-time coordinates are marked as a 'star', are binned into nearest spatial locations of the grid. Subsequently, the DART descriptor is formed using the overall interpolated event count within each bin of the logpolar grid. As the neuromorphic camera responds to changes in log-intensity, a brighter contrast or a faster motion results in an increased event rate. Thus, normalization of the DART descriptor is critical to capture the relative distribution of the surrounding events, and to account for camera and object motion, the descriptor is updated on an event-by-event basis using a queue to capture precise space-time information. This method presents a significant step forward in computing a structural descriptor for event-based data, and can be applied to four different problems, namely object classification, tracking, detection and feature matching.

These descriptors are much sensitive to noise and strongly depend on the type of object motion in scene. Moreover, these descriptors have only proven to be useful for static object recognition, and they fail to take temporal information into account and maintain a representation of dynamics over a long time so that they may not be useful for long-term application such as action recognition evaluated in this thesis. However,our proposed work is an end-to-end learning system that is robust to the noise and motion, also it can cover a long temporal extent so that the extracted feature can be used for long-term application such as action recogntion and activity similarity labeling.

2.1.2 Frame-based feature leaning for NVS streams

Two end-to-end feature learning methods are proposed. The first type is frame-based methods: the main idea is to construct events to grid frame form, then apply convolution neural networks for the feature learning. i.e., converting the neuromorphic events into synchronous frames of spike events, on which conventional computer vision techniques can be applied.

Zhu *et al.* [38] introduced a four-channel image form with the same resolution as the neuromorphic vision sensor: the first two channels encode the number of positive and negative events that have occurred at each pixel, while last two channels as the timestamp of the most recent positive and negative event. This proposed novel image-based representation of an event stream fits into any standard image-based neural network architecture. The event stream is summarized by an image with channels representing the number of events and the latest timestamp at each polarity at each pixel. This compact representation preserves the spatial relationships between events, while maintaining the most recent temporal information at each pixel and providing a fixed number of channels for any event stream. Moreover, Authors presented a self-supervised learning method for feature estimation given only a set of events and the corresponding gray scale images generated from the same camera. As a result, the network can be trained using only data captured directly from an event camera.

Inspired by the functioning of Spiking Neural Networks (SNNs) to maintain memory of past events, leaky frame integration has been used in recent work [4, 39, 40], where the corresponding position of the frame is incremented of a fixed amount when a event occurs at that event address. These methods are to make use of frame reconstruction procedures and conventional frame-based neural networks that can instead rely on optimized training procedures. Cannici integrated events into a volatile frame, a spatial structure to maintain events information through time [39]. Specifically, When an event is received, the corresponding pixel of the integrated frame is incremented of a fixed amount. Meanwhile the whole frame is decremented of a quantity that depends on the time elapsed between the last received event and

the previous one. Similar frame construction procedure is proposed in [74], which divides the time in constant and predefined intervals. Frames are obtained by setting each pixel to a binary value (depending on the polarity) if at least an event has been received within the reconstruction interval. With this mechanism, however, time resolution is lost and the same importance is given to each event, even if it is noise. Instead method proposed by Cannici in [39] does not distinguish the polarity of the events, obtaining frames invariant to the object movement, and performs continuous and incremental integration, characteristics that allowed to develop the event based framework. Later, Cannici focused on enhancing conventional architectures by designing attention mechanisms that can be used to make these networks focus only on relevant instants of events recordings and only on the salient portions of frames [40]. In this work, Cannici developed an algorithm that detects peaks of events activity and uses them to extract patches from reconstructed frames. This approach takes inspiration from the spiking neural network where a peak detection mechanism is used to decide when to output predictions. Instead of leaky integrate-and-fire neurons, however, this method makes use of region-wise events statistics to identify and localize peaks, which can efficiently identify regions of interest from events while improving the translation invariance properties.

Peng *et al.* [75] proposed bag-of-events (BOE) feature descriptors, which is a statistical learning method that firstly divides the event streams into multiple segments and then relies on joint probability distribution of the consecutive events to represent feature maps. Inspired by information theory and document analysis, this proposed method uses the joint probability distribution of the consecutive events to represent each stimulus. Therefore, BOE does not extract any visual features such as lines or shapes as many existing methods did, and BOE is a probability-based feature extraction method that has the advantage of good interpretability in mathematics. Moreover, BOE is an online learning algorithm, which does not require the whole training data set to be provided in advance. In other words, when the labeled (i.e., training data) and unlabeled events (i.e., testing data) are alternately received, BOE can smoothly handle the data and will not repeatedly train the feature extrac-

tion module. The problem is that BOE requires accumulating events into segments which does not offer a good solution for high-speed application where tasks are normally realized in real time.

Amir *et al.* used temporal filter to process the events, regarded the filter cascades as stacking frames and input these frames to Convolution Neural Networks [32]. Specifically, to capture the sequence information required to following tasks, a cascade of K delayed temporal filters collects a sequence of events. The first filter outputs a stream of events delayed by one tick, and creates a second copy of its input which is passed to the next filter in the cascade. Finally, the output event streams of all K filters are concatenated to form the input features for the first convolution layer. The neurons in these filters are configured to generate events stochastically, using a constant leak to decay the membrane potential linearly with time. Using a stochastic decay makes the rate of filter output events proportional to the time since the corresponding event was received at the filter input. These temporal filter cascades may be compared to stacking frames to create a spatio-temporal input to CNN.

Similary, Ghosh *et al.* partitioned events into a three dimensional grid of voxels where spatio-temporal filters are used to learn the feature, and learnt feature are as input to feed to convolution neural networks for action recognition [76]. To make CNNs work with event-based data, one first needs to structurize the spike-events, which arrives asynchronously as an ever growing set of three dimensional points. To convert the spike-event point cloud data obtained from a neuromorphic camera into a structured matrix form, authors proposed to structurizing the spike event data into a 3D grid of voxels, which form a 3D spatio-temporal matrix, where each voxel contains the number of spike-events within it. In this case, 3D convolution based filters can be seamlessly integrated to work with such spatio-temporal data. In this method, a convolutional kernel normally spans a long temporal constant, which leads to loss of temporal precision of each spike-event. To reduce this information loss, a possible workaround is to increase the number of time partitions by decreasing the voxel size along the temporal dimension.

Chadha *et al.* [77] generated frames by summing the polarity of events in each address as pixel, then fed them into a multi-modal teacher-student framework, where it employs a pre-trained optical flow stream as a teacher network to transfer knowledge to the NVS student network. By representing the events as frames, authors can leverage the conventional CNNs to realize the tasks. Specifically, this framework takes the advantage of optical flow, which firstly initializes the optical flow I3D with the Kinetics trained weights and then fine-tuning on the target action recognition datasets and then initialize the student NVS CNN with the teacher weights. We will discuss the drawbacks of this work in the last of this section.

Rebecq *et al.* established the bridge between vision with event cameras and conventional computer vision [78]. Specifically, they reconstructed natural videos from a stream of events (i.e. learn a mapping between a stream of events and a stream of images), which allows to apply off-the-shelf computer vision techniques to event cameras. Instead of embedding handcrafted smoothness priors into reconstruction framework, they directly learn video reconstruction from events using a large amount of simulated event data, but this method, to some extent, requires complex computation to form the conventional videos, which loses the advantage of using event cameras.

Recently, Gehrig [37] proposed a general framework that converts asynchronous event-based data into grid-based representations. Instead of assuming the input event representation as fixed, this conversion process is fully differentiable, allowing to learn a representation end-to-end from raw event data to the task loss. To achieve this, authors expressed the conversion process through kernel convolutions, quantizations, and projections, where each operation is differentiable. Specifically, to derive a meaningful signal from the event measurement field, this method firstly convolve events with a suitable aggregation kernel; after kernel convolutions, a grid representation of events can be realized by sampling the convolved signal at regular intervals. This framework has following advantages. First, it makes the conversion process fully differentiable, allowing to learn a representation end-to-end from raw event data to the task loss. In contrast, prior work assumes the input event repre-

sentation as fixed. Second, it lays out a taxonomy that unifies the majority of extant event representations in the literature and identifies novel ones.

While useful for early-stage attempts, all of these methods are frame-based methods that are not suitable 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 in order to use conventional downstream processing. Essentially, event is basic processing unit for neuromorphic vision sensing so the data amount needed to be processed is small, while frame-based methods take one frame as basic processing unit by looping over entire frame pixel, thus not providing an efficient and power-saving learning systems. Therefore, our proposed graph-based feature learning can efficiently keep the sparsity of NVS streams and provide efficient systems.

2.1.3 Event-based feature leaning for NVS streams

The second type of end-to-end feature learning method is event-based methods. The most commonly used architecture is based on spiking neural networks (SNNs) [41–43, 79, 80]. Lee introduce a novel supervised learning technique, which can train general forms of deep SNNs directly from spike signals [42]. This includes SNNs with leaky membrane potential and spiking winner-takes-all (WTA) circuits. The key idea of our approach is to generate a continuous and differentiable signal on which SGD can work, using low-pass filtered spiking signals added onto the membrane potential and treating abrupt changes of the membrane potential as noise during error back-propagation. Besides, authors addressed particular challenges of SNN training: spiking neurons typically require large thresholds to achieve stability and reasonable firing rates, but this may result in many "dead" neurons, which do not participate in the optimization during training. Therefore, novel regularization and normalization techniques are presented, which contribute to stable and balanced learning. This techniques lay the foundations for closing the performance gap between SNNs and ANNs, and promote their use for practical applications.

Neftci proposed a method to construct Restricted Boltzmann Machines

(RBMs) using Integrate & Fire (I&F) neuron models and to train them using an online, event-driven adaptation of the Contrastive Divergence (CD) algorithm. Authors took inspiration from computational neuroscience to identify an efficient neural mechanism for sampling from the underlying probability distribution of the RBM and identify the conditions under which a dynamical system consisting of I&F neurons performs neural sampling, suggesting that they can achieve similar performance. To train the networks easily, authors traind the neural RBMs using an online adaptation of CD. Specifically, they exploit the recurrent structure of the network to mimic the discrete "construction" and "reconstruction" steps of CD in a spike-driven fashion, and Spike Time Dependent Plasticity (STDP) to carry out the weight updates. Compared to standard CD, no additional connectivity programming overhead is required during the training steps, and both testing and training take place in the same dynamical system.

Bichler proposed a novel approach that fully embraces the asynchronous and spiking nature of these sensors and is able to extract complex and overlapping temporally correlated features in a robust and completely unsupervised way [80]. Authors presented a new way of using Spike-Timing-Dependent Plasticity (STDP) to process real life dynamic spike-based stimuli recorded from a physical AER sensor. Motion pattern can be learned from complex moving sequences with a feed-forward multilayer unsupervised learning spiking neural network. To do this, authors proposed a new network topology with spatially localized neurons, providing similar performances with only a tenth of the synapses required compared to a fully-connected network, Besides, by using the same network topology, receptive fields quickly emerge from "walking through the environment" recorded sequences even though no pattern is continuously repeating at a global scale.

While SNNs are theoretically capable of learning complex representations, they have not achieved the performance of gradient-based methods because of 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 fol-

low an intermediate step [44, 74, 81, 82]: a neural network is trained off-line using continuous/rate-based neuronal models with state-of-the-art supervised training algorithms and then the trained architecture is mapped to an SNN. Perez focused on vision systems comprising an event-driven sensor and a large number of event-driven processing modules [74]. However, training of event-driven processing modules is still an open research problem and its its application to large-scale systems is presently not practical. Therefore, authors presented an intermediate solution. First, they built a database of training images (frames) by collecting events from a DVS camera during fixed time intervals. Second, they trained a frame-driven ConvNet with this database to perform object recognition. Third, they mapped the learned parameters of the frame-driven ConvNet to an event-driven ConvNet, and finally, they fine-tuned some extra available timing-related parameters of the event-driven ConvNet to optimize recognition. Authors illustrated this with two example ConvNet exercises: one for detecting the angle of rotated DVS recordings of walking human silhouettes, and the other for recognizing the symbols of poker cards when browsing the card deck in about 1 second in front of a DVS, showing the effectiveness of the proposed training method.

Similar works are done in [81,82]. In [81], authors proposed a method based on the Siegert approximation for Integrate-and-Fire neurons to map an offline-trained Deep Belief Networks (DBNs) onto an efficient event-driven spiking neural network suitable for hardware implementation. The method is demonstrated in simulation and by a real-time implementation of a 3-layer network with 2694 neurons used for visual classification of MNIST handwritten digits. In [82], to reduce the performance losses due to the conversion from analog neural networks (ANN) without a notion of time to sparsely firing and event-driven SNNs, authors analyzed the effects of converting deep ANNs into SNNs with respect to the choice of parameters for spiking neurons such as firing rates and thresholds. They presentd a set of optimization techniques to minimize performance loss in the conversion process for ConvNets and fully connected deep networks, which yields networks that outperform all previous SNNs on the MNIST database. However, until now, despite

their substantial implementation advantages at inference, the obtained solutions are complex to train and have typically achieved lower performance than gradient-based CNNs.

There are some other initial attempts on event-based feature learning for neuromorphic vision sensing. Besides using SNNs, Wang interpreted an event sequence as a 3D point clouds in space and time, event cloud is hierarchically fed in PointNet [83] to capture spatio-temproal structure of motion, then the learned feature is used for recognition [2]. Specifically, authors proposed a novel representation to interpret an event sequence as a 3D point clouds in space and time. In their proposed method, each event becomes a point in a three dimension continuum. Each gesture generates a distinctive cloud of events coordinate system and they call it space-time event clouds. By interpreting event streams as space-time event clouds, spatial features and temporal features are fused in a 3D space-time continuum. To robustly differentiate point clouds and recognize corresponding gestures, authors proposed to adapt PointNet to analyze event-camera data, i.e., event clouds. The event cloud is hierarchically analyzed using a PointNet-based architecture to capture the essential spatio-temproal structure of the hand motion, then the learned feature is used for classification.

Gao designed a broad learning network to deal with the event-based data for the object classification [84]. They firstly used an asynchronous peak-and-fire mapping to depict the event-based data. Then a basic broad learning system (BLS) is established in the form of a flat network, where the event-based inputs are transferred as 'feature nodes' and the structure is expanded as 'enhancement nodes'. The output layer is directly connected with the feature nodes and enhancement nodes by the weights. The broad network provides an alternative way of learning in a go-broad way, which is different from the deep CNNs models. With the event-based input data continuously coming, the network becomes broad by adding feature nodes, enhancement nodes, and additional enhancement nodes. A key advantage is that proposed incremental BLS can be remodeled by the increment of nodes without the entire retraining, which facilitates the BLS network extending from a basic network

to a large one. Another advantage is that bridges the event-based data and the broad learning, which successfully integrates the asynchronous data into a flexible broad network.

While providing useful insights, all these event-based methods were test on simple datasets with a small number of class and clean background: e.g., DVS128 Gesture Datasets used in [2, 32] comprises a set of 11 hand and arm gestures and the posture dataset in [85, 86] includes only three human actions, i.e., bend, sitsand, and walk. Moreover, all current datasets all were acquired from a relatively noise-free experimental environment that cannot represent complex real-life scenarios, which led to many algorithms achieving very high accuracy. It is therefore unlikely that these methods can obtain such high accuracy for real-world scenarios, as they cannot capture long-term temporal dependencies. Indeed when these are applied on more complex dataset (such as UCF101-DVS) for human action recognition, their performance degrades significantly as discussed in application experiments. Moreover, our graph-based learning methods can use the well-establish gradient-based learning rules that can make the training process easy and also provide a confident performance.

2.2 Graph-based Deep Learning

As our feature learning frameworks are based on graph input, we simply review the graph-based Deep Learning. Specifically, we will review the development of graph-based deep learning, types of graph neural networks and application of graph deep learning in computer vision community.

With the advance of deep neural networks (e.g. CNNs and RNNs) and computational resource (e.g. GPUs), many machine learning tasks have already changed from heavily relying on handcrafted feature engineering to extract informative feature set to various end-to-end deep learning paradigms, and also achieved promising performance in many applications. One key reason of such successes is that deep neural netowrks are able to take advantages of the hierarchical patterns and extract high-level features to achieve a great expressive capability because of grid-like na-

ture of data (known as Euclidean form). However, there is an increasing number of real-world applications, such as chemistry molecules, traffic networks and citation networks, where data are represented in Non-Euclidean form, thus emerges the graph-based deep learning methods. In this section, we will briefly describe the development of graph-based neural networks, taxonomy of graph neural networks and some applications in computer vision field.

The principle of constructing CNNs on graph generally follows two streams: the spectral perspective [87–93] and the spatial perspective [3, 94–98]. Spectral convolution applies spectral filters on the spectral components of signals on vertices transformed by a graph Fourier transform, followed by spectral convolution. Defferrard [89] provided efficient filtering algorithms by approximating spectral filters with Chebyshev polynomials that only aggregate local K-neighborhoods. This approach was further simplified by Kipf [87], who considered only the one-neighborhood for single-filter operation. Levie [88] proposed a filter based on the Caley transform as an alternative for the Chebyshev approximation. As to spatial convolution, convolution filters are applied directly on the graph nodes and their neighbors. Several research groups have independently dealt with this problem. Duvenaud [94] proposed to share the same weights among all edges by summing the signal over neighboring vertices followed by a weight matrix multiplication, while Atwood [95] proposed to share weights based on the number of hops between two vertices. Finally, recent works [3, 98] make use of the pseudo-coordinates of nodes as input to determine how the features are aggregated during locally aggregating feature values in a local patch. Spectral convolution operations require an identical graph as input, as well as complex numerical computations because they handle the whole graph simultaneously. Therefore, to remain computationally efficient, our work follows the spirit of spatial graph convolution approaches and extends them to NVS data for feature learning.

Graph neural networks (GNNs) can be categorized into following four types: recurrent graph neural networks, graph convolutional neural networks, graph autoencoders, and spatial-temporal graph neural networks. Recurrent Graph Neural

Networks (RecGNNs) are applied into the cases where the node in a graph constantly exchanges information/message with its neighbors until reaching a stable state. The aim of RecGNNs is to learn node representations with recurrent neural architectures, which is lately inherited by spatial-based graph convolutional neural networks. Similar to the CNNs, graph CNNs generate a node's representation by aggregating its own feature and neighbors' feature. Different from recurrent architectures, graph CNNs stack multiple graph convolutional layers to extract high-level node representations. Currently, graph CNNs play an significant role in graph deep learning field, which are largely used in the application related to node classification and graph classification. Graph Autoencoders encode nodes/graphs into a vector representation and reconstruct graph data from the encoded information in an unsupervised manner, which are widely used to learn network embedding and graph generative distributions. Spatial-temporal Graph Neural Networks (STGNNs) aim to learn representation from spatial-temporal graphs by considering spatial dependency and temporal dependency at the same time, which become increasingly important in a variety of applications such as traffic speed forecasting [99, 100], driver maneuver anticipation [101], and human action recognition [102]. Many current approaches integrate graph convolutions to capture spatial dependency with RNNs or CNNs to model the temporal dependency.

Graph-based deep learning technologies are becoming popular in the process and analysis of image, video and point cloud. In terms of image field, one application is image classification. Images are firstly converted to the structured graph data by carefully hand-crafted graph construction methods (e.g., KNN similarity graphs), then graph convolutional networks is leveraged as a classifier [89, 91, 103]. Another application on images is visual question answering that explores the answers to the questions on images [104]. Visual reasoning on images also is a hot topic. Since images contain multiple objects, understanding the relationships among the objects helps to characterize the interactions among them [105–107]. As to application on video, researchers are using graph convolutional neural networks to realise action recognition. video contents are firstly represented as graphs, e.g.

spatial-temporal graphs in [102], skeletons-based graphs in [108] and space-time region graphs in [109], then designed well-suitable graph convolutional networks to recognize action in videos. As for point clouds, the state-of-the-art deep neural networks only consider the local features of point clouds and ignore the geometric relationships among points; therefore researchers took advantage of graph neural network to solve this problem. For instance, EdgeConv [110] is able to capture the local geometric structure and maintain the permutation invariance, thus achieving progressive performance in point cloud segmentation. Recently, a regularized graph convolutional network where graph Laplacian is dynamically updated to capture the connectivity of the learned features is proposed for segmentation on point clouds [111]. Wang *et al.* proposed a local spectral graph convolutional network for both point cloud classification and segmentation [90], and Valsesia *et al.* proposed a localized generative model by using graph convolution to generate 3D point clouds [112].

Chapter 3

PIX2NVS: Parameterized Conversion of Pixel-domain Video Frames to Neuromorphic Vision Streams

3.1 Introduction

One of the major obstacles in developing neuromorphic-based advanced machine learning algorithms for recognition, classification and retrieval is the lack of widely-available event-based neuromorphic vision streams with reliable annotations to train and test with. Recent work has attempted to resolve this issue by recording limited-scale annotated datasets in controlled conditions [33,74,113,114], i.e., video frames displayed in a monitor under controlled frame-rate and brightness/contrast conditions and are recorded with a DVS camera. While such experimental approaches provided for the first available annotated video datasets in neuromorphic vision sensing (NVS) format, their three issues are that: *(i)* the recording is affected by environmental and monitor conditions (e.g., lighting, monitor flicker, vibrations, etc.); *(ii)* high-accuracy synchronization between the played-out video frames and the corresponding NVS may be difficult to resolve because of drift between the timing of the playout device and the DVS camera; *(iii)* due to their hardware na-

ture, such measurement-based approaches cannot scale to large datasets containing millions of videos, such as the recently-released Youtube-8M datasets [115]. To this end, recent work [8, 116, 117] proposed models to generate NVS events using pixel-wise linear interpolation of the pixel intensity given by successively rendered images. However, these approaches have one or more of the following detriments: use of custom bias settings, requirement to have pixel-domain frames captured by a co-existing active pixel sensor (APS) camera, such as the bundled APS of the DAVIS240C device, and lack of distortion metrics to quantify the accuracy of the generated NVS events.

In this chapter, we propose and make available online[1] the pixel-to-NVS (PIX2NVS) framework in Section 3.2, which is a software that can be used to generate neuromorphic vision streams from any pixel-domain video format. We also propose and verify three new metrics in Section 3.3 (Chamfer distance, ε-repeatability and Earth Mover's Distance (EMD)) to quantify the accuracy of the model-generated NVS event streams against ground-truth event streams available from experimental setups, such as '.aedat' files from DAVIS camera deployments. Moreover, the parameter optimization of PIX2NVS and efficacy of emulated NVS datasets is put to the test in Section 3.5. Via the use of a 3D convolutional neural network (CNN), we elaborate on the training and testing aspects for multi-class human action recognition and sign language recognition applications. Our results show that training and testing with emulated spike events leads to marginal loss of accuracy for applications. Importantly, the obtained accuracy gap is within 4.5%.

3.2 PIX2NVS Framework

NVS devices like the iniLabs DAVIS [20] and Pixium Vision ATIS sensors [12, 21] output asynchronous spike events indicating temporal intensity contrast changes. Spike events are recorded in pixel coordinates, timestamped with microsecond resolution and labeled as ON or OFF [20] (denoted by +1 and -1). They are produced in a format compliant with the address event representation protocol (AER) [118].

[1] http://www.github.com/pix2nvs

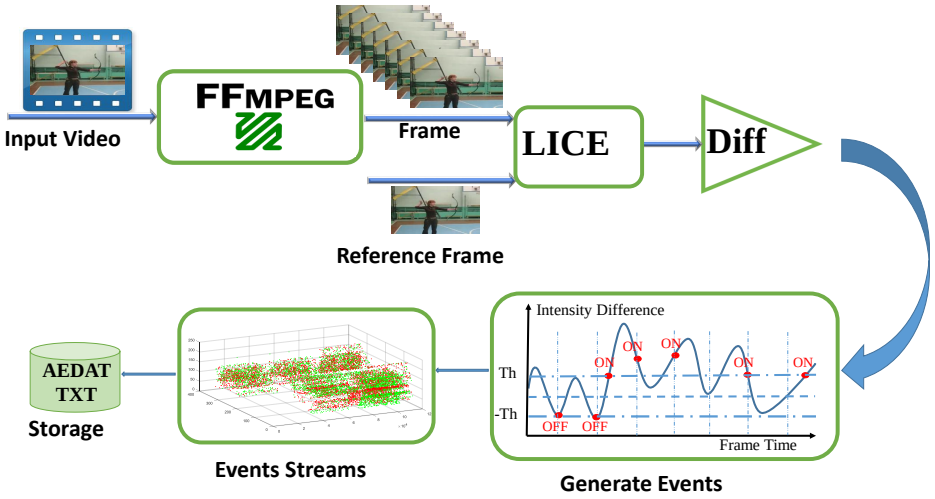


Figure 3.1: Illustration of PIX2NVS framework: videos are firstly extracted as frames by FFmpeg library, then proposed by LICE module detailed in Section 3.2.1 and Diff module detailed in Section 3.2.2. Generate Events module also is illustrated in Section 3.2.2 and reference update is in Section 3.2.3.

Our emulation framework produces such spike event streams by processing any standard compressed format container (e.g., MP4, MKV, etc.) based on the FFMPEG library[2] [121]. The process begins by the decoding of the APS video frames. These frames are then processed sequentially in order to produce a stream of spike events, which are stored in text or AEDAT format (AER data file). The aim of our framework is to produce NVS data that is as similar as possible to an equivalent scene would have been captured with an NVS-generating device. Similarity is assessed based on the metrics of the next section. The core operation of PIX2NVS is shown in Algorithm 1 and also illustrated in Fig. 3.1. PIX2NVS consists of frame extraction module by using FFMPEG, intensity change module, comparison and events generation module and stream storage. And the details of the operation are described in the following parts of this section. For clarity, we also summarize the main emulation parameters in Table 3.1.

