POLITECNICO DI MILANO

Facoltà di Ingegneria

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Doctoral Programme in Information Technology

Learning Efficient and Effective Representations for Event-based Cameras

Advisor: PROF. MATTEO MATTEUCCI

Tutor: PROF. FRANCESCO AMIGONI

Doctoral Dissertation of:

MARCO CANNICI

2022 - XXXIV Cycle

To my family,
Isidoro, Claudia, Chiara.

ABSTRACT

Event-based cameras are bio-inspired sensors that emulate the functioning of biological retinas. Unlike traditional cameras, which generate dense frames at a constant and predefined rate, these sensors, similarly to the photoreceptors in the retina, output data only when a change in brightness is detected. The result is a sensor able to sparsely and incrementally encode visual changes with microseconds resolution, high dynamic range, and minimum requirements for power consumption and bandwidth. Nevertheless, due to their fundamentally novel way of recording appearance, these sensors cannot be directly used with typical computer vision systems, which must be redesigned to work with events. That is the case with deep neural networks for vision, which express their full potential when hierarchical representations can be computed from input data, such as when dealing with images. However, learning to achieve this level of abstraction efficiently and effectively from events is far more difficult than doing the same with images. In fact, while rich visual data is directly accessible from a single frame, reconstructing appearance from events requires additional computation and temporal reasoning. Visual information is indeed spread temporally through incremental and sparse updates, making learning effective network representations harder.

This thesis addresses this challenge by focusing on three aspects of designing deep neural networks for event-based vision. First, we look at how to efficiently compute hidden neural representations by preserving event-based cameras' properties during computation. We accomplish this by designing a framework for converting deep neural networks into systems with identical expressiveness but capable of performing asynchronous and incremental processing, thus retaining the event camera's asynchronous and data-driven nature. Then, we focus on performance and study how to learn effective input representations for a given task. We propose a recurrent mechanism that automatically learns to interface with any convolutional network by sparsely and incrementally building a frame-like representation from asynchronous events. Finally, we focus on the challenging task of training neural networks to operate effectively on a real-world event-based camera when the only source of training supervision comes from simulation. We tackle the problem from a domain adaptation perspective by learning to extract domain-invariant intermediate representations. This learning strategy enables the network to attain performance comparable to that potentially achieved by directly learning from real annotated samples, yet without performing any finetuning on a real device. Throughout this thesis, we explore the importance of representations in event-based networks, at both the input and hidden layers, and show that by focusing on these aspects, considerable gains can be achieved toward more effective and efficient processing.

CONTRIBUTIONS

MAIN THESIS CONTRIBUTIONS

[1] **Cannici, M.**, Ciccone, M., Romanoni, A., and Matteucci, M. (2019). "Asynchronous convolutional networks for object detection in neuromorphic cameras." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. **Best Paper Award**.

Online Resources: *Paper*, *Video*, *Code*

Personal contribution: main author; code implementation, experimental validationn, results' analysis, and paper writing.

[2] **Cannici, M.**, Ciccone, M., Romanoni, A., and Matteucci, M. (2020). "A differentiable recurrent surface for asynchronous event-based data." In: *European Conference on Computer Vision, (pp. 136-152)*. Springer, Cham.

Online Resources: *Paper*, *Video*, *Project Page*

Personal contribution: main author; code implementation, experimental validation, results' analysis, and paper writing.

[3] Planamente*, M., Plizzari*, C., **Cannici*, M.**, Ciccone, M., Strada, F., Bottino, A., Matteucci, M., and Caputo, B. (2021). "DA4Event: towards bridging the Sim-to-Real Gap for Event Cameras using Domain Adaptation". In: *IEEE Robotics and Automation Letters, (vol. 6, no. 4, pp. 6616-6623)*.

Online Resources: *Paper*, *Code*

Personal contribution: main author (the first three authors contributed equally); code implementation, experimental validation, results' analysis, and paper writing.

[4] **Cannici*, M.**, Plizzari*, C., Planamente*, M., Ciccone, M., Bottino, A., Caputo, B., and Matteucci, M. (2021). "N-ROD: A Neuromorphic Dataset for Synthetic-to-Real Domain Adaptation." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (pp. 1342-1347)*.

Online Resources: *Paper*, *Video*, *Project Page*

Personal contribution: main author (the first three authors contributed equally); data acquisition, code implementation, experimental validation, results' analysis, and paper writing.

* *indicates equal contribution.*

OTHER CONTRIBUTIONS

In parallel to the primary research on event-based vision, my Ph.D. research has also explored general topics in deep learning. Part of the research [5, 6] has focused on skeleton-based action recognition, which involves classifying actions based on a sparse point-cloud-based representation of the human skeleton, partially similar to that of an event-based camera. Later research [7] has also involved studying how to integrate deep neural networks and algorithms in end-to-end trainable pipelines to improve robustness and explainability.

Although this thesis does not cover these contributions in detail, ongoing research and prospective applications of these studies to event-based vision, such as the one proposed in [8], are discussed in the concluding chapter. The following is a list of the additional research contributions not explicitely covered in the thesis:

[5] Plizzari, C., **Cannici, M.**, and Matteucci, M. (2021). "Spatial temporal transformer network for skeleton-based action recognition." In: *Pattern Recognition. ICPR International Workshops and Challenges. Proceedings, Part III (pp. 694-701)*. Springer International Publishing.

Online Resources: Paper, Code

[6] Plizzari, C., **Cannici, M.**, and Matteucci, M. (2021). "Skeleton-based action recognition via spatial and temporal transformer networks." In: *Computer Vision and Image Understanding, 208, 103219*.

Online Resources: Paper, Code

[7] Archetti, A., **Cannici, M.**, and Matteucci, M. (2021). "Neural Weighted A*: Learning Graph Costs and Heuristics with Differentiable Anytime A*." In: *International Conference on Machine Learning, Optimization, and Data Science*. **Honorable Mention**.

Online Resources: Post-Print, Code

[8] Plizzari, C., Planamente, M., Goletto, G., **Cannici, M.**, Gusso E., Matteucci, M., and Caputo, B. (2022). "E^2(GO)MOTION: Motion Augmented Event Stream for Egocentric Action Recognition." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Online Resources: Pre-Print

ACKNOWLEDGMENTS

Part of the works presented in this thesis have been supported by research grants under the research project "TEINVEIN: TEcnologie INnovative per i VEicoli Intelligenti", CUP (Codice Unico Progetto - Unique Project Code): E96D17000110009 - Call “Accordi per la Ricerca e lInnovazione”, cofunded by POR FESR 2014 – 2020 (Programma Operativo Regionale, Fondo Europeo di Sviluppo Regionale - Regional Operational Programme, European Regional Development Fund).

First, I would like to thank my advisor, Prof. Matteo Matteucci, for introducing me to the world of research and for providing me with the flexibility and resources to explore my own ideas. I would also like to thank Guillermo Gallego and Cornelia Fermüller for reviewing my thesis and for having spent very nice words about my work.

Thank you to everyone I met and collaborated with during my studies, especially everyone at the AIRLab, for making the journey of being a Ph.D. student easier and more enjoyable. Thanks to Andrea, who assisted me in publishing my first articles and pushed me to submit even when I was hesitant. Thanks to Marco for all your help. You always had valuable advice to give, and none of the research projects we worked on together would have been possible without you. A special thank goes to Francesco, who shared with me every teaching experience and day at the office. I learned a lot from your perseverance and dedication. Thanks also to the people at VANDAL I collaborated with. To Mirco, for his passion and dedication, and to Chiara for being first a student, then a co-author, and ultimately a friend.

Infine, un ringraziamento speciale va alla mia famiglia e ai miei amici. Questo non sarebbe stato possibile senza il vostro aiuto e supporto costante. Grazie a tutti i miei parenti per avermi supportato in ogni modo durante tutti questi anni di studi e ad Elena per avermi sempre incoraggiato e per essere stata la mia guida in questi ultimi mesi difficili.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DVS	Dynamic Vision Sensor
CD	Change Detection
EM	Exposure Measurement
ATIS	Asynchronous Time Based Image Sensor
DAVIS	Dynamic and Active pixel Vision Sensor
AER	Address-Event Representation

VLSI	Very Large Scale Integration
DL	Deep Learning
SNN	Spiking Neural Network
ANN	Artificial Neural Network
LIF	Leaky Integrate-And-Fire
MLP	Multi Layer Perceptron
CNN	Convolutional Neural Network
GCNN	Graph Convolutional Neural Network
LSTM	Long Short-Term Memory
UDA	Unsupervised Domain Adaptation
SLAM	Simultaneous Localisation and Mapping
BII	Brightness Increment Image
TS	Time Surface
FSAE	Filtered Surface of Active Events
SAE	Surface of Active Events
HOTS	Hierarchy Of Event-Based Time-Surfaces
HATS	Histograms of Time Surfaces
IETS	Inceptive Event Time-Surface
TBR	Temporal Binary Representation
EST	Event Spike Tensor
AMAE	Adaptive Motion-Agnostic Encoder
DiST	Discounted Sorted Timestamp Image
MVSEC	Multivehicle Stereo Event Camera Dataset

1 INTRODUCTION

Vision is arguably the most developed sense in primates. Through a complex network of receptors, neurons, and other specialized cells, the eyes and the brain interact almost simultaneously in the intricate physical process of viewing something. The eye first collects light, projects it to a membrane of receptors employing a system of adjustable lenses, converts it into a sequence of electrical signals, and finally transmits it to a system of complex neural pathways connecting the eye to the visual cortex. Roughly 50% of the cerebral cortex in macaque and 20-30% in humans is solely devoted to visual understanding and processing [9, 10], clearly showing both its intricacy and importance for our everyday life.

However, despite hundreds of years of research and investigation, the visual system is still far from being completely understood. The first studies on visual feature detectors date back to 1959 to the seminal works of Hubel and Wiesel [11–13], who discovered the presence of orientation-selective responses in the primary visual cortex. Based on these discoveries, several visual processing models have later been proposed [14, 15]. All these models rely on the core idea that high-level vision capabilities trace back to a cascade of hierarchically organized layers of cells providing progressively structured signals.

Modern computer vision relies on these same principles. However, despite recent breakthroughs in artificial intelligence, image processing, and micro-electronics, our brain still processes and interprets information in a far more efficient manner than computers do. Exploiting both analog and digital communication through trillions of synapses, it reaches an estimated processing speed of $\approx 10^{16}$ floating point operations per second (FLOPs), which is comparable to that of some of the fastest supercomputers ever built [16] ($\approx 5 \cdot 10^{17}$ FLOPs of the Fugaku, or $\approx 10^{16}$ FLOPs of the less powerful Marconi-100 [17]). However, while the brain consumes only ≈ 20 W of power, of which only a fraction is devoted to vision and its communication [18], these supercomputers require much more energy ($1\text{-}30 \cdot 10^6$ W), making the brain far more energy-efficient than supercomputers ($\approx 10^{15}$ FLOPs W^{-1} vs. $\approx 10^{10}$ FLOPs W^{-1}) [16]. It is thus apparent that we need to take more and more inspiration from neuro-biological systems if we aspire to make machines achieve the same levels of efficiency as our brains.

1.1 NEUROMORPHIC VISION DEVICES

Neuro-biological systems process information in an asynchronous, sparse, and energy-efficient manner, as opposed to computers, which employ synchronized logics, high-speed clock rates, and energy-demanding computation. The difference between biological retinas and traditional vision devices offer a clear illustration of this disparity. Conventional cameras gather visual information by taking full-frame pictures collecting light at a constant and preset rate. While these images closely resemble what we picture in our minds, they typically contain highly redundant information, unlike neural signals. Indeed, all sensor's pixels are synchronously read to form the final image regardless of whether something has changed since the last image has been taken.

This way of capturing visual information deviates from biological retinas, which leverage a much more efficient operating principle. Light hitting the retina initiates a sequence of chemical and electrical processes triggering impulses that ultimately reach the vision centers in the brain. Retinal ganglion cells collect neural signals from photoreceptors, altering their electrical charge and causing them to fire output signals after reaching a certain potential threshold. The same paradigm is adopted in the following processing layers of the visual cortex, which trigger neural impulses only when enough relevant visual information has accumulated. This procedure results in an entirely asynchronous and energy-efficient system capable of performing computation only when required.

Inspired by these mechanisms, neuromorphic vision devices, also known as event-based cameras, are vision sensors that attempt to emulate the functioning of biological retinas. An array of independent pixels generates an output signal, i.e., an event, anytime the local brightness level changes by a given threshold, simulating a simplified model of the retina's photosensitive membrane. Like in a biological vision system, information is produced asynchronously and only when needed, resulting in a very efficient device with many advantages over traditional ones. Many of the operating principles of these artificial retinas are shared with biological ones, giving us hope that we will soon be able to create visual systems with the same precision and efficiency as biological ones.

1.2 MOTIVATION: EFFICIENT AND EFFECTIVE NEURAL REPRESENTATIONS FOR ASYNCHRONOUS DATA

The reason for the success of modern computer vision systems resides in their ability to learn to reason directly from experience, without any prior knowledge on the task. Inspired by early neural computation models, many *Machine Learning* systems accomplish this by extracting meaningful features and combining them in complex and general patterns through a hierarchy of increasingly sophisticated representations. *Deep neural networks*, which are nowadays the beating heart of most vision systems, are capable of naturally extracting this hierarchy as they are organized on several processing layers that progressively refine information. Thanks to ad-hoc training algorithms, these layers can be jointly trained to extract effective intermediate representations that make even the most difficult visual tasks simple to solve.

How well such artificial systems perform on a particular task is often linked to how rich, informative, and general their internal representations are. However, the paradigm used to extract such representations is specifically designed to operate on dense visual encodings. As a result, learning to extract effective representations from the dense images acquired by standard cameras is remarkably easier than performing the same from the sparse and asynchronous output produced by event-based cameras. Indeed, while a single image directly conveys rich visual data, the same appearance information needs to be reconstructed from events through temporal reasoning, as it is spread across the sequence of asynchronous and incremental updates, making the task of learning such representations far more challenging. Designing novel mechanisms to extract effective representations and new computing paradigms capable of exploiting the asynchronous nature of events is thereby critical for unlocking event-based cameras' potential in modern computer vision architectures.

Spiking Neural Networks, a type of artificial neural network that mimics both the learning and dynamics of biological neurons, are appealing in event-based vision due to their energy efficiency and asynchronous processing paradigm. However, their complex dynamic makes learning with these architectures very challenging to accomplish, limiting their usage in complex visual tasks. As an alternative, and thanks to the success of deep learning in frame-based computer vision, researchers have recently started exploring the potentiality of deep neural networks, such as the representation power previously discussed, even in event-based vision. They are, however, less efficient than spiking networks and typically designed to process synchronous and dense data streams, making it difficult to exploit their ability to learn when asynchronous data is employed. The tradeoff between these two solutions, combined with the almost complementary benefits they provide, begs the question of whether it is possible to draw inspiration from the asynchronous and incremental processing of spiking networks to make deep neural network representations more suited at processing events. This research direction raises a number of questions that have yet to be answered. In particular, *is it possible to exploit deep neural networks at their full potential while still preserving the sparse and event-driven paradigm of event-based cameras? Can these neural networks be trained to extract effective representations even from such a sparse visual encoding?* And finally, *can these representations be trained to guarantee good performance even when target conditions deviate from the data used during training?*

1.3 OUTLINE AND MAIN CONTRIBUTIONS

This thesis describes our effort to provide an answer to the previous questions. We develop a framework based on a hybrid approach for converting traditional convolutional neural networks into event-based networks capable of asynchronous computation (Chapter 3). We propose a recurrent layer that maximizes task performance by learning to extract a task-specific event representation incrementally, driven by incoming events (Chapter 4). Finally, we demonstrate how domain adaptation techniques can be used to achieve good performance on real event-based data even when training is performed solely on simulated samples (Chapter 5). These techniques enable to train with supervision on a set of samples from a different distribution (i.e., *domain*) than that provided by the real camera while still attaining high performance at deployment time. We focus once again on representations and show that when these are domain-invariant, generalization to real-world even cameras improves significantly.

The thesis is structured as follows:

- Chapter 2 provides a general overview of the functioning of event-based cameras and their advantages over traditional vision devices. Recent advances in deep learning architectures for event-based processing are also discussed, focusing on event representations and processing approaches.

- Chapter 3 addresses the problem of combining event-driven computation with traditional convolutional neural networks. We develop a framework for building fully convolutional neural networks that efficiently handle events. We accomplish this by reformulating the traditional convolution and max-pooling operations. Thanks to an additional internal memory of previously extracted representations, these layers perform incremental computation sparsely, updating their internal state in an event-driven manner. We showcase these layers on the object detection task by implementing an event-based version of the YOLO [19] detection network.

  The chapter is based on [1], published at the Second International Workshop on Event-based Vision and Smart Cameras, held at the 2019 Conference on Computer Vision and Pattern Recognition (CVPR19).

- Chapter 4 presents a novel representation for event-based data that enables end-to-end learning of task-specific event representations. The proposed MatrixLSTM is a grid of LSTM [20] cells that process events as soon as they arrive by incrementally building a two-dimensional representation. This encoding can be attached to any state-of-the-art frame-based architecture for interfacing optimally with events. We show that MatrixLSTM can provide powerful representations in several tasks, including object recognition, optical flow prediction, and object detection.

The chapter is based on [2], published at the 2020 European Conference on Computer Vision (ECCV20).

- Chapter 5 addresses the scenario of training deep event-based neural networks when the only source of annotated event data comes from simulation. Event-based cameras have been commercialized only very recently [21] and remain a costly prototype sensor if compared to standard cameras, limiting the availability of large-scale datasets and making simulation a viable option for training deep neural networks. However, since simulation does not perfectly match a real event camera, networks trained solely on simulated events typically underperform when deployed on an actual device. Our research shows that this issue can be solved by repurposing standard domain adaptation techniques to the event-based paradigm. We propose DA4Event, a general procedure for training event-based neural networks on simulated data with minor performance loss. We study how the use of simulated data during training affects different event representations and how DA4Event can help in reducing performance degradation on object recognition and semantic segmentation tasks. We further extend the research by studying the more challenging scenario of simulating events from synthetic renderings. As datasets enabling this type of analysis are still lacking in the field, we develop N-ROD, a novel dataset obtained by extending the popular RGB-D Object Dataset [22], to foster future research on these topics.

  The chapter is based on [3] and [4]. The first is published at the IEEE Robotics and Automation Letters (RA-L) and presented at the 2021 International Conference on Intelligent Robots and Systems (IROS21). The second is published at the Third International Workshop on Event-based Vision, held at the 2021 Conference on Computer Vision and Pattern Recognition (CVPR21).

- Chapter 6 summarizes the work carried out during the thesis and suggests possible future directions.

2 EVENT-BASED CAMERAS IN DEEP LEARNING VISION

This chapter introduces the working principles of event-based cameras and their application in deep learning architectures for computer vision. Event-based cameras are described in the first half of the chapter, along with a discussion of their benefits over traditional vision devices and the fundamental differences between currently available models. The second part provides an overview of recent uses of event-based cameras in deep learning applications, with particular emphasis on event-based representations. Indeed, these constitute alternative solutions to the problems discussed in this thesis and the main elements on which we build upon. We refer the readers to the papers of Posch et al. [23] and Delbruck [24] for an in-depth overview of neuromorphic vision devices and to the survey of Gallego et al. [25] for a broader overview of recent event-camera models and their use in computer vision tasks.

2.1 SILICON RETINAS

The retina, a complex, multilayered neural network located at the rear hemisphere of the eye bulb, is the foundation of every biological visual system. It is composed of specialized cells and receptors that convert visual stimuli into electrical signals. Pigment molecules in the retina's innermost membrane layer absorb light as it travels through the semi-transparent layers of neurons that compose the retina epithelium. These photopigment cells are sensitive to different wavelengths of visible light, giving humans the ability to perceive primary colors. When light strikes these pigment molecules, chemical changes occur that modify the membrane potential of subsequent photoreceptor cells, commonly known as cones and rods, located in the inner synaptic layer. On/off bipolar cells connect these receptors to ganglion cells located in the outer synaptic layer, which react to visual stimuli by sending action potentials down the optic nerve. A schematic of the human retina is provided in Figure 2.1 on the following page.

Ganglion photoreceptors cleverly encode spatio-temporal visual information into action potential patterns known as spike-trains. These signals do not directly encode light or color intensity, contrary to what we picture in our minds, but instead they are correlated with movement and changes in brightness. In particular, as the magnitude of brightness changes increase, the rate of spike-trains produced by ganglion cells also increases, while it settles

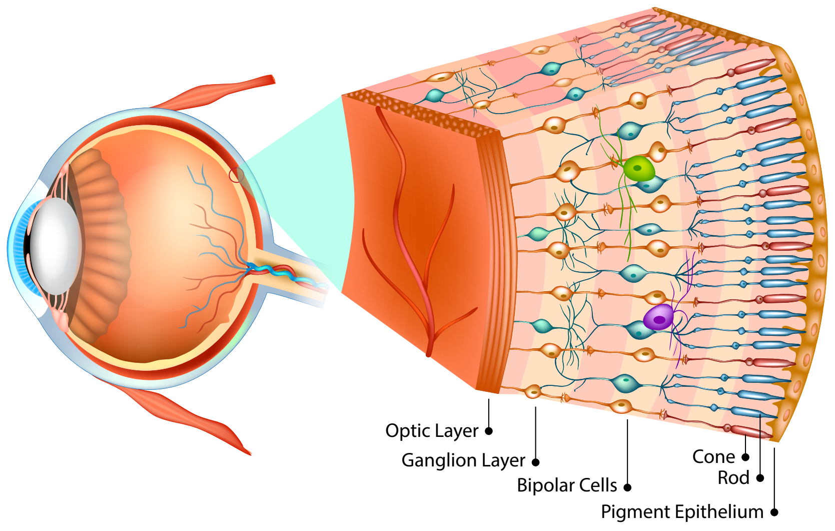


Figure 2.1: Cross section of the human eye's retina. The light striking the retina travels through all of the semi-transparent neural layers before reaching and activating the innermost rods and cones. These photoreceptors initiate communication back toward the ganglion cells and eventually through the optic nerve. *Image adapted from Sakurra, Adobe Stock.*

when no movement or brightness variations are detected. Spike-trains are then converted back into continuous signals in the first layers of the visual cortex by dendritic integration in postsynaptic terminals, where high-level visual understanding happens. This mechanism is complemented by other highly developed biological adaptive filtering and sampling processes, which contribute to making the spike-train encoding scheme incredibly efficient. These include, among many others [26], spatiotemporal filters reducing redundancy and noise and a rectification mechanism limiting the spike-firing rates of bipolar cells.

This highly efficient biological transmission protocol makes the vision system very hard to beat by modern digital systems. As reported by Posch et al. [23] and Echeverri [27], in order to match the human retina characteristics in terms of dynamic range (over 100 dB), spatial ($\sim$ 92 million brightness-sensitive rod cells and $\sim$ 4.6 million color-sensitive cone cells [28]) and temporal ($\sim$ 24 fps [27]) resolution, a typical image sensor sampling at the Nyquist frequency[1] would have to transmit more than 20 Gb/s. The optic nerve, on the other hand, transports only around 20 Mb/s to the visual cortex by encoding two bits of information each spike, three orders of magnitude less.

1 In signal processing, the theorem of Nyquist-Shannon defines the Nyquist frequency as the minimum frequency needed to sample a continuous signal without losing information. In particular, the theorem states that the minimum sampling frequency necessary to reconstruct the original analog signal must be greater than double its maximum frequency.

The efficiency of biological neural systems inspired researchers to design new types of architectures and algorithms that, by mimicking neuro-biological processes, could benefit from their efficient computation and communication. This research area has yielded a number of interesting and successful outcomes [29–33], with silicon retinas [34, 35] being the most mature technology. The following sections will cover the functioning of neuromorphic retinas and how information is processed and conveyed in these devices.

2.1.1 Address-Event Representation

What makes the biological vision system very efficient is, to a large extent, its highly evolved communication system. Reproducing similar mechanisms in the design of neuromorphic devices is an essential step in the journey toward replicating their performance. The brain is composed of millions of neural cells interconnected with point-to-point connections arranged in a three-dimensional space. Due to wiring complexity and space constraints, however, this configuration is regrettably not practical to implement with current Very Large Scale Integration (VLSI) technologies. Nevertheless, as discussed earlier, neurons are not constantly active, and they generate spike-trains at a far slower rate than the bandwidth of existing digital buses. Bio-inspired units in neuromorphic devices (such as asynchronous neurons and photoreceptors) behave similarly, typically generating spikes at only $10 - 1000$ Hz [23]. Neuromorphic devices exploit this behavior and replace explicit point-to-point connections with just a few high-speed physical buses, shared across all units, that are time-multiplexed to enable multiple units to interact simultaneously [23].

The most widespread protocol following this operating principle for intra-chip and inter-chip communication in neuromorphic devices is the Address-Event Representation (AER). The AER was developed 30 years ago by Caltech institute [36, 37] as an asynchronous communication mechanism, and it is nowadays used in many retinomorphic vision sensors. Each unit (e.g., a neuron or a pixel) is given a unique address that identifies the source of the signal. Whenever a unit outputs a signal, an event is generated consisting of its unique address and any extra information produced. As time is intrinsically stored in the timing of the events, it is usually not included within the packets, and it is only added when processing takes place offline or in a synchronous device, such as FPGAs or traditional digital processors [23].

The AER protocol suits neuromorphic computation very well, as sensing and processing units typically interact through asynchronous computation. An example is event-based cameras, where the protocol is implemented by assigning each pixel to an address encoding its spatial position.

2.1.2 Neuromorphic Imaging Devices

Primarily aimed at reproducing biological models, the earliest neuromorphic imaging devices were employed as a way of testing and verifying neurobiological theories [23]. The first silicon retina is due to Mahowald and Mead [38], who implemented a VLSI architecture featuring silicon bipolar and photoreceptors cells that react to light, generating output spike-trains encoded using the AER protocol. This design was later improved by Zaghloul and Boahen [39] [40], who implemented more realistic models of the retina, even including high-level visual processing layers. Recent designs are now increasingly prioritizing performance over faithfully reproducing biological models. In general, they can be categorized by the mechanism used to encode visual stimuli [41]. Although several prototypes have been proposed [42], here we focus on the most popular ones, distinguishing between those that only detect brightness changes (*Change Detection (CD)* devices) and those that additionally encode grayscale, or *Exposure Measurement (EM)*, readings alongside changes.