[2]MPEG coding frameworks and FFMPEG are chosen because of their wide availability and support. However, the proposed framework can also support non-MPEG codecs [119, 120] if a decoder is made available.

Table 3.1: Table of emulation parameters.

Symbol	Definition
`LICE_mode` $\in$ {`LI`, `CE`, `LICE`}	method for mapping luminance, `LI` refers to log intensity, `CE` refers to contrast enhanced intensity
$T_{\log}$	threshold to control switch between linear and log mappings of luminance for `LICE_mode=LI`
`dif` $\in$ {`0`, `avg`, `min`}	method for computing the difference value between LICE values at position (x,y)
B	size of max-pooling window for local inhibition
$M_{\max}$	maximum number of spike events to generate per position
T_{map}	threshold for computing the number of spike events to generate per position
`new` $\in$ {`TRUE`, `FALSE`}	method for generating the LICE values for reference frame F_n
`tstamp`	method for generating spike timestamps: {`RAND`, `LINEAR`, `FIXED`}
`exp` $\in$ {`TRUE`, `FALSE`}	whether to use exponential moving average for reference frame update
α	moving average update rate, $0 < \alpha < 1$

3.2.1 Converting Pixels to Intensity

For every spatial position (x,y) of each frame F_n, the RGB pixel values $(r_{x,y}, g_{x,y}, b_{x,y})$, typically ranging between 0 to 255, are first converted into luminance values via $q_{x,y} = 0.299r_{x,y} + 0.587g_{x,y} + 0.114b_{x,y}$ or, if `hue = TRUE`, hue values via $h_{x,y} = b_{x,y}/(r_{x,y} + g_{x,y})$. Without loss of generality, for the remainder of this work we shall be focusing on luminance values. Given that the NVS devices detect changes in the logarithm of the captured intensity of each spatial position,

Algorithm 1 Conversion of video frame pixels to NVS.

1: **Input:** Pixel-domain video frames $F_0, F_1, ..., F_N$ extracted from a video format container using FFMPEG, parameters: `hue`, `LICE_mode`, $T_{\log}, \gamma, T_{\text{map}}$, `dif`, `new`, `exp`, `tstamp`, `fps`
2: **Output:** Spike event tuples $E_e = \langle x_e, y_e, t_e, P_e \rangle$ stored in a text file and/or in an AEDAT stream
3: **Operation:** Read F_0, convert pixels to LICE values, produce spike event tuples and optionally update LICE values
4: **for** $n = 1:1:N$ **do**
5: Read F_n and convert the RGB pixel values to LICE values using Section 3.2.1
6: Find differences of LICE values of successive frames using (3.3), (3.4) and (3.5)
7: If the difference is equal or exceeds threshold T_{map}, then output spike events with coordinates, polarity and timestamp using (3.6) and (3.7), (3.8)
8: Optionally update LICE values using (3.9) and (3.10)
9: **end for**

the first mode of PIX2NVS is to convert these values into *log-intensity* values via:

$$l_{x,y} = \begin{cases} q_{x,y}, & q_{x,y} \leq T_{\log} \\ \ln(q_{x,y}), & q_{x,y} > T_{\log} \end{cases} \tag{3.1}$$

with $T_{\log}$ the threshold used to control the switch between the linear and the log mapping. For log intensity, $T_{\log} = 0$, while for lin-log intensity, the threshold is set to a value close to 10% of the maximum value, e.g., $T_{\log} = 20$.

The limited dynamic range in the APS hardware may mask local scene luminance changes and lead to a different response than log-intensity. Therefore, we provide a second mode for PIX2NVS via *contrast-enhanced* intensity values [122]. Here, we first define the perceptual luminance of each pixel as $l'_{x,y} = 100 \times \sqrt{(q_{x,y}/255)^{\gamma}}$, with $\gamma = 2.2$, and then calculate the contrast-enhanced intensity at coordinate (x, y) by

$$l_{x,y} = \frac{\sum_{p=0}^{1} |l'_{x,y} - l'_{x+2p-1,y}| + \sum_{p=0}^{1} |l'_{x,y} - l'_{x,y+2p-1}|}{4} \tag{3.2}$$

The choice between log-intensity and contrast enhancement (LICE) is controlled by setting parameter `LICE_mode` $\in \{$`LI`, `CE`, `LICE`$\}$, with the third mode applying

the `CE` and `LI` modes sequentially.

3.2.2 Spike Event Generation

In our proposed framework, we provide emulated events by differencing the video frame $l_{x,y}[n]$ with a reference frame $\bar{l}_{x,y}[n-1]$. To this end, we propose three approaches: *(i)* co-located differencing (which is enabled by setting parameter `dif = 0`):

$$d_{x,y} = l_{x,y}[n] - \bar{l}_{x,y}[n-1] \tag{3.3}$$

(ii) two variants of differencing that utilize the average or the minimum value of the weighted-neighborhood of reference frame values (`dif = avg`, `dif = min`):

$$\begin{cases} d_{x,y} = l_{x,y}[n] - \frac{\Sigma_{p=0}^{1} \bar{l}_{x+2p-1,y}[n-1] + \Sigma_{p=0}^{1} \bar{l}_{x,y+2p-1}[n-1]}{4} \\ d_{x,y} = l_{x,y}[n] - \min_{p \in 0,1}(\bar{l}_{x+2p-1,y+2p-1}[n-1]) \end{cases} \tag{3.4}$$

As correlation in small spatial neighborhoods of natural images is known to be high [123], we assume that neighboring pixels transmit redundant information. We therefore emulate local inhibition in our framework by applying a local maximum on non-overlapping patches of the differences $d_{x,y}$, with dimensions $B \times B$ (B is a parameter):

$$(x^*, y^*) = \arg\max(d_{(x+s_i),(y+s_j)}) \quad \forall s_x, s_y \in \{0, 1, ..., B\} \tag{3.5}$$

Hence, when local inhibition is enabled, within each patch, we only keep $d_{x,y}$ for the (x^*, y^*) positions corresponding to the locally maximum difference values using (3.3) or (3.4).

The eth spike corresponding to frame F_n (out of $e_{\text{tot}}[n]$ spikes detected in that frame) is generated if and only if $|d_{x,y}| \geq T_{\text{map}}$; in such a case, the polarity of the spike is:

$$P_e = \text{sgn}(d_{x,y}) \tag{3.6}$$

and the coordinates of the spike are $(x_e, y_e) = (x, y)$. Concerning the timestamp of

the spike, we can generate it as: *(i)* a random number between the timestamps of frames F_{n-1} and F_n; *(ii)* a linearly-scaled value between the timestamps of frames F_{n-1} and F_n; *(iii)* fixed to the timestamp of frame F_n. Parameter `tstamp` controls this:

$$t_e = \frac{1}{\text{fps}} \times \begin{cases} U([n-1,n]), & \texttt{tstamp} = \texttt{RAND} \\ n-1+\frac{e}{e_{\text{tot}}[n]}, & \texttt{tstamp} = \texttt{LINEAR} \\ n, & \texttt{tstamp} = \texttt{FIXED} \end{cases} \tag{3.7}$$

where fps stands for the frame-rate of the video and $U([a,b])$ returns a uniformly-distributed number within $[a,b]$.

Following the steps defined in (3.6) and (3.7), up to one spike is generated at each spatial position of each new frame. However, in this way, the number of generated spikes per position are limited by the frame rate; this does not encapsulate cases where the pixel intensity difference $d_{x,y}$ is high [117]. We address this by allowing more than one spike to be generated between frames following an approach similar to Furber's work [117]. First, we assign $M_{\max}$ as the maximum number of spikes per position. We then compute the number of spikes to generate per position (x,y), $M_{x,y}$, as:

$$M_{x,y} = min(M_{\max}, \left\lfloor \frac{d_{x,y}}{T_{\text{map}}} \right\rfloor) \tag{3.8}$$

and these additional spikes are timestamped using one of the methods of (3.7).

3.2.3 Reference Frame Update

The default option is to update the reference frame $\bar{l}_{x,y}[n]$ from the LICE values generated by (3.1) for all positions (x,y) in F_n, i.e.,

$$\bar{l}_{x,y}[n] = l_{x,y}[n] \tag{3.9}$$

However, the above reference update method only refers to the current frame and does not consider the transient response of neuromorphic sensors. Similar to Furber *et al.* [117], the reference frame update can alternatively be modeled by considering an exponential moving average over past frames. This provides a po-

tentially more accurate representation by accounting for the capacitive memory of neuromorphic sensors. We define the new reference update at frame F_n as:

$$\bar{l}_{x,y}[n] = \begin{cases} l_{x,y}[0], & \text{if } n = 0 \\ \alpha l_{x,y}[n] + (1-\alpha)\bar{l}_{x,y}[n-1], & \text{otherwise} \end{cases} \tag{3.10}$$

where $0 < \alpha < 1$ is the update rate, which can be tuned to match the capacitive properties of a neuromorphic sensor. The moving average update is enabled by setting parameter `exp = True`.

Current NVS cameras only update the log-scaled values of recently-detected positions. If we follow this approach then we only update the reference frame with 3.9 or 3.10 for positions (x,y) where a spike is detected, i.e., $\forall(x,y) \in \{(x_1,y_1),\ldots,(x_{e_{\text{tot}}},y_{e_{\text{tot}}})\}$. In our framework, we can toggle between the default option of updating all spatial positions, versus the selective updating described above by setting the parameter `new` to `True` or `False` respectively.

3.3 Distance Metrics to evaluate PIX2NVS

To evaluate the performance of PIX2NVS against ground truth NVS data generated by hardware experiments, we propose the use of three distance metrics that quantify spatial correspondences between emulated and experimental spike events [124]. Firstly, for the n-th frame, we denote the set of I emulated spikes as $\mathscr{E}\{E_1^{\text{emu}}(x,y,P_\Sigma),\ldots,E_I^{\text{emu}}(x,y,P_\Sigma)\}$ and the set of J experimental (i.e., real) spikes as $\mathscr{R}\{E_1^{\text{exp}}(x,y,P_\Sigma),\ldots,E_J^{\text{exp}}(x,y,P_\Sigma)\}$.

3.3.1 Chamfer Distance

In the case of the Chamfer distance, for each emulated spike $E_i^{\text{emu}}(x,y,P_\Sigma)$ of the n-th frame, we first search for experimental spike $E_{j^*}^{\text{exp}}(x,y,P_\Sigma)$ (with E_j^{exp} taken as all the spikes allocated to the same frame as E_i^{emu}) with the minimum Euclidean distance, calculated based on their spatial coordinates and the weighted polarity λP_Σ, where λ is a positive constant ensuring that the polarity component has comparable amplitude to the mean Euclidean distance between $(x_i^{\text{emu}},y_i^{\text{emu}})$ and $(x_j^{\text{exp}},y_j^{\text{exp}})$. Therefore, $\forall i, 1 \le i \le I$:

$$j^* = \arg\min_j \left\| (x_i^{\text{emu}}, y_i^{\text{emu}}, \lambda P_{\Sigma,i}^{\text{emu}}) - (x_j^{\text{exp}}, y_j^{\text{exp}}, \lambda P_{\Sigma,j}^{\text{exp}}) \right\| \tag{3.11}$$

Then, the Chamfer distance of every emulated frame is computed as the mean Euclidean distance between all corresponding spike events $\{i, j^*\}$, $1 \leq i \leq I$:

$$C(n) = \frac{\sum_{i=1}^{I} \left\| (x_i^{\text{emu}}, y_i^{\text{emu}}, \lambda P_{\Sigma,i}^{\text{emu}}) - (x_{j^*}^{\text{exp}}, y_{j^*}^{\text{exp}}, \lambda P_{\Sigma,j^*}^{\text{exp}}) \right\|}{I} \tag{3.12}$$

The Chamfer distance for the entire video is found as the average of the distances over all frames in the video.

3.3.2 Epsilon Repeatability

The ε-repeatability metric is defined per frame as the percentage of emulated spikes of the same polarity type (positive or negative) that are found within ε distance of at least one experimental spike. For each emulated spike E_i^{emu}, we first find whether at least one spike E_j^{exp} (of the same polarity type) exists in the same frame with spatial coordinates that have Euclidean distance smaller or equal to ε. We create a subset of emulated spikes $\mathscr{E}^{\varepsilon}$, where:

$$\mathscr{E}^{\varepsilon} = \{E_i^{\text{emu},\varepsilon} \in \mathscr{E} \mid \left\| (x_i^{\text{emu}}, y_i^{\text{emu}}) - (x_j^{\text{exp}}, y_j^{\text{exp}}) \right\| \leq \varepsilon\} \tag{3.13}$$

Then the ε-repeatability rate for each frame is defined as the ratio of set cardinality between $\mathscr{E}^{\varepsilon}$ and $\mathscr{E}$:

$$r^{\varepsilon}(n) = \frac{|\mathscr{E}^{\varepsilon}|}{I} \tag{3.14}$$

The final ε-repeatability for a video sequence is the mean of the frame ε-repeatability rates $r^{\varepsilon}(n)$.

3.3.3 Earth Mover's Distance (EMD)

We introduce EMD as a final distance metric to quantify the accuracy of pixel-to-NVS conversion. EMD has been proposed as the means to quantify the dissimilarity between two signatures [125], which is defined as the minimum 'cost' that must be

paid to transform one signature into the other, where there is a 'ground distance' between the basic features that are aggregated into the signature. The EMD is an effective method of measuring the domain shift between the real and emulated spike datasets. Essentially, we want to find the flow $\mathbf{F} = [f(i,j)]$ between spikes in sets $\mathscr{E}$ and $\mathscr{R}$ (as defined previously) that minimizes the 'work optimization' problem stated below:

$$\begin{aligned}
\text{minimize} \quad & W(\mathscr{E},\mathscr{R},\mathbf{F}) = \sum_{i=1}^{I}\sum_{j=1}^{J} f(i,j)d(i,j) \\
\text{subject to} \quad & f(i,j) \geq 0, \quad 1 \leq i \leq I, 1 \leq j \leq J, \\
& \sum_{i=1}^{I} f(i,j) \leq \left|P_{\Sigma,j}^{\text{exp}}\right|, \quad 1 \leq j \leq J \\
& \sum_{j=1}^{J} f(i,j) \leq \left|P_{\Sigma,i}^{\text{emu}}\right|, \quad 1 \leq i \leq I \\
\text{and} \quad & \sum_{i=1}^{I}\sum_{j=1}^{J} f(i,j) = \min\left(\sum_{\forall i}\left|P_{\Sigma,i}^{\text{emu}}\right|, \sum_{\forall j}\left|P_{\Sigma,j}^{\text{exp}}\right|\right)
\end{aligned} \tag{3.15}$$

After initializing the flow uniformly, this optimization problem can be solved using linear programming [125]. In our study, we set the "ground distance", $d(i,j) = \left\|(x_i^{\text{emu}}, y_i^{\text{emu}}) - (x_j^{\text{exp}}, y_j^{\text{exp}})\right\|$. The EMD can thus be interpreted as the minimum work required to "transport" the polarity between emulated and real spike event sets $\mathscr{E}$ and $\mathscr{R}$ such that both sets are evenly distributed, normalized by the total optimum flow $\mathbf{F}_{\text{opt}}$, i.e.,

$$\text{EMD}(\mathscr{E},\mathscr{R}) = \frac{\sum_i \sum_j f_{\text{opt}}(i,j)d(i,j)}{\sum_i \sum_j f_{\text{opt}}(i,j)} \tag{3.16}$$

The size of the flow matrix $\mathbf{F}$ grows exponentially with the number of spike events and, as such, becomes non-trivial to compute. We are able to partially offset the complexity by dividing each frame into a grid of spike blocks and computing the EMD between spatially corresponding blocks of real and emulated spikes. Denoting the subset of emulated and real spikes in block k as $\mathscr{E}_k^{\text{S}} \in \mathscr{E}$ and $\mathscr{R}_k^{\text{S}} \in \mathscr{R}$ respectively, the distance D for frame n is now computed over $1 \leq k \leq K$:

$$D(n) = \sum_{k=1}^{K} \text{EMD}(\mathscr{E}_k^{\text{S}}, \mathscr{R}_k^{\text{S}}) \tag{3.17}$$

In this paper, we set $K = 16$. The final distance for a video sequence is computed by averaging $D(n)$ over all frames.

3.3.4 Discussion

Each distance metric of this section measures a notion of dissimilarity between the experimental and the emulated spike event sets. The Chamfer distance is an asymmetric measure that is maximally biased towards the subset of experimental spikes that are closest to the emulated ones. This bias is relaxed in the ε-repeatability metric, which, however, remains parametric to the choice of ε. In addition, repeatability is a zero-order metric – it only accounts for the number of 'valid' spikes – whereas Chamfer distance is a first order metric as it incorporates the average distance between spike events. Finally, the EMD can be seen as an unbiased distance metric that, within each spatial area, considers each spike event distribution as a whole. However, EMD is substantially more expensive to compute on large datasets and will also be more affected by the contribution of outliers in the $\mathscr{E}$ and $\mathscr{R}$ sets than the Chamfer distance and the ε-repeatability.

We can take advantage of the complementary aspects of the three distances in order to rank the accuracy of each parameter setup of PIX2NVS in a more robust manner. As we described before, Chamfer distance is a local metric which can measure fine-grained difference of two sets of events, while EMD is a global distance metric that globally quantify the difference of polarity distribution. Epsilon repeatability measure the patch difference. Specifically, when utilizing these distances for emulation parameter selection, we calculate a "Weighted Score" measure, where, for each experiment, we average the three distances after L2-normalizing their values.

3.4 Experimental Emulation and Metrics Validation

3.4.1 Emulation of PIX2NVS

We use an iniLabs DAVIS240C camera to record pixel-domain video frames and experimental NVS events simultaneously, the latter serving as ground truth. We then deploy our PIX2NVS model based on the captured video frames in order to generate artificial NVS events to compare against the ground truth. Beyond this, we also validate the accuracy of our model based on a recently-released dataset [8]. The parameters used for the reported experiments were: `hue=FALSE`, `LICE_mode=LI`, $T_{\log} = 20, T_{\text{map}} = 0.4$, `dif=0`, `new=TRUE`, `tstamp=FRAME`, and `fps` is set according to the frame rate of the utilized video content.

A qualitative comparison between the real and model-generated events is shown in Fig.3.2. It is evident that the model-generated NVS events are clustered around frame times (since we use `tstamp=FRAME`). In addition, real NVS events contain flicker noise due to the underlying electronics, while our model-generated NVS datasets do not include such noise since they are based on threshold differencing of LICE values. moreover, the amount of emulated events are less then real one, we largely ascribe this difference to the setting of threshold. If we lower the threshold, them more events are generated. Beyond these effects, the qualitative comparison shows that our model appears to be generating events that resemble the spatio-temporal structure of real NVS events from the DAVIS240C. The biggest limitation with generated data is that timestamp resolution is inherently restricted by the frame rate of videos, which is typically in the regime of frame rate per second. Therefore, we can see the cascades of events in the generated data, while the real events are more smooth in the temporal extent.

3.4.2 Effectiveness of Proposed Metrics

Because of the presence of such flicker noise in the experimentally derived NVS, we measure the proposed metrics in reference to the model-generated data. In order to evaluate the suitability of the proposed Chamfer distance, ε-repeatability and Earth Mover's Distance in the domain of NVS data, before measurement with each metric,

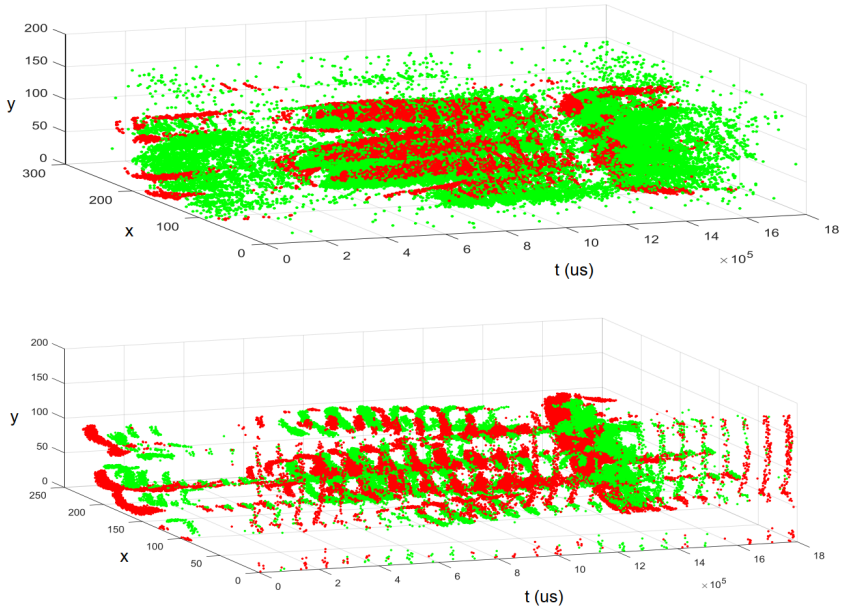


Figure 3.2: Experimental NVS events (top) and model-generated ones (bottom). Green/Red points: Trigger ON/OFF.

we impose artificial spatio-temporal distortions in the model-generated events by: *(i)* spatial downsampling (SD) of the events' coordinates; *(ii)* temporal downsampling (TD) by reduction of the `fps` value used for the grouping of NVS events into frames; *(iii)* pseudo-random injection of additive noise (AN) NVS events at 1% to 7% of the possible spatial coordinates within each model NVS frame. Experiments are conducted using the dataset of Mueggler *et al.* [8] and real DVS events and video frames captured with a DAVIS240C camera in our laboratory. For the cases of SD and TD, measurement is carried out by first upscaling the downscaled NVS events to the original spatio-temporal resolution before using the process described in Section 3.3. If the proposed metrics are appropriate for the utilized NVS data, we expect that, as we impose such SD/TD/AN distortions: *(i)* the Chamfer distance and Earth Mover's Distance will increase; *(ii)* the ε-repeatability will decrease. Indeed, the results, shown in Table 3.2, validate this expectation for all cases. Therefore, we conclude that these three metrics are appropriate for the quantification of the accuracy of model-generated NVS events.

Table 3.2: Average Chamfer distance, ε-repeatability ($\varepsilon = 2.5$) and Earth Mover's Distance w.r.t. spatial downsampling (SD), temporal downsampling (TD) and additive noise (AN) from samples in lab tests with DAVIS240C and the Mueggler *et al.* dataset [8].

Datasets		Lab Tests			Mueggler [8]		
Methods		Chamfer	ε-Rep.	EMD	Chamfer	ε-Rep.	EMD
Original data		1.81	0.86	2.27	1.27	0.89	2.31
SD	120×90	2.71	0.73	2.34	1.86	0.80	2.37
	80×60	2.74	0.72	2.42	1.90	0.78	2.44
	60×45	3.12	0.66	2.50	2.17	0.72	2.51
TD	fps/2	2.07	0.82	2.36	1.38	0.87	2.34
	fps/3	2.24	0.78	2.41	1.44	0.87	2.39
AN	1%	2.31	0.80	2.30	1.76	0.85	2.33
	3%	3.11	0.74	2.34	2.11	0.81	2.36
	5%	3.28	0.70	2.39	2.34	0.78	2.40
	7%	3.55	0.67	2.45	2.52	0.69	2.43

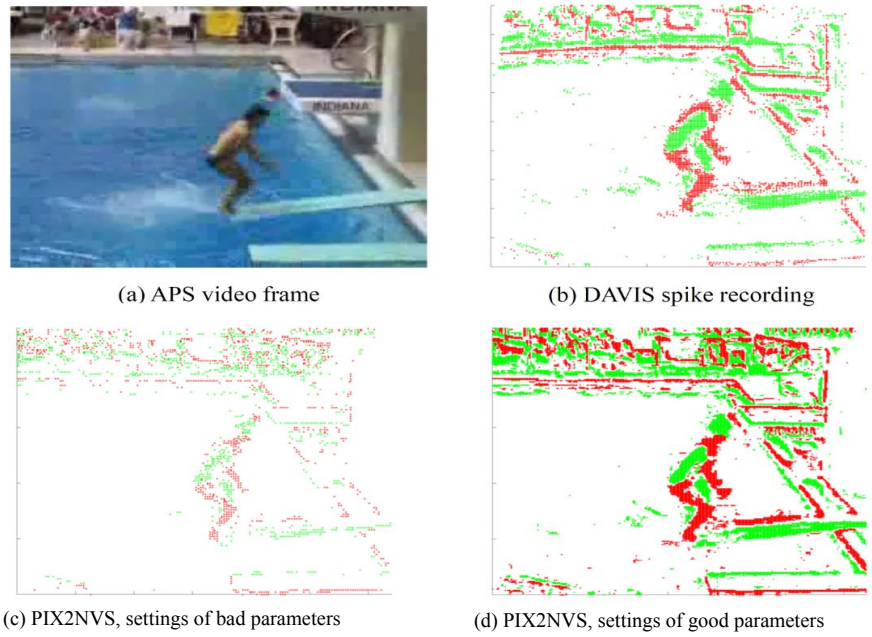
3.4.3 Parameter Optimization with Random Search

Before carrying out any action recognition experiments with emulated and real NVS, we must decide on the emulation parameters to use from the multitude of options of Table 3.1. To this end, we utilize the proposed distance metrics of Section 3.3 and the recently-released UCF-50 NVS recordings, i.e., a subset of UCF-101 consisting of 50 action categories [126], recorded with a DAVIS camera setup under the conditions described by Hu *et al.* [113]. We perform an adaptive random search over the emulation parameter space by repeatedly computing the distance metrics on the available UCF-50 NVS dataset and combining them into the "Weighted Score" measure described in Section 3.3. As this measure converges, we reduce the hypersphere radius from where we select the next hyperparameter set around the parameter options that led to decreased weighted distance.