CHANGE DETECTION (CD) DEVICES. In the same way as the neural layers of the biological retina create a spike whenever they sense a change in brightness, pixels in change detection devices generate a signal, i.e., an event, in response to changes in light. The sensor is composed of a matrix of pixels that independently track the brightness level hitting their photodiodes. Whenever the logarithmic intensity $L = log(I)$ at pixel location (x_i, y_i) changes of a quantity above or below a predefined logarithmic threshold $C > 0$, an event

$$e_i = \{x_i, y_i, t_i, p_i\} \tag{2.1}$$

is generated, specifying the position (x_i, y_i) of the pixel, the time instant t_i the change has been detected, and a polarity bit $p_i \in \{-1, 1\}$ encoding whether the intensity decreased or increased. Two consecutive events e_i and e_j generated from the same pixel location (x, y) safisfy the following equation [25]:

$$\Delta L(e_i, e_j) = L(x, y, t_i) - L(x, y, t_j) = p_i \cdot C, \quad \text{with } \Delta t_{ij} = t_i - t_j > 0, \tag{2.2}$$

where Δt_{ij} is the temporal difference between the two events and L is the logarithmic brightness intensity. The output of the sensor $\mathcal{E} = \{e_i \mid t_i \geq t_j, \quad \forall j < i\}$ is thus an ordered sequence of asynchronous events describing the dynamics of what is changing or moving in the scene. The rate of events generated by CD devices is data-driven, just like the spike-train frequency in photoreceptors of the retina. That is, many events are produced when capturing a scene involving fast movements, while little to no events are generated in a static one.

The *Dynamic Vision Sensor (DVS)* falls in the class of CD devices. It was first introduced by Lichtsteiner et al. [21] and then later improved by Serrano-Gotarredona and Linares-Barranco [43], who increased the sensitivity of the

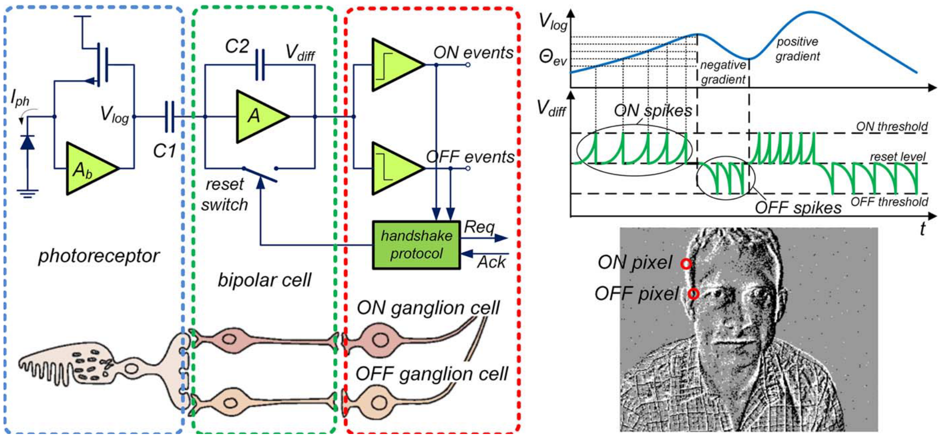


Figure 2.2: A typical DVS sensor compared to a schematized structure of the retina. **(left)** Photoreceptors, bipolar and ganglion cells are implemented in hardware with dedicated circuits to replicate the retina's functioning. **(top-right)** A log-intensity signal V_{log} at a particular DVS's pixel and the generated output events. The change in brightness at each pixel location is monitored through V_{diff}. A new event is produced whenever V_{diff} reaches a preset positive or negative theshold C. **(bottom-right)** Positive and negative events generated by an event-camera moved in from of a person. *Figure taken from Posch et al. [23]. Copyright* ©2014 *IEEE.*

pixels and reduced their size at the expense of higher power consumption. A schematic of the sensor, taken from Posch et al. [23], is provided in Figure 2.2. An amplified photodiode generates a voltage signal V_{log} proportional to the light's log intensity. This information is amplified to V_{diff} according to the ratio $C1/C2$ and read by two comparators that generate an output event whenever V_{diff} exceeds the predefined thresholds. In that case, a reset switch is activated, which pulls the gain amplifier's output back to an initial voltage V_0 to enable the reading of a new intensity value [21, 44]. Other variants of the sensor exist, like the sensor developed by Berner and Delbruck [45] which is sensitive to color variations, and its extension, named *cDVS* [46], that combines log-intensity and color changes detection, thus producing events with color information. The industry is actively pushing the pixel resolution of DVS sensor, with the latest sensor from Prophesee [34] and Samsung [35] now reaching 1Mpx resolution.

CHANGE DETECTION AND EXPOSURE MEASUREMENT (EM) DEVICES. Neurobiological studies revealed the existence of two primary pathways in the primary visual cortex [47, 48], one that quickly responds to motion and luminance changes, and a slower one sensitive to textures and chromatic modulation [2]. Event-based cameras such as the Dynamic Vision Sensor (DVS),

2 The retina is composed of three distinct types of ganglion cells known as magno, parvo and konio cells, which give rise to three homonymous pathways. The magno and parvo pathways are those discussed in the text, with magno cells producing low-latency motion-sensitive responses and parvo cells processing details, textures, and colors. The third pathway, which

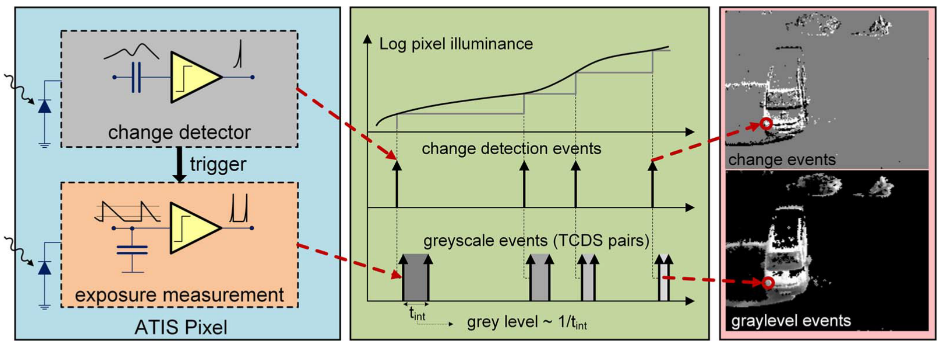


Figure 2.3: Functioning of an Asynchronous Time Based Image Sensor. Whenever the DVS detects a change in brightness, an exposure's measurement at that pixel location is triggered. The ATIS sensor produces two types of events, visualized on the right, which encode change and exposure information. *Figure taken from Posch et al. [23]. Copyright* ©2014 *IEEE.*

being low latency and sensitive to dynamic information, clearly mimic the first processing pathways. In contrast, conventional frame-based cameras can be functionally attributed to the second means of visual processing, as they convey slow but texture-rich information. Inspired by these studies, more advanced event-based camera designs propose combining brightness change detection with the direct measurement of pixels' intensity values, thus exploiting the benefits of both vision paradigms and more faithfully reproducing information available in the primary visual cortex.

The *Asynchronous Time Based Image Sensor* (*ATIS*) developed by Posch et al. [47] is the first device to provide this sort of combined visual information. Besides traditional events encoding brightness changes (CD events), this sensor generates additional events encoding exposure measurements (EM events) analogous to that conveyed by grayscale images. However, instead of synchronously sensing intensity values as in traditional images, these sensors still exploit an asynchronous behavior to encode pixel brightness values only when these change. This sensing paradigm is realized by extending a traditional DVS pixel with an additional exposure measurement circuit, which is triggered by the DVS, independently and asynchronously, whenever it detects a change. When this happens, two consecutive AER events are generated, encoding the absolute value of the pixel as the relative time difference between the two events, measured as the time a photodiode takes to integrate between two threshold values. A schematic is provided in Figure 2.3.

Combining the asynchronous approach of the DVS sensor with intensity readouts, ATIS sensors are capable of very high video compression and temporal resolution. However, since the time encoding is inversely proportional to the intensity, dark objects may produce artifacts, as the intensity readout could be interrupted by new events. Moreover, the size of the pixel limits the

comes from konio cells, has far fewer cells than the other two and, as a result, its physiological response has yet to be fully characterized [48].

fill factor[3] in ATIS sensors, as they effectively combine two sub-pixels for CD and EM measurements.

An alternative hybrid approach is that implemented in the *Dynamic and Active pixel Vision Sensor (DAVIS)* developed by Berner et al. [49]. Contrary to the ATIS, which only provides asynchronous information, the DAVIS sensor combines synchronous frame-based intensity readings with the asynchronous events of a DVS. The same pixel is used both to produce full-frame grayscale pictures with traditional global or rolling shutter mechanisms and to detect event-based illumination changes asynchronously. Since only a small readout circuit is required to extend a DVS pixel to operate in both modalities, increasing the pixel size by only 5% [23], the fill factor is not harmed as much as in an ATIS device. The latest *DAVIS346* model [25] features a 346×260 resolution and a variant named *CDAVIS* [50] extends the sensor to output RGBW frames at VGA resolution as well as asynchronous monochrome QVGA events. A similar design is also proposed in the SDAVIS192 [51], which integrates a DAVIS sensor with a proper color filter array.

2.1.3 Advantages of Event-Based Cameras

Thanks to their unconventional way of capturing the scene, event-based cameras possess a number of benefits over traditional imaging devices [25]. These benefits, which originate from the event-based camera functioning, as depicted in Figure 2.4 on the following page, are detailed in the following:

- *Low latency and temporal resolution*: pixels in DVS sensors implement a simple yet very fast analog circuitry that enables detection of brightness changes at microsecond temporal resolution. As every pixel in the sensor operates asynchronously and independently, brightness changes (i.e., events) are transmitted as they occur, with minimum delay. This is different from standard cameras, where pixels must wait a global exposure time before reading. As a result, event-based cameras can capture rapid movements without motion blur, a common problem with frame-based devices.
- *High Dynamic Range (HDR)*: the exposure time in an event-based camera is not fixed everywhere, but each pixel is instead free to operate at its own exposure. As a consequence, event cameras exhibit a dynamic range over 120 dB, which is well above the 60 dB of traditional cameras. Moreover, since event-based cameras sense light at a logarithmic scale, their response to light is consistent over a wide range of intensities. These two characteristics give event-based cameras an edge over standard devices when operating in unfavorable lighting conditions.

3 The fill factor of an imaging sensor is the percentage of its total surface that is sensitive to light. Transistors, capacitors, wiring, and registers reduce this area, affecting the sensitivity of the sensor to light, and thus its ability to sense in low light conditions.

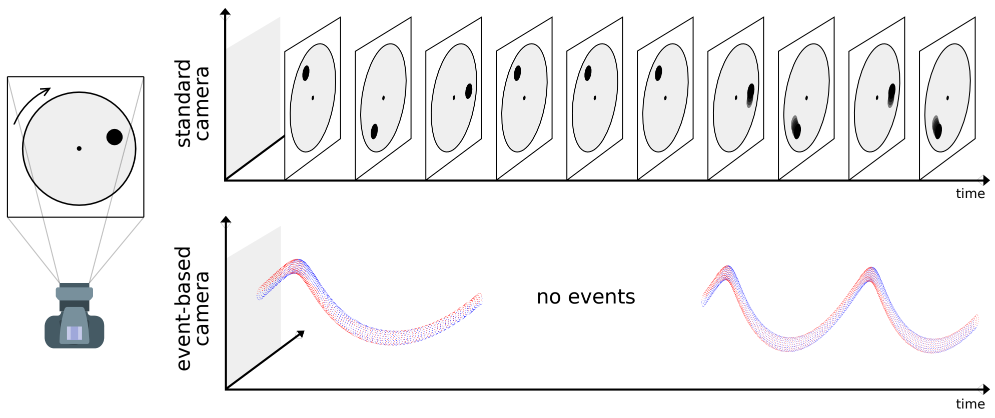


Figure 2.4: Comparison between a standard camera's output with that of an event-based camera capturing a rotating disk. While a standard camera captures full-frame pictures at predefined rates, an event camera only encodes changes in the scene. As a result, the background of the disk is not captured since it does not change intensity, thus highly reducing information redundancy. Similarly, when the disk stops, no event is generated. Finally, the high temporal resolution eliminates motion blur effects, which instead affect standard devices in the presence of high-speed motion. *Figure re-created from Kim [52] and inspired by Mueggler et al. [53].*

- *Low power consumption*: as event-cameras are data-driven, power is only consumed when something changes. As a result, power consumption is in the order of 100 mW in most cameras and can reach even 10μW in some prototypes. This is a very desirable property, especially in wearable devices and mobile robots.

2.2 DEEP LEARNING APPROACHES TO EVENT-CAMERAS

Drawing inspiration from their functioning, event-based cameras share several benefits of biological retinas, which give promise in many situations where standard devices are limited in their capabilities. Fast motions, like in drones or moving cars, and abrupt brightness changes, such as when exiting a dark tunnel or driving with frontal sunlight, are all challenging conditions in which they excel. Taking advantage of these strengths, researchers have started creating new algorithms to tackle traditional computer vision problems under this new way of sensing the world. These algorithms range from designs that focus on exploiting and maintaining event-camera properties during computation, often using specialized hardware to accomplish asynchronous and minimum delay computing, to algorithms that use events in combination with standard devices to boost the performance of existing computer vision algorithms.

Despite event-based cameras advantages, developing algorithms that efficiently interface with event measurements is not straightforward. Events are asynchronous and spatially sparse, while traditional vision algorithms often require dense and information-rich representations to operate effectively. Moreover, event measurements do not only depend on the scene brightness but also on the relative movement between the camera and the scene, which poses new challenges and opportunities in algorithm designs.

Several hand-engineered methods, designed ad-hoc for event-based processing, have shown great performance on a number of computer vision tasks, such as optical flow prediction [54–56], depth estimation [57, 58], pose estimation [59], and many others. These solutions completely rethink the problem under the paradigm shift imposed by event-based cameras. On the contrary, Deep Learning (DL) approaches have shown in the past to provide a high level of performance on several traditional computer vision tasks without the need of explicitly modeling the problem. Artificial Neural Networks (ANNs) are often easier to adapt than conventional algorithms, and they have proven to work well when applied to similar visual encodings, such as point clouds [60, 61].

Although the lack of large annotated datasets is still hindering their true potential in the field, DL approaches are becoming very popular in the event-based research community. This section provides a general overview of recent applications of DL approaches to event-based cameras, focusing on the aspect relevant to this thesis. As a thorough discussion of DL techniques is out of the scope of this thesis, we refer the reader to the comprehensive Deep Learning textbook of Goodfellow et al. [62] for details on the DL tools here discussed. Moreover, a broader overview of event cameras applications, covering the full spectrum of computer vision problems, is also provided by Gallego et al. [25] in their recent survey.

2.2.1 Processing Paradigms

Events produced by neuromorphic cameras convey very little information, as, if considered alone, they only signal a change of brightness at a particular location in space and time. Computer vision algorithms must rely on some aggregation mechanism to correlate events that occurred in a spatio-temporal neighborhood to perform any meaningful prediction. In this respect, two general paradigms for event processing have emerged in the research community, which differentiate depending on how events are consumed during computation.

EVENT-BY-EVENT COMPUTATION. This approach processes each event as it arrives, incrementally and asynchronously updating the algorithm's output, thus achieving minimum reaction times. These algorithms often rely on an internal state that is updated upon the event's arrival, which constitutes the algorithm's belief about the content of the scene and its evolution in time.

Spiking Neural Networks (SNNs) [63] are the leading approach when it comes to neural systems for asynchronous spike-based computation. Indeed, being inspired by the functioning of biological neural networks, they naturally fit event-based processing. They are composed of independent neuron-like units that asynchronously react to incoming spikes events by aggregating spikes coming from other neurons and firing themself whenever enough relevant information is accumulated. The neurons' membrane potential constitutes the SNN's internal memory, which gets updated each time an event arrives, asynchronously, in an event-by-event manner. They have been applied to event-based processing in several tasks, such as edge detection [64, 65], object classification [66, 67] and hand-gestures recognition [68]. Given their origin, they are typically trained with unsupervised biologically inspired learning rules [69, 70], but they often achieve better performance when hybrid approaches leveraging traditional gradient-based learning methods are applied. Several works [67, 71–73] use standard ANNs as a proxy for learning their synaptic weights and overcoming their non-differentiability, enabling the training of even complex, multilayered architectures.

Another typical class of event-by-event algorithms relies on filtering algorithms for estimating some system's unknowns. They are traditionally designed to operate on an incomplete and potentially noisy set of observations, and they commonly incorporate the notion of a state, which is continuously updated as new observations become available. These two characteristics make them particularly suited for event-based asynchronous processing. For these reasons, a number of event-based algorithms based on filters have been proposed over the years, whether deterministic or stochastic. Typical applications include *Simultaneous Localisation and Mapping (SLAM)* algorithms [74–76], where filters represent the standard even in conventional computer vision approaches, *noise filtering* mechanisms [77, 78], as well as *image and video reconstruction* algorithms [79, 80] converting event camera output to grayscales.

Deterministic filters have also been proposed to implement asynchronous convolution for event-based artificial neural networks [71, 81–83] and feature extraction in general [84, 85]. They exploit the sparse encoding provided by event cameras to implement fast operations at low computational cost, processing only local neighborhoods around incoming events rather than the entire image. Scheerlinck et al. [81] formulate the problem as a continuous-time filter which is evaluated asynchronously, at discrete time instances, by solving associated ordinary differential equations. In this thesis (see Chapter 3 and the related paper [1]), we propose a recurrent formulation of the convolution and max-pooling operations that leverage an internal state to asynchronously and locally compute convolutions in ReLU [86] based deep neural networks. The approach was later extended by Messikommer et al. [87] who focus on a different class of convolutional networks and propose a procedure agnostic of the input event representation.

BATCH-BASED COMPUTATION. This second approach waits for a batch of events to arrive and process them all simultaneously, sacrificing response time for performance. When processing the event stream offline, a sliding window approach is commonly used to construct the batch by either fixing the number of events or the time period of each window. As in the event-by-event approach, an internal state may also be used to extend the context beyond the batch and integrate information from previous computations. Batch-based approaches usually convert the input stream into structured representations providing richer information than single events, as they enable events to be correlated in space and time. Some pre-processing mechanisms still preserve an asynchronous and sparse encoding [88–91], while others prefer to convert the event stream into a densified representation [92–94].

Several batch-based algorithms leverage the concept of *time surface* [84, 85], a dense representation extracting local spatio-temporal motion footprints from each event. While these technically fall under the event-by-event approaches, as they can be computed asynchronously leveraging a memory mechanism, in practice, they are often used within batch-based methods that accumulate and process them to accomplish several computer vision tasks. Applications of time surfaces include 3D stereo reconstruction [95, 96], stereo depth estimation [97], SLAM [98], corner detection and tracking [99–101] as well as object classification [84, 85].

Few batch-based approaches avoid creating densified intermediate representations to preserve event cameras' benefits given by their sparse encoding. Methods based on graphs interpret events as vertices of a graph interconnected in local spatio-temporal neighborhoods. Graphs have recently been used in motion segmentation algorithms [88], and graph neural networks have successfully been applied in object classification [89, 90] and action recognition [90, 91] tasks. A similar approach interprets event streams as 3D point clouds where the depth dimension is replaced with the temporal one. Deep learning networks, such as PointNet [61] and PointNet++ [60], have shown good performance when applied to small temporal windows of events in object recognition and semantic segmentation [102], as well as gesture recognition [103] tasks.

Far more popular are methods that convert the event stream into dense representations commonly known as *event frames*. These representations resemble conventional frames and are thus easy to integrate into conventional computer vision pipelines. DL approaches based on these grid-like encodings have been applied to a number of tasks, including object classification [2, 92] and detection [104], semantic segmentation [105], depth and optical flow [54–56] estimation, and image reconstruction [106–108].

Grid-like event representations are discussed in detail in the next section, as they constitute the basis of some of the works presented in this thesis. We present a novel bio-inspired representation in Chapter 3, an end-to-end trainable layer for learning to extract an event-frame optimized for a given task in

Table 2.1: Comparison of grid-like event representations used in deep learning frameworks. H and W denote the representation's spatial dimensions while B the number of optional temporal bins. The *Time* and *Polarity* columns indicate how the corresponding event feature is encoded: *None* if the value is not included, *Chan* if it is represented as a channel dimension, and *Feat* if it is encoded in the pixel value. The column *Motion Invariant* reports if either speed or direction invariant processing is performed, with details discussed in the text. Finally, the Matrix-LSTM [2] representation is discussed extensively in Chapter 4 on page 71. *Table adapted from Gehrig et al. [92].*

Event Representation	Dimensions	Time	Polarity	Motion Invariant	Voxel Based	Learned	Characteristics
Event Count [109]	$H \times W \times 2$	None	Chan				Number of events
BII [110]	$H \times W$	None	Feat				Sum of event polarities
SAE [111]	$H \times W \times 2$	Feat	Chan				Most recent timestamp
FSAE [100]	$H \times W \times 2$	Feat	Chan				Filtered most recent timestamp
DiST [112]	$H \times W \times 2$	Feat	Chan	✓			Discounted sorted timestamp
IETS [113]	$H \times W \times 2$	Feat	Chan				Edge-only filtered timestamps
Motion-comp. counts [114]	$H \times W$	None	None	✓			Motion-compensated event counts
Motion-comp. time [114]	$H \times W$	Feat	None	✓			Avg. of Motion-compensated time
Linear TS [41]	$H \times W$	Feat	Feat				Linearly decaying time
Exp. TS, HOTS [41, 84]	$H \times W$	Feat	Feat				Exp. decaying time
Accum. TS [41]	$H \times W$	Feat	Feat				Exp. decaying time with accum.
HATS [85]	$H \times W \times 2$	Feat	Chan				Histogram of avg. time surfaces
Motion-corrected [115]	$H \times W \times 2$	None	Chan	✓			Motion-corrected event counts
Voxel-Grid Image [55]	$H \times W \times B$	Feat	Feat		✓		Weighted sum of event polarities
Multi-Channel BII [106]	$H \times W \times B$	Feat	Feat		✓		Sum of event polarities
TBR [93]	$H \times W$	None	None				Number of events
AMAE [94]	$H \times W$	Feat	None	✓		✓	Adaptive temporal feature
EST [92]	$H \times W \times B \times 2$	Feat & Chan	Chan		✓	✓	Temporal encoding
Matrix-LSTM [2]	$H \times W \times B \times C$	Feat & Chan	Feat & Chan		✓	✓	Spatial-temporal encoding

Chapter 4, while in Chapter 5 we compare the generalization performance of several event representations when domain shifts are considered.

2.2.2 Grid-like Event Representations

Given a stream of asynchronous events $\mathcal{E} = \{e_i = (x_i, y_i, t_i, p_i)\}_{i=1}^{N}$ spanning a temporal window ΔT, the process of extracting a grid-like representation can be described as a function $\Phi_{\mathcal{R}}$ mapping $\mathcal{E}$ into a volume $\mathcal{R}_{\mathcal{E}} \in \mathbb{R}^{H \times W \times F}$ with F features per pixel.

Several representations of this kind have been proposed over the years, mostly differing in how pixel features are computed and aggregated over time. Gehrig et al. [92] show that most of them can be unified under a unique framework by rephrasing representations as kernel convolutions on the *event field*, i.e., a discretized four-dimensional manifold spanning the two spatial dimensions, as well as the time and polarity dimensions. Grid-like representations are usually hand-crafted, meaning that the transformation mapping the event stream into $\mathcal{R}_{\mathcal{E}}$ is fixed, and it does not depend on the task at hand. Recently, a few works [2, 92, 94] propose to embed neural layers within $\Phi_{\mathcal{R}}$ and jointly train them together with the rest of the network to learn to extract task-specific representations.

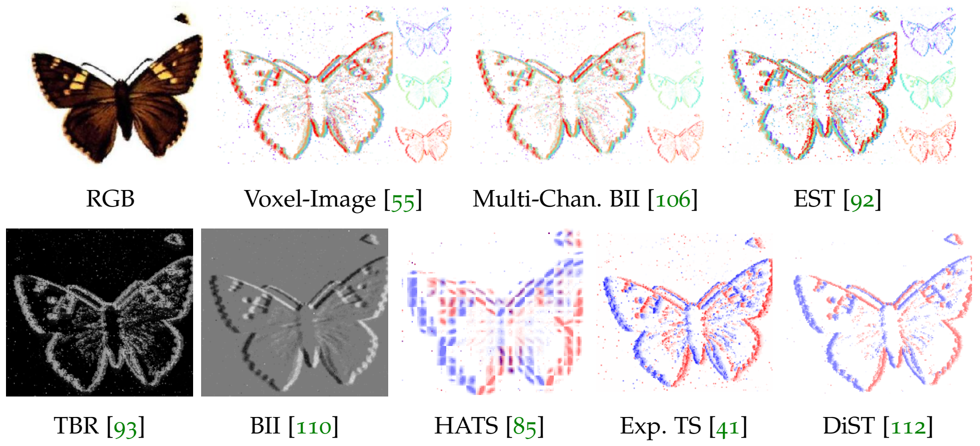


Figure 2.5: Qualitative comparison between different event representations using the last 100ms (third saccade) of the *butterfly_0006* N-Caltech101 [116] sample. The first row shows three voxel-based representations making use of 3 bins, where channels are represented with different colors and shown both combined and separately at the side of the figure. The Voxel Image [55] and the multi-channel BII [106] are very similar, with the Voxel Image introducing an additional temporal scaling whose effect is noticible in the first channel, in this example. The second row depicts two single channel representations followed by three two-channel representations, where features of positive events are shown in red and that of negative events in blue.