We report the obtained distances for an indicative set of parameters in Table 3.3. The results confirm that, as we tend towards the optimal configuration, the domain shift between emulated and real spike datasets decreases. The only exception was the EMD metric, which was slightly increased in some parameter options. This is attributed to the contribution of outlier experimental points (or noise) and it emphasizes the significance of using the Weighted Score in Table 3.3 as the metric to

Table 3.3: Inverted ε-repeatability (i.e., 1 minus the ε-repeatability score, with $\varepsilon = 3.5$), Chamfer distance (with $\lambda = \sqrt{3}$), EMD and weighted distance score (smaller is better) between UCF-50 real and emulated spikes, w.r.t. different PIX2NVS options.

θ						Similarity Metrics			
T_{log}	B	M_{max}	T_{map}	`new`	`exp`	Inv. ε-rep	Chamfer	EMD	Score
20	2	1	0.4	`False`	`False`	0.575	5.455	2.448	0.500
0	2	1	0.2	`False`	`False`	0.496	4.479	2.597	0.454
0	2	1	0.2	`False`	`True`	0.469	4.387	2.467	0.435
0	2	1	0.2	`True`	`True`	0.464	4.253	2.434	0.427
0	1	3	0.2	`True`	`True`	0.455	4.172	2.307	0.414



(a) APS video frame

(b) DAVIS spike recording

(c) PIX2NVS, settings of bad parameters

(d) PIX2NVS, settings of good parameters

Figure 3.3: Comparison of PIX2NVS conversion against the spike events recorded with a DAVIS camera. Green/red points correspond to +1/-1 (or ON/OFF) spike polarity. Best viewed in color.

perform parameter optimization. Fig. 3.3 presents a visual comparison between the original APS frame [Fig. 3.3(a)], the recorded spike events with the DAVIS camera [Fig. 3.3(b)], and the emulated spike events with the worst and best PIX2NVS parameter settings from Table 3.3 (first and last row), shown in Fig. 3.3(c) and Fig. 3.3(d), respectively. Overall, it is clear that the best performance is achieved when opting for exponential moving average and full-reference frame update {`exp = True, new = True`}, as this effectively accounts for past frames and longer

motion cues. We find that varying the local inhibition window size B has minimal effect on accuracy, whilst setting $T_{\log} = 0$ [i.e., using $\ln(q_{x,y})$ in (3.1)] for `LICE_mode = LI` is beneficial in comparison to the other two modes. Importantly, the main source of similarity gain and domain alignment is achieved by increasing the maximum number of spikes generated, $M_{\max}$, and decreasing the filter threshold, T_{map}, at each position and per reference frame. As shown by (3.8) and confirmed by the visual example given in Fig. 3.3(c+d), this generates more spikes and thus increases the per-frame granularity of the emulator. Importantly, while the settings of the last row of Table 3.3 worked well for the utilized APS video content and NVS hardware (DAVIS camera), the use of the proposed distance metrics allows for the tuning of the emulation parameters under any APS input video and NVS camera hardware as long as indicative measurements are available.

3.5 Evaluating the effectiveness of Emulated NVS Data Streams

In this section, we evaluate the proposed framework for training a CNN based on emulated NVS data that has been generated with the PIX2NVS framework. Specifically, we test two studies here, namely CNN-based human action recognition in Section 3.5.1 and American Sign Language recognition in 3.5.2.

3.5.1 Human Action Recognition

Network Input: Deep CNNs requires the input as the grid images, so we firstly group the events into frames as input. We use the PIX2NVS framework to extract the emulated NVS events from RGB video frames, which provides us with training data correspondences for the NVS domain. The emulator generates a set of event tuples $E_e = \{\langle x_e, y_e, t_e, P_e \rangle\}$ over the video sequence, where the event polarity $P_e = \pm 1$ (i.e., representing ON or OFF). Let us denote the complete set of event and non-event tuples as $E = \{< x, y, t, P > | (< x, y, t, P > \notin E_e) \rightarrow P = 0\}$. We can aggregate the polarities into a single NVS frame $f[n]$ corresponding to the nth $(H \times W)$ RGB video frame by summing the polarities at each spatial position (x, y) for events

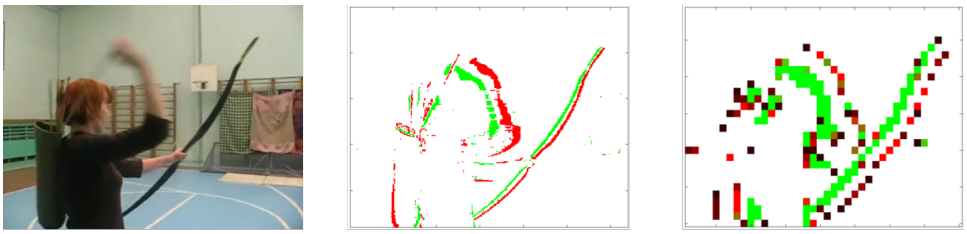


Figure 3.4: Extracted frame from PIX2NVS for an archery video from UCF-101; (a) RGB frame; (b) NVS frame (native resolution); (c) NVS frame (downsampled by 8). The pseudo-color in (c) corresponds to the continuous range generated after downsampling.

falling in the $(n-1)$th and nth video frame time interval I. For the nth video frame, the summed polarity at position (x,y), $f[n](x,y)$ is denoted as:

$$f[n](x,y) = \sum_{t \in I} P(x,y,t) \tag{3.18}$$

This enables spatio-temporal correspondence with the video frames. While this frame grouping is artificial, it allows for the use of CNNs for the recognition task [127], and we can now aggregate consecutive NVS frames into training volumes v_s.

The UCF-50 [126] dataset is composed of 320×240 RGB pixels per frame. Contrary to dense RGB frames, the NVS frames are notably sparse, which means several convolutional layers will be are required in order to extract complex representations from the input. The bulk of the execution complexity of a multilayer CNN is typically in the lower convolutional layers; doubling the input resolution to the CNN quadruples the number of activation in these layers. One option to reduce the running time is to memorize the convolutional operations in the lower layers and hash the non-zero entries and corresponding locations [128, 129]. Instead, when dealing with action recognition tasks, we argue that it is not necessary to model fine spatial dependencies. As such, we propose to decrease the input sparsity by downsampling the NVS frames by a factor of 8. This draws parallels to the sensing resolution of the human eye which, except for the fovea, is mostly considered to be low. Beyond such analogies, the practical benefit of downsampling is that it reduces the number of activation and the overall complexity of the network

substantially. For the datasets considered in this paper, this results in an NVS frame $\Phi \in \mathbb{R}^{W \times H}$, with $W = 40$ and $H = 30$. Fig. 3.4 shows an NVS frame at native and downsampled resolution for one of the frames of a UCF-101 video.

In order to compensate for the low spatial resolution, we take a long temporal extent of $T > 100$ consecutive NVS frames. This is contrary to recent proposals using high-resolution optical flow [130, 131], which typically ingest only a few frames per input (typically around 10) and inline with recent work that proposed such metrics for motion-based video classification [132]. A longer temporal extent is more likely to encompass the relevant action for classifying the video, without clipping it out. We therefore fix the temporal extent T to 160, which is close to the average number of extracted NVS frames per video in UCF-101. Finally, in order to make our CNN input independent of the NVS resolution, we use a fixed spatial size $N \times N$, which is cropped/resized from Φ; in this paper we set $N = 24$. Our final network input $\hat{\Phi} \in \mathbb{R}^{N \times N \times T}$ is thus 3D and can be ingested by a 3D CNN[3].

Network Architecture: The resized network inputs can be considered analogous to the motion vector magnitude that can be extracted from standard compressed video formats such as MPEG/ITU-T AVC/H.264 [133] and HEVC [134]. We therefore opt to utilize a recent 3D CNN architecture proposed for action recognition with motion vector inputs [132]. As demonstrated therein:

- the 3D CNN architecture achieves an optimal balance between complexity and classification performance, requiring substantially less parameters than other CNN based classifiers [130, 135];
- all convolutions and pooling are spatiotemporal in their extent;
- all convolutional layers and the first two fully-connected layers use the parametric ReLU activation function [136].

[3] As validation for our chosen setup, we have experimented with doubling the spatial resolution to 60×80 for our 3D CNN input, whilst lowering the temporal extent to 64 and varying the temporal strides, in order to maintain an equivalent complexity. When evaluating on UCF-50 we found that results are within 1% of the results reported in this paper, which suggests that, when using a longer temporal extent, downsampling has marginal effect in the accuracy during inference.

However, whereas two channels are required to ingest the δx and δy motion vector components as extracted from the video codec, we modify the 3D CNN to ingest a single $N \times N \times 1 \times T$ stream corresponding to the NVS frames produced by our PIX2NVS emulation framework (during training) or the DAVIS camera hardware (during testing).

3D CNN Training: We train the network using stochastic gradient descent with momentum set to 0.9. For the initialization and hyperparameters, we follow the protocol of previous work that utilized motion vector inputs [132] for video classification using UCF-101 and HMDB. It is important to note that, for our experiments on UCF-50, because this dataset is a subset of UCF-101 and is therefore prone to overfitting, we train the network from scratch. As such, we follow the same training configuration as UCF-101 but reduce the training time, decaying by a factor of 0.1 every 20k iterations, for 50k iterations. To further minimize the chance of overfitting due to the low spatial resolution and the small size of the training split for all datasets, we supplement the training with heavy data augmentation. To this end, we concatenate the NVS frames into a single $W \times H \times 1 \times T$ volume and apply the following steps; *(i)* a multi-scale random cropping to fixed size $N_c \times N_c \times 1 \times T$ from this volume, by randomly selecting a value for N_c from $N \times c$ with $c \in \{0.5, 0.667, 0.833, 1.0\}$; as such, the cropped volume is randomly flipped and spatially resized to $N \times N \times 1 \times T$; *(ii)* normalizing the input by subtracting the global mean and dividing by the global standard deviation over all values in the input. This normalizes the active sites in the NVS volume and is crucial for the training loss to decrease monotonically. During training, we apply additional regularization in the network by using dropout ratio of 0.8 on the first two fully-connected layers together with weight decay of 0.0005.

Testing based on Maximal Motion Activity Crops: During testing, per video, we generate 5 random volumes of temporal size T from which to test on. Rather than using the standard 10-crop testing (center and four corners, unflipped and flipped) [137], we propose to extract the salient region in each volume, which we identify as having continuous spike activity over consecutive NVS frames. In

order to locate this region, we first sum over the absolute values of all frames in a volume to generate a spike activity map $M \in \mathbb{R}^{W \times H}$. We sum-pool the map intensities in M by sliding an $N \times N$ window over the activity map, thus generating a map of summed motion per potential crop M'. We then locate the salient region by finding the index of the maximum value in M' . Per volume, we thus extract a crop at this position and a center crop, both of size $N \times N \times T$, and consider both horizontally flipped and unflipped counterparts. As such, we average the scores to produce a single score for the video. In this respect, we only have to process 4 crops per volume to generate the score. When evaluating on UCF-101 (split 1), the difference in performance between our proposal and 10-crop testing is less than 1%, whilst our proposal requires less than half the processing to generate the test results.

Accuracy of NVS-based Action Recognition: Table 3.4 presents the results on UCF-50 when using real & emulated spike events to train & test. The recognition accuracy only drops up to 5.1% when training & testing between different spike event sets, in comparison to using the same spike event types. To the best of our knowledge, we are the first to evaluate on UCF-50 with a deep CNN ingesting NVS inputs and our results are competitive or surpass well-known results on this dataset [126]. Specifically, the most competitive results to ours is the APS-based work of Todorovic *et al.* [138], which achieves 81.03% on UCF-50 by modelling extracted video frame features on a stochastic Kronecker graph that requires 50-100W [139] for the APS capture and preprocessing on a dual-core CPU to extract video frame features. Therefore, despite the loss of 6.16% in accuracy, our approach is four orders of magnitude more power efficient for the capturing and preprocessing stages, as there is no preprocessing beyond spike aggregation and framing and the DAVIS camera incurs only 10mW of active power consumption [20, 113].

More recent proposals focus on UCF-101, which is a bigger and more challenging dataset, and tend to utilize motion activity or APS frames as inputs to a deep learning framework [132]. To assess our proposal against such inputs, we compare its recognition accuracy with the one achieved when the same CNN is

Table 3.4: Recognition accuracy on UCF-50 between training and testing with real and emulated NVS frames.

		Test	
		Real	Emulated
Train	Real	79.92%	75.32%
	Emulated	74.87%	78.36%

trained with the motion vector magnitude input $I = \sqrt{\delta x^2 + \delta y^2}$ that was extracted from the video coding format [132]. On UCF-101 (split 1), both NVS and MV-based CNNs were found to achieve comparable accuracy, i.e., 70.7% and 71.0% respectively. This result can also be interpreted as initial experimental validation that NVS and motion vector inputs from the video codec comprise scene activity representations that can be found to be equally informative for a 3D CNN. With regards to benchmarks against the state-of-the-art, recent work utilizing optical flow and CNNs [130] is able to achieve 81.2% on UCF-101, which is 11% higher than our results. However, the APS frame capture and preprocessing for optical flow extraction requires a GPU, which incurs extreme power and latency overheads in comparison to spike ingestion. For instance, Brox flow estimation is reported to run at only 0.15 FPS on a Tesla K80 GPU [132] (with active power of more than 200W). Similarly as before, this corresponds to four orders of magnitude of power increase for the sensing and preprocessing stages in comparison to our approach that only requires power of the order of 10mW for the spike events to be captured, aggregated and framed to be ingested by our CNN (which has 3 times less weights than optical-flow oriented CNN processing [132]).

3.5.2 American Sign Language Recognition

In order to further validate our proposal against APS inputs, we introduce a new sign language dataset for NVS-based classification. The dataset consists of 1200 real and 1200 emulated NVS recordings, each representing a different static sign of 24 letters (A-Y, excluding J) from the American Sign Language. The emulated spike events are generated using the PIX2NVS framework on APS video recordings, which are captured using a standard laptop camera. The real spike events are generated with

an iniLabs DAVIS240c NVS camera set up in the same environment and at the same position and orientation to the person. Each signed letter recording lasts a couple of seconds and is produced with different static hand positioning and hand motion relative to the camera in order to introduce natural variance into the dataset. Note that the experiments done in this section are different from the the American sign language recognition experiments in Chapter 4. Specifically, the purpose here is to evaluate our emulator PIX2NVS and using frame-based CNNs as tool, while in Chapter 4 we design experiments to evaluate the graph representation and our graph CNNs and also we largely increase the amount of ALS datasets used in the experiment.

Network Input and Architecture: Fig. 3.5 shows the required hand pose for each letter of the alphabet. For some letters (e.g., M and N), there is little variation in finger positioning. Similar to prior work on sign language recognition [140], letters J and Z are excluded from our dataset as their ASL designation requires motion. Due to the relatively static (and short) nature of the sign recordings, having a long temporal extent over the video is less important when evaluating the signed letter, as the hand can be considered a rigid body with mainly translational motion and limited rotational motion over the video duration. Therefore, contrary to the UCF-101 action recognition task, spatial resolution is more important than temporal extent, in order to distinguish between the individual fingers. As such, for this dataset we opt for the standard VGG16 architecture pre-trained on ImageNet and fine-tune all convolution layers and the first two fully connected layers on the sign language dataset. However, as the NVS frames comprise only one channel, we repurpose the RGB channels of the VGG16 input to take three consecutive frames, in order to introduce local motion cues that may assist recognition performance.

Fig.3.6 compares the three-frame real NVS, emulated NVS and APS (grayscale) inputs for two examples from the datasets. Under our experimental setup, the DAVIS camera turned out to produce higher-than-expected ”salt & pepper” noise. Therefore, prior to CNN ingestion, we denoised each NVS frame by median filtering with a block size of 2×2. Such denoising is commonly applied

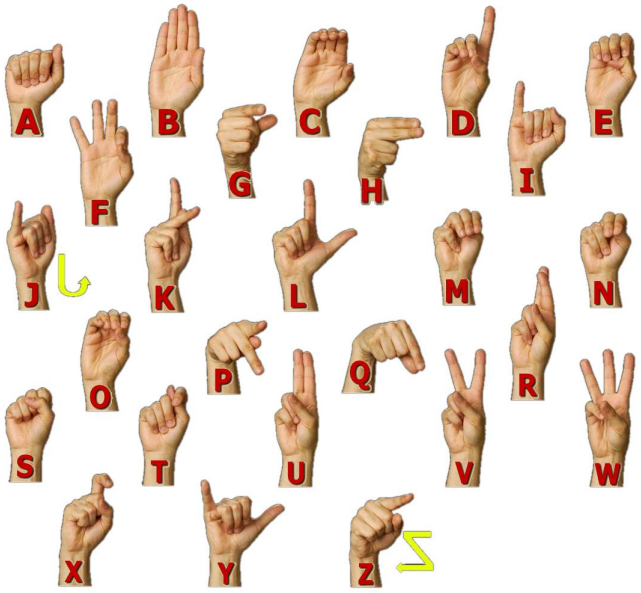


Figure 3.5: Signs for letters A-Z from the American Sign Language (ASL). Some letters such as M and N only have subtle differences. Letters J and Z are not static signs and require motion.

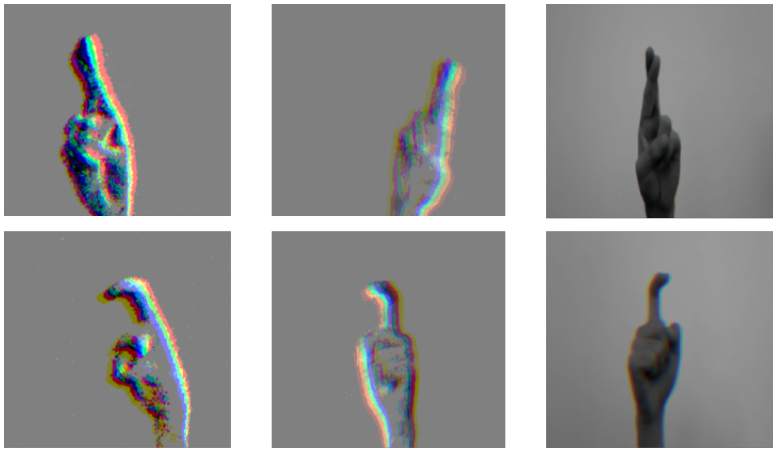


Figure 3.6: Example of standardized 3-frame inputs to VGG16 for letters R (top row) and X (bottom row): (a) Real NVS after 2×2 median filter; (b) Emulated NVS; (c) APS (grayscale)

when converting NVS sensing to frame representations [141], and, in this case, we have found that it improves the visual similarity with the emulated NVS frames. This is illustrated in Fig. 3.6 for letters R and X.

Training and Testing: We train the VGG16 architecture using stochastic gradient descent with momentum set to 0.9. The learning rate is initialized at 10^{-3}

and decayed by a factor of 0.1. We complete training at 15k iterations. In order to combat overfitting, we set dropout and weight decay on the first two fully connected layers to 0.8 and 0.005 respectively. We first resize the input frames such that the smaller side is 256 and keep the aspect ratio. We then use a multi-scale random cropping of the resized RGB frame; the cropped volume is subsequently randomly flipped, and normalized according to its mean and standard deviation, as in Section 3.5.1. During testing, as the background is relatively uniform, we only take a single maximal motion activity crop following the method described in Section 3.5.1.

Accuracy of NVS-based Sign Language Recognition: We further motivate the efficacy of the proposed NVS emulation on the sign language recognition. First, in order to show that transfer learning on the grayscale frames does not suffice (since the domain shift between the pixel and NVS is too large), Table 3.5 compares the accuracy achieved on the real NVS frames when fine-tuning pre-trained VGG16 on grayscale frames versus fine-tuning on emulated NVS frames. The emulated spikes are generated using the best parameter set measured on UCF-50 (i.e., the bottom row of Table 3.3). Table 3.5 shows that the use of PIX2NVS allows for more than 35% increase in accuracy by converting pixels to a representation that resembles experimental spike events.

In order to assess this result against other approaches, we compare against recent work on real-time sign language recognition using consumer depth cameras [140]. The paper uses data recorded from an Intel Creative Gesture Camera. This device is a low-cost time-of-flight camera with range from 10cm to 1m, maximal frame rate of 50 fps, resolution of 240×320 pixels (for depth data), and dynamic power consumption in the order of the Kinect sensor [142], i.e., 2.25–4.7W [143] (versus only 10mW for the DAVIS sensor). Their dataset consists of 3 subjects performing the same 24 signs from the ASL sign language as in our dataset (letters A-Y except J). The recognition accuracy is reported at 78%, which is only 4.5% higher than our result in Table 3.5. In summary, by using NVS-based inputs we are able to achieve comparable accuracy on a larger sign language recognition dataset, whilst our CNN inputs are: *(i)* of lower resolution, *(ii)* generated by the

DAVIS camera at significantly higher framerate and *(iii)* with two orders of magnitude lower power consumption.

Table 3.5: Recognition accuracy on sign language dataset when training on grayscale and emulated NVS frames and and testing on real NVS frames.

		Test NVS (Real)
Train	APS (Grayscale)	38.28%
	NVS (Emulated)	73.52%

3.6 Conclusion

A key challenge when attempting recognition tasks with NVS data is the lack of annotated datasets to train advanced machine learning frameworks with. In this paper, we attempt to address this by proposing NVS emulation framework that can convert APS videos from action recognition datasets to emulated NVS spike events. Our main observations are: *(i)* we demonstrate how downsampling the converted NVS and grouping them into 'frame' representations of long temporal extent leads to dense representations that are suitable for 3D CNN training for action recognition; *(ii)* our results show that the gap between *emulated spike events* for training and *real spike events* for testing is approximately five percentile points, which indicates minimal domain shift after the proposed PIX2NVS framework; *(iii)* training and testing with emulated NVS spike events is shown to achieve comparable performance to the equivalent network that uses motion vector magnitudes extracted from the compressed bitstream of the utilized videos; *(iv)* we demonstrate how emulated NVS inputs can be used with a pre-trained ImageNet CNN for sign language recognition and introduce a new NVS-based sign language recognition dataset for evaluation; *(v)* overall, via PIX2NVS and appropriate parameter setting, for the first time, NVS-based action recognition is shown to be within range of the best results of recent proposals that ingest APS data. Also, we proposed and evaluated three distance metrics to quantify the accuracy of the model-generated events against ground-truth and optimize the parameter of PIX2NVS with random search.

Acknowledge that there are shortcomings in our proposed methods. First of all, the timestamp of generated events are largely restricted by the frame of rate of video collections, which may increase the latency of NVS streams. Even though we used linear interpolation method to assign timestamps, the temporal resolution still is a challenge to improve. In addition, generated events are much cleaner compared to the real events captured from NVS device. This may not imitate the nature of real events so that it may not make the developed algorithms have the same performance when using them in real life applications. As we all know, the noise definitely can increase the robustness of the algorithms. These drawbacks are definitely needed to be improved in the future work.

Chapter 4

Graph-based Object Classification for Neuromorphic Vision Sensing

4.1 Introduction

Object classification finds numerous applications in visual surveillance, human-machine interfaces, image retrieval and visual content analysis systems. Following the prevalence and advances of CMOS active pixel sensing (APS), deep convolutional neural networks (CNNs) have already achieved good performance in APS-based object classification problems [5, 137]. However, APS-based sensing is known to be cumbersome for machine learning systems because of limited frame rate, high redundancy between frames, blurriness due to slow shutter adjustment under varying illumination, and high power requirements [144]. Inspired by the photoreceptor-bipolar-ganglion cell information flow in low-level mammalian vision, researchers have devised cameras based on neuromorphic vision sensing (NVS) [21, 144, 145]. NVS hardware outputs a stream of asynchronous ON/OFF address events (a.k.a., 'spikes') that indicate the changes in scene reflectance. An example is shown in Fig 4.1, where the NVS-based spike events correspond to a stream of coordinates and timestamps of reflectance events triggering ON or OFF in an asynchronous manner. This new principle significantly reduces the memory usage, power consumption and redundant information across time, while offering low latency and very high dynamic range.

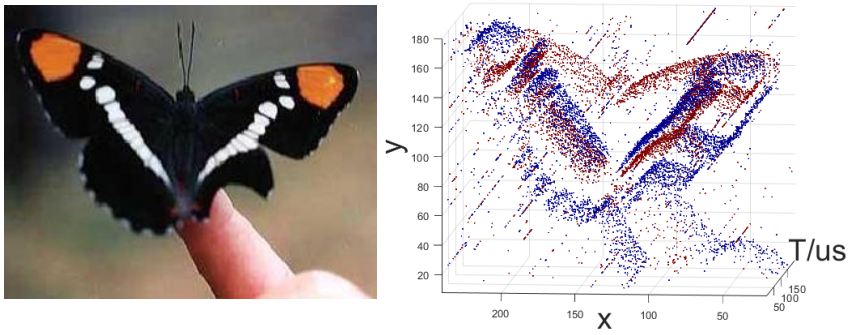


Figure 4.1: Examples of objects captured by APS and neuromorphic vision sensors. Left: Conventional APS image. Right: Events stream from NVS sensor (Red:ON, Blue:OFF).

However, it has been recognized that current NVS-based object classification systems are inferior to APS-based counterparts because of the limited amount of work on NVS object classification and the lack of NVS data with reliable annotations to train and test with [21, 144, 146]. In this Chapter, we improve on these two issues by firstly proposing graph-based object classification method for NVS data. Previous approaches have either artificially grouped events into frame forms [4, 38–40] or derived complex feature descriptors [35, 72, 147], which do not always provide for good representations for complex tasks like object classification. Such approaches dilute the advantages of compactness and asynchronicity of NVS streams, and may be sensitive to the noise and change of camera motion or viewpoint orientation. To the best of our knowledge, this is the first attempt to represent neuromorphic spike events as a graph, which allows us to use residual graph CNNs for end-to-end task training and reduces the computation of the proposed graph convolutional architecture to one-fifth of that of ResNet50 [5], while outperforming or matching the results of the state-of-the-art.

With respect to benchmarks, most neuromorphic datasets for object classification available to date are generated from emulators [70, 117, 148], or recorded from APS datasets via recordings of playback in standard monitors [113, 149, 150]. However, datasets acquired in this way cannot capture scene reflectance changes as recorded by NVS devices in real-world conditions. Therefore, creating real-world NVS datasets is important for the advancement of NVS-based computer vision. To this end, we create and make available a dataset of NVS recordings of 24 letters (A-

Y, excluding J) from the American sign language. Our dataset provides more than 100K samples, and to our best knowledge, this is the largest labeled NVS dataset acquired under realistic conditions.

In this chapter we address two issues. First, in Section 4.2, we briefly explain the framework of graph-based object classification for neuromorphic vision sensing. Specifically, in Section 4.2.1, we propose a graph-based representation for neuromorphic spike events, allowing for fast end-to-end task training and inference; in Section 4.2.5, we introduce *residual graph CNNs* (RG-CNNs) as classifier for object classification. This network requires less computation and memory in comparison to conventional CNNs, while achieving superior results to the state-of-the-art in various datasets as shown in Section 4.4. Second, we source one of the largest and most challenging neuromorphic vision datasets, acquired under real-world conditions, and make it available to the research community in Section 4.3.

We summary the novelties and contributions of this chapter here:

1. Graph representation for NVS streams pave a new ways for researchers to explore as this kind of presentation can maintains compactness and sparsity of events.

2. There are many potential applications by coupling graph representation with graph-based deep learning methods, which not only can be implemented in an end-to-end manners, but also can use the well-established gradient-based learning rules to train a system with better performance.

3. Graph-based learning methods for NVS streams do not dilute the advantages of neuromorphic vision sensing, which can offer a efficient system that require less memory and computation.

4.2 Methodology

Our goal is to represent the stream of spike events from neuromorphic vision sensors as a graph and perform convolution on the graph for object classification. Our model is visualized in Fig. 4.2: a non-uniform sampling strategy is firstly used to

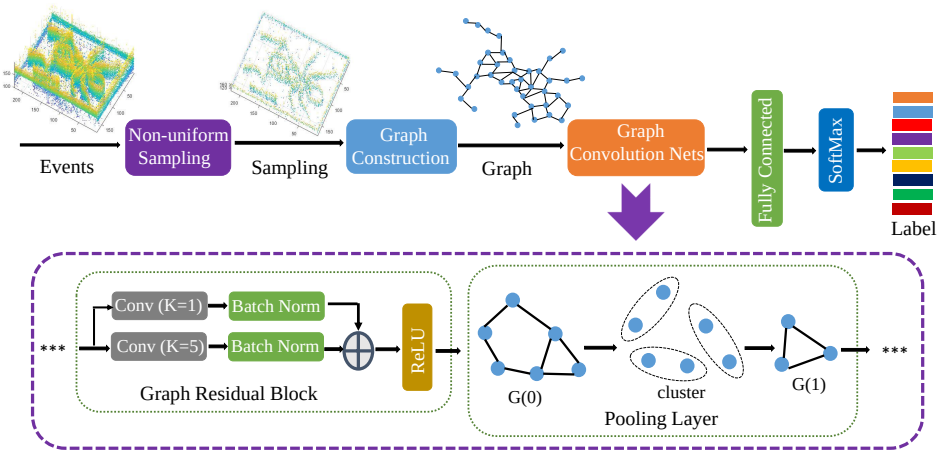


Figure 4.2: Framework of graph-based object classification for neuromorphic vision sensing, indluding non-uniform sampling, graph constrnction and residual-graph CNNs.

obtain a small set of neuromorphic events for computationally and memory-efficient processing; then sampling events are constructed into a radius neighborhood graph, which is processed by our proposed residual-graph CNNs for object classification. As to residual-graph GNNs, it is mainly stacked by graph residual block, graph pooling layers and fully connected layers as a classifier. The details will be described in the following section.

4.2.1 Non-uniform Sampling & Graph Construction

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

$$\{e_i\}_N = \{x_i, y_i, t_i, p_i\}_N \tag{4.1}$$

where $(x_i, y_i) \in \mathbb{R}^{H \times W}$ indicates the spatial address at which the spike event occured, t_i is the timestamp indicating when the event was generated, $p_i \in \{+1, -1\}$ is the event polarity (with +1, -1 signifying ON and OFF events respectively), and N is the total number of the events. To reduce the storage and computational cost, we use non-uniform grid sampling [151] to sample a subset of M representative events $\{e_i\}_M$ from $\{e_i\}_N$, where $M \ll N$. Effectively, one event is randomly selected from a space-time volume with the maximum number of events inside. If we consider

$\mathbf{s}\{e_i\}_{i=1}^{k}$ to be such a grid containing k events, then only one event e_i ($i \in [1,k]$) is randomly sampled in this space-time volume. We then define the sampling events $\{e_i\}_{\{m\}}$ on a directed graph $\mathrm{G} = \{\mathcal{V}, \mathcal{E}, \mathrm{U}\}$, with $\mathcal{V}$ being the set of vertices, $\mathcal{E}$ the set of the edges, and U containing pseudo-coordinates that locally define the spatial relations between connected nodes. The sampling 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 $v_i : (x_i, y_i, t_i)$, with $v_i \in \mathcal{V}$. We define the connectivity of nodes in the graph based on the radius-neighborhood-graph strategy. Namely, neighboring nodes v_i and v_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:

$$d_{i,j} = \sqrt{\alpha(|x_i - x_j|^2 + |y_i - y_j|^2) + \beta |t_i - t_j|^2} \leq \mathrm{R} \tag{4.2}$$

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 $u(i, j) = \left[\left|x_i - x_j\right|, \left|y_i - y_j\right|\right] \in \mathrm{U}$. After connecting all nodes of the graph $\mathrm{G} = \{\mathcal{V}, \mathcal{E}, \mathrm{U}\}$ 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 $f^{(0)}(i) = p_i \in \{+1, -1\}$.

4.2.2 Graph Convolution

Generalizing neural networks to data with graph structures is an emerging topic in deep learning research. The principle of constructing CNNs on graph generally follows two streams: the spectral perspective [87–93] and the spatial perspective [3, 94–98]. Spectral convolution applies spectral filters on the spectral components of signals on vertices transformed by a graph Fourier transform, followed by spectral convolution. Defferrard [89] provided efficient filtering algorithms by ap-

proximating spectral filters with Chebyshev polynomials that only aggregate local K-neighborhoods. This approach was further simplified by Kipf [87], who consider only the one-neighborhood for single-filter operation. Levie [88] proposed a filter based on the Caley transform as an alternative for the Chebyshev approximation. As to spatial convolution, convolution filters are applied directly on the graph nodes and their neighbors. Several research groups have independently dealt with this problem. Duvenaud [94] proposed to share the same weights among all edges by summing the signal over neighboring vertices followed by a weight matrix multiplication, while Atwood [95] proposed to share weights based on the number of hops between two vertices. Finally, recent work [3, 98] makes use of the pseudo-coordinates of nodes as input to determine how the features are aggregated during locally aggregating feature values in a local patch. Spectral convolution operations require an identical graph as input, as well as complex numerical computations because they handle the whole graph simultaneously, so it is not suitable for the variable and large graphs constructed from NVS. Therefore, to remain computationally efficient, our work follows the spirit of spatial graph convolution approaches and extends them to NVS data for object classification.

Similar to conventional frame-based convolution, spatial convolution operations on graphs are also an 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 $f(i)$, $\mathfrak{N}(i)$ denote the set of neighbors of node i and $g(u(i,j))$ denote the weight parameter constructed from the kernel function $g(.)$. The graph convolution operator $\otimes$ for this node can then be written in the following general form

$$(f \otimes g)(i) = \frac{1}{|\mathfrak{N}(i)|} \sum_{j \in \mathfrak{N}(i)} f(j) \cdot g(u(i,j)) \tag{4.3}$$

where $|\mathfrak{N}(i)|$ is the cardinality of $\mathfrak{N}(i)$. We can generalize (4.3) 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 opera-

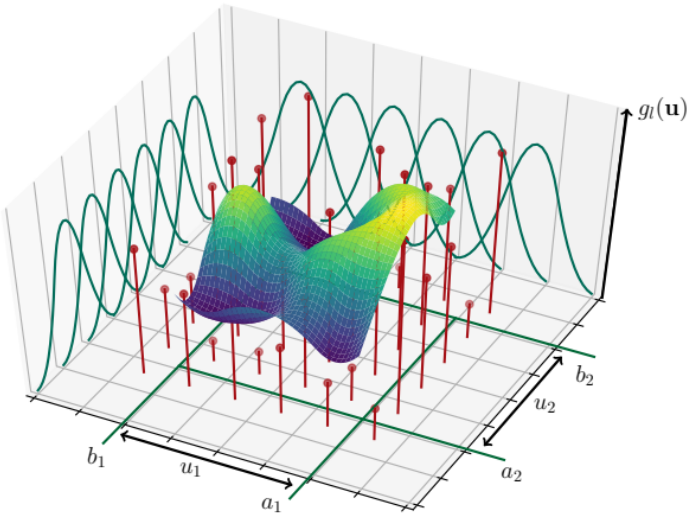


Figure 4.3: Quadratic B-spline basis functions (reproduced from [3]): for kernel dimensionality, The heights of the red dots represent trainable parameters, which are multiplied by the elements of the B-spline tensor product basis.

tion $\otimes$ for the node i with M_{in} feature maps is defined as:

$$(\mathbf{f} \otimes \mathbf{g})(i) = \frac{1}{|\mathfrak{N}(i)|} \sum_{l=1}^{M_{in}} \sum_{j \in \mathfrak{N}(i)} f_l(j) \cdot g_l(u(i,j)) \tag{4.4}$$

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. Therefore, several spatial convolution operations [3,96–98] on graphs were proposed by using different choice of kernel functions $\mathbf{g}$. Among them, SplineCNN [3] 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 as shown in Fig. 4.3 to efficiently filter NVS graph inputs of arbitrary dimensionality. Let $((N^m_{1,i})_{1 \le i \le k_1}, \ldots, (N^m_{d,i})_{1 \le i \le k_d})$ denote d open $\mathbf{B}$-spline bases of degree m with $\mathbf{k} = (k_1, \ldots, k_d)$ defining d-dimensional kernel size [152]. Let $w_{p,l} \in \mathbf{W}$ denote a trainable parameter for each element p from the Cartesian product $\mathfrak{P} = (N^m_{1,i})_i \times \cdots \times (N^m_{d,i})_i$ of the B-spline bases and each of the M_{in} input feature maps indexed by l. Then the kernel function $g_l : [a_1, b_1] \times \cdots \times$

$[a_d, b_d] \to \Re$ is defined as:

$$g_l(\mathbf{u}) = \sum_{p \in P} w_{p,l} \cdot \prod_{i=1}^{d} N_{i,p_i}(u_i) \tag{4.5}$$

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, activated by activation function $\xi(t)$, can be written as:

$$\begin{aligned} \text{Conv}_{l'} &= \xi\Big(\frac{1}{|\mathfrak{N}(i)|} \sum_{l=1}^{M_{\text{in}}} \sum_{j \in \mathfrak{N}(i)} f_l(j) \cdot \sum_{p \in P} w_{p,l} \\ &\cdot \prod_{i=1}^{d} N_{i,p_i}(u_i) + b_{l'}\Big) \end{aligned} \tag{4.6}$$

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

Finally, to accelerate deep network training, we use batch normalization [153] before the activation function in each graph convolutional layer. That is, the whole node feature $f_{l'}$ over $l^{'}$ channel map is normalized individually via:

$$f_{l'}^{'} = \frac{f_l - E(f_{l'})}{\sqrt{\text{Var}(f_{l'}) + \varepsilon}} \cdot \gamma + \beta \tag{4.7}$$

where $l^{'} = 1, .., M_{\text{out}}$, $E(f_{l'})$ 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.

4.2.3 Pooling Layer

The utility of a pooling layer is to compact feature representations, in order to preserve important information while discarding irrelevant details [154]. In conventional APS-oriented CNNs, because of the uniform sampling grid (e.g., regular

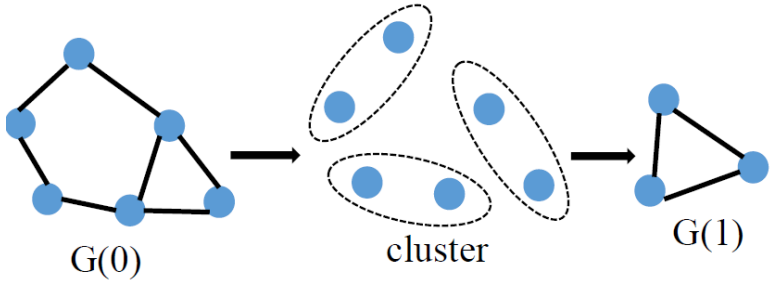


Figure 4.4: Illustration of graph pooling operation.

pixel array in images), pooling layers can be easily implemented by performing a max, average, or sum operation over neighbouring features. Similar to recent work in graph pooling [155], we apply pooling in order to obtain a coarser NVS graph. As shown in the pooling layer of the Fig. 4.4, we first derive fixed-size clusters for graphs based on the node coordinates, then aggregate all nodes within one cluster, followed by the computation of new coordinates and features for the new nodes.

Given a graph representation, let us denote the spatial coordinates for node i as $(x_i', y_i') \in \mathbb{R}^{H' \times W'}$ 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, leading to $\left\lceil \frac{H'}{s_h} \right\rceil \times \left\lceil \frac{W'}{s_w} \right\rceil$ clusters. Given there are num nodes $\{v_1, \ldots, v_{\text{num}}\}$ in one cluster, only one new node is generated on each cluster. For this new node, the coordinates $(x_{\text{new}}, y_{\text{new}})$ are the average of coordinates of these num nodes,

$$\begin{cases} x_{\text{new}} = \lfloor \sum_{i=1}^{\text{num}} x_i / \text{num} \rfloor \\ y_{\text{new}} = \lfloor \sum_{i=1}^{\text{num}} y_i / \text{num} \rfloor . \end{cases} \tag{4.8}$$

And the feature is the average or maximum of feature of these num nodes, according to whether a max pooling (MaxP) or average pooling (AvgP) strategy is used.

$$\begin{cases} f_{v_{l,new}} = MaxPool(f_{v_{l,i}}) = max_{i=1}^{num}(f_{v_{l,i}}) \quad OR \\ f_{v_{l,new}} = AvgPool(f_{v_{l,i}}) = \sum_{i=1}^{num} f_{v_{l,i}} / num \end{cases} \tag{4.9}$$

Importantly, if there are connected nodes between two clusters, we assume the new generated nodes in these two clusters are connected with an edge. Let's denote a max-pooling/average-pooling layer using above strategy with MaxP(O)/AvgP(O), where O is the number of output channels. s_h and s_w are the

dimension size of clusters, allowing for various down-scaling.

4.2.4 Fully Connected Layer

Given M_{in} feature maps $\mathbf{f} \longrightarrow \mathbb{R}^{P \times M_{\text{in}}}$ from a graph with P nodes, similar to CNNs, a fully connected layer in a graph convolutional network is a linear combination of weights linking all input features to outputs. Let us denote $f_l^p(x)$ as the feature in lth feature map of the pth node, then we can derive a fully connected layer for $q = 1, ..., Q$ as:

$$f_q^{\text{out}}(x) = \xi\left(\sum_{p=1}^{P}\sum_{l=1}^{M_{\text{in}}} F_{P \times M_{in} \times Q} f_l^p(x)\right) \tag{4.10}$$

where Q is the number output channels indexed by q, F is trainable weight with size $\mathrm{P} \times \mathrm{M_{in}} \times \mathrm{Q}$, $\xi(t)$ is the non-linear activation function, e.g. ReLU: $\xi(t) = \max(0,t)$. For the remainder of the paper, we use $\mathrm{FC}(Q)$ to indicate a fully connected layer with Q output dimensions, comprising the results of (4.10).

4.2.5 Residual Graph CNNs

Inspired by the idea of ResNet [5], we propose residual graph CNNs in order to resolve the well-known degradation problem inherent with increasing number of layers (depth) in graph CNNs [156]. We apply residual connections for NVS-based object classification, as shown in the related block of Fig. 4.2. Consider the plain (non-residual) baseline is a graph convolutional layer with the kernel size of 5 in each dimension, followed by a batch normalization [153] that accelerates the convergence of the learning process. We consider a "shortcut" connection as a graph convolution layer with kernel size of 1 in each dimension, which matches the dimension of the output future maps, and is also followed by batch normalization. Then we perform element-wise addition of the node feature between shortcut and the baseline, with ReLU activation function. We denote the resulting graph residual block as $\mathrm{Res_g(c_{in}, c_{out})}$, with c_{in} input feature maps and c_{out} output feature maps.

We follow the common architectural pattern for feed-forward networks of interlaced convolution layers and pooling layers topped by fully-connected layers. For an input graph, a single convolutional layer is firstly applied, followed by batch normalization, and max pooling. This is then followed by L graph residual blocks,

each followed by a max pooling layer. Finally, two fully connected layers map the features to classes. For example, for $L = 2$, we have the following architecture: Conv $\longrightarrow$ MaxP $\longrightarrow$ Res_{g} $\longrightarrow$ MaxP $\longrightarrow$ Res_{g} $\longrightarrow$ MaxP $\longrightarrow$ FC $\longrightarrow$ FC.

4.3 Proposed American Sign Language Dataset

In this section, we first describe the existing NVS object classification datasets and then we introduce our dataset that provides for an enlarged pool of NVS training and testing examples for handshape classification.

4.3.1 Existing Neuromorphic Datasets

Many neuromorphic datasets for object classification are converted from standard frame-based datasets, such as N-MNIST [114], N-Caltech101 [114], MNIST-DVS [149] and CIFAR10-DVS [150]. N-MNIST and N-Caltech101 were acquired by an ATIS sensor [145] 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 [1]. 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 [70, 117, 148]. 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 [35] was created by directly recording objects in urban environments with an ATIS sensor. This two-class real-world dataset comprises 12,336 car samples and 11,693 non-car samples (background) with 0.1 second length. Despite its size, given that it only corresponds to a binary classifier problem, N-CARS cannot represent the behaviour of object classification algorithms on more complex NVS-based tasks.

4.3.2 Description of ASL-DVS

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

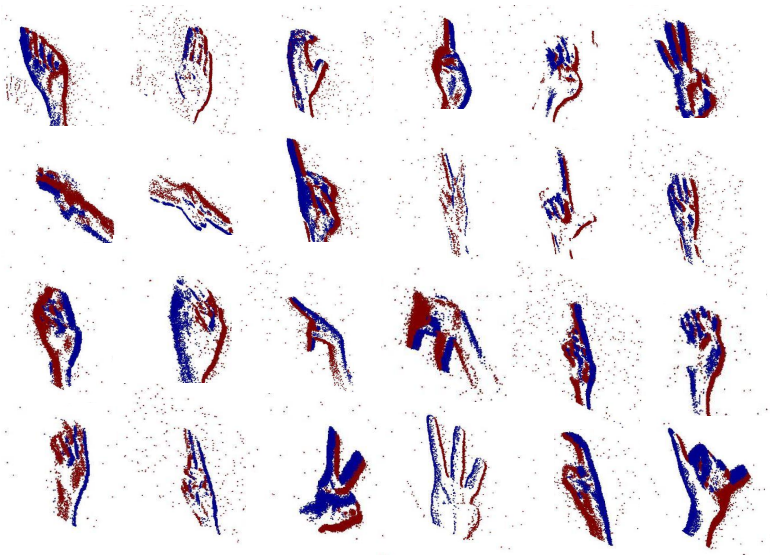


Figure 4.5: Examples of the ASL-DVS dataset (the visualizations correspond to letters A-Y, excluding J, since letters J and Z involve motion rather than static shape). Events are grouped to image form for visualization (Red/Blue: ON/OFF events).

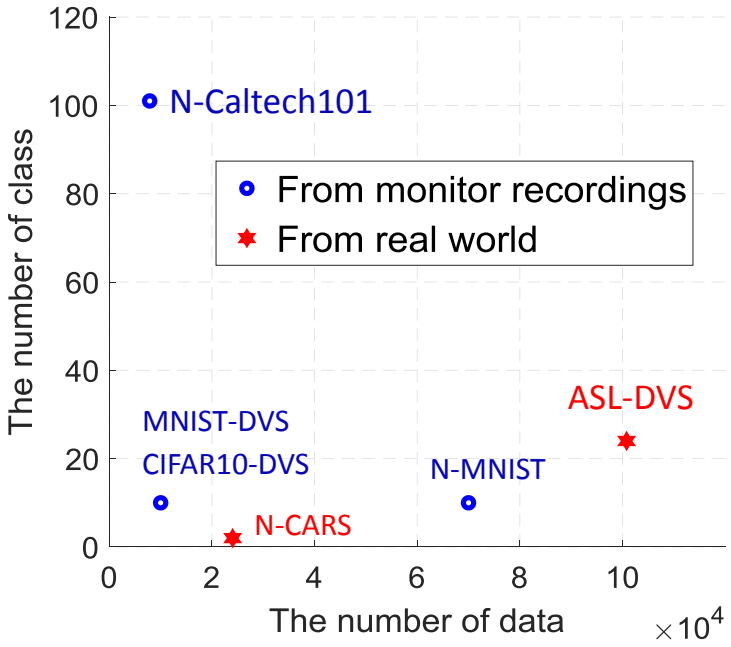


Figure 4.6: Comparison of proposed NVS dataset w.r.t. the number of class and the number of total size.

Sign Language (ASL), which we call ASL-DVS. Examples of recordings are shown in Fig 4.5. The ASL-DVS was recorded with an iniLabs DAVIS240c NVS camera set up in an office environment with low environmental 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. As is evident from Fig. 4.5, our ASL-DVS dataset presents a challenging task for event-based classifiers, due to the subtle differences between the finger positioning of certain letters, such as N and O (first two letters in row 3). Fig. 4.6 shows a comparison of existing NVS datasets w.r.t. the number of classes and 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 model for NVS-based object recognition, especially given the fact that it comprises real-world recordings. ASL-DVS and related code can be found at this link: https://github.com/PIX2NVS/NVS2Graph.

4.4 Experimental Results

4.4.1 Parameters Searching w.r.t. Performance and Complexity

In this section, we explore how performance and complexity are affected when varying the key parameters of our approach. Via the ablation studies reported here, we justify the choice of parameters used for our experiments in the final experiments.

With regards to the parameters of the proposed graph CNNs, we experiment with the non-residual (i.e., plain) graph architecture (G-CNN) as a representative example, and explore the performance when varying the depth of graph convolution layer and the kernel size of graph convolution. Concerning the graph construction, our studied parameters are the time interval under which we extract events, the event sample size and the radius distance (R) used to define the connectivity of the nodes. All experiments reported in this supplementary note were conducted on the

N-Caltech101 dataset, since it has the highest number of classes among all datasets. Finally, training methods and data augmentation follow the description given in Section 5.1 of the paper.

Event Sample Size for Graph Construction: The primary source of input compression is the non-uniform sampling of the events prior to graph-construction, which is parameterized by k in the paper. We explore the effects of this input compression by varying k and evaluating the accuracy to complexity (GFLOPs) tradeoff in Table 4.1. No compression (i.e., k = 1) gives accuracy/GFLOPs = 0.636/3.74, whereas increasing compression with k = 12 gives accuracy/GFLOPs = 0.612/0.26 (i.e., 93% complexity saving). This suggests that the accuracy is relatively insensitive to compression up to k = 12 (with k = 8 providing an optimal point) and it is the graph CNN that provides for state-of-the-art accuracy.

Table 4.1: Top-1 accuracy and complexity (GFLOPs) w.r.t. event sample size, parameterized by k.

k	Accuracy	GFLOPs
1	0.636	3.74
8	0.630	0.39
12	0.612	0.26

Radius Distance: When constructing graphs, the radius-neighborhood-graph strategy is used to define the connectivity of nodes. The radius distance (R) is an important graph parameter: when the radius is large, the number of generated graph edges increases, i.e., the graph becomes denser and needs increased GFLOPs for the convolutional operations. On the other hand, if we set a small radius, the connectivity of nodes may decrease to the point that it does not represent the true spatio-temporal relations of events, which will harm the classification accuracy. In this ablation study, we varied the radius distance to $\mathrm{R} = \{1.5, 3, 4.5, 6\}$, to find the best distance with respect to accuracy and complexity. The results are shown in Table 4.2, where we demonstrate that radius distance above 3 cannot improve the model performance while incurring significantly increased complexity. Therefore, in our paper we set the radius distance to 3. Note that when radius distance changes from 4.5 to 6, the required computation increases only slightly because of the max-

imum connectivity degree D_{max} that is set to 32 to constrain the edge volume of graph.