We provide an overview of the most popular representations in the following, focusing on those that have been used in previous works as the input of deep neural networks. A comparison between these representations is given in Table 2.1 on the preceding page, while visual depictions are provided in Figure 2.5.

SIMPLE AGGREGATION METHODS. Early deep neural networks' applications to event-based cameras employed simple aggregation procedures to condense event data. These were often inspired by other event-based algorithms [41, 100, 110, 115, 117] that focused on providing low-latency solutions to several computer vision applications.

In these representations, the sequence of events $\mathcal{E}^{(x,y,p)} = \{e_i \in \mathcal{E} \mid x_i = x, y_i = y, p_i = p\}$ arriving at each pixel location, often split by polarity, is aggregated into a single pixel feature through basic aggregation methods.

The *event counts* representation [54, 109] uses the cardinality $|\cdot|$ of $\mathcal{E}^{(x,y,p)}$ to aggregate event sequences, discarding any temporal information:

$$\mathcal{R}_{\mathcal{E}}^{\text{count}}(x, y, p) = |\, \mathcal{E}^{(x,y,p)} \,| \,. \tag{2.3}$$

On the contrary, the *Surface of Active Events (SAE)* [41, 111] only preserves recent temporal information by keeping track of the timestamp of the last event received at each pixel:

$$\mathcal{R}_{\mathcal{E}}^{\text{SAE}}(x,y,p) = \max_{i \in \mathcal{E}^{(x,y,p)}} t_i, \tag{2.4}$$

where, to simplify notation, we indicate as $i \in \mathcal{E}^{(x,y,p)}$ the indices of the events e_i in the sequence. As time is inherently monotonically increasing, the function in Equation 2.4 describes a monotonically increasing surface [41, 111], hence its name. When time-modulated kernel functions are applied over this representation, highlighting more descriptive features of the event stream, the resulting representations are usually called *time surfaces* [41, 84], as described later in the text.

Finally, polarity information is used within the *Brightness Increment Image (BII)* [110] in which the pixel feature is computed summing together the polarity value $p_i \in \{-1, 1\}$ of all the events that fired at the same location:

$$\mathcal{R}_{\mathcal{E}}^{\text{BII}}(x,y) = \sum_{i \in \mathcal{E}^{(x,y)}} C \cdot p_i, \tag{2.5}$$

where C is the camera's contrast threshold (sometimes set equal to 1 [106, 118]), and we dropped the parameter p in $\mathcal{E}^{(x,y)}$ to indicate the set of events fired at pixel location (x, y), regardless of their polarity value.

While still effective in some applications [105], these representations exhibit some issues that limit their performance when used with ordinary convolutional neural networks. They are usually not resilient to noise and condense all events into a single two-dimensional frame, thus sacrificing temporal resolution. Extensions of these simple representations were later proposed to address some of these limitations. The *max* operator in SAE is commonly replaced with an average over the events' timestamps [113], and filtering mechanisms are optionally used [100, 113]. The Filtered Surface of Active Events (FSAE) [100] reduces artifacts occurring at high event rates by artificially limiting the number of representation's updates, while the Inceptive Event Time-Surface (IETS) [113] further discards events to focus more on the shape of moving objects than the magnitude of intensity changes[4]. With a similar aim, the extension of the event counts representation proposed by Rebecq et al. [115] introduces motion compensation from IMU measurements, thus reducing motion-blur-like artifacts caused by moving edges. More sophisticated solutions involve motion compensation algorithms [114, 119–121] that estimate the optical flow of the events and can be employed as pre-processing steps to adjust for motion prior to computing the event representations, as done by Mitrokhin et al. [114] on regular count

4 Notice that these representations, contrary to what the names suggest, do not apply a time-dependent kernel over the surface of most recent events. Instead, a simple average is used to aggregate all pixel timestamps into a unique feature value, effectively replacing the *max* operator in SAE with an average.

and timestamp images. When used with deep neural networks, all these representations are typically normalized to lie within the $[0, 1]$ value ranges, as commonly done with images.

DISCOUNTED SORTED TIMESTAMP IMAGE (DiST). Another extension of the SAE representation is that proposed very recently by Kim et al. [112]. They exploit the spatial neighborhood around each pixel to suppress the contribution of noisy events and quantize temporal features in such a way as to remove the time scale from temporal delays. A Discounted Time Image (DiT) is first computed starting from SAE as follows:

$$\mathcal{R}_{\mathcal{E}}^{\text{DiT}}(x, y, p) = \mathcal{R}_{\mathcal{E}}^{\text{SAE}}(x, y, p) - \alpha D(x, y, p), \tag{2.6}$$

where α is a constant scale parameter and $D(x, y, p)$ is a pixel-wise discount factor defined as:

$$D(x, y, p) = \frac{\max\limits_{i \in \mathcal{N}_r(x,y,p)} \mathcal{R}_{\mathcal{E}}^{\text{SAE}}(x_i, y_i, p) - \min\limits_{i \in \mathcal{N}_r(x,y,p)} \mathcal{R}_{\mathcal{E}}^{\text{SAE}}(x_i, y_i, p)}{\mid \mathcal{N}_r(x, y, p) \mid}. \tag{2.7}$$

$D(x, y, p)$ computes the difference between the latest and oldest timestamps in a neighborhood $\mathcal{N}_r(x, y, p)$ of radius r around each pixel location (x, y), normalized by the events count of the neighborhood. Events triggered by hot pixels are associated with a high discount factor since, when isolated, their neighborhood has a low events count. The same happens in regions affected by background noise, as the event frequency is typically low, and thus the numerator in $D(x, y, p)$ large.

A Discounted Sorted Timestamp Image (DiST) representation is finally computed from $\mathcal{R}_{\mathcal{E}}^{\text{DiT}}$ by first building a matrix for each polarity $ArgSort_p(\mathcal{R}_{\mathcal{E}}^{\text{DiT}}) \in \mathbb{Z}_0^{+H \times W}$ replacing each value in $\mathcal{R}_{\mathcal{E}}^{\text{DiT}}$ with the position that value has in the global sorting of all the discounted time features. The final representation is the normalized version of that matrix:

$$\mathcal{R}_{\mathcal{E}}^{\text{DiST}}(x, y, p) = \frac{ArgSort_p(\mathcal{R}_{\mathcal{E}}^{\text{DiT}})(x, y)}{\max_{x,y} ArgSort_p(\mathcal{R}_{\mathcal{E}}^{\text{DiT}})}. \tag{2.8}$$

By replacing temporal features with their normalized sorting index, the relative order between features is maintained, but their difference, which originally encoded the delay between two events, is replaced with a constant value. As a result, the representation becomes invariant to the scale of temporal delays and, therefore, less sensitive to the motion speed [112].

TIME SURFACES. Building upon SAE, several extended representations can be computed by applying time-sensitive kernels over the events' activity surface. The SAE representation, keeping track of the timestamp of latest event in each pixel, successfully encapsulates many information about the motion of brightness gradients in the scene. Indeed, in this representation, a moving edge creates a trail pattern whose precise timestamp information can be exploited to derive both direction and speed. However, in certain applications, it may be desirable to forget about events that occurred far in the past, decaying old information as time passes and giving the latest events greater importance. The resulting representations are commonly known as *Time Surface (TS)*.

Several temporal kernels have been proposed over the years [41, 84, 85], with the majority of them differing in terms of the approach employed to decay information. These same, or very similar, activations are also often called *membrane potentials* when used to define the behaviour of SNN neurons [66, 122, 123]. The simplest of these decay timestamps either linearly [41]

$$\Lambda(x,y,t) = \begin{cases} P(x,y)\cdot\left(1+\frac{\mathcal{R}^{\text{SAE}}(x,y)-t}{\tau}\right), & \mathcal{R}^{\text{SAE}}(x,y)-t >= \tau \\ 0, & \mathcal{R}^{\text{SAE}}(x,y)-t < \tau \end{cases}, \tag{2.9}$$

or exponentially [41, 84]

$$\Gamma(x,y,t) = \begin{cases} P(x,y)\cdot \mathrm{e}^{\left(\frac{\mathcal{R}^{\text{SAE}}(x,y)-t}{\tau}\right)}, & \mathcal{R}^{\text{SAE}}(x,y) <= t \\ 0, & \mathcal{R}^{\text{SAE}}(x,y) > t \end{cases} \tag{2.10}$$

as time progresses, with the time constant τ defining the duration after which each event's contribution is decayed to 0, and $P(x,y)$ providing the polarity of the latest event in each pixel location. The result is a temporal feature within the $[-1,1]$ range, with the polarity determining the initial value (either 1 or -1) of each pixel feature. In order to avoid overriding previous information when a new event arrives, the accumulating exponential kernel [41] extends the exponential kernel by incorporating the value the pixel had at the time $\tau_e^{(x,y)}$ a new event was received at (x,y):

$$\Phi(x,y,t) = \begin{cases} \left(\Phi\left(x,y,\tau_e^{(x,y)}\right)+P(x,y)\right)\cdot \mathrm{e}^{\left(\frac{\mathcal{R}^{\text{SAE}}(x,y)-t}{\tau}\right)}, & \mathcal{R}^{\text{SAE}}(x,y) <= t \\ 0, & \mathcal{R}^{\text{SAE}}(x,y) > t \end{cases}. \tag{2.11}$$

A two-dimensional time surface is then computed by evaluating the previous equations at a particular time instant t, typically the timestamp of the latest event in $\mathcal{E}$, obtaining $H \times W$ representations $\mathcal{R}^{\Lambda}$, $\mathcal{R}^{\Gamma}$, and $\mathcal{R}^{\Phi}$. Although these representations are rarely used as input to deep neural networks, a similar application is that proposed by Lagorce et al. [84], who design a multi-layer architecture named Hierarchy Of Event-Based Time-Surfaces (HOTS) similar to a convolutional neural network but made of exponentially decaying surfaces $\mathcal{R}^{\Gamma}$. In this version, two sets of time surfaces are built, one for each polarity, by removing the explicit dependency from the polarity $P(x,y)$ in Equation 2.10, and locally defining them over a small neighborhood around each event as in a convolutional kernel.

HISTOGRAMS OF TIME SURFACES (HATS). The Histograms of Time Surfaces (HATS) [85] is a two-channel representation extending the concept of time surface with a memory mechanism resilient to noise. HATS are built by first dividing the event stream into C non-overlapping cells of size $K \times K$ pixels each. Then, from the events originating in each cell c, a grid of $(2\rho+1) \times (2\rho+1)$ histograms $\mathbf{h}_{c,p}$ is built, one for each polarity p. These histograms are computed by aggregating time surfaces $\mathcal{T}_{e_i}$ defined as:

$$\mathcal{T}_{e_i}(p) = \begin{cases} \sum_{j \in \mathcal{N}_{e_i}(p)} e^{-\frac{t_i - t_j}{\tau}} & \text{if } p_i = p \\ 0 & \text{otherwise} \end{cases}, \tag{2.12}$$

where $\mathcal{N}_{e_i}(p)$ is the cell's memory providing the set of events preceding e_i in a $[-\rho, \rho]$ spacial neighborhood.

The final two channels representation is obtained by rearranging together the normalized time surface histograms based on the location of the originating cell:

$$\mathcal{R}_{\mathcal{E}}^{\text{HATS}} = \{\mathbf{h}_{\mathcal{C}_j}(p)\}_{j=1}^{C}, \quad \mathbf{h}_{\mathcal{C}_j}(p) = \frac{1}{|\mathcal{C}_j|} \sum_{e_i \in \mathcal{C}_j} \mathcal{T}_{e_i}(p). \tag{2.13}$$

Despite being originally designed to provide local features for support vector machines [85], few works use HATS as an event representation in deep neural classifiers [92]. The ρ parameter is often such that $2\rho + 1 < K$, thus reducing the initial grid resolution. Temporal resolution is also lost, as the entire temporal window is condensed into a single frame with no bins retaining temporal resolution. For these reasons, other event representations are usually preferred in deep learning applications.

VOXEL-GRID-BASED REPRESENTATIONS. When designing deep neural networks for tasks involving fine-grained temporal predictions, such as when performing optical flow and ego-motion estimation [54, 55], image reconstruction [107, 108], and frame interpolation [124], input representations must be capable of preserving the event stream's temporal resolution as much as possible. However, all preceding representations aggregate the event stream $\mathcal{E}$ into either a single-channel or a two-channel event-frame depending on whether the polarity is kept separate. Temporal information is used as the pixel feature, which is typically obtained by aggregating the timestamps of all the events occurring at the same pixel location. Although time is used as a feature, this aggregation process condenses the whole event stream in just a few channels, thus losing any temporal resolution, especially when the event stream spans several milliseconds, and potentially hurting performance.

In order to alleviate these issues, several works [55, 106, 118] propose to extract multiple representations from the same event sequence, each aggregating a different temporal window of the original stream. A voxel grid structure splits the event stream into a spatio-temporal grid $H \times W \times B$ having the original spatial resolution but discretizing time into B consecutive bins. Two general mechanisms have been proposed [55, 106] to define the bins. The first partitions the time frame ΔT *based on time* into B equally-sized sub-windows from which representations $\mathcal{R}_b$ are extracted, each aggregating the subset $\mathcal{E}_b = \left\{ e_i \in \mathcal{E} \mid t_i \in [\frac{(b-1)\Delta T}{B}, \frac{b \cdot \Delta T}{B}] \right\}$ and corresponding to a single $H \times W$ slice of the voxel grid. The second is similar, but fixes the *number of events* in each bin to a predefined number N_e, thus splitting the sequence into $\mathcal{E}_b = \{e_i \in \mathcal{E} \mid i \in [(b-1)N_e, b \cdot N_e]\}$ intervals.

Any of the previous aggregation mechanisms can potentially be used to condense the events $\mathcal{E}_b$ into the corresponding bin's representation $\mathcal{R}_b$. The multi-channel image of Wang et al. [106] [118] aggregates events by polarity, using the same procedure of BII in Equation 2.5, while the voxel-grid image proposed by Zhu et al. [55] also introduces an interpolation strategy that gives recent events greater importance:

$$\begin{aligned} &\mathcal{R}_{\mathcal{E}}^{\text{vox}}(x,y,b) = \textstyle\sum_{i=1}^{N} p_i k_b(x-x_i) k_b(y-y_i) k_b(b-t_i^*) \\ &\text{with } t_i^* = (B-1)\frac{t_i - t_1}{t_N - t_1}, \text{ and } k_b(a) = max(0, 1-|a|), \end{aligned} \tag{2.14}$$

where t_i^* are the event timestamps rescaled into $[0, B-1]$, and $k_b(a)$ is the bilinear sampling kernel proposed by Jaderberg et al. [125]. Spatial interpolation is often removed since it only plays a role if sub-pixel events are present in the event stream, often resulting from undistortion and rectification transformations, or from augmentation procedures used in training, such as random rotations, translations, and scaling.

Voxel-grid images [55] are arguably the most popular grid-like event representation in deep learning applications. They have been used as input to deep architectures in a variety of tasks, including optical flow, depth, and egomotion [54, 55], object detection [104], classification [92], and image reconstruction [107, 108, 126].

TEMPORAL BINARY REPRESENTATION (TBR). Recently proposed by Innocenti et al. [93], the Temporal Binary Representation (TBR) provides an alternative solution to maintain temporal resolution without increasing the number of channels. TBR first computes an intermediate multi-channel representation $\mathcal{B} \in \{0,1\}^{H \times W \times B}$ by stacking together two-dimensional frames extracted from B independent bins of equal temporal duration. These frames simply indicate the occurrence of at least one event in every pixel location, regardless of its polarity. Each pixel's feature vector $\mathcal{B}(x,y) = [\mathcal{B}(x,y,0), \mathcal{B}(x,y,1), \dots, \mathcal{B}(x,y,B-1)]$ is then interpreted as a binary number with B digits, and converted into its decimal representation to obtain the final representation:

$$\begin{aligned} \mathcal{B}(x,y,b) &= \mathbb{1}(x,y,b), \\ \mathcal{R}_{\mathcal{E}}^{\text{TBR}}(x,y) = \tfrac{1}{B}\text{bin2dec}(\mathcal{B}(x,y)) &= \tfrac{1}{B} \sum_{b=0,\dots,B-1} 2^b \cdot \mathcal{B}(x,y,b), \end{aligned} \tag{2.15}$$

where $\mathbb{1}(x,y,b)$ is the indicator function returning 1 if at least one event is received at pixel (x,y) within the temporal bin b. The resulting representation has a single channel, but it condenses all the B event frames with

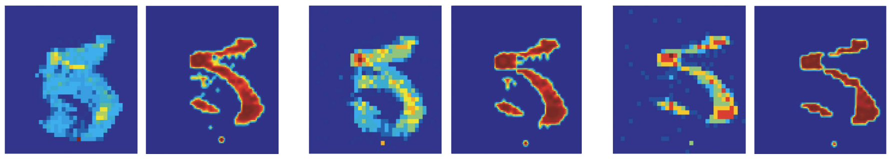


Figure 2.6: AMAE [94] representation by varying motion patterns. For each group of two images, the left one reports the event counts ($\mathcal{R}^{\text{count}}$), while the right one is AMAE. The representation remains almost unchanged despite different motion trajectories. *Figure adapted from that of Deng et al. [94]. Copyright* ©2020 *IEEE.*

lossless compression. This property is particularly ideal in tasks where a high temporal resolution is a prerequisite for a good prediction, such as in action recognition tasks [93]. Increasing the number of bins, and thus the resolution, has no impact on the dimensions of the $\mathcal{R}^{\text{TBR}}_{\mathcal{E}}$ tensor, contrary to other representations.

EVENT SPIKE TENSOR (EST). The Event Spike Tensor (EST) was proposed by Gehrig et al. [92] as the first end-to-end trainable grid-like representation. The procedure is similar to a voxel-grid image, with the difference that event timestamps are used as pixel features, and the weighting of each event's contribution is learned with a Multi Layer Perceptron (MLP) network instead of being fixed. Events are grouped by polarity to extract a two-channel representation from each bin:

$$\mathcal{R}^{\text{EST}}_{\mathcal{E}}(x,y,b,p) = \sum_{e_i \in \mathcal{E}^{(x,y,p)}} \hat{t}_i \cdot \mathcal{K}\left(\hat{t}_i - \frac{b}{B-1}\right), \tag{2.16}$$

where $\mathcal{E}^{(x,y,p)}$ is the set of all events of polarity p received in a specific (x,y) pixel, and $\hat{t}_i = t_i/t_N$ is the normalized event timestamp.

EST effectively enhances event representations by incorporating learnable components within the transformation, thus making automatic the process of tuning the representation for a task. It has been applied successfully to object recognition [92, 127], optical flow prediction [92], as well as semantic segmentation [127].

ADAPTIVE MOTION-AGNOSTIC ENCODER (AMAE). Deng et al. [94] pointed out that an object classification network should be invariant to the movement patterns of the object being classified. A good representation for classification should thus disregard any motion-related feature and only focus on encoding the object's appearance. In other words, representations extracted from the same object moving with different patterns should look the same. While event cameras exhibit very little motion blur, using different bins to condense events introduces back similar artifacts that could make deep neural networks suffer from motion biases if not properly trained.

Adaptive Motion-Agnostic Encoder (AMAE) is a learnable representation that adapts to the scene's movement to extract motion-invariant features. A time-sensitive tensor $\mathcal{T}_p$ is first computed, one for each polarity, using a transformation analogous to the voxel image in Equation 2.14, but making use of the Dirac delta in place of a bilinear kernel, and time instead of polarity. Pixels in this representation are then re-weighted to get rid of motion information by an adaptive scoring function $\mathcal{F}_{Adapt,p}$ defined as:

$$\mathcal{F}_{Adapt,p}(x,y) = \begin{cases} e^{\lambda \cdot \left(\mathcal{T}_p(x,y) - \theta_- \mathcal{M}_p\right)} & \text{if } \mathcal{T}_p(x,y) < \theta_- \mathcal{M}_p \\ e^{\lambda \cdot \left(-\mathcal{T}_p(x,y) + \theta^- \mathcal{M}_p\right)} & \text{if } \mathcal{T}_p(x,y) > \theta^- \mathcal{M}_p \\ \tanh\left(\mathcal{T}_p(x,y) - \theta_- \mathcal{M}_p\right) + 1 & \text{otherwise} \end{cases} \tag{2.17}$$

where $\mathcal{M}_p = \frac{\sum_{x,y} \mathcal{T}_p(x,y)}{\text{nonzero}(\mathcal{T}_p)}$ is the sum of the intensity values normalized by the number of active pixels, and $\{\theta_+, \theta_-, \lambda, \beta\}$ are adaptive parameters predicted from $\mathcal{T}_p$ by a convolutional neural network.

The final representation, computed as

$$\mathcal{R}_{\mathcal{E}}^{\text{AMAE}}(x,y) = \mathcal{F}_{Adapt,p}(x,y) \times \mathcal{T}_p(x,y), \tag{2.18}$$

demonstrated better performance than other learnable representations, such as EST, when the classification network is tested on motion patterns that differ from that used in training [94]. A visual representation of motion invariant features extracted by AMAE is given in Figure 2.6 on the previous page.

2.2.3 Datasets and Simulators

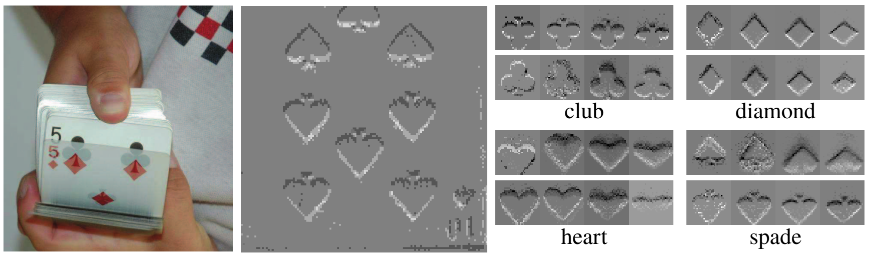
As it is typical whenever a novel sensor is first launched, very few datasets featuring event data were accessible in the first several years. This lack limited event cameras adoption, especially in deep learning applications where the quality and complexity of state-of-the-art algorithms traditionally follow those of the datasets. Deep neural networks require precise annotations and a large amount of data to be adequately trained, which initially lacked in the field, also due to the increased difficulty in annotating such a sparse visual encoding.

The availability of event-based datasets is today rapidly increasing. Basic vision tasks, such as object recognition, were among the first to benefit from this increased availability, fueling the deep learning community's interest in the sensor. However, in challenging tasks requiring pixel-level annotations, large, high-quality datasets are still missing. This lack has recently encouraged researchers to exploit unsupervised deep learning techniques and new event-camera simulators to train effective neural models, even in complex tasks.

This section provides an overview of the event-based datasets and simulators available in the literature, focusing on those related to this thesis. A description of the primary datasets, grouped by category, is provided in the following, while an overview is given in Table 2.2 on the facing page.

Table 2.2: Comparison between available datasets for classification, gesture and action recognition, detection, optical flow prediction, and segmentation.

Dataset	Task	Acquisition	# Classes	# Labels	Camera	Resolution (pix)	Total time (h)
Poker-DVS [128]	Classification	real-world, still cam	4	131	DVS128 [43]	31×31	2.1 sec
N-MNIST [116]	Classification	LCD, still image, moving cam	10	70,000	ATIS [47]	34×34	5.83
MNIST-DVS [71]	Classification	LCD, moving image, still cam	10	30,000	DVS128 [43]	128×128	16.67
CIFAR10-DVS [129]	Classification	LCD, moving image, still cam	10	10,000	DVS128 [21]	128×128	3.33
N-Caltech101 [116]	Classification	LCD, still image, moving cam	101	9,146	ATIS [47]	302×245 (avg)	0.76
DVS-Caltech256 [130]	Classification	LCD, moving image, still cam	257	30,607	DAViS240C [131]	240×120	8,58
N-Cars [85]	Classification	real-world, moving cam	2	24,029	ATIS [47]	55×65 (avg)	0.68
N-ImageNet [112]	Classification	LCD, still image, moving cam	1,000	1,781,167	DVS Gen3 [132]	480×640	24.74
N-ROD [4]	Classification	LCD, still image, moving cam	51	41,877	ATIS Gen3 [25]	256×256	3.49
ASL-DVS [89]	Gesture Recog.	real-world, still cam	24	100,800	DAViS240C [131]	240×120	2.80
DVS128 Gesture [133]	Gesture Recog.	real-world, still cam	11	1,342	DVS128 [21]	128×128	2.24
DVS-UCF-50 [130]	Action Recog.	LCD, moving image, still cam	50	6676	DAViS240C [131]	240×120	13.81
VOT Challenge 2015 [130]	Tracking	LCD, moving image, still cam	–	21455	DAViS240C [131]	240×120	0.20
Pedestrian Detection [134]	Detection	real-world, still cam	1	11,667	DAVIS346 [25]	346×260	0.10
Gen1 Automotive [135]	Detection	real-world, moving cam	2	255,781	ATIS [47]	304×240	39,32
1Mpx Detection [104]	Detection	real-world, moving cam	3	$25M$	Gen4 CD [34]	1280×720	14,65
MVSEC-OF [54, 136]	Optical Flow	real-world, moving cam	–	$20Hz$	DAVIS346 [25]	346×260	0.32
DVSMOTION20 [137]	Optical Flow	real-world, moving cam	–	$1kHz$	DAVIS346 [25]	346×260	0.02
DDD17-EvSeg [105, 138]	Semantic Seg.	real-world, moving cam	6	19,840	DAVIS346 [25]	346×260	2.15
DDD17-EISD [138, 139]	Instance Seg.	real-world, moving cam	1	19,000	DAVIS346 [25]	346×260	2.06



(a) Fast browsing **(b)** Uncut sequence **(c)** Samples

Figure 2.7: Samples in Poker-DVS [128] are obtained by first quick browsing [71] a deck in front of an event camera **(a)-(b)** and then extracting 31×31 motion-compensated pips with a tracking algorithm. Two samples from each class are shown in **(c)**. *Image in (a) is taken from Perez-Carrasco et al. [71]. Copyright ©2018 IEEE.*

OBJECT AND GESTURE RECOGNITION DATASETS. Event-based camera's recognition datasets, by definition, present an extra level of complexity when compared to typical frame-based image benchmarks. Because a DVS sensor is only sensitive to changes, brightness changes or movement must be present in the scene in order for visual features to appear.

The *Poker-DVS* [128] was one of the very first classification datasets ever to be released. Originally designed to show off the temporal characteristics of event-based cameras, it was later used to benchmark simple classification architectures [71, 84, 140]. In its latest version, the dataset features just 131 samples in which poker pips at a 31×31 pixel resolution move for about 20-30 ms. Pips were extracted from three recordings in which a specially made poker deck was quickly browsed in front of an event camera. Examples of Poker-DVS samples are provided in Figure 2.7.

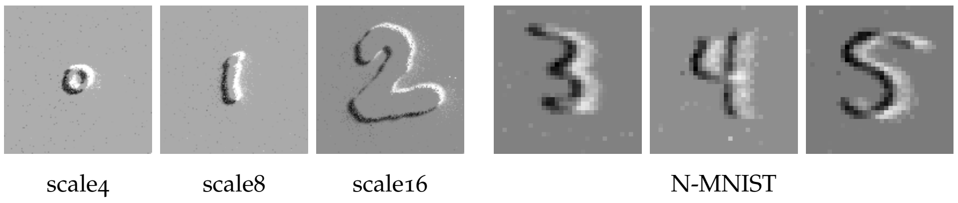


Figure 2.8: MNIST-DVS [71] (left) and N-MNIST [116] (right) samples. MNIST-DVS samples are provided in three different scales at 128×128 resolution, while N-MNIST contains 34×34 recordings at fixed scale.

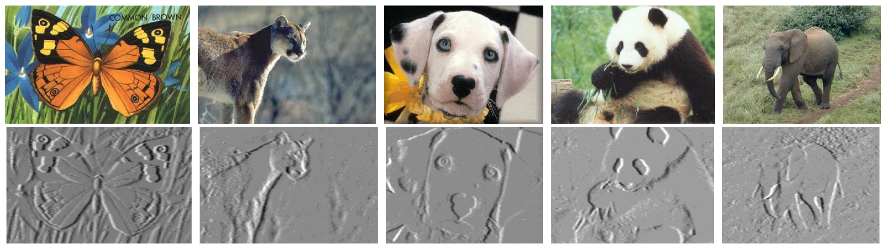


Figure 2.9: N-Caltech101 [116] samples. The first row shows RGB images from the original Caltech101 [142] dataset, while the second provides N-Caltech101 event conversions as voxel-grids.

Motivated by the need of having large datasets for benchmarking new classification approaches, Serrano-Gotarredona and Linares-Barranco [128] propose to convert existing frame-based datasets into their event-based version by artificially moving image samples on an LCD monitor and recording them with an event-based camera. The *MNIST-DVS* [71] is a partial conversion of the popular MNIST [141] dataset obtained with this technique, featuring 10 different digit classes equally split among 10,000 samples, each recorded at three different resolutions.

Recordings obtained by moving images on a monitor, however, are characterized by discontinuous movements since image trajectories depend on the refresh rate of the LCD screen used. To compensate for this issue, Orchard et al. [116] propose to keep the image still and move the camera instead. A pan-tilt mechanism moves the camera with a motion resembling the retinal saccadic movements observed in primates and humans. Using this procedure, they performed a new conversion of the MNIST dataset, called *N-MNIST* [116], as well as a novel conversion of the Caltech101 [142] image-based dataset, named *N-Caltech*101 [116], both featuring all the original samples. The first provides 60,000 training and 10,000 testing samples at a 34×34 resolution, as in the original dataset, while the second is composed of 9,146 samples of varying size, split into 100 categories and an extra *background* class. Figure 2.8 compares MNIST-DVS and N-MNIST in terms of resolution and scale, while Figure 2.9 shows few examples of N-Caltech101 recordings together with the original RGB images.

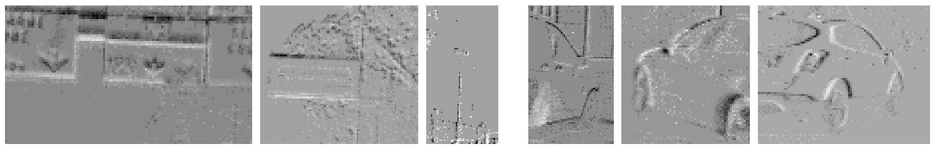


Figure 2.10: Samples from the N-Cars [85] classification dataset. The first three samples are from the *background* class, while the remaining ones are *cars*.

Following very similar procedures, a number of event-based conversions of popular image-based datasets was later proposed. Li et al. [129] released the *CIFAR*10-*DVS* dataset, a conversion of the CIFAR-10 [143] benchmark with a total of 10,000 128×128 samples. Hu et al. [130] converted a number of frame-based datasets, including the *Caltech*-256 [144], featuring 30,607 images split roughly equally into 257 categories, and the *UCF-50* [145] Action Recognition Dataset consisting in 6676 samples split in 50 action classes with an average length of 6.64s. Very recently, Kim et al. [112] released the first event camera conversion of the large-scale ImageNet [146] dataset. N-ImageNet is the largest object recognition dataset for event-based cameras currently available by number of classes and samples. It features 1,781,167 event recordings split into 1,000 different classes, each lasting 50ms. The recording setup is similar to that used in previous dataset conversions. However, images are captured with different camera trajectories and brightness conditions, two of the primary aspects affecting event generation, thus enabling the analysis of classifiers' robustness under varying conditions.

The use of automatized procedures for converting well-known image-based datasets simplified the effort of collecting samples while also enabling the creation of benchmarks that could be used to compare event-based networks' performance with that of image-based recognition architectures directly. However, these conversion procedures introduce non-idealities that one would rarely see in real-world recordings. Camera movements are generally fixed, inducing motion biases when training neural networks, and recordings from LCD screens are typically affected by unrealistic noise patterns caused by the refresh rate.

For these reasons, researchers started developing realistic datasets by recording objects in real-world scenes. An example is the *N-Cars* [85] dataset, a collection of urban recordings lasting 100ms each and featuring two object categories: *cars* and urban *background*. The dataset comes split into 7,940 car and 7,482 background training samples, and 4,396 car and 4,211 background testing samples, obtained by cropping bounding boxes around objects in driving scenes. Examples from both classes are provided in Figure 2.10.

Another dataset is the *ASL-DVS* [89], a set of handshape recordings for American Sign Language (ASL) classification. The dataset features 24 classes corresponding to the 24 letters (from *A* to *Z*, excluding the *J*) of the ASL, each with 4,200 samples lasting approximately 100 ms and recorded in real-world conditions. Far more complex are the recordings in the *DVS*-128 *Gesture* [133] dataset, which constitutes the first proper gesture recognition event-based benchmark. The DVS-128 Gesture consists of 1,342 instances of 11 different hand and arm gestures, collected from 29 subjects in 122 trials under 3 distinct lighting conditions.

Algorithms proposed in this thesis are, for the most part, validated on object recognition benchmarks. We use most of the datasets presented in this section, except for the gesture recognition ones, with the algorithms discussed in Chapter 4 and Chapter 5. In Chapter 5, we also adopt the procedure proposed by Orchard et al. [116] to design N-ROD [4], the first dataset for studying Synthetic-to-Real domain shifts in event-based recordings.

OBJECT DETECTION DATASETS. Object detection [147–149] is typically regarded as a fundamental task in computer vision, as it serves as the foundation of many other more advanced problems, including object tracking [150, 151], instance segmentation [152–154], image captioning [155, 156], and many others. It is characterized by the dual goal of determining the location of the objects in a given scene (*object localization*) as well as the category to which each object belongs (*object classification*).

Among the typical deep learning vision tasks involving fine-grained predictions, object detection is nowadays the one enjoying the largest hand-annotated event-based datasets. However, this achievement is only very recent [104, 135]. *N-Caltech*101 [116] is the first dataset to provide bounding boxes annotations for event-based data. Because the dataset was produced capturing samples from a monitor in a controlled setting, it was also possible to transfer all of the original image-based annotations to events by compensating for the known camera motion. However, objects in N-Caltech101 are often centered, and they occupy most of the frame, making learning to detect objects driven more by recognition than localization. Besides the recognition datasets discussed previously, Hu et al. [130] also converted the VOT Challenge 2015 Dataset [157] by recording RGB video clips with an event camera. Since the dataset is primarily meant for object tracking, it contains bounding boxes annotations for just 60 single-object sequences, making it impractical for training robust object detectors.

The first proper real-world event-based detection dataset is due to Miao et al. [134], who developed a pedestrian detection dataset featuring both indoor and outdoor sequences, two weather conditions (sunny and rainy), and three different environments. The dataset is composed of 12 sequences with an average length of 30 sec, recorded with a DAVIS camera and manually annotated every 20 ms. The setting used in all these sequences involves a fixed camera pointed at walking pedestrians in a static environment. Object detectors trained in this context can reach good performance despite the restricted number of training sequences since only events related to moving

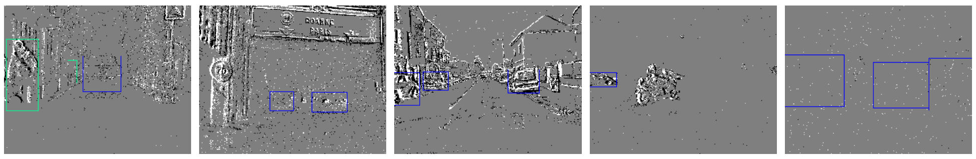


Figure 2.11: Samples from the Gen1 Automotive Detection dataset [135]. Each image depicts 20ms of events, together with *cars* and *pedestrian* bounding boxes in blue and cyan, respectively. The dataset contains recordings from different environments and moving conditions. Two-wheelers are not considered as pedestrians, and bounding boxes are also provided for objects in static scenes, making detection very challenging.

people are recorded, not the background, but they are still confined to operate in this simplified setting.

Tournemire et al. [135] released the Gen1 Automotive Detection Dataset, the first large-scale real-world automotive dataset for object detection. The dataset contains more than 39 hours of automotive sequences recorded with the QVGA Prohpesee sensor [25]. It features hand-annotated bounding boxes for cars and pedestrians at about 4 Hz, yielding more than $255,000$ labels in total. Samples showing different environments and challenging conditions for prediction are shown in Figure 2.11. An improved version was released in 2020 by Perot et al. [104] who recorded their dataset using the latest Prophesee event camera with 1-megapixel resolution and extended the set of labels also to include two-wheelers for a total of over 25M bounding boxes labeled at high frequency. The dataset is, to date, one of the most advanced automotive datasets currently available for event-based sensing.

Object detection is one of the benchmarks used to validate some of the neural architectures proposed in this thesis. The work discussed in Chapter 3 collocates before the release of real-world detection datasets, and it thus relies on simplified benchmarks. We perform tests on the N-Caltech101 dataset and propose different evaluation procedures based on detection variants of Poker-DVS and N-MNIST, as well as a simulated dataset. Finally, we evaluate the learnable representation discussed in Chapter 4 on the Gen1 Automotive Detection Dataset.

OPTICAL FLOW AND SEMANTIC SEGMENTATION DATASETS. Tasks requiring pixel-level predictions are by far the most challenging in deep learning research. Among these, some of the most prominent applications include optical flow prediction [158], which requires the network to predict the motion vector of each pixel in the image, and semantic segmentation [159, 160], which extends the detection task at the pixel level, asking the network to classify each pixel based on the category of the object to which it belongs. However, as the task complexity increase, so does the complexity and accuracy of the annotations required for such tasks. As a result, very few real-world datasets for event-based vision featuring pixel-level annotations are currently available in the literature.

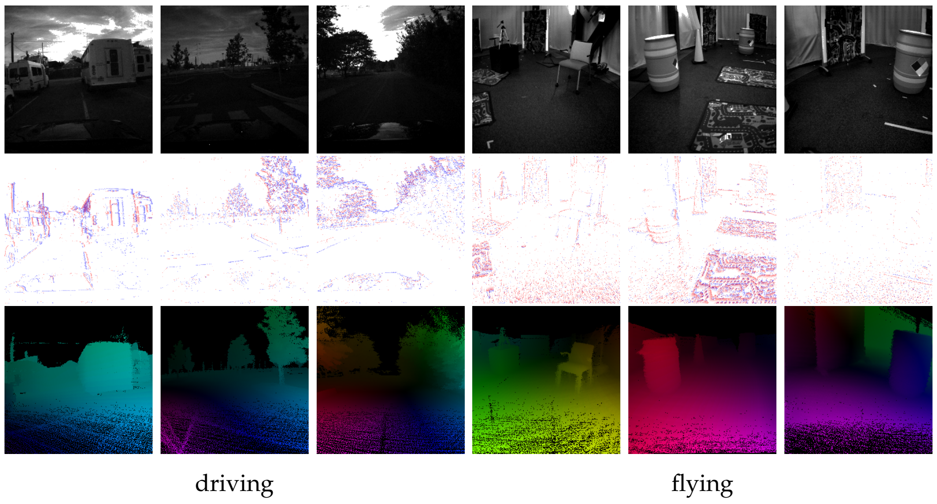


Figure 2.12: Samples from the MVSEC [136] optical flow extension proposed by Zhu et al. [54]. Driving sequences (left) are used for training, while sequences captured with a drone (right) for testing. The top row shows grayscale images taken by the DAVIS, followed by event counts and the ground truth flow.

Optical flow prediction has traditionally been tackled with analytical solutions exploiting the event-based camera's ability to detect motion. These approaches do not need training data and are usually tested on limited recordings, such as those used by Rueckauer and Delbruck [161] and Bardow et al. [162]. Zhu et al. [54] proposed the first large-scale optical flow dataset for self-supervised training of deep neural networks. They extended the Multivehicle Stereo Event Camera Dataset (MVSEC) [136] by deriving ground truth flow labels for testing, and used grayscale images from the DAVIS camera with a photometric loss for training. Optical flow labels were generated by calculating the motion field from the available camera motion and LiDAR depth maps, thus assuming motion is performed in a static environment. The dataset features a range of different vehicles, both indoor and outdoor scenarios, and different lighting conditions. Car driving scenes are typically used for training, while indoor hexacopter flying scenes for testing, enabling a robust cross-environment evaluation. Samples from both these sets are shown in Figure 2.12. Recently, Almatrafi et al. [137] proposed the *DVSMOTION20* dataset, a collection of event recordings captured with a DAVIS camera mounted on a gimbal and performing random motions at varying speeds. The dataset features four indoor static scenes as well as two sequences with moving objects.

Recent is also the application of deep learning in event-based semantic segmentation. To date, very few real-world datasets featuring ground truth labels are currently available in the literature. Alonso and Murillo [105] published an extension of the DAVIS Driving Dataset (DDD17) [138], initially intended for steering angle prediction, that adds semantic segmentation

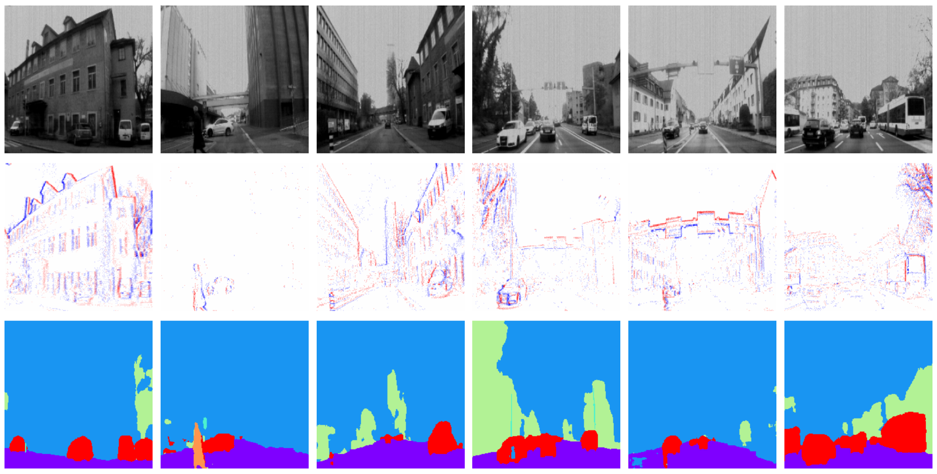


Figure 2.13: Samples from the DDD17 [138] extended by Alonso and Murillo [105]. The first two rows show DAVIS grayscales and events, displayed as normalized event counts, while the last row displays autolabeled segmentation masks.

masks on a portion of the recordings. These were obtained by labeling the DAVIS camera's grayscales with a network pre-trained on Cityspaces [163], and then transferring annotations to the events for a total of 15, 950 training and 3, 890 testing labeled frames. A few samples from this extended subset are displayed in Figure 2.13. Recently, Yang et al. [139] further extend the DDD17 dataset providing car instance segmentation labels for a subset of 14, 900 training and 4, 100 testing labeled images. As images from the DAVIS visually differ from traditional grayscale images, predicted labels are usually not particularly accurate, thus only providing coarse supervision.

We evaluate some of the methods presented in this thesis using datasets covered in this section. In Chapter 4 we test our proposed event representation on the optical flow prediction task with the *MVSEC* dataset, and perform a preliminary analysis of the domain adaptation technique discussed in Chapter 5 on an extended version [127] of the DDD17 segmentation dataset.

EVENT-CAMERA SIMULATORS. Although the number of annotated event-based datasets is constantly increasing, as testified by the recent release of the N-ImageNet [112] dataset, their availability still lags far behind that of standard image-based datasets, limiting the generability and complexity of deep neural networks. This lack gets even more critical in tasks requiring pixel-level annotations, where large hand-annotated datasets are still to be released due to the complexity and costs of their production. As a result, there is a need in event-based vision to develop alternative solutions that compensate for this deficiency.

Simulation, together with unsupervised training, is to date one of the few established solutions for training deep neural networks in complex event-based vision tasks. Katz et al. [164] were the first to demonstrate how to emulate both DVS and color sensitive (cDVS [46]) sensors. They used a 125 frames per seconds camera to capture high-speed videos that were then converted into event streams. The system processes the video stream and keeps track of the logarithmic brightness at each pixel location. Whenever the brightness differs from that of the last event generated at that location by a quantity above or below the preset threshold, a new ON/OFF event is generated with timestamp interpolated from that of the frames, and the pixel brightness is stored for future events generation. The cDVS sensor is emulated similarly but by keeping track of variations in color intensities.

Following a similar principle, the DAVIS [165] and the *ViSim* [166] simulators extend simulation also to produce additional modalities, such as grayscales, depth, and inertial measurement unit (IMU) readings. These frameworks directly interface with a renderer (either Blender [167] or a custom-made one [166]), enabling event simulation to produce datasets with potentially infinitely different scenes owing to synthetic rendering.

These approaches all rely on synchronously sampling frames at a very high frame rate to estimate the visual signal precisely. Rebecq et al. [168] improve over these approaches by adaptively varying the frame rate of the source video based on the predicted dynamics of the visual signal, thus reducing the number of frames to be generated and analyzed. The *ESIM* simulator supports a number of different rendering engines, and it was also recently integrated within the CARLA [169] driving simulator, easing the generation of realistic driving scenes. An extension to ESIM named Vid2E [127] enables the conversion of traditional 30-60 fps videos by employing a slow-motion neural network to interpolate the frames at an arbitrary framerate, adaptively, prior to event simulation.

All these engines rely on an ideal model of the DVS sensor and do not consider noise and non-idealities. Event-based cameras are extremely sensitive sensors that may behave differently based on environmental conditions. The brightness level [170] or even the temperature [171] can cause the camera to produce different event dynamics. For instance, at very low brightness regimes, the event camera pixels transition between two thresholds more slowly, causing latencies and artifacts equivalent to motion blur. The very recent *v2e* [170] simulator models some of these non-idealities, providing realistic event simulation, even under non-ideal operating conditions.

In this thesis, we used simulation at different stages to evaluate and train the proposed methods. In Chapter 3, we use the *DAVIS* simulator to produce object detection scenes and train our asynchronous convolutional networks. In Chapter 5, instead, we use the *ESIM* simulator to study how much simulation's non-idealities affect the robustness of deep neural networks when they are tested on real event-based cameras. For this purpose, we propose a novel dataset that supplements N-ROD [4] with events generated with *ESIM* as well as a method to tackle domain shift issues during training.

2.2.4 Applications

Thanks to their recent growth in popularity, the increase in the availability of event-based datasets, as well as the development of increasingly more accurate simulation procedures, event-based cameras are nowadays successfully applied in several computer vision tasks. These range from applications emphasizing their temporal resolution and low latency characteristics, typically leveraging hand-engineered algorithms, to general vision understanding involving low-level and high-level learning tasks. This section focuses on learning-based vision tasks and reviews state-of-the-art approaches on the four primary tasks investigated in this thesis: object recognition and detection, optical flow prediction, and semantic segmentation.

OBJECT CLASSIFICATION AND DETECTION. Motivated by the same biological inspiration that drove event-based development, early research on event-based object recognition focused on mimicking the processing principles of the first visual cortex layers. SNNs were quickly adopted to process event-based visual data, as they intend to provide a biologically inspired model of the neuron and, thus, they naturally fit the event-based encoding. Based on biological models of the visual cortex [172, 173], several simple architectures for event recognition were proposed [140, 174, 175] extracting a hierarchy of visual features by leveraging *simple* and *complex* neural cells. These feature extractors were typically based on hand-engineered filters, such as Gabor filters [176, 177], or learned using unsupervised, biologically realistic learning rules [178, 179]. The complexity and effectiveness of spiking recognition networks evolved in step with their training procedures [73, 180–182]. Starting from simple networks with only a few layers capable of classifying basic shapes [66, 140, 183] (such as those in Poker-DVS and N-MNIST), recent SNNs can now handle challenging visual patterns and complex architectures [180, 184].

Despite the recent advances in the design and training of deep SNNs, the performance of these neural models still lags behind that of regular ANNs when applied to the complex event streams produced by neuromorphic cameras [184]. Another line of research advocates moving away from the biological realism of SNNs in favor of exploiting the flexibility and ease of training of traditional machine learning pipelines for vision. Early efforts [84, 85] relied on unsupervised and hand-engineered methods for extracting features, which were then paired with Nearest Neighbor and Support Vector Machine classifiers for recognition. Encouraged by the success of frame-based deep learning systems, researchers have increasingly focused on how to interface events with standard neural networks for image processing effectively. The vast majority of them converts event streams into dense three-dimensional representations, which are then regarded as multi-channel images and processed by traditional Convolutional Neural Networks (CNNs) [2, 92, 94, 112, 185]. An overview of event representations typically used in such approaches is provided in Section 2.2.2.

Nevertheless, although delivering state-of-the-art performance, these approaches do not take advantage of event streams' sparse and asynchronous nature. Inspired by the parallels between event streams and three-dimensional point clouds, another research line studies the possibility of adapting point cloud classification methods to event data. These approaches [102, 103, 186] avoid densifying event information by relying on PointNet [61] and PointNet++ [60] architectures for sparse processing. Sekikawa et al. [102] extend PointNet with a recursive computation scheme for asynchronous processing, while Yang et al. [186] integrate Transformer self-attention [187] to learn spatio-temporal subsampling. Finally, following the research on point cloud processing, Graph Convolutional Neural Networks (GCNNs) have also been recently applied to event data [90, 188] with the same goal of preserving sparsity during computation. Either raw events [90], or aggregated features describing small event neighborhoods [188], are first connected in a spatio-temporal graph, and then standard GCNNs are used for processing.

Because of the novelty of these sparse processing architectures, even within the deep learning community, extending these techniques to more complicated visual tasks often necessitates a substantial amount of work. As a result, frame-based architectures are commonly employed in learning-based vision for event-based cameras. That is the case, for instance, of the object detection task, where traditional CNNs are commonly used as backbone architectures [104, 189, 190]. Li et al. [189] extend the image-based Faster R-CNN detector [191] with an adaptive method to dynamically collect information from several consecutive event frames based on the motion, while Perot et al. [104] integrate recurrent convolutional layers [192] to improve the network's temporal consistency.

Methods presented in this thesis fall under frame-based architectures for event-based processing. In Chapter 3, we focus on *object detection* and propose an alternative to point clouds inspired methods for performing sparse computation, which extends traditional CNNs with layers capable of sparse and incremental computation. In Chapter 4, instead, we design a learnable representation that effectively interfaces with CNN for various tasks, including object *classification*, while in Chapter 5, we propose a general procedure to effectively learn from simulated events and demonstrate its effectiveness extensively in *classification* tasks.

OPTICAL FLOW PREDICTION. The operating principle of event-based cameras is intrinsically tied with motion [193]. Events are indeed triggered by either changes in light intensity or by moving edges (in general, brightness gradients) in the image plane. Several works take advantage of this property, together with the very high temporal resolution of event-based cameras, to accurately estimate the optical flow using ad-hoc algorithms. Benosman et al. [194] propose to extend the Lucas-Kanade [195] algorithm by adapting

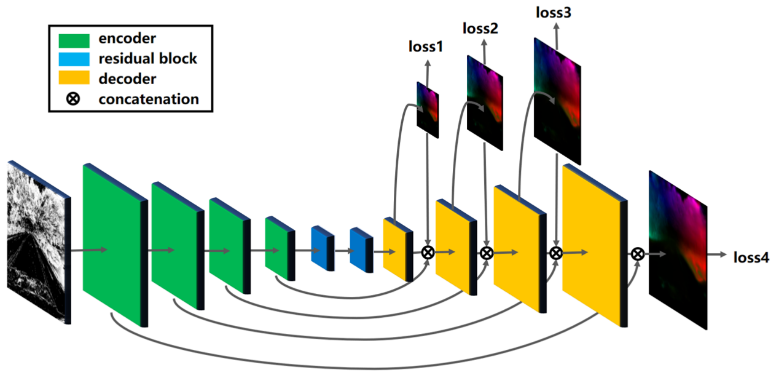


Figure 2.14: Architecture of the EV-FlowNet [54] optical flow prediction network. Event frames are processed by an encoder composed of four strided convolutional layers, followed by a sequence of two residual blocks and a decoder module composed of four upsample convolutional layers. Residual connections connect encoder and decoder layers, as in a UNet [203], and the output of each decoder layer is processed by a depthwise convolution to obtain multi-scale predictions. These optical flows are used to warp the input image and compute a photometric loss at multiple scales. *Figure taken from Zhu et al. [54]. Copyright* ©2018 *IEEE.*

its constraint on differential flow brightness consistency. Gehrig et al. [110] extend this approach to perform feature tracking by combining events with standard cameras for increased robustness, while Orchard et al. [196] implement a similar procedure by exploiting motion-sensitive receptive fields extracted with an SNN. Several variants of the Lucas-Kanade algorithm have also been proposed and evaluated against ground truth data by Rueckauer and Delbruck [161].