Table 4.2: Top-1 accuracy and complexity (GFLOPs) w.r.t. radius distance

Radius distance	Accuracy	GFLOPs
1.5	0.551	0.33
3	0.630	0.39
4.5	0.626	0.98
6	0.624	1.19

Time Interval of Events: For each sample, events within a fixed time interval are randomly extracted to input to our object classification framework. In this study, we test under various time intervals, i.e., 10, 30, 50 and 70 milliseconds, to see their effect on the accuracy and computation. The results are shown in Table 4.3. When extracting 30ms-events from one sample, the model achieves the highest accuracy, with modest increase in complexity over 10ms-events. Therefore, we opted for this setting in our paper.

Table 4.3: Top-1 accuracy and complexity (GFLOPs) w.r.t. the length of extracted events

length (ms)	Accuracy	GFLOPs
10	0.528	0.31
30	0.630	0.39
50	0.613	0.92
70	0.625	1.27

Depth of Graph Convolution Layers: As to the architecture of graph convolution networks, experimental studies by Li *et al.* [156] show that the model performance saturates or even drops when increasing the number of layers beyond a certain point, since graph convolution essentially pushes representations of adjacent nodes closer to each other. Therefore, the choice of depth of graph convolution layers (D) affects the model performance as well as its size and its complexity. In the following experiment, we tested various depths from 2 to 6, each followed by a max pooling layer, and subsequently concluding the architecture with two fully connected layers. The number of output channels (C_{out}) in each convolution layer and the cluster size ($[s_h, s_w]$) in each pooling layers were as follows: *(i)* $D = 2$: $C_{out} = (128, 256)$, $[s_h, s_w] = (16 \times 12, 60 \times 45)$; *(ii)*

D = 3: $C_{out} = (64, 128, 256)$, $[s_h, s_w] = (8 \times 6, 16 \times 12, 60 \times 45)$; *(iii)* D = 4: $C_{out} = (64, 128, 256, 512)$, $[s_h, s_w] = (4 \times 3, 16 \times 12, 30 \times 23, 60 \times 45)$; *(iv)* D = 5: $C_{out} = (64, 128, 256, 512, 512)$, $[s_h, s_w] = (4 \times 3, 8 \times 6, 16 \times 12, 30 \times 23, 60 \times 45)$; *(v)* D = 6: $C_{out} = (64, 128, 256, 512, 512, 512)$, $[s_h, s_w] = (2 \times 2, 4 \times 3, 8 \times 6, 16 \times 12, 30 \times 23, 60 \times 45)$. For all cases, the number of output channels of the two fully connected layers were 1024 and 101 respectively. The results are show in Table 4.4: while the highest accuracy is obtained when the depth is 5, complexity (GFLOPs) and size (MB) of the network is substantially increased in comparison to D = 4. Therefore, in our paper, we set the depth of graph convolution layer to D = 4.

Table 4.4: Top-1 accuracy, complexity (GFLOPs) and size (MB) of networks w.r.t. depth of convolution layer.

Depth	Accuracy	GFLOPs	Size (MB)
2	0.514	0.11	5.53
3	0.587	0.16	6.31
4	0.630	0.39	18.81
5	0.634	1.05	43.81
6	0.615	2.99	68.81

Kernel Size: Kernel size determines how many neighboring nodes' features are aggregated into the output node. This comprises a tradeoff between model size and accuracy. Unlike conventional convolution, the number of FLOPs needed is independent of the kernel size. This is due to the local support property of the B-spline basis functions [3]. Therefore we only report the accuracy and model size with respect to various kernel sizes. In this comparison, the architecture is the same as the G-CNNs in Section 5.1, with the only difference being that the kernel size is increasing between 2 to 6. The results are shown in the Table 4.5. When kernel size is set as 3, 4, 5 and 6, the networks achieve the comparable accuracy, while the size of network increases significantly when the kernel size increases. In our work, we set kernel size in the graph convolution to 5, due to the slightly higher accuracy it achieves. It is important to note that, even with a kernel size of 5 that incurs a larger-size model in comparison to size of 3, our approach is still substantially less complex than conventional deep CNNs, as shown in Table 4.7 in the following section.

Table 4.5: Top-1 accuracy and size (MB) of networks w.r.t. kernel size

Kernel size	Accuracy	Size (MB)
2	0.543	5.02
3	0.626	8.30
4	0.621	12.90
5	0.630	18.81
6	0.627	26.02

Table 4.6: Accuracy/GFLOPs of networks w.r.t. input size on N-Caltech101, for conventional deep CNNs with event image inputs.

Input Size	VGG_19	Inception_V4	ResNet_50
224×224	0.549/19.63	0.578/9.24	0.637/3.87
112×112	0.457/4.93	0.4272/1.63	0.595/1.02
56×56	0.300/1.29	0.343/0.22	0.517/0.28
G-CNNs	0.630/0.39	RG-CNNs	0.657/0.79

Input Size for Deep CNNs: We investigate how the input size controls the tradeoff between accuracy and complexity for conventional deep CNNs trained on event images. We follow the training protocol and event image construction described in Section 5.2 of the paper, but now downsize the event image inputs to various resolutions prior to processing with the reference networks. The accuracy and complexity (GFLOPs) is reported on N-Caltech101 in Table 4.6. ResNet-50 offers the highest accuracy/GFLOPs tradeoff for conventional CNNs, ranging from 0.637/3.87 to 0.517/0.28. However, our RG-CNN trained on graph inputs surpasses accuracy of ResNet-50 for all resolutions, whilst offering comparable complexity (0.79 GFLOPs).

4.4.2 Comparison to the State-of-the-Art

In our experiments, the datasets of Fig. 4.6 are used to validate our algorithm. 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 [35]: 20% of the data is randomly selected for testing and the remaining is used for training. For each sample, we randomly extract a single 30-millisecond time window of events, as input to our object classification framework. 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, weighted parameters α and β are set to 1 and 0.5×10^{-5}, respectively, and the maximal connectivity degree $D_{\max}$ for each node is 32. As to the architecture of graph convolution networks, we choose two residual graph blocks for simple datasets N-MNIST and MNIST-DVS ($L = 2$). The architecture of our network for these datasets is $\text{Conv}(1,32) \longrightarrow \text{MaxP}(32) \longrightarrow \text{Res}_\text{g}(32,64) \longrightarrow \text{MaxP}(64) \longrightarrow \text{Res}_\text{g}(64,128) \longrightarrow \text{MaxP}(128) \longrightarrow \text{FC}(128) \longrightarrow \text{FC}(Q)$, with Q is the number of classes of each dataset, and the cluster size in each pooling layer is 2×2, 4×4 and 7×7, respectively. For the remaining datasets, three residual graph blocks (L=3) are used, and the utilized network architecture is $\text{Conv}(1,64) \longrightarrow \text{MaxP}(64) \longrightarrow \text{Res}_\text{g}(64,128) \longrightarrow \text{MaxP}(128) \longrightarrow \text{Res}_\text{g}(128,256) \longrightarrow \text{MaxP}(256) \longrightarrow \text{Res}_\text{g}(256,512) \longrightarrow \text{MaxP}(512) \longrightarrow \text{FC}(1024) \longrightarrow \text{FC}(Q)$. 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 in two categories; *(i)* N-Caltech101 and ASL-DVS: 4×3, 16×12, 30×23 and 60×45; *(ii)* CIFAR10-DVS and N-CARS: 4×4, 6×6, 20×20 and 32×32. We also compare the proposed residual graph networks (RG-CNNs) with their corresponding plain graph networks (G-CNNs) that stacked the same number of graph convolutional and pooling layers. The degree of B-spline bases m of all convolutions in this work is set to 1.

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 for 150 epochs, with batch size of 64 and learning rate of 0.001. The learning rate is decayed by a factor of 0.1 after 60 and 110 epochs.

To compare with the state-of-the-arts, we compute the accuracy of classification accordingly. We compare Top-1 classification accuracy obtained from our

Table 4.7: Top-1 acccuracy of our CNNs w.r.t. the state of the art & other graph convolution networks.

Model	N-MNIST	MNIST-DVS	N-Caltech101
H-First [33]	0.712	0.595	0.054
HOTS [147]	0.808	0.803	0.210
Gabor-SNN [42, 157]	0.837	0.824	0.196
HATS [35]	**0.991**	0.984	0.642
G-CNNs (this work)	0.985	0.974	0.630
RG-CNNs (this work)	0.990	**0.986**	**0.657**
Model	CIFAR10-DVS	N-CARS	ASL-DVS
H-First [33]	0.077	0.561	-
HOTS [147]	0.271	0.624	-
Gabor-SNN [42, 157]	0.245	0.789	-
HATS [35]	0.524	0.902	-
G-CNNs (this work)	0.515	0.902	0.875
RG-CNNs (this work)	**0.540**	**0.914**	**0.901**

model with that from HOTS [147], H-First [33], SNN [42, 157] and HATS [35]. HOTS relies on a time-oriented approach to extract spatio-temporal features from the asynchronously acquired dynamics of a visual scene, which describes the recent time history of events in the spatial neighborhood of an event. Specifically, HOTS [147] considers the times of most recent events with the same polarity in the spatial neighbourhood and extracts a spatial receptive field, allowing to build the event-context, then exponential decay kernels are applied to the obtained values to constitute the time-surface. H-First [33] takes advantage of timing information provided by AER sensors and uses spike timing to encode the strength of neuron activation, with stronger activated neurons spiking earlier. This enables to implement a MAX operation using a simple temporal Winner-Take-All rather than performing a synchronous MAX operation. Inspired by time-surfaces, HATS [35] is 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.

For SNN, the results are previously published, while for HOTS, H-First and HATS, we report results from Sironi [35], since we use the same training and testing methodology. The results are shown in Table 4.7. On five out of the six evaluated datasets, our proposed RG-CNNs consistently outperform these methods and sets

a new state-of-the-art, achieving near-perfect classification on smaller datasets, N-MNIST and MNIST-DVS.

4.4.3 Comparison to Other Graph Convolution

Graph convolution generalizes the traditional convolutional operator to the graph domain. Similar to frame-based convolution, graph convolution has two types [158]: spectral and spatial. Spectral convolution [89–93] defines the convolution operator by decomposing a graph in the spectral domain and then applying a spectral filter on the spectral components. Spatial convolution [3, 96–98] aggregates a new feature vector for each vertex using its neighborhood information weighted by a trainable kernel function. Similar to conventional frame-based convolution, spatial convolution operations on graphs are also an one-to-one mapping between kernel function and neighbors at relative positions w.r.t. the central node of the convolution. In this experiment, we compare the performance with respect to different graph convolution operation. Here we consider four other graph convolution operations: GCN [87], ChebConv [89], MoNet [98] and GIN [159].

ChebConv is a spectral graph convolution operation. As we illustrated before, spectral convolution [89–93] defines the convolution operator by decomposing a graph in the spectral domain and then applying a spectral filter on the spectral components. Therefore, we have the following useful property that if the spectral filter is an order of K polynomial, it is exactly K-hop localized in the spatial domain. Defferrard et al. [89] exploited this property and designed localized filters of the form of polynomial parametrization, however, the computational complexity is still high because of the multiplication with the Fourier basis. Thus, designed the spectral filter using Chebyshev polynomial so that filter can thus be parametrized as the truncated expansion, which largely reduce the computational complexity. GCN [87] is proposed by Kipf and Welling who simplified the Chebyshev polynomial of order K to the linear form. Such a model can alleviate the problem of overfitting on local neighborhood structures for graphs with very wide node degree distributions, Additionally, this layer-wise linear formulation allows to build deeper models with a fixed computational budget. MoNet [98] is a Spatial Graph Convolution opera-

tion, which generalises the previous spatial domain framework on non-Euclidean domains by introducing a local system of d-dimensional pseudo-coordinates and defines parametric kernels using Gaussian kernel instead of using fixed kernel constructions. GIN characterizes the representational capacity of GNNs via a slightly weaker criterion: a powerful heuristic called Weisfeiler-Lehman (WL) graph isomorphism test; hence enables GNNs to not only discriminate different structures, but also to learn to map similar graph structures to similar embeddings and capture dependencies between graph structures [159]. Capturing structural similarity of the node labels is helpful for generalization particularly when the co-occurrence is sparse across different graphs or there are noisy edges and node features.

Table 4.8: Top-1 acccuracy of our CNNs w.r.t. the state of the art & other graph convolution networks.

Model	N-MNIST	MNIST-DVS	N-Caltech101
GIN [159]	0.754	0.719	0.476
ChebConv [89]	0.949	0.935	0.524
GCN [87]	0.781	0.737	0.530
MoNet [98]	0.965	0.976	0.571
G-CNNs (this work)	0.985	0.974	0.630
RG-CNNs (this work)	0.990	**0.986**	**0.657**
Model	CIFAR10-DVS	N-CARS	ASL-DVS
GIN [159]	0.423	0.846	0.514
ChebConv [89]	0.452	0.855	0.317
GCN [87]	0.418	0.827	0.811
MoNet [98]	0.476	0.854	0.867
G-CNNs (this work)	0.515	0.902	0.875
RG-CNNs (this work)	**0.540**	**0.914**	**0.901**

Table 4.8 includes the classification results stemming from other graph convolutional networks. The architectures of all control networks are the same as our plain graph networks (G-CNNs) in this section, with the only difference being the graph convolutional operation. The training details and data augmentation methods are the same as illustrated before. The classification accuracy stemming from all networks of Table 4.8 indicates that our proposed RG-CNN and G-CNN outperform all other graph convolutional networks.

4.4.4 Comparison to Deep CNNs

In order to further validate our proposal, we compare our results with conventional deep convolutional networks trained on event-based frames. We train/evaluate on three well-established CNNs; namely, VGG_19 [160], Inception_V4 [161] and ResNet_50 [5]. Given that the format of the required input for these CNNs is frame-based, we group neuromorphic spike events to frame form over a random time segment of 30ms, similar to the grouping images of Zhu [38]. The two-channel event images have the same resolution as the NVS sensor, with each channel encoding the number of positive and negative events respectively at each position. To avoid overfitting, we supplement the training with heavy data augmentation: we resize the input images such that the smaller side is 256 and keep the aspect ratio, then randomly crop, flip and normalize 224×224 spatial samples of the resized frame. 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}, and the learning rate is initialized at 10^{-3} and decayed by a factor of 0.1 every 10k iterations.

The Top-1 classification accuracy of all networks is reported in Table 4.9, with the implementation of our proposed G-CNNs and RG-CNNs being the same as in Section 4.4.2. As to reference networks, 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. 4.1). However, the accuracy of our proposals surpasses that of conventional frame-based deep CNNs on nearly all datasets.

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 [162]:

$$\text{FLOPs} = 2HW(C_{\text{in}}K^2 + 1)C_{\text{out}} \tag{4.11}$$

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 [3]; *(i)* for computation of B-spline

Table 4.9: Top-1 acccuracy of our graph CNNs with graph input w.r.t. CNNs with image form input.

Model	N-MNIST	MNIST-DVS	N-Caltech101
VGG_19 [160]	0.972	0.983	0.549
Inception_V4 [161]	0.973	0.985	0.578
ResNet_50 [5]	0.984	0.982	0.637
G-CNNs (this work)	0.985	0.974	0.630
RG-CNNNs (this work)	**0.990**	**0.986**	**0.657**
Model	CIFAR10-DVS	N-CARS	ASL-DVS
VGG_19 [160]	0.334	0.728	0.806
Inception_V4 [161]	0.379	0.864	0.832
ResNet_50 [5]	**0.558**	0.903	0.886
G-CNNs (this work)	0.515	0.902	0.875
RG-CNNNs (this work)	0.540	**0.914**	**0.901**

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

$$\text{FLOPs} = N_{\text{edge}}(m+1)^d(3C_{\text{in}}C_{\text{out}}+7d)+(N_{\text{edge}}+N_{\text{node}})C_{\text{out}}$$

For fully connected layers, in both conventional CNNs and GCNs, we compute FLOPs as [162] FLOPs $= (2I-1)O$, where I is the input dimensionality and O is the output dimensionality. With regards to the number of parameters, for each convolution layer in both CNNs and GCNs, 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 (4.12), 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 4.10. G-CNNS and RG-CNNs have a smaller number of weights and require the less computation compared to deep CNNs. The main reason is that the graph

Table 4.10: Complexity (GFLOPs) and size (MB) of networks.

Model	GFLOPs	Size (MB)
VGG_19 [160]	19.63	143.65
Inception_V4 [161]	12.25	42.62
ResNet_50 [5]	3.87	25.61
G-CNNs	0.39	18.81
RG-CNNs	0.79	19.46

representation is compact, which in turn reduces the amount of data needed to be processed. For N-Caltech101, the average number of nodes of each graph is 1,000, while grouping events into a 2-channel image makes the input size equal to 86,400.

4.5 Conclusion

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, object classification with NVS streams cannot leverage on state-of-the-art convolutional neural networks (CNNs), since NVS does not produce frame representations. To circumvent this mismatch between sensing and processing with CNNs, we propose a compact graph representation for NVS, which allows for condensed representations, and in turn allow for end-to-end task training and fast post-processing that matches the compact and non-uniform sampling of NVS hardware. We couple this with novel residual graph CNN architectures and show that, when trained on spatio-temporal NVS data for object classification, such residual graph CNNs preserve the spatial and temporal coherence of spike events, while requiring less computation and memory. Finally, to address the absence of large real-world NVS datasets for complex recognition tasks, we present and make available a 100k dataset of NVS recordings of the American sign language letters, acquired with an iniLabs DAVIS240c device under real-world conditions.

Chapter 5

Spatio-Temporal Feature Learning for Neuromorphic Vision Sensing

5.1 Introduction

Beyond event sparsity and asynchronicity, neuromorphic event streams are naturally encoding spatio-temporal motion information [144]; as such, they are extremely adaptable to tasks related to moving objects such as action analysis/recognition, object tracking or high-speed moving scenes. As an illustration, Fig. 5.1 shows a neuromorphic event stream, overlaid with the corresponding RGB frames recorded at the video frame rate; 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 frame rate achievable with an NVS device and asynchronous outputs. Furthermore, the asynchronicity removes the data redundancy from the scene, which reduces to the power requirement to 10mW, compared to several hundreds of mW for APS sensors. Remarkably, NVS devices achieve this with microsecond-level latency and robustness to uncontrolled lighting conditions as no synchronous global shutter is used. We therefore look to perform feature learning directly on the raw neuromorphic events. This is in contrast to recent work on action analysis/recognition [163], which relies on extracting spatio-temporal features from RGB frames and thus inherits the limitations associated with APS cameras.

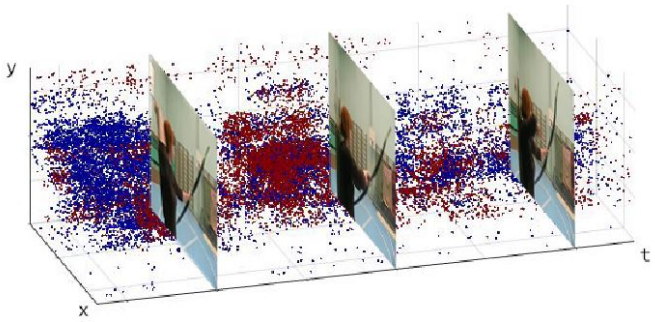


Figure 5.1: Examples of archery action captured by APS and NVS devices. APS devices capture images at fixed frame rates, while NVS devices output a stream of events. (Red:ON, Blue:OFF)

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 a limited research in the NVS domain, as well as a lack of NVS data with reliable annotations to train and test on [21, 144, 146]. Yet, more so, the sheer abundance of asynchronous and sparse events means that feature learning directly on events can be particularly cumbersome and unwieldy. Much of the recent success of computer vision comes from the definition of robust and invariant feature or interest point extractors and descriptors. Thus far, most approaches have attempted to solve this issue by either artificially grouping events into frame forms [4, 38–40] or deriving complex feature descriptors [35, 72, 147], which do not always provide for good representations for complex tasks like action recognition. 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 and maintain a representation of the feature dynamics over time, thus rendering them less viable for action recognition and action similarity-based tasks. More recent methods have employed end-to-end feature learning, where a convolutional neural network (CNN) [32, 76] or spiking neural network (SNN) [79, 80] is trained to learn directly from raw observations. While these methods show great promise, CNN-based learning methods require event grouping into frames and therefore suffer from the same drawbacks as above. On the other hand, SNN-based methods are

complex to train, which results in lower performance compared to gradient-based alternatives.

In this chapter, we propose an end-to-end feature learning framework trained directly on neuromorphic events. Instead of using CNNs or SNNs, we propose to leverage on graph-based learning. By representing events as graphs, we are able to maintain event asynchronocity 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, and the first time neuromorphic events have been trained with graph convolution neural networks and end-to-end feature learning. Our proposed graph based framework is able to accommodate both appearance and motion-based tasks; in this paper, we focus on action recognition, action similarity labeling and scene recognition as representative tasks. For object classification in Chapter 4, we design a spatial feature learning module, comprising graph convolutional layers and graph pooling layers, for processing a single input event graph. While in this chapter 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 CNN. This is inspired by recent work in APS-based action recognition [163] 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 action recognition and action similarity labeling. We leverage on existing APS-based datasets such as UCF101 [126], HMDB51 [164], ASLAN [165] and YUPENN [166], and convert these to the NVS domain by recording the display with an NVS camera. The generated NVS datasets, UCF101-DVS, HMDB51-DVS and ASLAN-DVS, represent the largest NVS datasets for

human action. We evaluate our framework on these three tasks in Sec. 5.3 and show that our framework achieves state-of-the-art results on both tasks compared to recent work or conventional frame-based approaches.

5.2 Methodology

The architecture for our graph-based spatio-temporal feature learning network is shown in the Fig. 5.2, and it consists of four parts: graph construction and sampling, spatial feature learning module, Graph2Grid module and temporal feature learning module. The neuromorphic events are firstly sampled and connected with a sequence of graphs. For object classification, a single graph is typically constructed, whereas for action recognition with longer temporal extent, multiple graphs are extracted over the event stream duration. The 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 our Graph2Grid module 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.

5.2.1 Sampling and Graphs Construction

To maintain a representation of the feature dynamics over time, We introduce the parameter S to represent the number of graphs constructed from one sample. Given that an application, such as object classification in Chapter 4, focuses on appearance-based feature 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, in this chapter, we are exploting the both spatial and temporal feature learning, thus we divide the event stream into S vol-

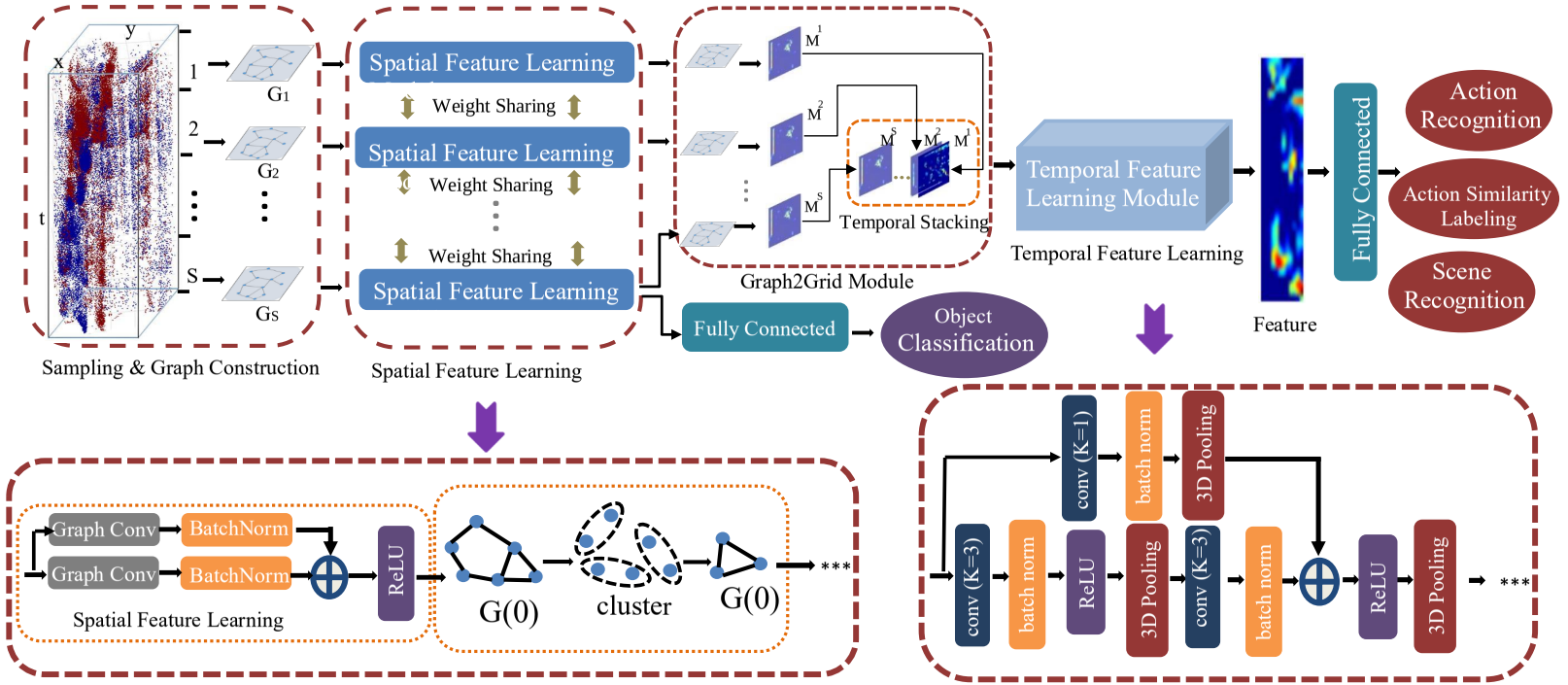


Figure 5.2: Framework of graph-based action recognition for neuromorphic vision sensing. Our framework is able to accommodate both object classification and action recognition tasks. We first construct S graphs from the event stream (where $S = 1$ for object classification), and each graph is passed through a spatial feature learning module, comprising graph convolutional and pooling layers. For object classification, the output of this module is mapped to object classes directly by fully connected layers. For action recognition, action similarity labeling and scene recognition, 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 conventional 3D CNN, before mapping features to action classes with fully connected layers.

umes with the same time duration, then we construct a graph for each volume in which T_{vol} ($T_{\text{vol}} < T/S$, where T is sample duration) length events are randomly extracted to construct a graph, giving us a set of graphs $\mathscr{G} = \{\mathrm{G}_n\}_{n=1}^{S}$. 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 [163], 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. As to the graph construction progress for each volume, we follow the technology in 4.2.1 for implementation. Specifically, extracted events are firstly sampled by using non-uniform sampling that largely reduces the amount of data for post-processing and allows for a more computation-efficient system; then sampled events are regarded as nodes of graph and we define the connectivity of nodes using the radius-neighborhood-graph strategy as illustrated in 4.2; finally the polarity of events are regarded as feature that is inside each node. We just make simple description here and for more details please refer to the Section 4.2.1.

5.2.2 Spatial Feature Learning Module

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

Inspired by the ResNet architecture [5], 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 [156]. Especially, graph CNNs easily suffer from the problem of over-smoothing; this is,

when the networks go deeper, the feature in all nodes turn to be the same. This largely degrade the performance of model. Our residual graph CNN (RG-CNN) is effectively composed of a series of residual blocks and pooling layers. Considering equations (4.3) and (4.7) denote a single graph convolutional layer with batch normalization [153] 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. A residual block is illustrated at the bottom right of Fig. 4.2. We denote the resulting graph residual block as $\mathrm{Res_g}(\mathrm{c_{in}}, \mathrm{c_{out}})$, 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 $(x_i', y_i') \in \mathbb{R}^{H' \times W'}$ 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.

5.2.3 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 5.2.1, given a long video 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_V. 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. 4.2. In this way. we are effectively able to create pseudo frames from the graphs, with M_{in} channels and timestamp $(n-1)T_V$, corresponding with 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 $(x_i', y_i') \in \mathbb{R}^{H_{\text{spatial}} \times W_{\text{spatial}}}$, we define a grid representation $\mathbf{f}^{\text{grid}}(i)$ of spatial size $H_{\text{spatial}} \times W_{\text{spatial}}$ as follows:

$$f_{a,b,l}^{\text{grid}} = \begin{cases} f_l^{\text{spatial}}(i), & \text{when } a = x_i', b = y_i' \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

where $(a,b) \in \mathbb{R}^{H_{\text{spatial}} \times W_{\text{spatial}}}$. 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 simple concatenate over a fourth temporal dimension. We denote the resulting grid feature over S graphs as $\mathbf{F}^{\text{grid}} = \mathbf{f}^{\text{grid},1} || \mathbf{f}^{\text{grid},2} || \ldots || \mathbf{f}^{\text{grid},S}$, where $||$ denotes concatenation over the temporal axis. Thus, the dimensions of $\mathbf{F}^{\text{grid}}$ is thus $H_{\text{spatial}} \times W_{\text{spatial}} \times M_{\text{in}} \times S$. 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. The key reason why we start from events to graph, then back to grid feature matrix is that we do need to construct multiple graphs to cover the long-temporal extent and meanwhile it is difficult to learn temporal dependencies from multiple various graphs. One advantage is that constructed grid feature matrix has the much smaller resolution compared to the original spatial resolution so that we still can effectively learn the temporal feature as we discussed in the end of section 5.2.4.

5.2.4 Temporal Feature Learning Module

The output feature matrix $\mathbf{F}^{\text{grid}}$ contains both spatial and temporal information over the entire video duration, which can be effectively encoded with a conventional 3D CNN [135] in order to generate a final spatio-temporal representation of the video input for action recognition. In this work, we consider three network ar-

chitectures 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. Our 3D residual block design is illustrated in Fig. 4.2; 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 $c \in \{3,5\ldots C-2,C\}$, 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 5.3, describing 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.

5.3 Three Applications of Spatial-Temporal Feature Learning

In this section, we demonstrate the potential applications of our framework as a method of representation learning for high-level computer vision tasks with NVS inputs. We firstly focus on the large-scale multi-class human action recognition in Section 5.3.1, then turn to action similarity labeling in Section 5.3.2, finally move to scene recognition in Section 5.3.3.

5.3.1 Human Action Recognition

The understanding and recognition of human action have gained a substantial research interest among the computer vision community in last decades which can be applied in many areas including intelligent surveillance, human behavior analysis, and so on. For human action recognition, appearances and dynamics are crucial

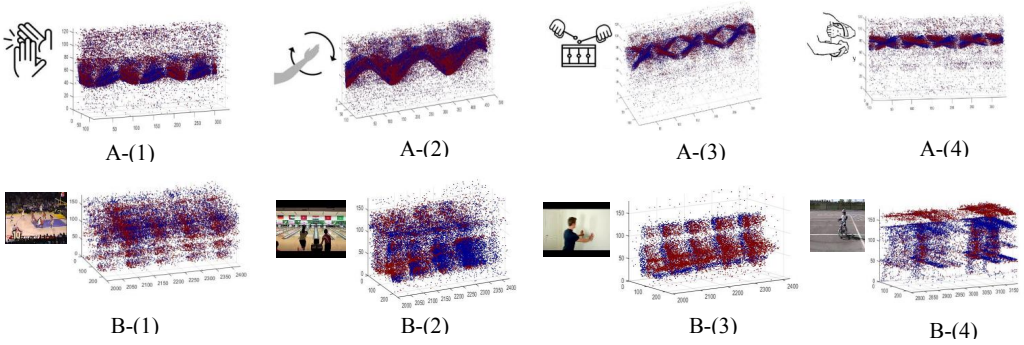


Figure 5.3: Visualization of samples from DVS128 Gesture Dataset and UCF101-DVS. (A) DVS128 Gesture Dataset: A-1: hand clap; A-2: right hand rotation clockwise; A-3: air drums; A-4: forearm roll. (B) UCF101-DVS: B-1: basketball dunk; B-2: bowling; B-3: wall pushups; B-4: biking

and complementary aspects, therefore proposed spatial-temporal feature learning framework can be seen as a way to model the human action.

Datasets: Most action recognition methods for neurmorphic vision sensing are tested on the DVS128 Gesture Dataset [32] and posture dataset [85]. The DVS128 Gesture Dataset comprises 1,342 instances of a set of 11 hand and arm gestures and posture dataset includes only three human actions; namely, consisting of 191 bend, 175 sitstand and 118 walk actions. Both of them are collected from experimental setting environment with clean background. These datasets are simple and have limited number in both size and class; as such, they cannot represent complex real-life scenarios and are not robust to evaluation for advanced algorithms. Moreover, while useful for early-stage attempts, algorithms [2, 32, 85, 86] evaluated on these datasets (including our framework) already achieve high accuracy. Therefore, it is necessary to establish larger and more complex datasets for algorithm evaluation in the NVS domain.

For the purpose of evaluating APS-based solutions, UCF101 [126] and HMDB51 [164] are widely used to evaluate the performance of the algorithms. Specifically, UCF-101 has about 13,300 videos with 101 different human actions and HMDB51 has about 6,600 videos with 51 human action categories. We therefore propose to convert these datasets to the neuromorphic domain; this requires recording the APS videos with an NVS camera. Recent work by Hu *et al.* [113] recorded UCF51, by displaying existing benchmark videos on a monitor and record-

ing with a stationary a neuromorphic vision sensor under controlled lighting conditions; however UCF51 only represents a small subset of UCF101 over 51 classes. Similarly, we follow the same recording protocol of [113] and record the *remaining* of UCF101 and HMDB51. Each video is displayed by a monitor that is set to its highest brightness and contrast setting. The display is recorded by a neuromorphic vision sensor DAVIS240c that is adjusted to cover the region of interest on the monitor. The recording is set in a dark room where only the monitor is the light source. In this way, we generate the largest neuromorphic datasets available for action recognition, which also correspond with the standard datasets evaluated on in the APS domain. We refer to these NVS datasets as UCF101-DVS and HMDB51-DVS respectively. These datasets will be released to the public domain as a contribution of the thesis.

Implementation Details: We select the number of graphs S constructed from the event stream from the set $\{8,16\}$, and present our results for both settings in the Table 5.1 and 5.2. For each volume, events within $T_{\text{vol}} = 1/30$ seconds are constructed into one spatial graph, and each node in the graph is connected to its nearest node. We utilize our proposed residual graph CNNs (RG-CNN) for the spatial feature learning module. For the DVS128 gesture dataset, only two residual blocks are stacked, each followed by a graph max-pooling layer, and for this module we use the architecture: $\text{Res}_{\text{g}}(1,64) \longrightarrow \text{MaxP}_{\text{g}}(2,2) \longrightarrow \text{Res}_{\text{g}}(64,128) \longrightarrow \text{MaxP}_{\text{g}}(4,4)$. Conversely, and for UCF101-DVS and HMDB51-DVS, three residual blocks are used 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 architecture as described in Section 5.2.4:

- *Plain 3D:* For plain 3D CNN, we 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 refer to a traditional 3D convolution layer with batch normalization and activation function as $\text{Conv}_{\text{3D}}(c_{\text{in}}, c_{\text{out}})$, 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 layer as FC and number of task classes as Q. The plain 3D convolution architecture can thus be represented as follows (assuming 128 input channels after the Graph2Grid block): $\text{Conv}_{3D}(128,128) \longrightarrow$ Pool3D $\longrightarrow \text{Conv}_{3D}(128,256) \longrightarrow$ Pool3D $\longrightarrow \text{Conv}_{3D}(256,512) \longrightarrow$ Pool3D $\longrightarrow \text{Conv}_{3D}(512,512) \longrightarrow$ Pool3D $\longrightarrow$ GlobAvgP $\longrightarrow$ FC(Q). With 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 in 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 there is not too aggressive a temporal downscaling early on).

- *Inception-3D(4):* We additionally consider an Inception-3D based architecture, comprising a series of four consecutive I3D blocks. In order to ensure that the temporal feature learning does not become a bottleneck, we restrict the number of I3D blocks to four. Our implementation of the I3D block is a concatenation of four streams of convolution layers with varying kernel size, and matches that of Carreira *et al.* [6]. 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 the following: $\text{Inc}_1(128,480) \longrightarrow$ Pool3D $\longrightarrow \text{Inc}_2(480,512) \longrightarrow$ Pool3D $\longrightarrow \text{Inc}_3(512,512) \longrightarrow$ Pool3D $\longrightarrow \text{Inc}_4(512,512) \longrightarrow$ Pool3D $\longrightarrow$ GlobAvgP $\longrightarrow$ 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: Inc_1=[[128],[128,192],[32,96],64], Inc_2=[[192],[96,208],[16,48],64], Inc_3=[[160],[112,224],[24,64],64], Inc_4=[[128],[128,256],[24,64],64].

- *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. 4.2; essentially, there are

two 3D convolutional layers in the base stream of the block, with a 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 $\text{Res}(c_{\text{in}}, c_{\text{inter}}, c_{\text{out}})$, 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 number of input and output channels respectively to the residual block. The 3D residual CNN is defined as follows: $\text{Res}(128, 256, 512) \longrightarrow \text{Pool3D} \longrightarrow \text{Res}(512, 512, 1024) \longrightarrow \text{Pool3D} \longrightarrow \text{GlobAvgP} \longrightarrow \text{FC}(Q)$. 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, and all 3D max_pooling layers are as defined for the plain 3D CNN.

Sampled graphs are spatially scaled by a randomly sampled factor within $(0.8, 1)$ and randomly left-right flipped with probability 0.5. We use the predefined training and test set for DVS128 Gesture Dataset and for UCF101-DVS and HMDB51-DVS, we use the training/test splits (standard 'Split1') defined for their APS counterparts (UCF101 and HMDB51). 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 and 100 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 with reference networks from the APS video domain repurposed for the NVS domain. Here, we include C3D [135], I3D [6], 3D ResNet with 34 layers [167], P3D with 63 layers [168], R2+1D [169] and 3D ResNext with 50 layers [7]. In contrast to our framework, these networks are entirely grid-based and require artificial grouping of events into frame form. Therefore, to feed these networks we follow a similar approach to Chadha *et al.* [77], and construct a single frame by summing events within a 1/30s duration at each spatial position of the NVS devices. The resulting event frame has two channels, as ON and OFF events are grouped independently.

We generate $S = 8$ or $S = 16$ frames from event volumes, in order to align with the number of input graphs utilized in our framework. To avoid over-fitting during training, we supplement the training with data augmentation: first, 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, then use a random cropping of 112×112 (160×160 for P3D, 224×224 for I3D) spatial samples of the re-sized frame, finally the cropped volume is randomly left-right flipped. We train all models from scratch using stochastic gradient descent with momentum set to 0.9, and the learning rate is initialized at 0.01 and decayed by a 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-the-art methods and reference networks. The results are shown in Table 5.1, and for all recent methods, considered event recording duration is 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 see that the LSTM-based method [170] is outperformed by other methods. We attribute this to the fact that the LSTM method regards event streams as pure temporal sequences and only learns the temporal features from the events, without encoding spatial dependencies. On the contrary, PointNet-based methods [2, 83, 171] take the input as a point cloud and learn to summarize the geometric features, which boosts accuracy. With regards to reference networks, although I3D [6] and 3D ResNet_34 [167] 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.

As shown in Fig. 5.3, DVS128 Gesture Dataset is too simple since we can see evident pattern difference, while UCF101-DVS is complex events volumes. And also, as shown in Table 5.1, the results on DVS128 Gesture Dataset are already close to perfect accuracy. Therefore, we further evaluate our algorithms on our newly

Table 5.1: Top-1 classification accuracy of DVS128 Gesture Dataset.

Method	Duration(0.25s)	Duration(0.5s)
LSTM [170]	0.882	0.865
PointNet [83]	0.887	0.902
PointNet++ [171]	0.923	0.941
Amir CVPR2017 [32]	-	0.945
Wang WACV2019 [2]	0.940	0.953
ResNet_34 [167]	0.943	0.955
I3D [6]	0.951	0.965
RG-CNN + Plain 3D	0.954	0.968
RG-CNN + Incep. 3D(4)	0.957	0.968
RG-CNN + Res. 3D	**0.961**	**0.972**

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 networks is reported in Table 5.2 for UCF101-DVS and HMDB51-DVS. We again present results on our framework for Plain 3D, Inecption-3D(4) and Residual 3D variants of our temporal feature learning module and compare directly with reference networks. As is evident, the reference networks are outperformed by our variants of our model. Specifically, the highest performance obtained from reference models is from I3D, while our base model (RG-CNN + Plain 3D) outperforms I3D by 3.3% and 6.1% in terms of UCF101-DVS and HMDB51-DVS when $S = 8$ inputs constructed from the event stream, respectively. Note that when varying the temporal feature learning architecture from Plain 3D, to Inception-3D(4) and Residual 3D, our model performance increases slightly, due to the higher capacity of these architectures.

Complexity Analysis: We compare the complexity of our proposed learning framework against external benchmarks, and do so with respect to the number of floating-point operations (FLOPs) and required parameter counts. For graph convo-

Table 5.2: 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 [135]	0.382	0.472	0.342	0.417
ResNet_34 [167]	0.513	0.579	0.350	0.438
P3D_63 [168]	0.484	0.534	0.343	0.404
R2+1D_36 [169]	0.496	0.628	0.312	0.419
ResNext_50 [7]	0.515	0.602	0.317	0.394
I3D [6]	0.596	0.635	0.386	0.466
RG-CNN + Plain ó6D	0.629	0.663	0.447	0.494
RG-CNN + Incep. 3D(4)	**0.632**	**0.678**	0.452	**0.515**
RG-CNN + Res. 3D	0.627	0.673	**0.455**	0.497

lutional and fully-connected layers, FLOPs and parameter numbers are computed as detailed in Section 4.4.4. For conventional 3D convolution, we compute FLOPs as $2HWT(C_{\text{in}}K^3+1)C_{\text{out}}$, where H, W, and T are the height, width, and temporal length, 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 accounts of conventional 3D convolution are expressed as $(C_{\text{in}}K^3+1)C_{\text{out}}$. Since FLOPs of graph convolutions depend on edge and node counts (see Section 4.4.4), we report exemplar results for UCF101-DVS in Table 5.3. 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 comparable size input volumes compared to all external benchmarks. As for complexity, though our models require more floating-point operations when compared to P3D-63 and ResNext-50, graph convolutions achieve better performance in all three datasets. On the other hand, accuracies of I3D are close to ours while requiring complexities which are two to three times higher.

5.3.2 Action Similarity Labeling

Action similarity detection is a binary classification task wherein predictions are made about the alignment of action pairs. In other words, models are required to learn to evaluate the similarity of actions rather than recognize particular actions.

Table 5.3: Comparison of networks w.r.t. complexity (GFLOPs) and size (MB) of networks.

Model	FLOPs($\times 10^9$)	#params($\times 10^6$)
C3D [135]	39.69	78.41
ResNet_34 [167]	11.64	63.70
P3D_63 [168]	8.30	25.74
R2+1D_36 [169]	41.77	33.22
ResNext_50 [7]	6.46	26.05
I3D [6]	30.11	12.37
RG-CNN + Plain 3D	12.46	6.95
RG-CNN + Incep. 3D(4)	12.39	3.86
RG-CNN + Res. 3D	13.72	12.43

The challenge of action similarity labeling lies in that the actions of the test set belong to separate classes and are not available during training [165]. That is to say, training does not provide an opportunity to learn models actions presented at test time. To the best of our knowledge, as of yet there is no work on action similarity classification in neuromorphic domain, and no existing dataset can be used for action similarity evaluation. We use the ASLAN [165] dataset which comprises 3,697 samples from 432 different action classes. Using the same experiment setting to the one described in Section IV-B, we captured ASLAN-DVS to be publicly provisioned for relevant research.

Training Details: We use the 'View-2' method as detailed in [165] 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 on 10 separate experiments in a leave-one-out cross validation scheme. In this application, we use models trained for action recognition as a feature extractor, and constructed 16 graphs from one sample to pass to the feature extractor. Specifically, we extract outputs as L_2-normalised features from the last *GlobalAvgP* and *Pool3D* layers as two types of features. Following the same setup as in [165], and we independently compute 12 different distances as similarity index as shown in Table 5.4 for every a input pair. 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

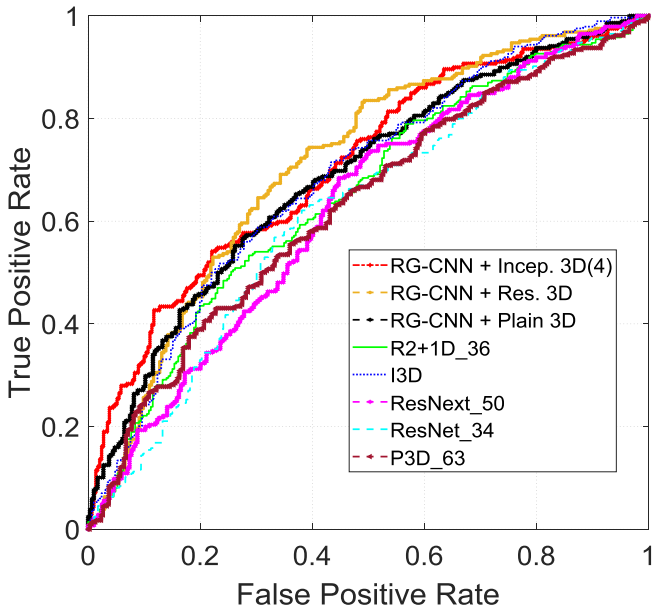


Figure 5.4: Action similarity labeling result. ROC curves of proposed and reference networks evaluated on ASLAN-DVS.

performance of the external benchmarks detailed in Section 5.3.1, where features are extracted as the outputs of the last two layers, and classifications are performed by support vector machines.

Table 5.4: 12 equations used for similarity computation

No.	Equation	No.	Equation
1	$\sum x_1 .* x_2$	7	$\sum x_1 .* x_2 / (\sqrt{(\sum x_1^2)} * (\sqrt{(\sum x_1^2)})$
2	$\sqrt{\sum x_1 .* x_2}$	8	$\sqrt{\sum (\sqrt{x_1} - \sqrt{x_2})^2}$
3	$\sqrt{\sum \sqrt{x_1} .* \sqrt{x_2}}$	9	$\sqrt{\sum (x_1 - x_2)^2}$
4	$\sqrt{\sum \frac{x_1 .* x_2}{x_1 + x_2}}$	10	$(\sum x_1 log\frac{x_1}{x_2} + \sum x_2 log\frac{x_2}{x_1})/2$
5	$\sum \lvert x_1 - x_2 \rvert$	11	$\sum \frac{\sqrt{(max(x_1, x_2))}}{\sqrt{(x_1 + x_2)}}$
6	$\sqrt{\sum \frac{(x_1 - x_2)^2}{x_1 + x_2}}$	12	$\sum min(x_1, x_2) / \sum x_1 \sum x_2$

Results: We reported the results including accuracy (Acc.) and area under ROC curve (AUC) and compare with the reference models in Table 5.5. Our RG-CNN + Incep. 3D(4) framework outperforms state-of-the-art acquired from I3D by 2.6% on accuracy and 3.1% on AUC. We also plot ROC curve in Fig.5.4, which clearly indicates that our graph-based feature learning methods made a improvement in this task. The complexity and computation analysis of our proposed spatio-

Table 5.5: Action similarity labeling result on ASLAN-DVS w.r.t. various model

Model	Acc.	AUC
ResNet_34 [167]	0.605	0.643
P3D_63 [168]	0.598	0.638
R2+1D_36 [169]	0.615	0.652
ResNext_50 [7]	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

temporal feature learning framework and reference networks are the same as in Section 5.3.1.

5.3.3 Scene Recognition

Scene recognition is a fundamental challenge that provides priors for the presence of actions, surfaces and objects, since ”scene” refers to a place where an action or event occurs [166], therefore scene recognition is one of the hallmark tasks of computer vision that allows definition of a context for object recognition.

Dataset: This is the first work on scene recognition in neuromorphic vision sensing area. In active pixel sensing area, YUPENN is widely used to evaluate the scene recognition algorithms, so we recorded YUPENN-DVS from YUPENN [166] using the same settings as described in Section 5.3.1 for UCF101-DVS and HMDB-DVS. YUPENN-DVS consists of 14 scene categories and each contains 30 samples.

Training Details: We evaluate the model on YUPENN-DVS and the training and testing procedures on YUPENN-DVS follows the standard leave-one-out evaluation protocol for YUPENN as in [166]. The parameter of architecture and reference models are the same as in 5.3.1, we also follow the training detail in that section and data augmentation as well. Similar to Section 5.3.1, we evaluate the performance with respect to various input length (8 and 16 graphs).

Results: Top-1 average accuracy and variance of YUPENN-DVS for scene recognition is reported in Table 5.6. When setting S is 8, our RG-CNN + Res. 3D framework outperforms state-of-the-art acquired from I3D by 5.6% on accuracy, setting as new benchmark. Likewise, our frameworks achieve better performance

compared to all reference networks when we set S as 16.

Table 5.6: Top-1 average recognition accuracy and variance of YUPENN-DVS w.r.t. various model

Model	S=8	S=16
ResNet_34	0.707±0.085	0.842±0.113
P3D_63	0.490±0.102	0.763±0.108
R2+1D_36	0.431±0.077	0.741±0.085
ResNext_50	0.562±0.089	0.815±0.135
I3D	0.824±0.062	0.900±0.098
RG-CNN + Plain 3D	0.873±0.072	0.946±0.120
RG-CNN + Incep. 3D(4)	0.877±0.160	0.950±0.105
RG-CNN + Res. 3D	0.880±0.060	0.945±0.091

5.4 Conclusion

In this chapter we develop an end-to-end trainable graph-based spatial temporal 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 develop three variants of network architectures for the 3D CNN; a plain architecture with interlaced 3D convolutional and pooling layers, an I3D-based architecture comprising multiple I3D blocks. We demonstrate how this framework can be employed for action recognition, action similarity labeling and scene recognition, and evaluate our framework on all tasks with proposed large-scale neuromorphic dataset, showing that our model outperforms the reference networks in alll tasks. We additionally propose and make available four 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.