A different line of research, first introduced by Benosman et al. [111] and then also used by Mueggler et al. [197], proposes to estimate the optical flow by locally fitting a plane over the spatio-temporal surface defined by coactive events. A similar approach is proposed by Barranco et al. [198] who estimate motion by tracking the objects' contrast edges, either reconstructed from events or localized from DAVIS intensity frames to reduce computation and increase robustness. A phase-based method is then introduced by Barranco et al. [199] that further improves robustness in highly textured areas. Alternative solutions are that proposed by Brosch et al. [200], who make use of filter banks sensitive to different motion speeds and orientations, the variational optimization method introduced by Bardow et al. [162], and the expectation-maximization framework proposed by Zhu et al. [201]. Another approach is that of Gallego et al. [202], who propose to estimate the optical flow by searching for the point trajectories that maximize contrast on the image plane.

With the rise of deep learning in event-based vision, many have attempted to exploit deep learning mechanisms to estimate optical flow from events. To alleviate the need for labeled data for training, Zhu et al. [54] propose to leverage grayscale images taken from the same scene to provide supervision at training time. The flow estimated from event data over the interval between two consecutive grayscale frames is used to warp the first frame forward in time, and a photometric loss is computed to compare the estimated frame against the second real one. By minimizing the difference in intensity between frames, EV-FlowNet [54] learns to predict an accurate optical flow without direct supervision. They achieve this by training on MVSEC [136], where synchronized grayscale frames and event streams are provided thanks to the DAVIS event camera sensor. A visual representation of the EV-FlowNet architecture is given in Figure 2.14 on the preceding page. Later works improve unsupervised training by either imposing both brightness and smoothness constraints [204], or by adding additional adversarial losses on top of the photometric supervision [205]. In contrast, Ye et al. [56] extend this approach by providing supervision directly from event-frames, thus removing the need for synchronized grayscale images, while Zhu et al. [55] accomplish a similar aim using a motion compensation loss. Recent approaches make use of additional recurrent layers to incorporate temporal priors [206] or to provide reliable predictions even in pixel locations where no event is received [207], where most earlier approaches fail.

Since SNNs can naturally capture the spatio-temporal dynamics encoded in the delay between successive events, several works have sought to exploit their functioning to learn optical flow estimation. Orchard et al. [196] exploit the synaptic delays in the response of motion-sensitive receptive fields to mimic the canonical Lucas-Kanade algorithm [195], while Paredes-Valles et al. [208] exploit biologically inspired learning rules to predict motion. An efficient hardware implementation has also been proposed by Haessig et al. [209], who demonstrate real-time performance on a TrueNorth [210] neuromorphic chip. However, all these approaches make use of simple architectures to mitigate training issues and rarely scale to complex visual tasks like the MVSEC benchmark commonly used in deep learning approaches. To solve these issues, Lee et al. [211] propose a hybrid approach that combines an SNN encoder with a regular ANN decoder, outperforming some deep learning approaches even in complex tasks such as the MVSEC.

In Chapter 4, we build upon the EV-FlowNet [54] architecture and show that its performance can be improved by acting on the input representation. We substitute its hand-engineered event encoding with our learnable Matrix-LSTM layer and demonstrate that learning to encode events end-to-end, without any extra modification, gives a relative increase in performance of up to 30.76%.

MOTION AND SEMANTIC SEGMENTATION. Because event-based cameras naturally respond to motion, a logical application is to categorize objects by how they move rather than by how they look. Indeed, while objects' semantic (i.e., appearance) is encoded incrementally and may require processing to be reconstructed, their motion is more readily available in event-based streams thanks to the very high temporal resolution. As a result, research in event-

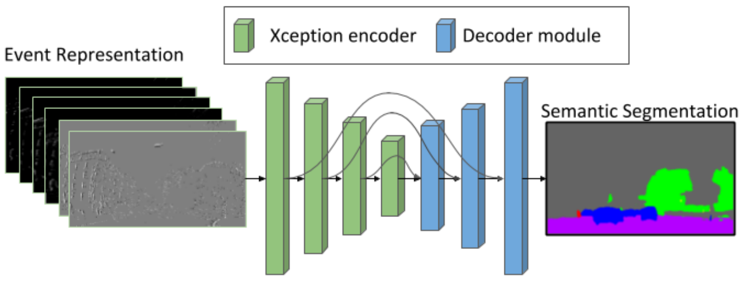


Figure 2.15: Architecture of the EV-SegNet [105] semantic segmentation network. The network is composed by a deep Xception-based encoder, followed by a five-layers decoder with skip connections in between. A pixel-wise cross-entropy loss is applied at the output of the network during training, on just one scale. *Figure taken from Alonso and Murillo [105]. Copyright ©2019 IEEE.*

based vision has mainly concentrated on motion segmentation rather than semantic segmentation.

Early works [212, 213] focus on distinguishing events caused by the camera motion from those produced by objects with a known shape, independently moving in the scene. When accessible, the ego-motion of the camera is also used [214], as identifying events caused by objects moving in other directions becomes easier. Another line of research relies on the concept of motion compensation to solve the task in the absence of prior knowledge about the motion or objects involved [114, 119–121]. In this case, motion is estimated, without supervision, in such a way to produce sharp edges when warping is applied to the event stream. Segmentation is performed by clustering events according to the motion either jointly [120] or after [114, 119, 121] the motion estimation process. Finally, a recent approach [88] proposes to connect events in a spatio-temporal graph and perform motion segmentation by minimizing an energy function through graph cuts [88].

While learning-based methods have also been used to accomplish motion segmentation [215], their primary applications in this context entail solving the semantic segmentation task, or the segmentation of objects based on their visual appearance. The most common approach is to adapt conventional ANN for frame-based semantic segmentation by converting event streams into grid-like representations. The first deep neural network for event-based semantic segmentation is due to Alonso and Murillo [105] who propose EV-SegNet, depicted in Figure 2.15, an Xception-based [216] encoder-decoder network, together with a semantic segmentation extension of the DDD17 [138] large scale dataset. The same approach is adopted by Gehrig et al. [127], where the network performance is improved by training on additional simulated data. Zhang et al. [217] focus instead on multi-modal semantic segmentation and propose to leverage event data under adverse conditions caused by accidents in driving scenarios. Finally, Kim et al. [218] are the first to apply SNNs on this task. Common frame-based architectures, such as the FCN [219] and DeepLab [220] architectures, are converted to perform spike-based processing, but still achieve lower performance than traditional methods on the DDD17 dataset.

In Chapter 5 we follow the approach of using simulated data during training proposed by Gehrig et al. [127]. However, we do not perform any finetuning with supervision on real event data. Instead, we leverage unsupervised domain adaptation methods to train a domain-invariant semantic segmentation network that achieves good performance even on real camera data.

3 ASYNCHRONOUS CONVOLUTIONAL NEURAL NETWORKS

The advantages of event-based cameras stem primarily from their ability to sense the world through sparse and asynchronous updates. This characteristic is very appealing in robotics since it opens up the possibility of designing computer vision systems with very short latencies and low power consumption. However, most vision systems, including deep neural networks, are traditionally designed to operate on synchronous and spatially dense representations, which vary fundamentally from the encodings provided by event-based cameras. Nonetheless, learning-based approaches have managed to achieve impressive results in event-based vision by collecting events into image-like dense representations and then applying traditional deep learning architectures with minor modifications [54, 92, 105, 112]. While this approach has contributed to advancing the field of event-based vision, it also comes at the cost of increased data redundancy, computational complexity, and latency [25].

This chapter proposes a method for converting deep neural networks trained on dense event-based representations into networks producing identical predictions but through incremental and asynchronous computation, thus fully preserving events' asynchronous and data-driven nature. We equip traditional max-pooling and convolution operations with an internal memory of the previous layer's output, as well as a set of update rules to asynchronously and sparsely update their internal representation once an event occurs. The resulting event-based layers are general and can be substituted to traditional ones, after training, without major modifications. We benchmark these layers on classification and object detection tasks using several extensions of publicly available datasets and additional recordings obtained in simulation.

3.1 INTRODUCTION

The ability to extract meaningful features from unstructured visual representations is at the heart of many fundamental techniques underlying visual understanding in computer vision. To this extent, Convolutional Neural

This chapter is based on: Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. "Asynchronous Convolutional Networks for Object Detection in Neuromorphic Cameras." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2019. DOI: 10.1109/cvprw.2019.00209. ©2019 IEEE. *Best Paper Award at the Second International Workshop on Event-based Vision and Smart Cameras.*

Networks (CNNs) have nowadays become the primary choice in many applications due to their natural ability to learn effective abstractions that make reasoning easy to accomplish even in challenging visual tasks. That is the case of many image-based visual tasks, such as object classification [220–223], object detection [19, 191, 224], and semantic scene labeling [219, 225, 226], among many others, but they have been recently extended also to non-Euclidean domains such as manifolds and graphs [227, 228].

The flexibility and success of such neural architectures in many fields have inspired the event-based vision community to apply these designs to event-based data as well. These approaches typically collect events in batches with fixed observation time or cardinality [106, 229] and then transform them into dense tensors for processing. An overview of popular event representations is provided in Section 2.2.2 on page 18. Since the resulting representation is structurally equivalent to a multi-channel image, applying CNNs becomes straightforward, and no modifications to the architectures or the learning procedures are necessary. This simple approach has aided in enabling the employment of event-based cameras in a variety of visual applications, including, but not limited to, semantic segmentation [105, 217], depth estimation [57, 58], high-speed video reconstruction [108, 126], as well as optical flow and ego-motion estimation [54–56]. All these works combine traditional state-of-the-art image-based architectures and tools with the benefits of event-based cameras to solve vision tasks even under challenging motion and lighting conditions. Despite all these advantages, however, collecting events into *synchronous* and *dense* representations discards all the sparsity and temporal resolution of the event stream, increasing latency, and computing complexity. Indeed, CNNs are not designed to deal with sparse inputs, and they perform computation even for parts of the scene that remain static throughout time.

As discussed in Section 2.2.1, a better approach in terms of efficiency is to make use of Spiking Neural Networks (SNNs) [63, 184, 230]. They are, indeed, very attractive in event based vision since dedicated neuromorphic devices [231–233] enable their implementation at extremely low power and latency regimes, opening the possibility to fully exploit all the benefits of event-based cameras. However, despite these advantages, their complex dynamics makes training spiking models remarkably challenging, which puts them at a disadvantage when compared to CNNs' performance, especially in complex vision tasks. Most Spiking Neural Networks (SNNs) solutions for event-based data are limited to shallow networks, and thus, show limited performance [184].

An ideal system for event-based processing should be capable of fully exploiting cutting-edge vision architectures and training techniques while also allowing for asynchronous, sparse, and data-driven computing. While most of the research has sought reaching this objective by improving SNNs' training [66, 182, 184, 218] to achieve the same accuracy of traditional CNNs, very few have explored the opposite path of bringing SNNs' efficiency within the CNNs' paradigm.

3.1.1 Main Contributions

This chapter takes a first step towards enabling asynchronous and sparse computing in traditional CNNs, improving efficiency without sacrificing prediction accuracy. We propose to achieve this by introducing a general processing framework that takes inspiration from spiking networks to enable conventional CNNs to perform asynchronous and data-driven computation. At training time, we leverage traditional deep learning tools to train state-of-the-art architectures on event-based representations, thus maximizing task performance while avoiding typical SNNs' restrictions. At deployment time, the proposed method allows these pre-trained networks to be converted into networks capable of performing incremental and sparse computation, retaining event-based advantages during processing. Crucially, networks implemented using the proposed procedure are asynchronous, meaning that computation only occurs when a sequence of events arrives and only where previous results need to be recomputed. More specifically:

- We propose improved event-based versions of the convolution and pooling layers of CNNs. In contrast to conventional layers in which features are stateless and always recomputed from scratch, these event-based layers maintain a memory of the intermediate representation produced at the previous iteration and update them locally only as a consequence of incoming events. Only the features that depend on values that changed in the preceding layer are recomputed, while the remaining ones are directly taken from the internal memory. The *e-conv* and *e-max-pool* layers are detailed respectively in Section 3.3.2 on page 47 and Section 3.3.3 on page 51 [1].

- We propose a novel event representation inspired by SNNs and a set of update rules to update layer's representations asynchronously as time passes. A leaking mechanism acting independently on each layer is used to allow past information to be forgotten, enabling features computed in the past to fade away as their visual information starts to disappear in the scene. We provide a description of this procedure in Section 3.3.1 on page 46.

- We showcased the framework on the object detection task, proposing the fcYOLE architecture, a fully-convolutional architecture making use of these novel event-based layers and inspired by the YOLO [19] single-shot detection network. The result is an asynchronous detector able to perform computation only when requested and at different rates.

- In Section 3.5 on page 56 we benchmark fcYOLE on object recognition, as well as several extensions of publicly available datasets for detection and additional recordings obtained in simulation. The proposed event-based networks can be used to produce an output only when new

1 Code available at `https://github.com/marcocannici/async-ev-cnn`

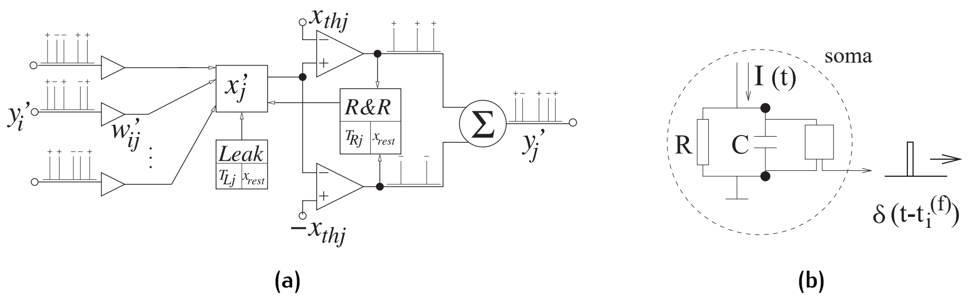


Figure 3.1: (a) Schematics of a LIF neuron [71]. **(b)** An simple circuit implementing the spiking neuron on the left [234]. A low-pass filter converts spikes into a current pulse $I(t)$ that charges the capacitor C, implementing the neuron's internal state. When the voltage goes over a threshold ϑ, the capacitor discharges, generating an output pulse δ. The integrate-and-fire circuit's voltage dynamics are analogous to that of an RC circuit. *Figures taken from Perez-Carrasco et al. [71]* ©2013 *IEEE, and Gerstner and Kistler [234]* ©2002 *Cambridge University Press.*

events arrive, dynamically adapting to the timings of the input, or to produce results at regular rates by using a leaking mechanism to update layers even in the absence of new events.

3.2 BACKGROUND

This section provides an overview of the operating principles and the primary computing units of SNNs. Although we do not explicitly implement a spiking network, these concepts are used throughout the chapter to design both a novel bio-inspired event representation and the proposed event-based layers for asynchronous computation.

BIOLOGICAL MODEL OF A NEURON. Neuronal action potentials, or spikes, constitute the primary means of communication between neurons in the biological brain. Spikes from other neurons travel along their axons and reach the post-synaptic neuron in unique sections known as synapses. There, information is chemically conveyed by neurotransmitters that modulate and amplify signals by inducing a variation in the membrane potential of the receiving neuron. When the sum of these potentials reaches a particular threshold, the neuron fully discharges, sending out a spike and entering a refractory phase during which no additional output spike can be produced. If no spike is received, the excitation leaks out, reducing the membrane potential over time until a resting state is reached [71, 235].

LEAKY-INTEGRATE-AND-FIRE (LIF) NEURON. A Spiking Neural Network (SNN) is a system of artificial neurons that attempts to imitate neuronal biological activity as closely as possible. Over the years, several models of the spiking neurons have been proposed [234, 236–239]. The simplest, but yet most widely used, is the Leaky Integrate-And-Fire (LIF) neuron [236]. Figure 3.1a on the preceding page, taken from Perez-Carrasco et al. [71], depicts a schematics of its primary components along with an equivalent electronic circuit [240]. A spiking neuron is characterized by an internal state x'_j that continuously evolves over time. Spikes y'_i coming from pre-synaptic neurons cause a positive or negative variation $\Delta x'$ on the neuron's state proportionate to the synaptic weight w'_{ij} connecting the two. When x'_j exceeds one of the predefined thresholds $\pm x_{th_j}$, the neuron generates an output spike y'_j, the sign of which depends on the value x'_j accumulated at that time instant, and then resets its state to a resting value x_{rest}. The internal state is also continuously subject to a leak, characterized by a constant rate $|x_{th}/T_{L_j}|$, which strives x'_j towards the resting state if no input spike is received. Finally, the rate of fire is also typically limited by a refractory period T_{R_j} during which spikes can be accumulated but not produced.

This simple description clearly highlights the connection between SNNs and biological systems. Some of these concepts, such as the thresholding-based spike generation and refractory states, have also influenced the design of early event-based camera models and motivated their use for processing events. However, traditional learning techniques based on backpropagation cannot be directly applied to spiking neurons. Indeed, SNNs operate on discrete spikes rather than smoothly differentiable functions of their inputs, which makes end-to-end training very challenging to accomplish. Although several solutions to these learning issues have been proposed [66, 182, 184, 218], the difficulty in training SNNs remains the primary factor limiting their effectiveness and applicability in event-based vision.

3.3 EVENT-BASED FULLY CONVOLUTIONAL NETWORKS

Conventional CNNs for video analysis treat every batch of frames independently and recompute all the feature maps entirely and from scratch, even if consecutive frame sequences differ from each other only in small portions. These architectures are often used for event-based processing by transforming event streams into a series of image-like representations similar to a video stream. Besides being a significant waste of power and computation, this approach does not match the asynchronous and sparse visual encoding produced by event-based cameras.

Inspired by SNNs's operational principles, we propose to alleviate these issues by enabling convolutional neural networks to perform incremental computation. First, we formulate a new event-based representation in which each pixel operates similar to a LIF neuron, accumulating incoming events incrementally and asynchronously while deactivating during time through a leaking mechanism. Then, instead of discarding the output of each layer after computation, we keep track of the networks' feature maps and use them for subsequent processing, only recomputing them locally in pixel locations affected by incoming events.

Layers maintain their state over time and communicate with each other with events, signaling where feature maps have been updated. We enable the leaking mechanism to directly act on each layer, allowing features to fade away as visual information starts to disappear in the input surface. We design a set of lightweight update rules that allow layers to be updated as time passes without recomputing feature maps from scratch.

The resulting architecture yields the same results of a network trained on synchronous frames extracted with the proposed event representation. As a result, training can be performed using any gradient-based learning procedure, taking full advantage of state-of-the-art learning tools. However, differently from a standard CNN, the network, once deployed on the proposed event-based layers, does not need synchronized inputs but can operate asynchronously, dynamically adapting to the rate on incoming events.

In Section 3.4 on page 52, the proposed framework is employed to detect objects captured by an event-based camera. Nonetheless, with certain constraints outlined in the next sections, any convolutional architecture can possibly be built to perform asynchronous event-by-event processing. A CNN trained to process frames can indeed be easily converted into an event-based network without any modification on its layers composition, and by using the same weights learned while observing frames, maintaining its output unchanged.

3.3.1 Leaky Event-Based Representation

The basic component of the proposed architecture is a procedure able to accumulate events. Sparse events generated by the neuromorphic camera are integrated into a *leaky surface*, a structure that takes inspiration from the functioning of Spiking Neural Networks (SNNs) to maintain a memory of past events. A similar mechanism has also been proposed by Cohen [41].

Every time an event with coordinates (x_e, y_e) and timestamp t is received, the corresponding pixel of the surface is incremented of a fixed amount Δ_{incr}. At the same time, as time passes, the whole surface is also decreased by a quantity Δ_{leak} proportional to the time elapsed since the previous event. More formally:

$$q^t_{x_s,y_s} = max(p^{t-1}_{x_s,y_s} - \Delta_{leak}, 0) \tag{3.1}$$

$$p^t_{x_s,y_s} = \begin{cases} q^t_{x_s,y_s} + \Delta_{incr} & if(x_s, y_s) = (x_e, y_e)^t \\ q^t_{x_s,y_s} & otherwise \end{cases}, \tag{3.2}$$

where $p^t_{x_s,y_s}$ is the pixel value in position (x_s, y_s) of the leaky surface and $\Delta_{leak} = \lambda \cdot (ts^t - ts^{t-1})$, with λ a constant scalar. Pixel values are prevented from becoming negative by means of the *max* operator. Notice that the effects of Δ_{leak} and Δ_{incr} are related: Δ_{incr} determines how much information is conveyed by every single event, whereas Δ_{leak}, through λ, defines the decay rate of activations. Indeed, given a certain choice of these parameters, similar results can be obtained using, for instance, a higher increment Δ_{incr} combined with a higher temporal λ. For this reason, we fix $\Delta_{incr} = 1$, and we vary only λ based on the dynamics of the input event stream.

Similar procedures capable of maintaining time resolution have also been proposed, such as those that make use of exponential decays [41, 84] to update surfaces and those relying on histograms of events [109]. The concept of *time surface* has also been introduced in [84] where surfaces are obtained by associating each event with temporal features computed applying exponential kernels to the event neighborhood. Extensions of this procedure making use of memory cells [85] and event histograms [54] have also been proposed. We discuss these event representations in detail in Section 2.2.2 on page 18.

Although these event representations may better describe complex scene dynamics, we make use of a simpler formulation to derive a linear dependence between consecutive surfaces. This choice allows us to design the event-based layers discussed in the following sections, in which time decay is applied to every layer of the network independently.

LEAKY SURFACE LAYER. We embed this procedure as an actual layer of the proposed framework, i.e., a layer that uses events to signal the occurrence of particular conditions (e.g., the update of a pixel) on top of its normal operation. To allow subsequent layers to locate changes inside the surface, the following information, together with the state of the reconstructed leaky frame, is also forwarded to the next layer:

i. the list of incoming events, to signal where new information needs to be processed;

ii. Δ_{leak}, which is directly sent to all the subsequent layers, to update feature maps at the new time instant;

iii. the list of surface pixels that have been reset to 0 by the max operator in Equation 3.1 on the facing page, as it will be clarified later in the section.

3.3.2 Event-based Convolutional Layer (e-conv)

The *event-based convolutional* (e-conv) layer we propose uses events to determine where the input feature map has changed with respect to the previous time step and, therefore, which parts of its *internal state*, i. e., the feature map computed at the previous time step, must be recomputed and which parts can be reused.

We use a procedure similar to the one described in the previous section to let features decay over time. However, while Δ_{leak} acts directly and identically on every pixel of the input representation, its effect may change in deeper layers based on the feature location. Indeed, the transformations applied by previous layers and the composition of their activation functions may change how Δ_{leak} acts in different parts of the feature map. For instance, the decrease of a pixel intensity value in the input surface may cause the value computed by a certain feature in a deep layer to decrease, but it could also cause another feature of the same layer to increase. Therefore, the update procedure must also be able to accurately determine how a single bit of information is transformed by the network through all the previous layers in any spatial location. We face this issue by storing an additional feature map, $F_{(n)}$, and by using a particular class of activation functions in the hidden layers of the network. In the following, we provide an intuition of the functioning of the *e-conv* layer by explicitly deriving its update rules for the first two convolutional layers of a CNN. Then, we generalize these update rules by induction to every layer of the network.

UPDATE RULES DERIVATION. Let us consider the first layer of a CNN, which processes the leaky surfaces described in the previous section through a convolution of filters W, bias b, and activation function $g(\cdot)$. The computation performed on each receptive field is:

$$y^t_{ij_{(1)}} = g\left(\sum_h \sum_k x^t_{h+i,k+j} W_{hk_{(1)}} + b_{(1)}\right) = g(\tilde{y}^t_{ij_{(1)}}), \tag{3.3}$$

where, with abuse of notation, h, k select a pixel $x^t_{h+i,k+j}$ in the receptive field of the output feature (i, j) and its corresponding value in the kernel W. The subscript (1) indicates the network's hidden layer; in this case, the first after the leaky surface layer.

When a new event arrives, the leaky surface layer decreases all the pixels by Δ_{leak}. Therefore, a pixel not directly affected by the event becomes $x^{t+1}_{hk} = x^t_{hk} - \Delta^{t+1}_{leak}$, with $\Delta^{t+1}_{leak} > 0$. At time $t+1$ Equation 3.3 becomes:

$$\begin{aligned} y^{t+1}_{ij_{(1)}} &= g\left(\sum_h \sum_k x^{t+1}_{h+i,k+j} W_{hk_{(1)}} + b_{(1)}\right) \\ &= g\left(\sum_h \sum_k (x^t_{h+i,k+j} - \Delta^{t+1}_{leak}) W_{hk_{(1)}} + b_{(1)}\right) \\ &= g\left(\tilde{y}^t_{ij_{(1)}} - \Delta^{t+1}_{leak} \sum_h \sum_k W_{hk_{(1)}}\right). \end{aligned} \tag{3.4}$$

If $g(\cdot)$ is (i) a piecewise linear activation function $g(x) = \{\alpha_i \cdot x \quad \text{if } x \in D_i\}$, as ReLU [241] or Leaky ReLU [242], and we assume that (ii) the updated value does not change which linear segment of the activation function the output falls onto and, in this first approximation, (iii) the leaky surface layer does not restrict pixels using $\max(\cdot, 0)$, then Equation 3.4 on the facing page can be rewritten as it follows:

$$y^{t+1}_{ij_{(1)}} = y^{t}_{ij_{(1)}} - \Delta^{t+1}_{leak} \alpha_{ij_{(1)}} \sum_{h} \sum_{k} W_{hk_{(1)}}, \tag{3.5}$$

where $\alpha_{ij_{(1)}}$ is the coefficient applied by the piecewise function $g(\cdot)$ which depends on the feature value at position (i, j). The equation thus obtained enables updating the feature map computed at the previous step as time passes, without having to recompute it from scratch. Its computation is much lighter than a convolution. Indeed, the summation $\sum_h \sum_k W_{hk_{(1)}}$ can be precomputed, being constant after training, and $\alpha_{ij_{(1)}}$ is also likely to remain the same through time. Computing Equation 3.5 is thus equivalent to an element-wise product with a scalar Δ^{t+1}_{leak} followed by the element-wise difference between two matrices, which can both be highly parallelized.