Chapter 6

Concluding Remarks

In this final chapter, we summarise the main contributions presented in this thesis, and further discuss their current limitations and potential improvements. Then we share some ideas of future research to move towards improvement of emulator PIX2NVS. Finally, we suggest some practical solutions for fully exploring the spatial-temporal feature extraction for neuromorphic vision sensing, which aim at being efficient and robust enough for more difficult tasks under environment with very rapid motion and extreme lighting variation.

6.1 Summary

In this thesis, we tried to bridge the gaps between the neuromorphic vision sensing (NVS) and active pixel sensing. As NVS is a newly proposed sensing technology, the performance of computer vision related tasks is far behind its counterpart. The reasons are lack of large-scale datasets for developing robust algorithms and the limited number of work in NVS domain. Therefore, we mainly solved these two problems by proposing an emulator to generate large-scale datasets from existing video collections and developing a feature learning model to extract feature for different computer vision tasks.

We began in Chapter 3 with PIX2NVS, a parameterized conversion of pixel-domain video frames to neuromorphic vision streams, which can convert frames from APS videos to emulated neuromorphic spike events. Importantly, we can generate large annotated NVS data from existing video frame collections used in

machine learning research using PIX2NVS emulator. Then we proposed and evaluated three distance metrics to quantify the accuracy of the model-generated events against ground-truth and optimized the parameter of PIX2NVS with random search. Finally, via the conventional deep CNNs, we test emulated data on two applications including human action recognition and American sign language recognition to evaluate the effectiveness of emulated NVS data.

In Chapter 4, we moved to developing vision-related algorithm for NVS. Specifically, we designed an object classification framework for NVS that is computationally efficient. In this framework, we proposed a graph based representation for neuromorphic events to keep their sparse and asynchronous nature and we couple this representation with novel residual graph CNN architectures that efficiently preserves the spatial and temporal coherence of spike events for the object classification. Also, we present and make available a 100k dataset of NVS recordings of the American sign language letters, to address the absence of large real-world NVS datasets for complex recognition tasks.

In Chapter 5, we extended spatial feature learning framework in chapter 4 to spatial-temporal feature learning framework so that the extracted feature can be used in both appearance and motion based applications. Specifically, we proposed a Graph2Grid block and temporal feature learning module for efficiently modelling temporal dependencies on multiple graphs over a long temporal extent. Then we accommodated this end-to-end feature learning framework to both appearance and motion based tasks. Specifically, we applied this model into three different applications including action recognition, action similarity labeling and scene recognition, and followed by experimental validation showing that our proposed framework outperforms all recent methods on intensive datasets. In this chapter, we also released largest neuromorphic human action datasets and scene recognition datasets.

6.2 Future Work

Several of the proposals of this thesis can be extended in future work. We note that in Chapter 3, due to the parameter tuning with the three proposed distortion met-

rics, our approach is reproachable for a number of applications and NVS camera configurations. We can improve domain adaptation in NVS emulation even further by embedding supervision from real NVS events into the emulator framework, e.g., by using a generative adversarial network (GAN) [172]. In addition, the PIX2NVS frame grouping (as described in Section 3.5.1) currently involves aggregating spikes within the same video interval by summing their polarities. While this provides an efficient method for CNNs to ingest the NVS spike events, in doing so, we potentially lose some of the per-interval motion dependencies that would be better described by modelling the spatio-temporal displacement between the spike events. Such modelling would further enable us to model a continuous NVS streams over the video duration. We shall be pursuing these ideas and analysis in future work. Moreover, the timestamp of events is restricted by the frame rate of videos, to increase the latency of events, it is possible to assign the dynamic timestamp for each events based on the interpolation of pixels.

Chapter 4 and Chapter 5 pave the way for more efficient feature extraction for NVS events. Events are firstly represented as graph as the input of framework. We observed that the size of graph is still large even though we apply non-uniform sampling over events before graph construction. The reason is that the sampling technology is still naive so that the size of events does not decrease largely. In the future work, we may explore more advanced sampling technology that can take use of the nature of neuromorphic vision sensing data to reduce the amount of data and keep more spatial and temporal information at the same time. Moreover, it would be interesting to see if we could design a more adaptive sampling technology with respect to various sensing scenarios and objects to effectively and efficiently reduce the number of events.

As to the graph construction in Chapter 4 and Chapter 5, the design of the methods utilised the well-established radius-neighborhood graph construction method that is defined as the weighted spatio-temporal distance. This method, to some extent, is computationally cumbersome, which make them difficult to be deployed on computationally limited platforms in real time. Therefore, one direction of this

work is to place more emphasis on exploring the nature of events to construct graph effectively for efficient and real-time hardware implementations, so that we will not lose the neuromorphic sensors' advantage of low latency. Moreover, an important future direction for this work is the extension of the graph applications to the other NVS-related computer vision tasks, e.g. visual tracking, mapping and 3D reconstruction, *etc.*. As we described, graph representation is able to keep the compact, sparse and asynchronous representation, which definitely is an option for post-processing of neuromorphic events.

In general, research in NVS is still in its infancy compared to APS, and we acknowledge that more efforts are needed to develop systems with higher accuracy. Our work is indeed one such strand of efforts, both via its proposals to use graph-based representations, but also by releasing new datasets. APS-based CNN methods benefit from pre-training and initialization of CNN architectures on ImageNet and similarly-large APS datasets, the likes of which are not yet available on the NVS domain. Our work in releasing datasets also aims to close this gap. Progress in commoditizing hardware is also under way, e.g., Samsung is devoting significant R&D in this space and has released new hardware, i.e., their recent Samsung SmartThings Vision camera, which may definitely advance the progress of the NVS community. Therefore, an important direction is to use the avaliable datasets to develop robust few-shot learning methods that can learn to make reliable predictions from small datasets as the new rise of the NVS hardware.

Moreover, there is a interesting a direction to considering APS or LiDAR as complementary and combing it with NVS to acquire high performance and maintain the high efficiency. We did initial exploration in our previous work [77] in which improvements can be obtained by transfer learning between information learnt from APS to NVS representations. However, this also comes at the loss of efficiency in the sensing and representation (as frames have to be used). Therefore, we believe more is to be gained in the accuracy-complexity sense by better datasets, better NVS hardware and better ways of aggregating NVS into compact representations. Effectively, this argument is extendable to APS as well: one can extend the efficiency

of APS systems by using LiDAR cameras, but this comes at significantly higher cost and significantly increased energy consumption. That is, similar to the APS-LiDAR comparison, NVS can be seen as a significantly faster and energy-efficient sensing modality in comparison to APS, which comes, however, with some loss in the descriptive power of the obtained signal. Therefore, corresponding algorithms that are still power-efficient and effective when involving more complementary are interesting to explore.

Appendix A

Reference Networks for Object Classification

In this appendix, we introduce the details of deep reference networks used in Section 4 for object classification. In Section 4, we include three typical deep CNNs: VGG_19 [160], Inception_V4 [161] and ResNet_50 [5], and all of them have their own specific architectures.

VGG_19: VGG_19 is combined by a stack of convolutional layers and followed by three Fully-Connected (FC) layers. The input to ConvNets is a fixed-size (224×224), and is passed through a stack of convolutional layers, where the filters is with a very small receptive field (3×3). The convolution stride is fixed to 1 pixel and spatial resolution is preserved after convolution. Spatial pooling is carried out by five max-pooling layers that is performed over a 2×2 pixel window and with stride 2. The details is as following (The convolutional layer parameters are denoted as "conv(receptive field size-number of channels)": input$(224 \times 224) \rightarrow 2 \times \text{conv}(3-64) \rightarrow$ maxpool $\rightarrow 2 \times \text{conv}(3-128) \rightarrow$ maxpool $\rightarrow 4 \times \text{conv}(3-256) \rightarrow$ maxpool $\rightarrow 4 \times \text{conv}(3-512) \rightarrow$ maxpool $\rightarrow 4 \times \text{conv}(3-512) \rightarrow$ maxpool $\rightarrow 3 \times$ FC.

Inception_V4: Inception_V4 is the combination of the two commonly used architectures: residual connections introduced by He *et al.* in [5] and the latest revised version of the Inception architecture in [173]. This straightforward integration enables the network to be more deeper and wider. Figure A.1 shows the

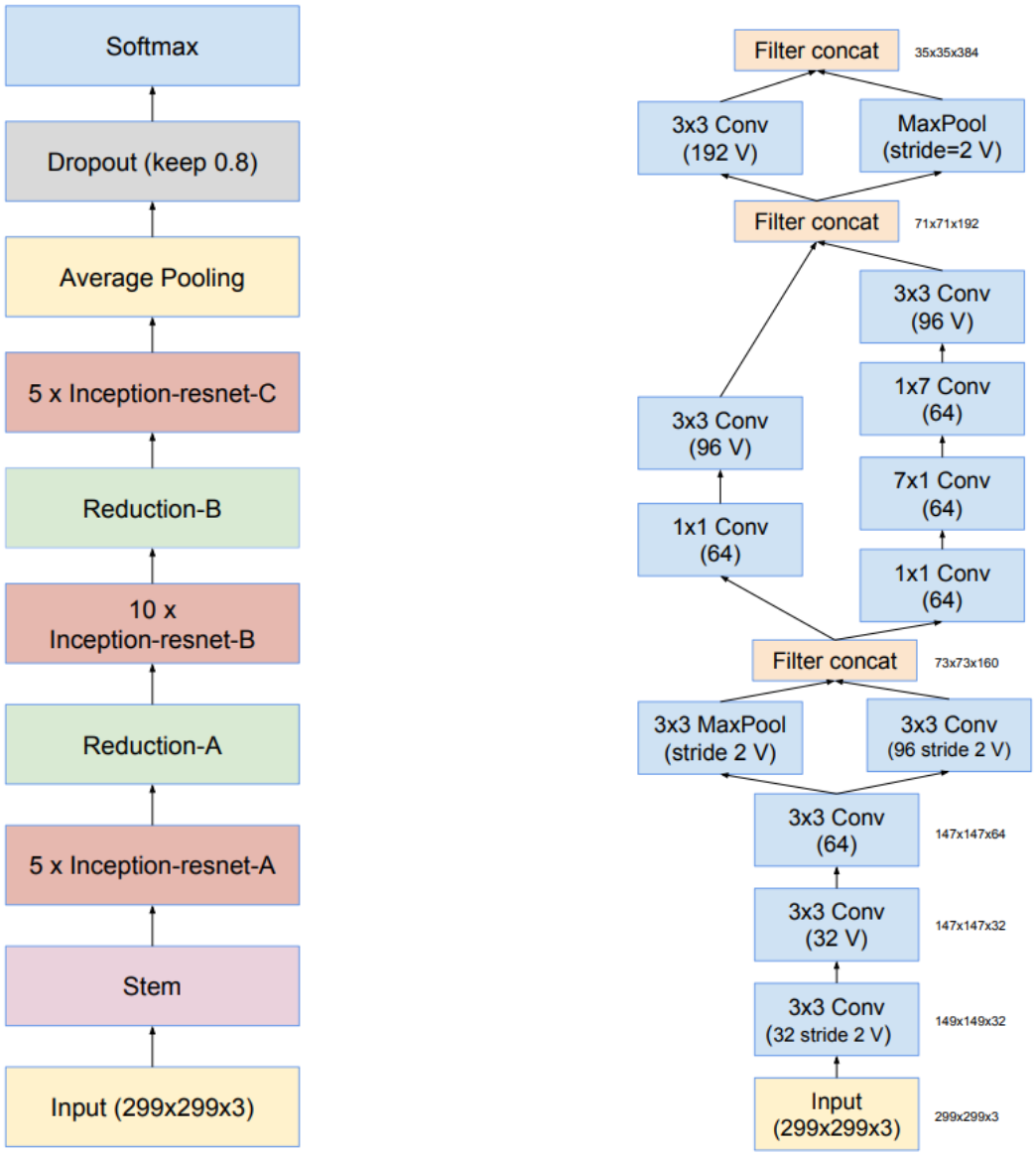


Figure A.1: The overall schema of the Inception_v4 network and the stem that is the input part of network. This figure is reproduced from [4]

architecture of Inception_v4 in details, and Figure A.2 shows the every components of Inception_v4.

ResNet_50: Residual connection is proposed to address the degradation problem when the networks go deeper. In stead of stacking layers directly to fit a desired underlying mapping, Residual learning hypothesizes that it is easier to optimize the residual mapping than to optimize the origina mapping. The residual block is shown as in Fig. A.3. In our work, we used 50 layers ResNet as referenced model, and

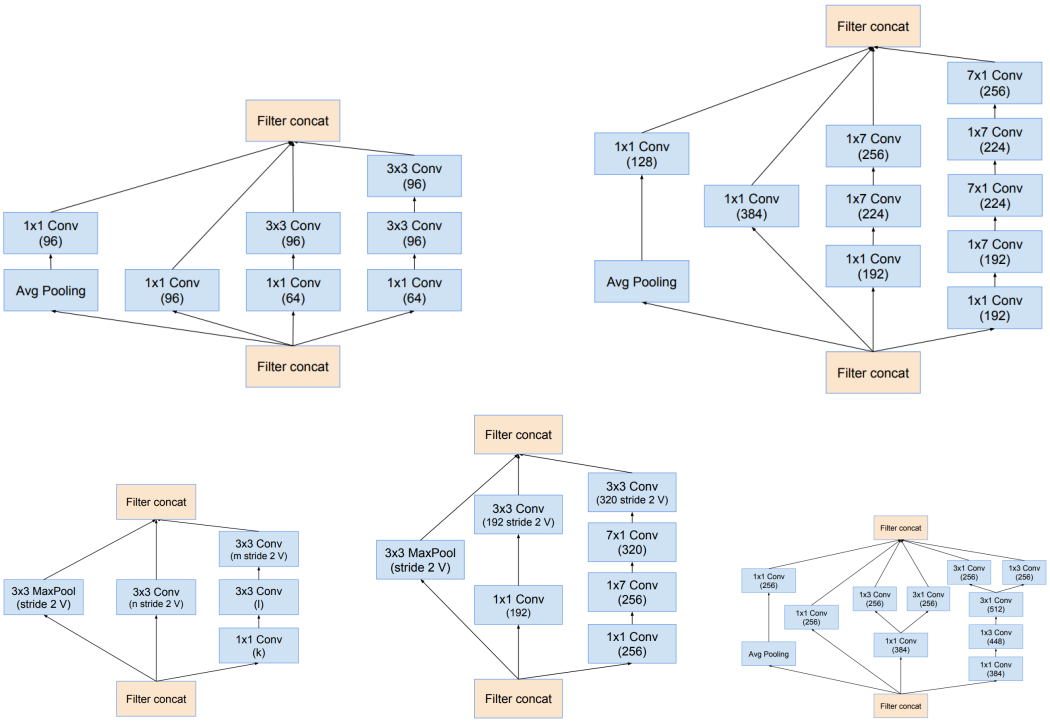


Figure A.2: The schema for 35×35 grid modules, 17×17 grid modules, 35×35 to 17×17 and 17×17 to 8×8 reduction module and 8×8 grid modules. Figures are reproduced from [4]

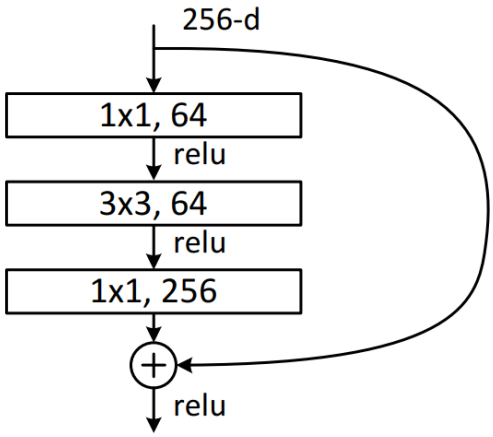


Figure A.3: A residual "bottleneck" building block, reproduced from [5]

Table A.1 illustrates the whole architecture of the ResNet_50.

Table A.1: Architectures for ResNet_50. Building blocks are shown in brackets with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

layer name	input size	Parameters
conv1	112×112	7×7,64, stride 2 3×3 max pool, stride 2
conv2_x	56×56	$\begin{bmatrix} 1 \times 1, & 64 \\ 3 \times 3, & 64 \\ 1 \times 1, & 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 1 \times 1, & 128 \\ 3 \times 3, & 128 \\ 1 \times 1, & 512 \end{bmatrix} \times 4$
conv4_x	14×14	$\begin{bmatrix} 1 \times 1, & 256 \\ 3 \times 3, & 256 \\ 1 \times 1, & 1024 \end{bmatrix} \times 6$
conv5_x	7×7	$\begin{bmatrix} 1 \times 1, & 512 \\ 3 \times 3, & 512 \\ 1 \times 1, & 2048 \end{bmatrix} \times 3$
average pool	1×1	average pool,1000-d fc,softmax

Appendix B

Reference Networks for Spatial Temporal Feature Learning

Here we mainly introduce the architecture of the reference networks (e.g. C3D [135], I3D [6], 3D ResNet with 34 layers [167], P3D with 63 layers [168], R2+1D [169] and 3D ResNext with 50 layers [7]) used in Sec.5. All of them were proposed for the spatial-temporal feature learning for APS video.

C3D: C3D is a simple, yet effective approach for spatio-temporal feature learning using deep 3-dimensional convolutional networks, which are good feature learning machines that model appearance and motion simultaneously. C3D net has 8 convolution, 5 max-pooling, and 2 fully connected layers, followed by a softmax output layer. All 3D convolution kernels are $3 \times 3 \times 3$ with stride 1 in both spatial and temporal dimensions and all pooling kernels are $2 \times 2 \times 2$, except for pool1 is $1 \times 2 \times 2$. The parameter details of C3D is as following: conv1a$(64) \rightarrow$ pool1 $\rightarrow$ conv2a$(128) \rightarrow$ poo2 $\rightarrow$ conv3a$(256) \rightarrow$ conv3b$(256) \rightarrow$ pool3 $\rightarrow$ conv4a$(512) \rightarrow$ conv4b$(512) \rightarrow$ pool4 $\rightarrow$ conv5a$(512) \rightarrow$ conv5b$(512) \rightarrow$ FC6$(4096) \rightarrow$ FC7$(4096) \rightarrow$ softmax. Note that the number in the brackets are the number of channels.

I3D: I3D is Inflated 3D Convolutional networks that is based on the 2D ConvNet inflation; this is, filters and pooling kernels of very deep image classification ConvNets are expanded into 3D, which is easily done by starting with a 2D architecture and inflating all the filters and pooling kernels – endowing them with an

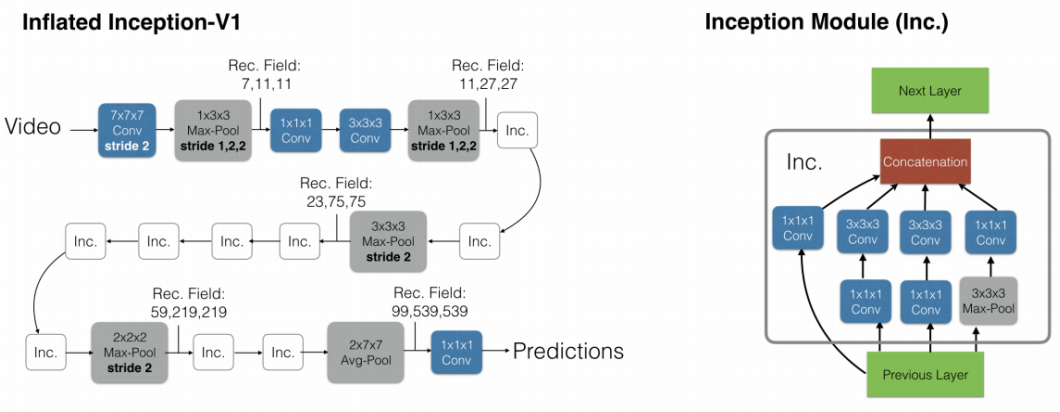


Figure B.1: The Inflated Inception-V1 architecture (left) and its detailed inception submodule (right), reproduced from [6]

additional temporal dimension. I3D is able to learn seamless spatio-temporal feature from video while leveraging successful ImageNet architecture designs and even their parameters. Figure B.1 shows the architecture of I3D in details.

3D ResNet: Inspired by the residual 2D CNNs [5], 3D ResNet [167] was proposed to explore the possibility of training a very deep 3D CNNs for action recognition. Deep 3D CNNs are difficult to train because of the large number of their parameters, while residual connection paves the way for the feasibility of learning spatial temporal feature. Table B.1 illustrates the architecture of 3D ResNet, and the difference between 3D ResNet and original 2D ResNets is the number of dimensions of convolutional kernels and pooling.

Table B.1: Architectures for 3D ResNet with 34 layers. Building blocks are shown in brackets with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

layer name	Parameters
conv1	$7 \times 7 \times 7$,64, stride 1(T), 2(XY) $3 \times 3 \times 3$ max pool, stride 2
conv2_x	$\begin{bmatrix} 3 \times 3 \times 3, & 64 \\ 3 \times 3 \times 3, & 64 \end{bmatrix} \times 3$
conv3_x	$\begin{bmatrix} 3 \times 3 \times 3, & 128 \\ 3 \times 3 \times 3, & 128 \end{bmatrix} \times 4$
conv4_x	$\begin{bmatrix} 3 \times 3 \times 3, & 256 \\ 3 \times 3 \times 3, & 256 \end{bmatrix} \times 6$
conv5_x	$\begin{bmatrix} 3 \times 3 \times 3, & 512 \\ 3 \times 3 \times 3, & 512 \end{bmatrix} \times 3$
average pool	average pool,400-d fc,softmax

P3D: It is a natural way to learn spatial temporal feature Using 3D CNNs, but training of 3D CNN is very computationally expensive and the model size also has a quadratic growth compared to 2D CNNs. P3D is designed to mitigate the above limitations by devising a family of bottleneck building blocks that leverages both spatial and temporal convolutional filters [168]. Specifically, P3D uses $1 \times 3 \times 3$ convolutional layer and one layer of $3 \times 1 \times 1$ convolutions in a parallel to replace a standard $3 \times 3 \times 3$ convolutional layer. As such, the model size is significantly reduced and the advantages of pre-learnt 2D CNN in image domain could also be fully leveraged by initializing the $1 \times 3 \times 3$ convolutional filters with 3×3 convolutions in 2D CNN. Moreover, P3D uses Pseudo-3D Residual Net that composes each designed block in different placement throughout a whole ResNet-like architecture to enhance the structural diversity of the network. As a result, the temporal connections in P3D ResNet are constructed at every level from bottom to top and the learnt video representations encapsulate information related to objects, scenes and actions in videos, making them generic for various video analysis tasks.

R2+1D: R2+1D introduces two new forms of spatiotemporal convolution that can be viewed as middle grounds between the extremes of 2D (spatial convolution) and full 3D [169]. Architecture of R2+1D is ResNet like 3D CNNs with two improved formulation. The first formulation is employing 3D convolutions only in the early layers of the network, with 2D convolutions in the top layers. As such, 3D convolutions in the early layers is a low/mid-level operation to model the motion, while 2D convolutions in the top layers realize spatial reasoning leading to accurate feature representation. The second spatio-temporal variant is a "(2+1)D" convolutional block, which explicitly factorizes 3D convolution into two separate and successive operations, a 2D spatial convolution and a 1D temporal convolution. From this module, an additional nonlinear rectification is added between these two operations, effectively doubling the number of nonlinearities. Besides, decomposition facilitates the optimization, yielding in practice both a lower training loss and a lower testing loss.

3D ResNext: 3D ResNext is ResNet-based architectures with 3D convolu-

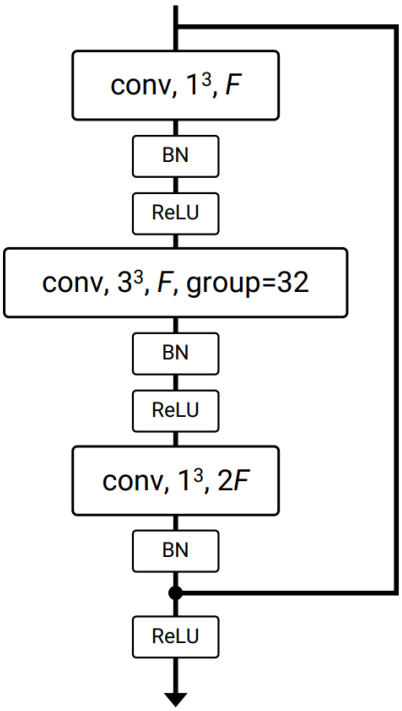


Figure B.2: Block of ResNext: x^3 and F are the kernel size and the number of feature maps, respectively; *group* is the number of groups of group convolutions, which divide the feature maps into small groups, reproduced from [7].

tions. Different from the original bottleneck block, the ResNeXt block introduces group convolutions, which divide the feature maps into small groups. Moreover, ResNext introduces cardinality, which refers to the number of middle convolutional layer groups in the bottleneck block. And increasing the cardinality of 2D architectures is more effective than using wider or deeper ones [174]. In our work, we use ResNext with 50 layers as reference and its residual block is shown in Figure B.2.

Appendix C

Datasets and Code for Algorithm Implementation

In this appendix, we provide reference link for the code used to make this thesis and our public datasets.

1: PIX2NVS software introduced in Chapter 3 can be found in this link:

https://github.com/PIX2NVS/PIX2NVS

2: Code for Chapter 4 and ASL-DVS dataset can be downloaded via this Github page:

https://github.com/PIX2NVS/NVS2Graph.

3: Code for Chapter 5 and UCF101-DVS, HMDB-DVS and ASLAN-DVS datasets are also public and can be found in following Github page:

https://github.com/PIX2NVS/NVS_FeatureLearning.

Bibliography

[1] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbrück. A 128x128, 120 dB 30mW latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[2] Qinyi Wang, Yexin Zhang, Junsong Yuan, and Yilong Lu. Space-time event clouds for gesture recognition: From RGB cameras to event cameras. In *Proc. IEEE Winter Conf. Appl. of Comput. Vision*, pages 1826–1835, 2019.

[3] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. SplineCNN: Fast geometric deep learning with continuous B-spline kernels. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 869–877, 2018.

[4] Gregory Kevin Cohen. *Event-based feature detection, recognition and classification*. PhD thesis, Paris 6, 2016.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 770–778, 2016.

[6] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 6299–6308, 2017.

[7] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Can spatiotemporal 3D CNNs retrace the history of 2D CNNs and ImageNet? In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 6546–6555, 2018.

[8] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. *arXiv preprint arXiv:1610.08336*, 2016.

[9] Christoph Posch, Teresa Serrano-Gotarredona, Bernabe Linares-Barranco, and Tobi Delbruck. Retinomorphic event-based vision sensors: bioinspired cameras with spiking output. *Proceedings of the IEEE*, 102(10):1470–1484, 2014.

[10] Richard H Masland. The fundamental plan of the retina. *Nature neuroscience*, 4(9):877, 2001.

[11] Stephen W Kuffler. Discharge patterns and functional organization of mammalian retina. *Journal of neurophysiology*, 16(1):37–68, 1953.

[12] Tobi Delbrück, Bernabe Linares-Barranco, Eugenio Culurciello, and Christoph Posch. Activity-driven, event-based vision sensors. In *Proc. IEEE Int. Symp. on Circuits and Syst.*, pages 2426–2429, 2010.

[13] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. A 240$\times$ 180 130 db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, 2014.

[14] Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. A dynamic vision sensor with 1% temporal contrast sensitivity and in-pixel asynchronous delta modulator for event encoding. *IEEE Journal of Solid-State Circuits*, 50(9):2149–2160, 2015.

[15] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. A 128*times*128 1.5% contrast sensitivity 0.9% fpn 3 μs latency 4 mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers. *IEEE Journal of Solid-State Circuits*, 48(3):827–838, 2013.

[16] Juan Antonio Leñero-Bardallo, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. A 3.6*mu* s latency asynchronous frame-free event-driven

dynamic-vision-sensor. *IEEE Journal of Solid-State Circuits*, 46(6):1443–1455, 2011.

[17] Christoph Posch, Daniel Matolin, and Rainer 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*, 46(1):259–275, Jan. 2010.

[18] Bongki Son, Yunjae Suh, Sungho Kim, Heejae Jung, Jun-Seok Kim, Changwoo Shin, Keunju Park, Kyoobin Lee, Jinman Park, Jooyeon Woo, et al. 4.1 a 640× 480 dynamic vision sensor with a 9μm pixel and 300meps address-event representation. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 66–67. IEEE, 2017.

[19] Chenghan Li, Christian Brandli, Raphael Berner, Hongjie Liu, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. Design of an rgbw color vga rolling and global shutter dynamic and active-pixel vision sensor. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 718–721. IEEE, 2015.

[20] T. Delbruck. Neuromorphic vision sensing and processing. In *Proc. Europ. Solid-State Dev. Res. Conf.*, pages 7–14. IEEE, 2016.

[21] E Neftci, C Posch, and E Chicca. Neuromorphic engineering. *Computational Intelligence-Volume II*, page 278, 2015.

[22] David Weikersdorfer and Jörg Conradt. Event-based particle filtering for robot self-localization. In *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 866–870. IEEE, 2012.

[23] Elias Mueggler, Basil Huber, and Davide Scaramuzza. Event-based, 6-dof pose tracking for high-speed maneuvers. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2761–2768. IEEE, 2014.

[24] Guillermo Gallego, Jon EA Lund, Elias Mueggler, Henri Rebecq, Tobi Delbruck, and Davide Scaramuzza. Event-based, 6-dof camera tracking for high-speed applications. *arXiv preprint arXiv:1607.03468*, 2, 2016.

[25] Andrea Censi and Davide Scaramuzza. Low-latency event-based visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 703–710. IEEE, 2014.

[26] Wenzhen Yuan and Srikumar Ramalingam. Fast localization and tracking using event sensors. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4564–4571. IEEE, 2016.

[27] Alex Zihao Zhu, Nikolay Atanasov, and Kostas Daniilidis. Event-based visual inertial odometry. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5816–5824. IEEE, 2017.

[28] Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. In *BMVC*, 2017.

[29] Matthew Evanusa, Yulia Sandamirskaya, et al. Event-based attention and tracking on neuromorphic hardware. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[30] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[31] Junhaeng Lee, T Delbruck, Paul KJ Park, Michael Pfeiffer, Chang-Woo Shin, Hyunsurk Ryu, and Byung Chang Kang. Live demonstration: Gesture-based remote control using stereo pair of dynamic vision sensors. In *2012 IEEE International Symposium on Circuits and Systems*, pages 741–745. IEEE, 2012.

[32] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, et al. A low power, fully event-based gesture recognition system. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 7243–7252, 2017.

[33] G. Orchard et al. HFirst: a temporal approach to object recognition. *IEEE Trans. on Patt. Anal. Mach. Intel.*, 37(10):2028–2040, 2015.

[34] Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E Shi, and Ryad B Benosman. HOTS: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Analysis and Machine Intell.*, 39(7):1346–1359, 2016.

[35] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. HATS: Histograms of averaged time surfaces for robust event-based object classification. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 1731–1740, 2018.

[36] Bharath Ramesh, Hong Yang, Garrick Michael Orchard, Ngoc Anh Le Thi, Shihao Zhang, and Cheng Xiang. DART: Distribution aware retinal transform for event-based cameras. *IEEE Trans. Pattern Analysis and Machine Intell.*, 2019.

[37] Daniel Gehrig, Antonio Loquercio, Konstantinos G Derpanis, and Davide Scaramuzza. End-to-end learning of representations for asynchronous event-based data. *arXiv preprint arXiv:1904.08245*, 2019.

[38] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. EV-FlowNet: Self-supervised optical flow estimation for event-based cameras. *arXiv:1802.06898*, 2018.

[39] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Event-based convolutional networks for object detection in neuromorphic cameras. *arXiv:1805.07931*, 2018.

[40] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Attention mechanisms for object recognition with event-based cameras. *arXiv:1807.09480*, 2018.

[41] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Trans. Comput.-Aided Design of Integrated Circuits and Syst.*, 34(10):1537–1557, 2015.

[42] Jun Haeng Lee, Tobi Delbrück, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10:508, 2016.

[43] Emre Neftci, Srinjoy Das, Bruno Pedroni, Kenneth Kreutz-Delgado, and Gert Cauwenberghs. Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in Neuroscience*, 7:272, 2014.

[44] Evangelos Stromatias, Daniel Neil, Francesco Galluppi, Michael Pfeiffer, Shih-Chii Liu, and Steve Furber. Scalable energy-efficient, low-latency implementations of trained spiking deep belief networks on SpiNNaker. In *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, pages 1–8, 2015.

[45] Hongjie Liu, Diederik Paul Moeys, Gautham Das, Daniel Neil, Shih-Chii Liu, and Tobi Delbrück. Combined frame-and event-based detection and tracking. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2511–2514. IEEE, 2016.

[46] Gregor Lenz, S Ieng, and Ryad Benosman. Event-based dynamic face detection and tracking based on activity. *CoRR*, 2018.

[47] João Carneiro, Sio-Hoi Ieng, Christoph Posch, and Ryad Benosman. Event-based 3d reconstruction from neuromorphic retinas. *Neural Networks*, 45:27–38, 2013.

[48] Ahmed Nabil Belbachir, Stephan Schraml, Manfred Mayerhofer, and Michael Hofstätter. A novel hdr depth camera for real-time 3d 360 panoramic vision. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 425–432. IEEE, 2014.

[49] Nathan Matsuda, Oliver Cossairt, and Mohit Gupta. Mc3d: Motion contrast 3d scanning. In *2015 IEEE International Conference on Computational Photography (ICCP)*, pages 1–10. IEEE, 2015.

[50] Henri Rebecq, Guillermo Gallego, and Davide Scaramuzza. Emvs: Event-based multi-view stereo. In *British machine vision conference (BMVC)*, number CONF, 2016.

[51] David Weikersdorfer, David B Adrian, Daniel Cremers, and Jörg Conradt. Event-based 3d slam with a depth-augmented dynamic vision sensor. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 359–364. IEEE, 2014.

[52] Beat Kueng, Elias Mueggler, Guillermo Gallego, and Davide Scaramuzza. Low-latency visual odometry using event-based feature tracks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 16–23. IEEE, 2016.

[53] Hanme Kim, Ankur Handa, Ryad Benosman, Sio-Hoi Ieng, and Andrew J Davison. Simultaneous mosaicing and tracking with an event camera. *J. Solid State Circ*, 43:566–576, 2008.

[54] Hanme Kim, Stefan Leutenegger, and Andrew J Davison. Real-time 3d reconstruction and 6-dof tracking with an event camera. In *European Conference on Computer Vision*, pages 349–364. Springer, 2016.

[55] Michael Milford, Hanme Kim, Stefan Leutenegger, and Andrew Davison. Towards visual slam with event-based cameras. In *The problem of mobile sensors workshop in conjunction with RSS*, 2015.

[56] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[57] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[58] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[59] Ryad Benosman, Charles Clercq, Xavier Lagorce, Sio-Hoi Ieng, and Chiara Bartolozzi. Event-based visual flow. *IEEE transactions on neural networks and learning systems*, 25(2):407–417, 2013.

[60] Francisco Barranco, Cornelia Fermüller, and Yiannis Aloimonos. Contour motion estimation for asynchronous event-driven cameras. *Proc. IEEE*, 102(10):1537–1556, 2014.

[61] Francisco Barranco, Cornelia Fermuller, and Yiannis Aloimonos. Bio-inspired motion estimation with event-driven sensors. In *Proc. Int. Work-Conf. Artificial Neural Networks*, pages 309–321, 2015.

[62] Tobias Brosch, Stephan Tschechne, and Heiko Neumann. On event-based optical flow detection. *Frontiers in Neuroscience*, 9:137, 2015.

[63] Garrick Orchard and Ralph Etienne-Cummings. Bioinspired visual motion estimation. *Proc. IEEE*, 102(10):1520–1536, 2014.

[64] Tobi Delbruck and Patrick Lichtsteiner. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. In *2007 IEEE international symposium on circuits and systems*, pages 845–848. IEEE, 2007.

[65] Jörg Conradt, Matthew Cook, Raphael Berner, Patrick Lichtsteiner, Rodney J Douglas, and T Delbruck. A pencil balancing robot using a pair of AER dynamic vision sensors. In *Proc. IEEE Int. Symp. on Circ. and Syst.*, pages 781–784. IEEE, 2009.

[66] Zhengming Fu, Tobi Delbruck, Patrick Lichtsteiner, and Eugenio Culurciello. An address-event fall detector for assisted living applications. *IEEE Transactions on Biomedical Circuits and Systems*, 2(2):88–96, 2008.

[67] Xavier Clady, Sio-Hoi Ieng, and Ryad Benosman. Asynchronous event-based corner detection and matching. *Neural Networks*, 66:91–106, 2015.

[68] Valentina Vasco, Arren Glover, and Chiara Bartolozzi. Fast event-based Harris corner detection exploiting the advantages of event-driven cameras. In *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, pages 4144–4149, 2016.

[69] Elias Mueggler, Chiara Bartolozzi, and Davide Scaramuzza. Fast event-based corner detection. In *Proc. Brit. Machine Vision Conf.*, volume 1, 2017.

[70] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbrück, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM. *The Int. Journal of Robotics Research*, 36(2):142–149, 2017.

[71] Elias Mueggler, Guillermo Gallego, Henri Rebecq, and Davide Scaramuzza. Continuous-time visual-inertial odometry for event cameras. *IEEE Transactions on Robotics*, 34(6):1425–1440, 2018.

[72] Xavier Clady, Jean-Matthieu Maro, Sébastien Barré, and Ryad B Benosman. A motion-based feature for event-based pattern recognition. *Frontiers in Neuroscience*, 10:594, 2017.

[73] Ryad Benosman, Charles Clercq, Xavier Lagorce, Sio-Hoi Ieng, and Chiara Bartolozzi. Event-based visual flow. *IEEE Trans. Neural Networks and Learning Syst.*, 25(2):407–417, 2014.

[74] José Antonio Pérez-Carrasco, Bo Zhao, Carmen Serrano, Begona 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 transactions on pattern analysis and machine intelligence*, 35(11):2706–2719, 2013.

[75] Xi Peng, Bo Zhao, Rui Yan, Huajin Tang, and Zhang Yi. Bag of events: An efficient probability-based feature extraction method for AER image sensors. *IEEE Trans. Neural Networks and Learning Syst.*, 28(4):791–803, 2017.

[76] Rohan Ghosh, Anupam Gupta, Andrei Nakagawa, Alcimar Soares, and Nitish Thakor. Spatiotemporal filtering for event-based action recognition. *arXiv:1903.07067*, 2019.

[77] 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.*, pages 7968–7972, 2019.

[78] Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. Events-to-video: Bringing modern computer vision to event cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3857–3866, 2019.

[79] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:99, 2015.

[80] Olivier Bichler, Damien Querlioz, Simon J Thorpe, Jean-Philippe Bourgoin, and Christian Gamrat. Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity. *Neural Networks*, 32:339–348, 2012.

[81] Peter O'Connor, Daniel Neil, Shih-Chii Liu, Tobi Delbrück, and Michael Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in Neuroscience*, 7:178, 2013.

[82] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks

through weight and threshold balancing. In *Proc. Int. Joint Conf. Neural Networks (IJCNN)*, pages 1–8, 2015.

[83] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proc. the IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 652–660, 2017.

[84] Shan Gao, Guangqian Guo, and CL Philip Chen. Event-based incremental broad learning system for object classification. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[85] Bo Zhao, Ruoxi Ding, Shoushun Chen, Bernabe Linares-Barranco, and Huajin Tang. Feedforward categorization on AER motion events using cortex-like features in a spiking neural network. *IEEE Trans. Neural Networks and Learning Syst.*, 26(9):1963–1978, 2014.

[86] Xi Peng, Bo Zhao, Rui Yan, Huajin Tang, and Zhang Yi. Bag of events: An efficient probability-based feature extraction method for AER image sensors. *IEEE Trans. Neural Networks and Learning Syst.*, 28(4):791–803, 2016.

[87] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proc. ICLR*, 2017.

[88] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. CayleyNets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Trans. Signal Process.*, 67(1):97–109, 2017.

[89] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Inf. Process. Syst.*, pages 3844–3852, 2016.

[90] Chu Wang, Babak Samari, and Kaleem Siddiqi. Local spectral graph convolution for point set feature learning. In *Proc. Eur. Conf. Comput. Vision (ECCV)*, pages 52–66, 2018.

[91] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv:1312.6203*, 2013.

[92] Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs. *IEEE Trans. Signal Process.*, 61(7):1644–1656, 2013.

[93] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *arXiv:1211.0053*, 2012.

[94] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Inf. Process. Syst.*, pages 2224–2232, 2015.

[95] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Inf. Process. Syst.*, pages 1993–2001, 2016.

[96] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Inf. Process. Syst.*, pages 3189–3197, 2016.

[97] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *Proc. IEEE Int. Conf. Comput. Vision Workshops*, pages 37–45, 2015.

[98] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 5115–5124, 2017.

[99] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

[100] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[101] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.

[102] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proc. 32nd AAAI Conf. Artificial Intell.*, 2018.

[103] Victor Garcia and Joan Bruna. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*, 2017.

[104] Medhini Narasimhan, Svetlana Lazebnik, and Alexander Schwing. Out of the box: Reasoning with graph convolution nets for factual visual question answering. In *Advances in Neural Information Processing Systems*, pages 2654–2665, 2018.

[105] Zhen Cui, Chunyan Xu, Wenming Zheng, and Jian Yang. Context-dependent diffusion network for visual relationship detection. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 1475–1482. ACM, 2018.

[106] Ting Yao, Yingwei Pan, Yehao Li, and Tao Mei. Exploring visual relationship for image captioning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 684–699, 2018.

[107] Danfei Xu, Yuke Zhu, Christopher B Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5419, 2017.

[108] Xiang Gao, Wei Hu, Jiaxiang Tang, Pan Pan, Jiaying Liu, and Zongming Guo. Generalized graph convolutional networks for skeleton-based action recognition. *arXiv preprint arXiv:1811.12013*, 2018.

[109] Xiaolong Wang and Abhinav Gupta. Videos as space-time region graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 399–417, 2018.

[110] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):146, 2019.

[111] Gusi Te, Wei Hu, Amin Zheng, and Zongming Guo. Rgcnn: Regularized graph cnn for point cloud segmentation. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 746–754. ACM, 2018.

[112] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning localized generative models for 3d point clouds via graph convolution. 2018.

[113] Y. Hu, H. Liu, M. Pfeiffer, and T. Delbruck. DVS benchmark datasets for object tracking, action recognition, and object recognition. *Frontiers in Neuroscience*, 10, 2016.

[114] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9, 2015.

[115] S. Abu-El-Haija et al. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*, 2016.

[116] M. L. Katz, K. Nikolic, and T. Delbruck. Live demonstration: Behavioural emulation of event-based vision sensors. In *Proc. IEEE Int. Symp. on Circ. and Syst.*, pages 736–740. IEEE, 2012.

[117] G. Pineda García, P. Camilleri, Q. Liu, and S. Furber. pydvs: An extensible, real-time dynamic vision sensor emulator using off-the-shelf hardware. In *Proc. IEEE Symp. Ser. Comp. Intel. (SSCI), 2016*, pages 1–7. IEEE, 2016.

[118] K. A. Boahen. Point-to-point connectivity between neuromorphic chips using address events. *IEEE Trans. Circ. and Syst. II: Analog and Digital Sig. Process.*, 47(5):416–434, 2000.

[119] Y. Andreopoulos and M. Van der schaar. Adaptive linear prediction for resource estimation of video decoding. *IEEE Trans. Circ. and Syst. for Video Technol.*, 17(6):751–764, 2007.

[120] J. Barbarien et al. Scalable motion vector coding. In *Proc. IEEE Int. Conf. Image Process., 2004. ICIP'04*, volume 2, pages 1321–1324. IEEE, 2004.

[121] S. Tomar. Converting video formats with ffmpeg. *Linux Journal*, 2006(146):10, 2006.

[122] K. Matkovic, L. Neumann, A. Neumann, T. Psik, and W. Purgathofer. Global contrast factor-a new approach to image contrast. *Computational Aesthetics*, 2005:159–168, 2005.

[123] Basabdatta Sen Bhattacharya and Stephen B Furber. Biologically inspired means for rank-order encoding images: A quantitative analysis. *IEEE Trans. Neural Netw.*, 21(7):1087–1099, 2010.

[124] Yin Bi and Yiannis Andreopoulos. PIX2NVS: Parameterized conversion of pixel-domain video frames to neuromorphic vision streams. *Proc. IEEE Int. Conf. Image Process., ICIP*, 2017.

[125] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comp. Vis.*, 40(2):99–121, 2000.

[126] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv:1212.0402*, 2012.

[127] Diederik Paul Moeys, Federico Corradi, Emmett Kerr, Philip Vance, Gautham Das, Daniel Neil, Dermot Kerr, and Tobi Delbrück. Steering a predator robot using a mixed frame/event-driven convolutional neural network. In *Proc. Int. Conf. Event-based Contr., Comm., and Sig. Process., EBCCSP*, pages 1–8. IEEE, 2016.

[128] Benjamin Graham. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014.

[129] Ben Graham. Sparse 3d convolutional neural networks. *arXiv preprint arXiv:1505.02890*, 2015.

[130] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Inf. Process. Syst.*, pages 568–576, 2014.

[131] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proc. IEEE Conf. on Comput. Vision and Patt. Rec., CVPR*, pages 1725–1732, 2014.

[132] Aaron Chadha, Alhabib Abbas, and Yiannis Andreopoulos. Compressed-domain video classification with deep neural networks: There's way too much information to decode the Matrix. *Proc. IEEE Int. Conf. Image Process., ICIP*, 2017.

[133] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the H.264/AVC video coding standard. *IEEE Trans. Circ. Syst. Video Technol.*, 13(7):560–576, 2003.

[134] Gary J Sullivan, Jens Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (HEVC) standard. *IEEE Trans. Circ. Syst. Video Technol.*, 22(12):1649–1668, 2012.

[135] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3D convolutional networks. In *Proc. IEEE Int. Conf. Comput. Vision*, pages 4489–4497, 2015.

[136] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proc. IEEE Int. Conf. Comp. Vis., ICCV*, pages 1026–1034, 2015.

[137] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Inf. Process. Syst.*, pages 1097–1105, 2012.

[138] Sinisa Todorovic. Human activities as stochastic kronecker graphs. *Computer Vision–ECCV 2012*, pages 130–143, 2012.

[139] Bill Bowhill, Blaine Stackhouse, Nevine Nassif, Zibing Yang, Arvind Raghavan, Oscar Mendoza, Charles Morganti, Chris Houghton, Dan Krueger, Olivier Franza, et al. The Xeon processor E5-2600 v3: A 22 nm 18-core product family. *IEEE J. of Solid-State Circuits*, 51(1):92–104, 2016.

[140] Alina Kuznetsova, Laura Leal-Taixé, and Bodo Rosenhahn. Real-time sign language recognition using a consumer depth camera. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 83–90, 2013.

[141] Tobi Delbruck Federico Corradi Luca Longinotti, Sim Bamford. jaersoftware. December 2016.

[142] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[143] Christian Brandli, Thomas Mantel, Marco Hutter, Markus Höpflinger, Raphael Berner, Roland Siegwart, and Tobi Delbruck. Adaptive pulsed laser line extraction for terrain reconstruction using a dynamic vision sensor. *Frontiers in neuroscience*, 7:275, 2014.

[144] Tobi Delbrück. Neuromorphic vision sensing and processing. In *Eur. Solid-State Dev. Res. Conf.*, pages 7–14, 2016.

[145] Christoph Posch, Daniel Matolin, and Rainer 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*, 46(1):259–275, 2011.

[146] Cheston Tan, Stephane Lallee, and Garrick Orchard. Benchmarking neuromorphic vision: Lessons learnt from computer vision. *Frontiers in Neuroscience*, 9:374, 2015.

[147] Xavier Lagorce, Garrick Orchard, Francesco Galluppi, Bertram E Shi, and Ryad B Benosman. HOTS: A hierarchy of event-based time-surfaces for pattern recognition. *IEEE Trans. Pattern Analysis and Machine Intell.*, 39(7):1346–1359, 2017.

[148] Yin Bi and Yiannis Andreopoulos. PIX2NVS: Parameterized conversion of pixel-domain video frames to neuromorphic vision streams. In *Proc. IEEE Int. Conf. Image Process. (ICIP)*, pages 1990–1994, 2017.

[149] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. Poker-DVS and MNIST-DVS. Their history, how they were made, and other details. *Frontiers in Neuroscience*, 9:481, 2015.

[150] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. CIFAR10-DVS: An event-stream dataset for object classification. *Frontiers in Neuroscience*, 11:309, 2017.

[151] KH Lee, H Woo, and T Suk. Point data reduction using 3D grids. *The Int. Journal of Adv. Manuf. Technol.*, 18(3):201–210, 2001.

[152] Les Piegl and Wayne Tiller. *The NURBS book*. Springer Science & Business Media, 2012.

[153] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015.

[154] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 2129–2137, 2016.

[155] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proc. the IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 3693–3702, 2017.

[156] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proc. 32nd AAAI Conf. Artificial Intell.*, 2018.

[157] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Inf. Process. Syst.*, pages 3882–3890, 2016.

[158] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Magazine*, 34(4):18–42, 2017.

[159] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proc. ICLR*, 2019.

[160] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[161] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-ResNet and the impact of residual connections on learning. In *Proc. 31st AAAI Conf. Artificial Intell.*, 2017.

[162] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *Proc. ICLR*, 2017.

[163] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 1933–1941, 2016.

[164] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. HMDB: A large video database for human motion recognition. In *Proc. IEEE Int. Conf. Comput. Vision*, pages 2556–2563, 2011.

[165] Orit Kliper-Gross, Tal Hassner, and Lior Wolf. The action similarity labeling challenge. *IEEE Trans. Pattern Analysis and Machine Intell.*, 34(3):615–621, 2011.

[166] Konstantinos G Derpanis, Matthieu Lecce, Kostas Daniilidis, and Richard P Wildes. Dynamic scene understanding: The role of orientation features in space and time in scene classification. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 1306–1313, 2012.

[167] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Learning spatio-temporal features with 3d residual networks for action recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3154–3160, 2017.

[168] Zhaofan Qiu, Ting Yao, and Tao Mei. Learning spatio-temporal representation with pseudo-3D residual networks. In *Proc. IEEE Int. Conf. Comput. Vision*, pages 5533–5541, 2017.

[169] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognit.*, pages 6450–6459, 2018.

[170] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Process.*, pages 4580–4584, 2015.

[171] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Inf. Process. Syst.*, pages 5099–5108, 2017.

[172] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[173] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[174] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.