Whenever the previous assumptions are not satisfied, i.e., when a new event arrives in (i, j)'s receptive field or its $\alpha_{ij_{(1)}}$ has changed, the feature $y_{ij_{(1)}}$ is recomputed by applying the filter W locally over x^{t+1}.

Consider now a second convolutional layer attached to the first one. Its output can be rewritten as

$$\begin{aligned}
y^{t+1}_{ij_{(2)}} &= g\left(\sum_{h,k} y^{t+1}_{i+h,j+k_{(1)}} W_{hk_{(2)}} + b_{(2)}\right) \\
&= g\left(\sum_{h,k} \left(y^{t}_{i+h,j+k_{(1)}} - \Delta^{t+1}_{leak} \alpha_{i+h,j+k_{(1)}} \sum_{h',k'} W_{h'k'_{(1)}}\right) W_{hk_{(2)}} + b_{(2)}\right) \\
&= y^{t}_{ij_{(2)}} - \Delta^{t+1}_{leak} \alpha_{ij_{(2)}} \sum_{h,k} \left(\alpha_{i+h,j+k_{(1)}} \sum_{h',k'} W_{h'k'_{(1)}}\right) W_{hk_{(2)}} \\
&= y^{t}_{ij_{(2)}} - \Delta^{t+1}_{leak} \alpha_{ij_{(2)}} \sum_{h,k} F^{t+1}_{h+i,k+j_{(1)}} W_{hk_{(2)}} = y^{t}_{ij_{(2)}} - \Delta^{t+1}_{leak} F^{t+1}_{ij_{(2)}},
\end{aligned} \tag{3.6}$$

where we combined Equation 3.5 and Equation 3.3 on the preceding page, and assumed once again the same conditions over $g(\cdot)$ and $\alpha_{ij_{(2)}}$.

The previous equation can be extended by induction as it follows:

$$\begin{gathered}
y^{t+1}_{ij_{(n)}} = y^{t}_{ij_{(n)}} - \Delta^{t+1}_{leak} F^{t+1}_{ij_{(n)}}, \\
\text{with } F^{t+1}_{ij_{(n)}} = \alpha_{ij_{(n)}} \sum_{h} \sum_{k} F^{t+1}_{i+h,j+k_{(n-1)}} W_{hk_{(n)}} \text{ if } n > 1,
\end{gathered} \tag{3.7}$$

where the subscript (n) indicates any convolutional layer of the network, and $F_{ij_{(n)}}$ expresses how visual inputs are transformed by the network in every receptive field (i, j), i.e., the composition of the previous layers activation functions. Since $F_{ij_{(n)}}$ only depends on the network's parameters $W_{(n)}$ and $\alpha_{ij_{(n)}}$, up to the current layer, the same considerations for computing Equation 3.5 also apply in this case.

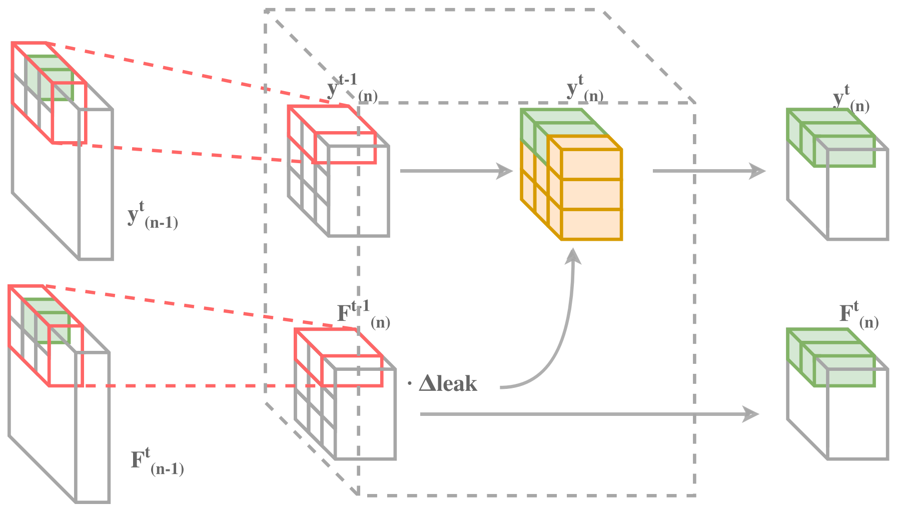


Figure 3.2: The structure of the e-conv layer. The internal state and the update matrix are recomputed locally only where events are received (green cells) whereas the remaining regions (yellow cells) are directly obtained from the previous state.

Given this formulation, the max operator applied by the leaky surface layer can be interpreted as a ReLU, and Equation 3.5 on the previous page becomes:

$$y^{t+1}_{ij_{(1)}} = y^{t}_{ij_{(1)}} - \Delta^{t+1}_{leak}\alpha_{ij_{(1)}} \sum_h \sum_k F^{t+1}_{i+h,j+k_{(0)}} W_{hk_{(1)}}, \tag{3.8}$$

where the value $F_{i+h,j+k_{(0)}}$ is 0 if the pixel $x_{i+h,j+k} \leq 0$ and 1 otherwise.

Notice that $F_{ij_{(n)}}$ needs only to be updated when the corresponding feature changes enough to make the activation function use a different coefficient α, e.g., from 0 to 1 in case of ReLU. In this case, $F_{(n)}$ is updated locally in correspondence of the change by operating on the update matrix $F_{(n-1)}$ of the previous layer and by applying Equation 3.7 on the preceding page only for the features whose activation function has changed.

INTERNAL STATE AND RECURSIVE COMPUTATION. As in every layer of the proposed framework, events are used to communicate the change to subsequent layers so that their features and update matrix can be updated accordingly. The internal state of the e-conv layer, therefore, comprises the feature maps $y^{t-1}_{(n)}$ and the update values $F^{t-1}_{(n)}$ computed at the previous time step. The initial values of the internal state are computed by making full inference on a blank surface; this is the only time the network needs to be executed entirely. As a new sequence of events arrives, the following operations are performed (see Figure 3.2):

i. Update $F^{t-1}_{(n)}$ locally on the coordinates specified by the list of incoming events (Equation 3.7). Note that we do not distinguish between actual events and those generated by the change of slope α in the linear activation function.

ii. Update the feature map $y_{(n)}$ with Equation 3.7 on page 49 in the locations which are not affected by an event, and generate an output event where the activation function's coefficient has changed after the update.

iii. Recompute $y_{(n)}$ by applying W locally in correspondence of the incoming events and output which receptive field has been affected through new output events.

iv. Forward the feature map and the events generated in the current step to the next layer.

3.3.3 Event-based Max Pooling Layer (e-max-pool)

In a max-pooling layer, the location of the maximum value in each receptive field is likely to remain the same over time. Hence, an event-based pooling layer can exploit this property to avoid recomputing the position of maximum values every time.

The internal state of an event-based max-pooling (e-max-pool) layer is composed by a *positional matrix* $I^t_{(n)}$. This matrix has the same shape as the layer's output feature map, and it stores the position of the maximum value in each receptive field. Every time a sequence of events arrives, the internal state $I^t_{(n)}$ is sparsely updated by only recomputing the position of the maximum values for the receptive fields affected by an incoming event. The internal state is then used both to build the output feature map and to produce the layer's *update matrix* $F^t_{(n)}$, by fetching the previous layer on the locations provided by the indices $I^t_{ij_{(n)}}$. For each e-max-pool layer, the indices of the receptive fields where the maximum value changes are communicated to the subsequent layers so that their internal states can be updated accordingly. This mechanism is depicted in Figure 3.3 on the next page.

Notice that the leaking mechanism acts differently in distinct regions of the input space. Pixel features inside the same receptive field can indeed decrease over time with different rates as their update values $F^t_{ij_{(n)}}$ could be different. Therefore, even if no event has been detected inside a region, the position of its maximum value might change as time passes. However, let us consider a pixel location $(i, j)^*$ in the previous layer's feature map having (i) the minimum update rate among features in a receptive field R, i.e., $(i, j)^* = argmin_{hk \in R}(F_{hk_{(n-1)}})$, while (ii) also corresponding to the maximum value in R, i.e., $(i, j)^* = argmax_{hk \in R}(y_{hk_{(n-1)}})$. This feature will decrease slower than all the others in R, and thus its value will always remain the maximum overtime. In this case, its index $I^t_{(n)_R}$ does not need to be recomputed until a new event arrives in R. We call these receptive fields *persistent maximum locations*.

Whenever a new event arrives at the e-max-pool layer, the following operations are performed:

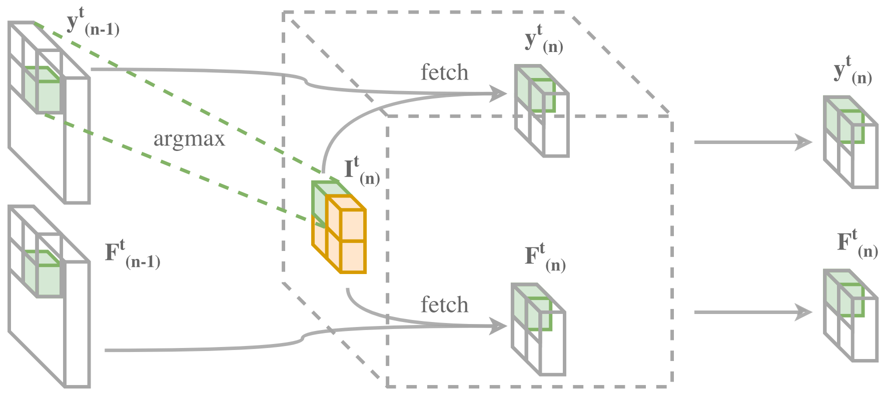


Figure 3.3: The structure of the e-max-pooling layer. The location $I^t_{(n)}$ of the maximum value in each receptive field is only recomputed for those regions affected by incoming events (green cells). The output is then obtained by fetching the layers' input at those locations.

- The positional matrix $I_{(n)}$ is updated by computing the position of the maximum value in all receptive fields R that are either affected by incoming events or are not persistent maximum locations.
- The list of persistent maximum locations is updated if new ones are discovered after the update.
- The matrices $F_{(n-1)}$ and $y_{(n-1)}$ are fetched at the locations of maximum values $I^t_{(n)}$ and forwarded to the next layer.
- All incoming events are forwarded to the next layer, but the events' coordinates are substituted with that of the receptive field R each event belongs to. Duplicate events arising from this re-indexing are removed before forwarding.

3.4 ALWAYS-ON EVENT-BASED OBJECT DETECTION

Object detection [147–149] is one of the most challenging problems in computer vision as it involves both semantic understanding and localization capabilities. Early deep learning methods have tackled the challenge in a multi-stage fashion. After predicting region proposals containing an object, a classification network infers the object category, and a second regression network predicts the object's size and position, i.e., the bounding box. The R-CNN [243] model is the first architecture of this sort to be proposed in the literature. A number of later works, such as the spatial pyramid pooling (SPP)-Net [244], the Fast [245] and Faster [191] R-CNNs, as well as the region-based fully convolutional network (R-FCN) [246], offer several refinements to this multi-stage design. Nevertheless, employing these pipelines in real-world applications is often a challenge. Each stage must be trained independently and, at deploy time, it needs to wait for the previous stage prediction to be completed, which adds to the overall time required and becomes the bottleneck in real-time applications [19].

One-stage frameworks offer a solution to this problem. Instead of optimizing each task separately, these architectures directly regress and classify both bounding boxes coordinates and class probabilities with a single network pass, significantly lowering the time expense. Pioneer works on this category are the YOLO [19] and SSD [224] networks, which inspired later works to improve robustness and efficiency of this kind of architectures [247–250].

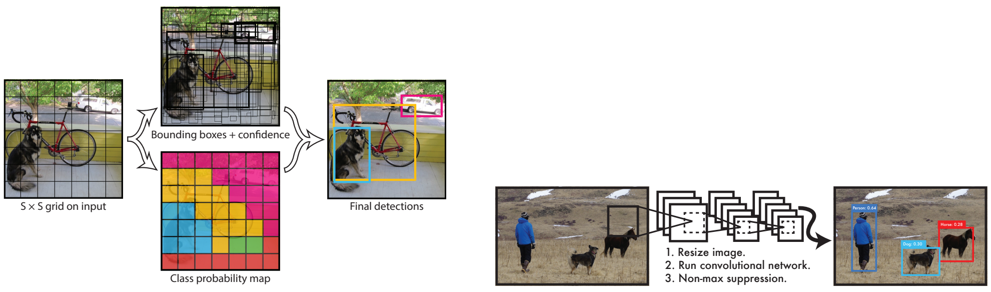
Despite the success of deep learning networks in this field, very few works propose to detect objects from event-based camera streams [189, 251, 252]. The primary obstacle that limited the research can be attributed to the lack of large-scale datasets, which constrained the design and complexity of early works in this area[2]. In the remainder of this chapter, we propose to exploit the proposed event-based layers to design an asynchronous network for event-based object detection. The resulting architecture can perform either batch-based or event-by-event computation, detecting objects as soon as enough information is accumulated. In the following, we first review the YOLO [19] object detector, as it constitutes the foundation of our architecture, and then present the proposed YOLE architecture.

3.4.1 YOLO: You Only Look Once

YOLO is a real-time object detection network proposed by Redmon et al. [19]. Its single-stage design is the result of combining a feedforward single-branch architecture, similar to traditional CNNs for object classification, with a novel prediction scheme and training loss. The core idea is detailed in Figure 3.4a on the next page.

OPERATING PRINCIPLES. YOLO divides the input image into a grid of $S \times S$ cells, each responsible for predicting the object, if present, centered on that cell. Each of these regions is associated with a fixed number B of predicted bounding boxes proposals (position and size), their confidence scores $\Pr(Object) * \text{IOU}_{\text{pred}}^{\text{truth}}$, and a class probability distribution $\Pr(Class_i \mid Object)$, shared among all boxes. The predicted confidence score reflects how much the network is confident that an object is actually contained in each predicted box, and it is set to be equal to the intersection-over-union between the predicted box and the ground truth during training. The network is thus asked to predict its own performance, making prediction easier to accomplish.

2 At the time this research was conducted, very few prior works [189, 251, 252] proposed to tackle object detection with events, and no proper object detection dataset was publicly available. With the recent release of large-scale datasets for event-based detection [104, 135], the interest in this area is constantly increasing, and new event-based solutions [104, 189, 190] are being proposed. Refer to Section 2.2.4 on page 35 for an overview of recent datasets and architectures.



(a) (b)

Figure 3.4: (a) The YOLO's bounding box prediction scheme. The image is first divided into a grid of $S \times S$ cells, each predicting: a class probability (depicted on the bottom) and a set of boxes proposals, together with their confidence score (depicted on top with thickness proportional to the confidence). **(b)** The final set of predictions is obtained through non-maximum suppression and confidence thresholding. *Figures taken from Redmon et al. [19]. Copyright ©2016 IEEE.*

Indeed, the confidence score can directly be used to discriminate between valid and invalid proposals, simplifying the proposal selection process to simple thresholding over the confidence scores. At test time, the object class is finally computed as

$$\Pr\left(Class_i \mid Object\right) * \Pr(Object) * \mathrm{IOU}_{\text{pred}}^{\text{truth}} = \Pr(Class_i) * \mathrm{IOU}_{\text{pred}}^{\text{truth}}, \tag{3.9}$$

by combining the cell's conditional probability $\Pr\left(Class_i \mid Object\right)$ with the objectness $\Pr(Object)$.

IMPLEMENTATION. During training, the following multi-part loss is optimized:

$$\begin{aligned}
\mathcal{L} = {} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{obj}} \left(C_{ij} - \hat{C}_{ij} \right)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{\text{nobj}} \left(C_{ij} - \hat{C}_{ij} \right)^2 \\
& + \sum_{i=0}^{S^2} 1_{i}^{\text{obj}} \sum_{c \in \text{ classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 .
\end{aligned} \tag{3.10}$$

Every cell $i \in S^2$ is associated with $5 * B + \mathcal{C}$ different values. The first set entails the $j = 1, \ldots, B-1$ bounding boxes proposals predicted by each cell, each defined by 5 values: the bounding box center (x_{ij}, y_{ij}) relative to the cell position, its size (h_{ij}, w_{ij}), and the confidence C_{ij}. The last set of $\mathcal{C}$ values corresponds to the predicted class probabilities $p_i(c)$. This multi-value

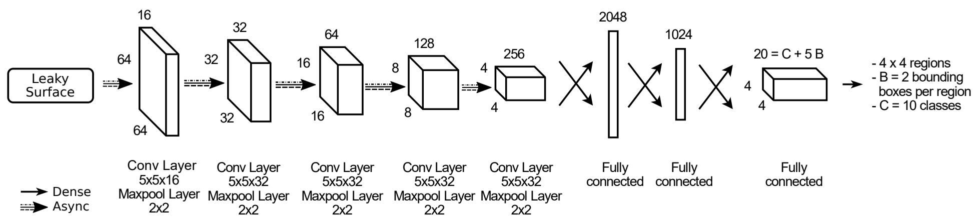

Leaky Surface
16
64
64
32
32
32
64
16
16
128
8
8
256
4
4
2048
1024
20 = C + 5 B
4
4
- 4 x 4 regions
- B = 2 bounding boxes per region
- C = 10 classes
Conv Layer 5x5x16 Maxpool Layer 2x2
Conv Layer 5x5x32 Maxpool Layer 2x2
Conv Layer 5x5x32 Maxpool Layer 2x2
Conv Layer 5x5x32 Maxpool Layer 2x2
Conv Layer 5x5x32 Maxpool Layer 2x2
Fully connected
Fully connected
Fully connected
Dense
Async

Figure 3.5: The YOLE detection network used to detect MNIST-DVS digits. The input 128×128 surfaces are divided into a grid of 4×4 regions at the output of the network, which predicts 2 bounding boxes and 10 class categories each.

prediction is accomplished in a single forward pass by directly predicting a dense $S \times S \times (5B + C)$ output through standard convolutional and fully connected layers.

During training, for each cell, only the proposal having the maximum IOU with a ground truth box is considered responsible for a prediction, indicated with single subscripts (h_i, y_i, w_i, h_i) in the loss function and with the indicator value $1_{ij}^{\text{obj}} = 1$. This proposal is asked to match the ground truth's values and predict a confidence score of $\hat{C}_{ij} = 1$ (first three terms of the loss). All remaining proposals are trained instead to minimize the predicted confidence ($\hat{C}_{ij} = 0$), while no constraint is enforced on their size and location.

By combining a proposal-based network with a multi-value regression loss, YOLO is capable of predicting the position, size, and class of all the objects in the scene with just a single forward pass of the network. Basic thresholding over the confidence scores and a non-maximum suppression refinement step are introduced during testing to remove any duplicate prediction, as shown in Figure 3.4b on the facing page. The network can process images in real-time at 45 fps, although simplified versions can even reach 155 fps [147].

3.4.2 YOLE: You Only Look at Events

Motivated by YOLO's fast prediction rates, we propose combining its single-stage design with our leaky surface layer to obtain an object detection network for event cameras. We call this approach "YOLE: You Only Look at Events." Networks thus obtained are fully-differentiable, meaning that we can exploit the YOLO's training procedure and backpropagation-based optimizer to learn to detect objects from event streams end-to-end.

We design two versions of this architecture. The first, *YOLE*, only partially exploits the proposed event-based framework as it also makes use of dense, not even-based, layers at the end of the network. We use this model as a reference to highlight the strengths and weaknesses of the event-based layers described in Section 3.3 on page 45. We extend this first version to a fully convolutional variant named *fcYOLE* and implement its architecture entirely with the proposed event-based layers as detailed below.

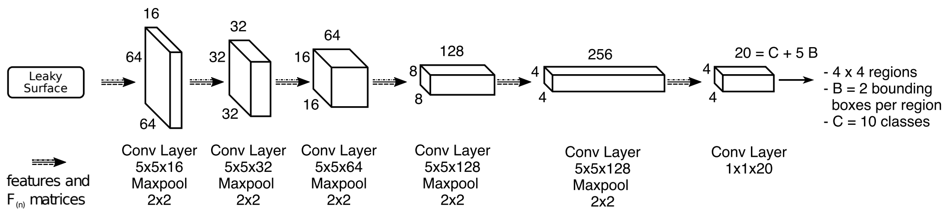

Leaky Surface
16
64
64
Conv Layer 5x5x16 Maxpool 2x2
32
32
32
Conv Layer 5x5x32 Maxpool 2x2
64
16
16
Conv Layer 5x5x64 Maxpool 2x2
128
8
8
Conv Layer 5x5x128 Maxpool 2x2
256
4
4
Conv Layer 5x5x128 Maxpool 2x2
20 = C + 5 B
4
4
Conv Layer 1x1x20
- 4 x 4 regions
- B = 2 bounding boxes per region
- C = 10 classes
features and F(n) matrices

Figure 3.6: fcYOLE: a fully-convolutional detection network based on YOLE. Fully-connected layers are replaced with 1×1 convolutions for directly predicting bounding box proposals.

YOLE NETWORK. The YOLE network combines the leaky surface proposed in Section 3.3.1 with a simpler version of the YOLO network [19]. Its architecture is depicted in Figure 3.5 on the preceding page. YOLE processes 128×128 surfaces, it predicts $B = 2$ bounding boxes for each region and classifies objects into C different categories. We follow the architectural designs of YOLO and add fully-connected layers at the end to expand the receptive field of cells in the output layer. This design choice enables them both to handle objects whose bounds exceed that of the cell and to reason using cues from the entire scene. We train the network on traditional layers and then reuse its weights within a network having the same structure but composed of the proposed *e-conv* and *e-max-pool* layers. Fully-connected layers are kept as dense, not event-based, operations in this version of the network.

FCYOLE NETWORK. Fully-connected layers cannot be easily extended to perform event-based processing. Indeed, every one of their output values depends on all the values of the input feature map. If just one input pixel is affected by an event, all the layer's output predictions must be completely recomputed. To address this issue, we propose replacing the last fully-connected layers with convolutional ones, obtaining what is generally known as a fully-convolutional architecture [219]. In particular, as depicted in Figure 3.6, we substitute the last set of fully connected layers with 1×1 convolutions that directly map feature maps into the bounding boxes predictions. As the network thus obtained is only composed of convolutional and max-pooling layers, we can be easily convert it into a fully event-based variant using *e-conv* and *e-max-pool* layers only. We call this variant fully-convolutional YOLE, or *fcYOLE*.

3.5 EXPERIMENTS

In this section, we provide details on the set of experiments we designed to test the proposed framework and detection networks. We begin by describing the custom detection benchmarks we created using available object recognition datasets. The performance of the proposed architectures is then discussed.

3.5.1 Datasets

At the time of this study, only a few event-based datasets for deep learning applications were publicly available in the literature, the most popular ones being: N-MNIST [116], MNIST-DVS [128], CIFAR10-DVS [189], N-Caltech101 [116] and POKER-DVS [128]. All of these datasets, except for POKER-DVS, are derived from their original frame-based versions [141, 143, 253] by capturing images displayed on a monitor with an event camera while moving the camera itself or the images to trigger events. Among these, N-Caltech101 is the only dataset providing bounding boxes for object detection. A comparison between these datasets is provided in Section 2.2.3 on page 27.

To benchmark the proposed detection networks in different scenarios, ranging from simple to complex, we propose extending available object classification datasets into object detection variants. These are created by combining multiple samples from the N-MNIST and MNIST-DVS datasets, obtaining the *Shifted N-MNIST* and *Shifted MNIST-DVS* extended datasets, and by providing bounding boxes for the original POKER-DVS, resulting in *OD-Poker-DVS*. A similar dataset, *Blackboard MNIST*, is also obtained in simulation by recording digits written on a blackboard. In the following, we go through how these datasets are created in further detail. Samples from each dataset are provided in Figure 3.7 on the next page.

SHIFTED N-MNIST The N-MNIST [116] dataset is a conversion of the popular MNIST [141] image dataset for computer vision. We enhanced this collection by building a slightly more complex set of recordings. Each new sample is composed of either one (*Shifted N-MNIST v1*) or two (*Shifted N-MNIS v2*) N-MNIST samples placed in random non-overlapping locations of a bigger 124×124 field of view.

As a preprocessing step, we generate bounding boxes for each N-MNIST digit individually. We first compute a sequence of surfaces as detailed in Section 3.3.1 on page 46, and then remove noisy locations by considering only non-zero pixels having at least other ρ active pixels within a circle of radius R around them. Then, with a custom version of the DBSCAN [254], we aggregate pixels into a single cluster to compute a bounding box that tightly frames the digit. We used $\rho = 3$, $R = 2$, and set a threshold $min_{surf} = 10$ to filter out small bounding boxes extracted in correspondence of low events activities.

To test the robustness of the proposed models, we further extend this dataset by introducing additional noise patterns. Following the procedure used to design the *Cluttered Translated MNIST* dataset [255], we add five 8×8 fragments of non-target objects (*Shifted N-MNIST v2fr*) cropped from random N-MNIST digits in random locations, and 200 additional random events per leaky frame (*Shifted N-MNIST v2fr+ns*).

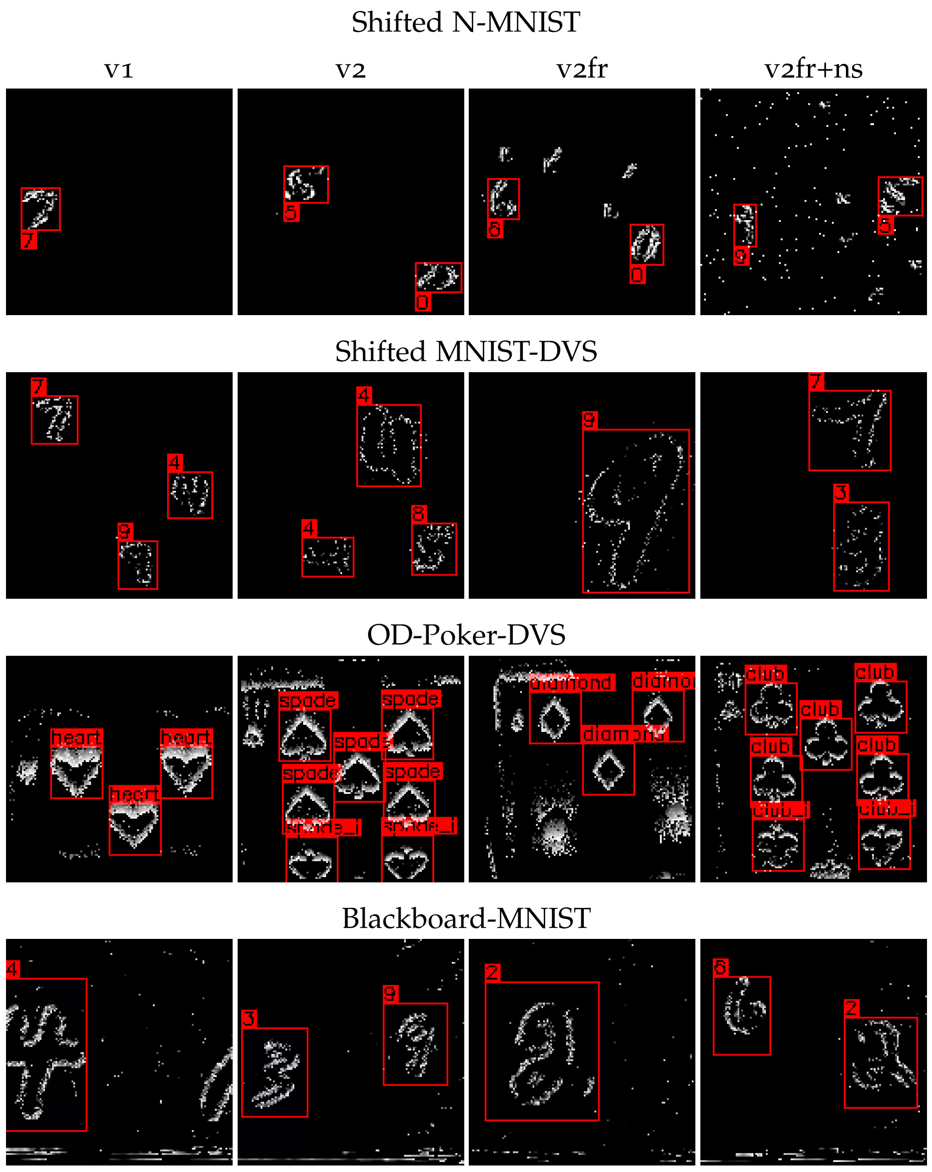


Figure 3.7: Examples of samples from the proposed detection datasets.

Each version of the dataset contains 60, 000 training and 10, 000 testing samples, as in the original N-MNSIT dataset. In Figure 3.7 on the preceding page we illustrate one sample for each of the four proposed variants.

SHIFTED MNIST-DVS MNIST-DVS [128] offers a set of recordings similar to that of N-MNIST [116]. However, differently from N-MNIST, digits are recorded at three different scales, namely, *scale4*, *scale8*, and *scale16*. We use a similar procedure to obtain Shifted MNIST-DVS recordings. We first extract bounding boxes with the same procedure detailed before and then place them in a 128×128 field of view by combining samples from the three different scales. In a given sample, four different scales combinations are possible: (i) three *scale4* digits, (ii) two scale8 digits, (iii) two scale4 digits mixed with one scale8 digit, and (iv) one scale16 digit placed in random locations of the field of view. The overall Shifted MNIST-DVS is composed of 30, 000 samples containing these four possible configurations.

OD-POKER-DVS The Poker-DVS [128] dataset is a small collection of neuromorphic recordings obtained by quickly browsing custom-made poker card decks in front of a DVS camera for $2 - 4$ seconds. The dataset is composed of 131 samples containing pips of four possible categories (spades, hearts, diamonds, or clubs) extracted from three decks recordings. Single pips are extracted using an event-based tracking algorithm that follows pips to extract 31×31 motion-compensated samples for classification.

With OD-Poker-DVS we extend its scope to object detection. To do so, we compute pips' bounding boxes within the original uncut deck recordings using the tracking algorithm [128] provided with the original dataset. We then split these recordings, following the procedure of Stromatias et al. [183] by dividing the sections containing visible digits into a set of shorter examples, each about 1.5 ms long. The final detection dataset is composed of 292 small examples, which we divided into 218 training and 74 testing samples.

BLACKBOARD-MNIST We use the DAVIS simulator released by Mueggler et al. [165] to build our own set of synthetic recordings. The resulting dataset consists of a number of samples showing digits written on a blackboard in random positions and with different scales. We preprocess a subset of images from the original MNIST dataset by removing their background and making them look like they were written with chalk. A random set of digits is then placed on a virtual blackboard, and the simulation is finally run, moving the virtual event camera in front of the blackboard to obtain event-based recordings together with bounding boxes of every visible digit. The rendering setup is shown in Figure 3.8 on the following page.

We built three sub-collections of recordings with increasing levels of complexity (easy, medium, and hard) which we then merged together to obtain the final dataset. In the easiest version, we place three digits of a fixed dimension (roughly corresponding to the middle scale of MNIST-DVS samples) in

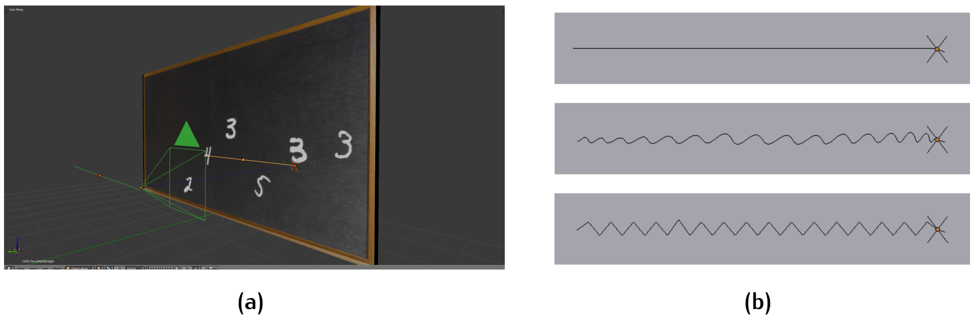


Figure 3.8: **(a)** The 3D Blender [167] scene used to generate the Blackboard MNIST dataset. The camera moves in front of the blackboard along a straight trajectory while following an hidden *focus object* that moves on the blackboard's surface, synchronized with the camera. The camera and its trajectory are depicted in green, the focus object is represented as a red cross and, finally, its trajectory is depicted as a yellow line. **(b)** The three types of focus trajectories.

the central region of the blackboard and use a single type of camera trajectory, moving the camera from right to left in a straight line. We collected a total of 1,200 samples (1,100 training and 100 testing).

The medium difficulty features more variability in the number and dimensions of the digits, and the types of camera movements. The portion of the blackboard on which digits are placed varies as well and can cover any region of the blackboard, including the edges. Camera motion, either from left-to-right or in the opposite direction, can be of the three kinds shown in Figure 3.8b. Finally, the number and dimensions of the digits are chosen following three possible configurations, similarly to the Shift MNIST-DVS dataset: either six small digits (with sizes comparable to scale4 MNIST-DVS digits), three intermediate-sized digits (comparable to the MNIST-DVS scale8), or two large digits (comparable to the biggest scale of the MNIST-DVS dataset, scale16). We generate 1,200 recordings with the same split structure as the previous set.

Finally, the hardest difficulty extends the second and third configuration described before by randomly resizing digits to a variable size spanning from the original configuration size down to the previous scale. A total of 600 new samples (550 training, 50 testing) are generated, 300 of them containing three digits each and the remaining consisting of two digits with variable size.

N-CALTECH101 The N-Caltech101 [116] collection is the only dataset we use which already comes with bounding boxes annotations. We split the dataset into 80% training and 20% testing samples using a stratified split. Since no ground truth bounding boxes are available for the background class, we do not use this category in our experiments.

Table 3.1: Classification performance (accuracy %) of YOLE against previous state-of-the-art synchronous and asynchronous architectures.

	Asynch.	Classifier	N-MNIST	N-Caltech101
H-First [140]	✓	spike-based	71.2	5.4
HOTS [84]	✓	histogram-based	80.8	21.0
Gabor-SNN [85]	✓	linear SVM	83.7	19.6
HATS [85]	✓	linear SVM	**99.1**	64.2
YOLE (Ours)	✓	MLP	94.9	**64.9**
Phased LSTM [258]		MLP	97.3	-

3.5.2 Experiments Setup

Event-based datasets, especially those based on MNIST, are simpler than those typically used to train image-based YOLO architectures [19]. Therefore, we design the MNIST object detection networks taking inspiration from the simpler LeNet [141] model composed of 6 conv-pool layers for feature extraction. Both YOLE and fcYOLE share the same structure up to the last regression and classification layers.

We adopt a more advanced design to detect objects on N-Caltech101, taking inspiration from the VGG16 architecture [222]. VGG16 is a feedforward classification network made of five consecutive blocks of convolutional layers separated by max-pooling. We simplify this structure utilizing only one convolution per block, obtaining better performance. Finally, we set the dimensions of the last fully connected layer to produce a grid of 5×7 regions predicting $B = 2$ bounding boxes and 100 classes each. As in the original YOLO architecture, we use Leaky ReLU in the hidden layers and a linear activation in the last one.

Following Redmon et al. [19], in all experiments, we first pre-train the first 4 convolutional layers to classify target objects, while we use a Xavier initialization [256] to initialize the remaining ones. All networks are trained optimizing the multi-objective loss in Equation 3.10 on page 54 using Adam [257], learning rate 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The batch size is chosen depending on the dataset to fill GPU memory optimally: 10 for Shifted N-MNIST, 40 for Shifted MNIST-DVS, and N-Caltech101, 25 for Blackboard MNIST, and 35 for Poker-DVS. Early-stopping is applied to prevent overfitting using validation sets of the same size as the test set.

3.5.3 Experimental Results

CLASSIFICATION PERFORMANCE OF YOLE. We first benchmark the object recognition capabilities of the proposed YOLE detection architecture on the N-MNIST [116] and N-Caltech101 [116] datasets. We compute the classifica-

Table 3.2: YOLE detection performance on S-N-MNIST variants in terms of both accuracy (%) and mean average precision.

	S-N-MNIST			
	v1	v2	v2fr	v2fr+ns
accuracy	94.9	94.7	88.6	85.5
mAP	91.3	90.5	81.5	77.4

tion accuracy by first selecting the predicted bounding box having the highest intersection over union (IOU) with the ground truth box and then using its class prediction to compute the classification score. We compare the classification predictions thus obtained with previous object recognition networks in Table 3.1 on the previous page. YOLE, despite not being trained only to perform classification, still achieves better results than previous recognition networks. In particular, it outperforms previous SNN-based solutions, such as HFirst [140] and Gabor-SNN [85], highlighting the advantage of using a deep architecture trained end-to-end to maximize performance compared to a shallow SNN architecture trained without backpropagation.

The performance of YOLE is also slightly better on N-Caltech101 than that of HATS [85], a prior work that combines an asynchronous mechanism to extract features with an SVM classifier for prediction. YOLE is similar, as it combines feature extraction through asynchronous event-based convolutional layers with a fully-connected classifier for prediction. However, as part of the YOLO training approach [19], we apply an MSE loss to train the network's output classification neurons (see Equation 3.10 on page 54), which is generally suboptimal if compared to the classification objective optimized by the SVM. Directly training to optimize a cross-entropy loss would most likely have resulted in better results.

DETECTION PERFORMANCE OF YOLE. We start by analyzing the detection performance of the YOLE architecture by gradually increasing the difficulty of the detection task. To do so, we use the different variants of the proposed Shifted N-MNIST dataset detailed in Section 3.5.1 on page 57. Results are reported in Table 3.2 in terms of both accuracies, computed as reported before, and mean average precisions [259]. We denote as *v1* the results obtained using scenes composed of a single digit and with *v2* those obtained with scenes containing two digits in random locations of the field of view. YOLE's 94.91% accuracy on single objects, which is higher than prior asynchronous networks reported in the previous paragraph, is achieved with a mean average precision of 91.3, indicating good localization capabilities as well. In the case of simultaneous prediction of multiple objects (Shifted N-MNIST v2), the network still shows good detection performance as the mean average precision remains nearly unchanged when switching from one object to two.

Table 3.3: Performance comparison between YOLE and fcYOLE on detection datasets.

	Classifier	Score	S-MNIST DVS	Blackboard MNIST	OD-Poker DVS	N-Caltech101
YOLE	MLP	mAP	92.0	87.4	82.2	39.8
		acc	96.1	90.4	87.3	64.9
fcYOLE	1×1 e-conv	mAP	87.4	84.7	78.7	26.9
		acc	94.0	88.5	79.1	57.1

The network performance starts deteriorating when additional non-target fragments (*v2fr*) and noisy (*v2fr+ns*) events are introduced. The maximum decrease in performance is recorded after adding fragments, which may indicate a bias toward detecting any object-like cluster and thus a poor ability in predicting reliable objectness scores (see Section 3.4.1 on page 53). However, the network still achieves good performance on both Blackboard-MNIST and OD-Poker-DVS, which include similar but significantly more realistic noise patterns.

YOLE VS. FCYOLE. We then compare the performance of YOLE against its fully-convolutional variant, fcYOLE. We use the proposed dataset extensions as well as N-Caltech101, and report detection results in Table 3.3. Qualitative results showing detection predictions on several samples are also provided in 3.9 on the next page.

Both variants achieve good detection results in most of the benchmarks. YOLE performs very well (mAP above 80%) on the Shifted-MNIST-DVS, Blackboard MNIST, and OD-Poker-DVS datasets, as expected from the previous analysis. The only dataset where the network struggles in recognizing objects is N-Caltech101. The increased difficulty of the task, but most significantly, the uneven nature of the dataset, explains this behavior. Classes in N-Caltech101 [116] are highly unbalanced, with samples per class statistics ranging from 800 (*Airplanes*) to just 31 samples (*Inline_Skate*). As a result, the network tends to specialize in predicting the most populous classes, such as *Airplanes*, *Motorbikes* and *Faces_Easy*, while it performs poorly on classes with just a few training samples, explaining the poor aggregate scores reported in Table 3.3. We analyze this trend in Table 3.4 on page 66, where we report average precisions for each distinct category. This behavior is common even in the original Caltech101 [253] dataset, and it is usually avoided by pre-training on additional data. This strategy has become applicable in event-based vision only very recently, thanks to the recent release of ImageNet-based datasets [112, 260], and were not available at the time of this research.

When using fcYOLE, instead, we registered a slight decrease in performance compared to the results we obtained using YOLE. This gap is caused

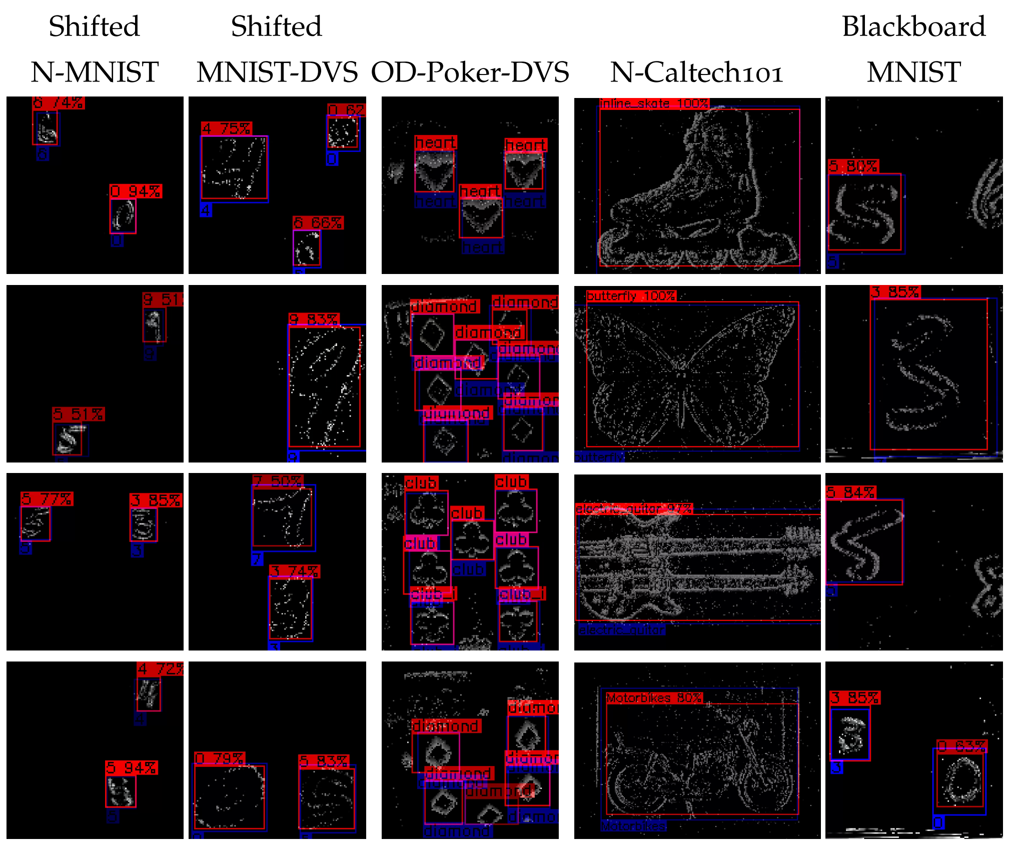


Figure 3.9: Examples of YOLE predictions. Ground truth boxes are depicted in blue, while predictions are indicated in red. The classified class is also reported, along with its class probability.

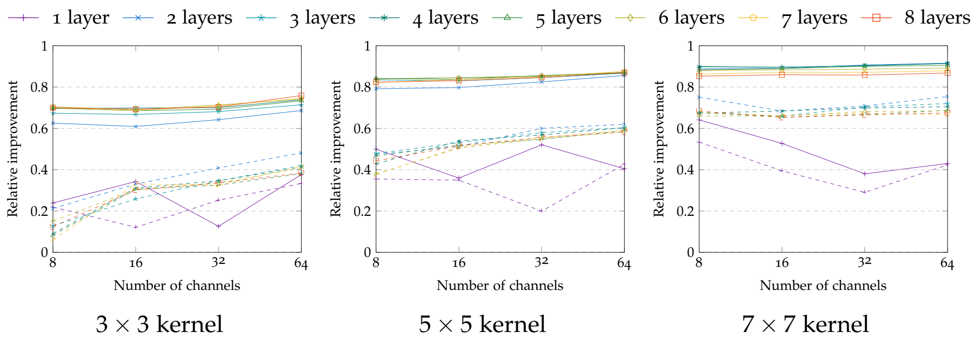


Figure 3.10: Time evaluation of the proposed e-conv and e-max-pool layers under different network configurations. The plots show the relative improvement when performing event-by-event computation of an event-based network, implemented with the proposed e-conv and e-max-pool layers, over a network with traditional layers. We run the evaluation with different kernel sizes, number of features, number of conv layers, and interleaving (dashed lines) or not (solid lines) a max-pooling layer after each convolution.

by the fact that every cell in fcYOLE generates predictions by only looking at the visual information in a limited portion of the field of view. Indeed, by replacing the fully connected layers at the end with 1×1 convolutions, cells' predictions become limited to the network receptive [261], in this case, 32×32. If an object is only partially contained inside this region, the network must estimate the object dimensions and class based on limited information. However, it should be stressed that the difference in performance between the two architectures does not come from the use of the proposed event layers, whose outputs are the same as the conventional ones, but rather from the reduced expressive power of the network. Nevertheless, by removing the last fully-connected levels, we created a detection network with fewer parameters, but most importantly, entirely made up of event-based layers.

ASYNCHRONOUS EVENT-BY-EVENT TIME PERFORMANCE. To identify the advantages and weaknesses of the proposed event-based framework in terms of inference time, we compare a network composed of a variable number of e-conv and e-max-pool layers against one made of regular network operations. Our event-based framework runs on CPU, and it is implemented in Python with combined NumPy [262] and C-based operations. For fairness in the evaluation, we implement both network versions with the same libraries to ensure an equal level of code optimization. Networks used in this analysis have a 256×256 input resolution, and they are composed of a variable number of layers. We start by comparing just one convolutional layer and then incrementally stack several of them by optionally interleaving a max-pooling layer in between. Several network configurations are tested by varying the kernel size (either 3×3, 5×5, or 7×7) and the number of output channels of each convolution (8, 16, 32, and 64). Convolutions are configured with

Table 3.4: YOLE and fcYOLE average precisions on N-Caltech101

	Motorbikes	airplanes	Faces_easy	watch	Leopards	chair	bonsai	car_side	ketch	flamingo	ant	chandelier	crocodile	grand_piano	brain	hawksbill	butterfly	helicopter	menorah	starfish
AP fcYOLE	97.5	96.8	92.2	75.7	57.2	7.5	30.2	70.3	42.3	2.3	2.4	34.8	0.0	69.5	35.3	19.6	33.5	8.6	67.7	23.2
AP YOLE	97.8	95.8	94.7	84.2	62.9	17.3	59.3	61.7	52.9	10.0	25.8	55.7	1.6	81.3	53.3	29.1	46.3	14.9	80.7	32.7
N_{train}	480	480	261	145	120	109	78	75	70	68	66	65	61	61	60	60	55	54	53	52

	scorpion	kangaroo	trilobite	sunflower	buddha	ewer	revolver	laptop	llama	ibis	minaret	umbrella	crab	electric_guitar	cougar_face	dragonfly	crayfish	dalmatian	ferry	euphonium
AP fcYOLE	2.5	3.4	41.2	29.1	46.5	35.3	20.3	40.0	1.4	1.5	59.5	61.0	5.0	23.2	21.8	55.6	7.3	24.9	29.7	43.5
AP YOLE	6.9	5.0	62.5	43.3	57.2	51.3	57.4	88.1	10.2	6.5	81.3	85.9	19.5	29.7	39.8	59.9	9.5	33.0	34.0	53.6
N_{train}	52	52	52	51	51	51	50	49	48	48	46	45	45	45	43	42	42	41	41	40

	lotus	stop_sign	joshua_tree	soccer_ball	elephant	schooner	dolphin	lamp	stegosaurus	rhino	wheelchair	cellphone	yin_yang	cup	sea_horse	pyramid	windsor_chair	hedgehog	bass	nautilus
AP fcYOLE	18.1	55.1	30.2	57.3	11.4	37.3	6.5	17.7	34.5	5.8	25.8	46.5	60.4	5.6	1.9	41.1	52.3	13.3	4.2	13.0
AP YOLE	27.6	61.7	29.8	51.5	6.0	56.8	11.5	45.3	44.6	11.3	25.3	54.6	63.3	17.5	8.8	48.6	65.2	9.8	4.0	50.4
N_{train}	40	40	40	40	40	39	39	37	37	37	37	37	36	35	35	35	34	34	34	33

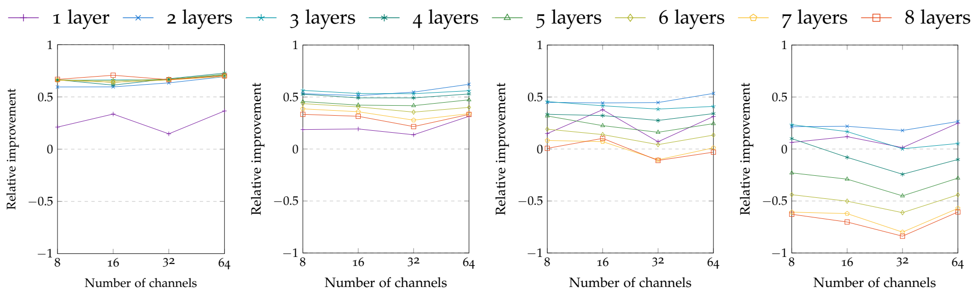
	pizza	emu	accordion	dollar_bill	tick	crocodile_head	gramophone	rooster	camera	pagoda	cougar_body	barrel	ceiling_fan	beaver	cannon	mandolin	flamingo_head	brontosaurus	stapler	pigeon
AP fcYOLE	17.4	5.6	48.3	74.4	28.2	9.8	30.6	26.6	15.1	34.0	0.0	30.7	40.8	0.5	0.0	6.6	2.7	0.2	46.0	6.2
AP YOLE	54.5	5.0	52.7	86.5	55.2	10.0	48.4	64.5	32.7	33.3	12.2	41.2	50.1	0.0	0.3	43.4	9.3	25.8	68.1	43.1
N_{train}	33	33	33	32	31	31	31	31	30	29	29	29	29	28	27	27	27	27	27	27

	headphone	anchor	scissors	wrench	okapi	lobster	panda	saxophone	mayfly	water_lilly	garfield	wild_cat	gerenuk	platypus	binocular	octopus	strawberry	snoopy	metronome	inline_skate
AP fcYOLE	10.7	14.6	28.2	14.7	10.0	0.0	4.7	59.7	0.5	3.1	23.4	0.0	4.8	13.1	0.4	14.0	0.5	8.3	63.4	37.2
AP YOLE	21.1	17.3	47.5	29.7	44.6	0.0	12.2	68.4	0.7	14.7	62.3	0.0	7.2	34.7	11.8	13.8	29.4	53.1	88.3	75.1
N_{train}	26	26	25	25	25	25	24	24	24	23	22	22	22	22	21	21	21	21	20	19

stride 1, and padding is always applied to maintain the resolution unchanged, making max-pooling layers the only downsampling operation. We run each network configuration with both event-based and regular layers and report the relative improvement as $(T_{frame} - T_{event})/T_{frame}$, which evaluates the decrease of prediction time, in percentage, when using event-based layers compared to the frame-based reference. We run the evaluation on CPU using an Intel(R) Xeon(R) Gold 6238R processor.

Since the proposed framework has been specifically designed to enable optimized event-by-event asynchronous computation in regular CNNs, we start by considering the setting in which networks are executed after every event's arrival. Results are provided in Figure 3.10 on page 65, where we report the timings obtained by processing sequences of 100 random events, one event at a time, uniformly sampled within a 256×256 frame and after having set the event-based layers in a random state by processing 10 random events with analogous characteristics. The proposed framework consistently outperforms the frame-based baseline in all network configurations, highlighting the advantage of using event-based layers for asynchronous computation and the benefits of performing sparse and localized updates over dense operations.

Intuitively, the improvement of the proposed layers is greater as the amount of computation that needs to be performed by a single kernel operation increases. In other words, by increasing the number of features produced by each convolution and its kernel size, the relative improvement grows consistently, reaching up to a 76.27% when 7×7 kernels are considered. The same considerations are also valid when max-pooling layers are removed (solid lines in the figure), as the same resolution is maintained throughout the whole architecture, and thus the computational load does not decrease with the depth (under an equal number of features). Under these network setups, that is, when max-pooling layers are removed, event-based layers achieve up to a 91.66% with 7×7 kernels. Notice that this configuration is very challenging for our event-based layers since, with large kernel sizes and without any subsampling operation, a single event is likely to affect many features in deep layers of the network, thus triggering recomputation instead of leveraging the proposed incremental update rules. Indeed, given a $K \times K$ kernel setup with stride 1, a single input pixel is contained in K^2 different receptive fields. In this setup, an e-conv layer receiving a new event triggers, in the worst case, K^2 new events in the next layer, which, in the absence of a subsampling layer in between, generate in turn $(K^2 + K - 1)^2$ new unique events, making sparseness in computation decrease with the depth. However, despite this behavior, the proposed event-based layers can still exploit lightweight computation in the remaining part of the feature map by exploiting their internal state and the proposed update rules. This mechanism lowers the computational cost compared to a traditional convolution where all receptive fields are always recomputed.



10 evs/batch 100 evs/batch 200 evs/batch 500 evs/batch

Figure 3.11: Time evaluation of the proposed e-conv layers under *synchronous* regimes. Events are grouped in batches of 10, 100, 200, and 500 events and used as input to a neural network with 3×3 kernel size, no max pooling, and variable number of layers.

SYNCHRONOUS BATCH-BASED TIME PERFORMANCE. Although we designed our framework to enable traditional CNNs to operate similarly to an SNN, enabling asynchronous and event-by-event computation, we also evaluate the proposed event-based layers when performing *synchnoronus* batch-based computation. We replicate the same setup as before, considering a network with no max-pooling layers, and focus on the setting that provided the smallest improvement in the previous evaluation to test the limits of event-based layers, i.e., when using 3×3 kernels. Evaluation is performed as before but grouping events into batches ranging from 10 to 500 events per batch. Results are reported in Figure 3.11.

As expected, the efficiency of the proposed event-based layers decreases as the number of events in each batch grows. This tendency is driven by the fact that different events may happen in distinct parts of the frame, triggering recomputation in different areas of the network and thus further decreasing the computational sparseness. Nevertheless, the proposed event-based layers can still deliver improved performance when batches contain up to 200 events, in most configurations, considering a 256×256 resolution. As the number of events increases, sparseness decreases, forcing feature maps to be recomputed almost entirely, especially in deep network layers. In this condition, the overhead of having to handle events makes the proposed layers underperform compared to the more straightforward strategy that always recomputes all the feature maps.

The condition studied in Figure 3.11 is indeed very challenging for the proposed event-based layers, as events are generated uniformly across the entire frame, without any spacial correlation. This increases the probability of having to recompute all the features and constitutes a worst-case scenario for our framework. To study how spatial correlation affects the proposed layers' performance, we focus on the 8 layers configuration, which provided the lowest improvement among configurations studied in Figure 3.11, and analyze how the improvement changes as a function of the events' sparsity. In

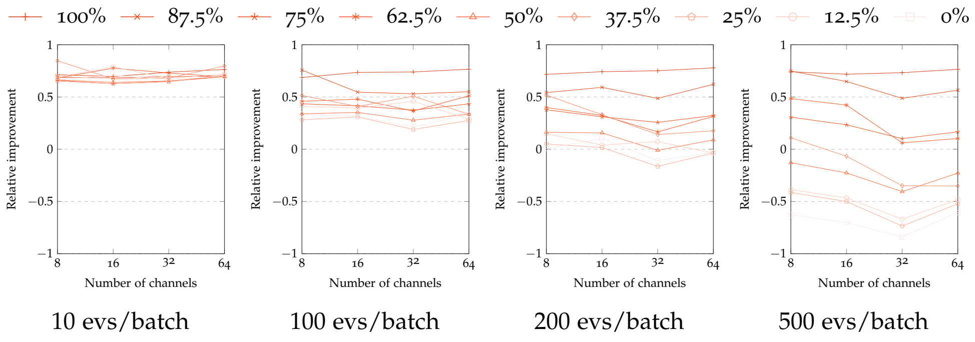


Figure 3.12: Time evaluation of the proposed e-conv layers under *synchronous* regimes and variable sparseness. The plots replicate the same setup in Figure 3.11 on the preceding page but consider *only the* 8 *layers configuration*. The network is executed under the same conditions, but varying the sparseness of the event stream, measured as $100 \cdot (1 - H_s * W_s/256^2)$, where the second term is the ratio between the region $H_s \times W_s$ from which events are sampled and the frame resolution 256×256.

this case, we call sparsity the value $100 \cdot (1 - H_s * W_s/256^2)$, which measures the ratio between the region $H_s \times W_s$ from which we sample random events, and the overall resolution 256×256 of the frame. We start by sampling all events from a squared region of 1×1 pixels (sparsity 100%) and then increase its size until covering all the frame (sparsity 0%).

As it can be seen from the results in Figure 3.12, the performance of the 8 layers configuration reported in the previous analysis (see red lines in Figure 3.11 on the preceding page) is only attained in the worst case when events are completely uncorrelated. In contrast, as sparsity grows, so does the improvement offered by the proposed event-based layers. This is true even when 500 events per batch are employed, where the proposed framework achieves worse performance only when sparsity is below 62.5%.

We analyze this behavior in more realistic conditions by measuring the inference time of the fcYOLE architecture against a network with the same structure but composed of traditional layers. We perform the evaluation on an Intel Xeon E5-2687W cpu on two datasets, namely Shifted N-MNIST and Blackboard MNIST. We group events into batches of 10ms and average timings across 1000 runs. Under our framework, fcYOLE achieves a 2x speedup (22.6ms per batch) on the first dataset, highlighting the advantage of using incremental sparse layers for computation. On Blackboard MNIST, however, it performs slightly slower (43.2ms per batch) compared to a network making use of conventional layers (34.6ms per batch).

The second benchmark is indeed challenging for our framework since changes are not localized in restricted regions. In noisy scenes, such as those in Blackboard MNIST, this condition limits the performance of the proposed framework and favors traditional CNNs, as they can leverage highly optimized implementations. We implemented our framework in Python, combining NumPy and C-based operations. However, with further optimizations (e.g., a C-only implementation) or specific hardware, we expect the overhead of handling events to decrease even in these edge cases, enabling the framework to better handle dense scenes, where recomputing feature maps may become necessary, and, at the same time, to fully exploit sparse event-based computation whenever possible, adaptively.

3.6 CONCLUSIONS

In this chapter, we proposed a framework for enabling asynchronous computation in traditional deep neural networks. We test the proposed event-based layers on object detection, an overlooked task in event-based vision, and design two asynchronous networks based on the YOLO one-stage detector. We take inspiration from SNNs to design an asynchronous leak-based event representation and incorporate its update mechanism into deep neural network layers to design asynchronous update rules that avoid recomputing feature maps from scratch.

The resulting detectors dynamically respond to input event activity by producing results only as a consequence of incoming events and by maintaining their internal states without performing any additional computation when no events arrive. The proposed event-based framework is not limited to object detection but can potentially be used to implement any state-of-the-art fully-convolutional architecture for any task. We analyzed the time performance of this formalization, outperforming frame-based layers when asynchronous event-by-event computation is performed and obtaining promising results even with batch-based computation.

A later approach, proposed by Messikommer et al. [87], builds upon the mechanisms presented in this chapter and achieves improved performance even in more challenging conditions. They depart from the LIF neurons' inspiration and propose to take advantage of recursive event representations. These do not enforce a temporal synchronization over the entire surface, contrary to the leaking mechanism we use, but only update the pixel corresponding to the incoming event, removing the need to synchronize internal features. Moreover, they focus on a different class of CNNs that make use of Submanifold Sparse Convolutions [263] to promote sparse activation maps. These are different from standard CNNs, which we use in our analysis, but produce comparable performance. Similar to our mechanism, their layers maintain a memory of the previous output and employ update rules to perform incremental computation. They achieve a 2.75x speedup on N-Caltech101 samples but perform slower than networks implemented with highly-optimized operations, as in our case. We believe further research on ad-hoc hardware implementations should be conducted to unlock the use of such methods in real-world settings and to evaluate their true potential against GPU-based and spike-based solutions.

4 LEARNING REPRESENTATIONS FOR EVENT-BASED NEURAL NETWORKS

The preceding chapter focused on event representations at different levels. We first presented a novel bio-inspired input representation and then designed a framework to enable asynchronous processing by relying on the concept of building intermediate hidden representations incrementally. We chose to use the suggested bio-inspired encoding because of its recursive formulation, which allowed us to create the proposed event-based layers. Nonetheless, the proposed representation is still hand-crafted, and careful adjustment of its hyperparameters may be required to achieve peak performance. That is also true for the vast majority of existing event representations, which, due to their hand-engineered design, may not convey all the information necessary for effectively solving the task at hand. While most of the research has focused on adapting downstream networks and learning procedures to event-based computation, relatively little effort has been put into learning how to encode raw event data optimally for maximizing task performance.

In this chapter, we propose addressing this challenge by framing the task of building an event representation as a layer of a deep neural network. Along the lines of the previous chapter, we propose to achieve this incrementally and sparsely. We design *MatrixLSTM*, a novel mechanism that uses Long Short-Term Memory (LSTM) cells to extract task-specific representations incrementally. We show that MatrixLSTM is capable of interfacing with any CNN, regardless of the task or the loss function to minimize. Our learned representation shows good flexibility and expressiveness in optical flow estimation and object detection compared to existing aggregation approaches. It improves the state-of-the-art of event-based object classification on the N-Cars dataset [85] and shows competitive performance in terms of efficiency thanks to a highly optimized encoding pipeline.

4.1 INTRODUCTION

Contrary to standard vision devices, event-based cameras do not only encode appearance (i.e., brightness), but they also provide very accurate motion

This chapter is based on: Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. "A Differentiable Recurrent Surface for Asynchronous Event-Based Data." In: *Computer Vision – ECCV 2020*. Springer International Publishing, 2020, pp. 136–152. DOI: 10.1007/978-3-030-58565-5_9 information. These two variables are strictly interconnected in the event stream [193, 264], and cleverly encoded with just polarity and time information within the spatio-temporal relations between concurrent events. Indeed, events alone only convey binary information (the increase or decrease in brightness at a specific point in time and space), and they only acquire meaning if related to previous event activity. This process of relating events gets even more critical in the presence of noise, which, as for any imaging device, may affect event-based cameras. Shot noise in photons or noise deriving from electronics, such as pixel fabrication mismatch, sub-threshold transistor current, and other nonidealities, can cause the camera to produce an unrelated or unideal event response.

Some works perform processing event-by-event by leveraging dynamical systems, such as SNNs, or filtering-based techniques [74–78] that are continuously and asynchronously updated to produce predictions with minimal latency. These systems typically rely on an internal state encoding the algorithm belief of the scene state and its evolution in time to perform temporal reasoning and handle noise. While very efficient, these mechanisms are often sensitive to noise and parameter tuning, and they rarely scale to challenging high-level vision tasks. For this reason, another line of research proposes to process events in batches to build a sufficient signal-to-noise ratio and produce reliable predictions. Packet-based processing has been successfully applied in many applications, including motion compensation and segmentation [88, 120, 121], optical flow prediction [162, 202], grayscale reconstruction [80, 162, 264], and many others. An overview is provided in Section 2.2.4 on page 35.

Along this line, as discussed in the preceding chapter, substantial progress has recently been achieved by converting short event sequences into image-like event representations. This approach exploits the similarity between these reconstructions and traditional images and directly unlocks many frame-based approaches to also process events. That is the case of learning-based methods, mostly CNNs, where the use of event representations has enabled advanced deep learning tools, like adversarial training [106, 265, 266], unsupervised learning [55, 56] and adaptation [3, 4, 190], as well as transfer learning [92, 126], to be directly used without modifications.

Nevertheless, choosing the correct representation to utilize for a specific task is not straightforward since it directly controls information accessible during training and inference and can thus have an impact on the system's performance. As a result, several task-specific representations have emerged over the past few years [93, 94, 112, 137]. However, these methods still rely on hand-engineered procedures for aggregating events, which cost work to design and may not transfer to other tasks. Deep learning algorithms have only recently been used to learn aggregate events in a data-driven manner [92] to enhance task performance.

4.1.1 Main Contributions

This chapter explores this new trend in event-based vision and presents a novel technique for learning to efficiently and effectively aggregate raw events. We employ an Long Short-Term Memory (LSTM) cell [20] as a convolutional filter and apply it over the stream of events to learn to extract 3D event representations that aggregate pixel information through time. The whole process is end-to-end differentiable, which means it can be trained alongside a state-of-the-art network to extract representations containing all the information required to tackle the task at hand. Most crucially, the proposed procedure is explicitly designed to maintain sparsity during the encoding process. It exclusively evaluates pixels receiving events and directly operates on raw event features without performing any intermediate densification, which is generally required when using similar computer vision mechanisms, such as ConvLSTM [192].

We consider state-of-the-art architectures for event processing and replace their hand-crafted representations with our learnable layer. We show this simple procedure is sufficient to significantly increase their performance even without special effort in hyper-parameter tuning, thus allowing for seamless integration with events regardless of the architecture or objective function to optimize. The resulting layer acts as a drop-in replacement for hand-crafted representations. It lowers the effort required for designing effective event encodings for the task at hand and enhances the performance of existing architectures with minor modifications. In summary, the contributions of this chapter are the following:

- We propose Matrix-LSTM, a task-independent approach for extracting grid-like event representations from asynchronous event streams. The framework is end-to-end differentiable. It can be used as input of any existing frame-based state-of-the-art architecture and jointly trained to extract the best representation from raw events. We detail the proposed encoding process and motivate our design choices, along with several other variants, in Section 4.3.1 on page 77.

- Section 4.4 on page 80 describes a set of custom CUDA kernels for efficiently aggregating events based on position and performing the proposed convolution-like operation over the events' stream. These are general order-aware group operations that enable to aggregate events in their pixel location while maintaining the arrival order. As such, they can be utilized even outside the proposed Matrix-LSTM layer.

- In Section 4.5 on page 82, we conduct experiments by replacing input representations with a Matrix-LSTM layer in existing architectures. We show that this simple modification is enough to improve the state-of-the-art on event-based object classification on N-CARS [85] by 3.3% and perform better than hand-crafted features on N-Caltech101 [116]. We also report a relative improvement on the MVSEC optical flow estimation benchmark [136] of up to 30.76% over hand-crafted features [136] and up to 23.07% over end-to-end differentiable ones [92]. Finally, we obtain object detection results on par with grayscale reconstruction mechanisms on the Gen1 Automotive Detection [135] dataset, yet without being able to fully exploit pre-trained feature extractors from images as opposed to these baselines.

- In Section 4.5.4 on page 94, we perform an extensive time performance evaluation of the proposed layer, showing competitive results over traditional recurrent architectures, such as ConvLSTM [192], on sparse event streams. Matrix-LSTM can also adapt to different aggregation intervals and can enable reliable predictions even at low latencies.
- We implement the Matrix-LSTM layer both in PyTorch [267] and TensorFlow [268] to enable easier integration of the proposed learnable representation in existing architectures [1].

4.2 RELATED WORK

Event-based vision is following a similar trajectory as that traditional image-based computer vision experienced in the last few years, where the field has steadily migrated from hand-engineered features to learning-based feature extractors. This section gives a brief overview of prior works, with an emphasis on event-based data representations and their relationship to the method proposed in this chapter. We recommend the reading of Section 2.2.2 on page 18 for a comprehensive summary on this subject.

HAND-CRAFTED REPRESENTATIONS. Several hand-engineered encodings for event-based data have been presented throughout the years, ranging from biologically inspired [1, 41, 269] representations, such as those used in SNNs [63], to mechanisms specifically designed for solving a task. To that purpose, the notion of *time-surface* was recently introduced by Lagorce et al. [84], who propose to extract events' spatio-temporal signatures by applying exponential kernels across a time-frame containing, in each location, the timestamp of the latest event. Sironi et al. [85] suggest extending this approach by using memory cells to store temporal information from previous events. As a result, their HATS representation is more noise-resistant as it can leverage a fixed-length buffer of events rather than relying just on the most recent one. These surface activations are then collected into histograms and classified using a Support Vector Machine (SVM). Although of a fixed length, the use of memory for computing representations closely relates HATS with the solution presented in this chapter. Crucially, the accumulation process

1 Code available at https://marcocannici.github.io/matrixlstm

used in HATS is hand-engineered, while we propose to learn one tailored for the task at hand through LSTM cells [20]. This design enables the memory to accommodate more events and, more critically, eliminates the need to tune the representation's hyperparameters manually.

Zhu et al. [54] propose the EV-FlowNet architecture for optical flow prediction, as well as a novel hand-crafted event-frame. A four-channel representation is built by storing, for each polarity, the number of events that occurred in each location besides temporal information. A similar aggregation strategy is also proposed by Ye et al. [56], while Zhu et al. [55] suggest discretizing time into consecutive bins to improve the temporal resolution of such representations. To that end, a voxel grid is constructed by allocating a channel for each bin and aggregating events using a linearly weighted accumulation similar to bilinear interpolation. A similar time discretization is also used in Events-to-Video [126], where the event representation is used within a recurrent-convolutional network to recover brightness from event sequences. Despite being more complex to compute than a simple event aggregation, the reconstructed frames closely resemble actual grayscale videos, allowing this approach to take full advantage of transferring features trained on natural images.

END-TO-END REPRESENTATIONS. Gehrig et al. [92] are the first to propose a method for learning a dense representation directly from raw events. The proposed EST mechanism is similar to a voxel grid, but the kernel employed to accumulate events is learned. In particular, they propose to implement a trilinear filter using a Multi Layer Perceptron (MLP) to learn extracting temporal information from each event independently. Features extracted from events occurring in the same pixel location are then added together to obtain the final pixel encoding. Among available event representations, EST is the one that most closely relates to the mechanism proposed in this chapter. However, it processes events independently and does not leverage the sequential nature of events during the encoding phase, preventing the network from adapting the aggregation strategy based on previous events. Instead, our method can integrate information conditioned on the current state and decide how relevant each event is to perform the task and how much information to retain from past events, thanks to the memory mechanism of LSTM cells.

Parallel to that of event representations, a recent trend in event-based processing is also investigating mechanisms that do not require explicit intermediate representations to perform a task [89, 102, 103]. Among these, Neil et al. [258] use a variant of the LSTM network, named PhasedLSTM, to learn sampling information at regular rates from asynchronous event streams. Despite collecting event data sequentially through LSTM cells as in our work, PhasedLSTM employs a single cell to encode the whole event stream and extracts a single encoding of the scene, disregarding spatial resolution. This approach typically fails to generalize to challenging tasks [269] and can not

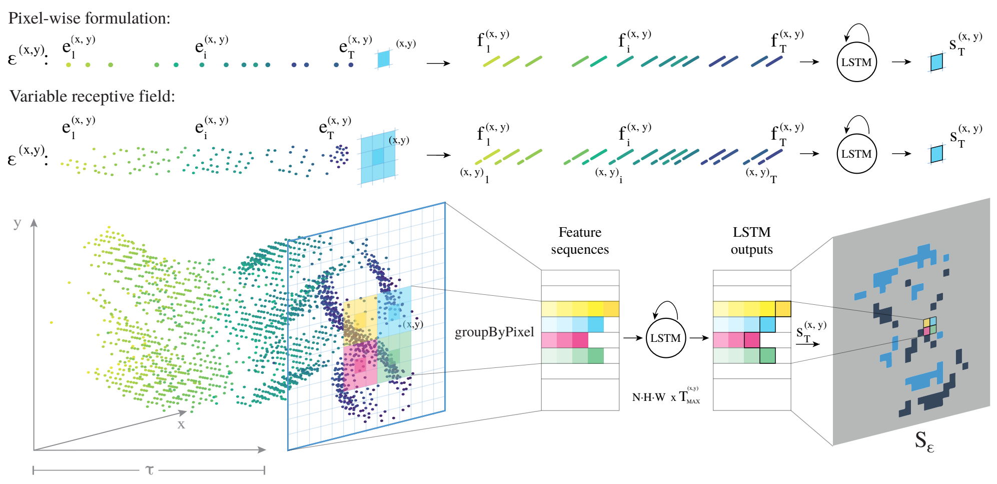


Figure 4.1: Overview of Matrix-LSTM (figure adapted from [258]). Events in each pixel are first associated with a set of features $f_i^{(x,y)}$, and then processed by the LSTM. The last output, $s_T^{(x,y)}$, is finally used to construct $\mathcal{R}_\mathcal{E}$. Larger receptive fields can optionally be used to extract spatio-temporal features in each pixel. In this case, event coordinates $(x, y)_i$, relative to the receptive field, are also added before processing. *GroupByPixel* is shown here on a single sample ($N = 1$) highlighting four non overlapping 3×3 kernels producing 2×2 features. Colors refer to pixel locations, while intensity indicates time. For clarity, the features' dimension is not shown in the figure.

be easily integrated with traditional network designs, such as standard CNNs. In contrast, this chapter proposes using an LSTM as a convolutional filter, obtaining a translation-invariant module that independently integrates local temporal features while still retaining spatial structures.

Finally, although it has never been adopted directly for aggregating raw events, we also mention the ConvLSTM [192] network, a convolutional variant of the LSTM that extends recurrent processing to grid-like structures. Despite its similarity with our method, since both extend the notion of convolution to LSTM cells, ConvLSTM is not straightforward to apply to sparse event-based streams and requires the input to be densified into frames before processing. This involves building either very sparse frames of simultaneous events, mostly filled with padding, or dense frames containing unsynchronized events. Our formulation, instead, preserves sparsity during computation and does not require events to be densified, even when large receptive fields are considered.

4.3 METHOD

Given a time interval τ (e.g., the length of an event recording), the set of events produced by an event-based camera can be defined as a sequence $\mathcal{E} =$

$\{(x_i, y_i, t_i, p_i) \mid t_i \in \tau\}$, ordered by the event timestamp t_i. The coordinates (x_i, y_i) indicate the location (within the $H \times W$ resolution) where events are generated and $p_i \in \{-1, 1\}$ defines their polarity. Since the event camera is composed of independent pixels producing events as soon as they detect a significant brightness change, in principle, multiple events can be generated at the same timestamp. However, the density of the events at a fixed timestamp t is likely to be very low. As a result, in order for traditional frame-based algorithms to interpret events, a mechanism capable of aggregating their information through time into an image-like representation $\mathcal{R}_\mathcal{E}$ is required.

4.3.1 Matrix-LSTM

Analogously to Gehrig et al. [92], our goal is to learn end-to-end a fully parametric mapping $\mathcal{M} : \mathcal{E} \rightarrow \mathcal{R}_\mathcal{E} \in \mathbb{R}^{H \times W \times C}$, between an event sequence and a dense event representation $\mathcal{R}_\mathcal{E}$ in such a way to provide the best features for a given task to be learned. In this chapter, we propose to implement $\mathcal{M}$ as an *$H \times W$ matrix of LSTM cells* [20]. We do not aim to recreate a frame that looks like a picture taken from the original scene, like a greyscale or an RGB image [126, 270], but rather to extract task-aware features that improve performance, regardless of their appearance. A graphical representation of the proposed solution is given in Figure 4.1 on the facing page and discussed in the following.

Let's define the ordered sequence of events $\mathcal{E}^{(x,y)}$ produced by a single pixel (x, y) during the interval τ as $\mathcal{E}^{(x,y)} = \{(x_i, y_i, t_i, p_i) \mid t_i \in \tau, x_i = x, y_i = y\} \subset \mathcal{E}$. We define its length as $T^{(x,y)} = |\mathcal{E}^{(x,y)}|$, which may potentially be different for each location (x, y). A set of features $f_i^{(x,y)} \in \mathbb{R}^F$ is first computed for each event occurring at location (x, y), typically the polarity and one or multiple temporal features (see Section 4.5 on page 82), resulting in a sequence $\mathcal{F}^{(x,y)} = \{f_i^{(x,y)} \mid i \in \mathcal{E}^{(x,y)}\}$. These are raw event features as we aim to learn extracting more complex ones by analysing spatio-temporal correlations between events.

To this purpose, we split the input frame into a grid and associate an LSTM cell $LSTM^{(x,y)}$ to each pixel location. Each of these recurrent units processes the sequence of features $\mathcal{F}^{(x,y)}$ asynchronously, keeping track of the current integration state and condensing all events into a single output vector $s^{(x,y)} \in \mathbb{R}^C$. In particular, at each time t_i, the $LSTM^{(x,y)}$ cell integrates the input feature f_i and produces as a result the updated pixel representation $s_t^{(x,y)}$. Once all the events are processed, the last output of the LSTM cell compresses the dynamics of the entire sequence $\mathcal{E}^{(x,y)}$ into a fixed-length vector $s_T^{(x,y)}$ that can thus be used as a pixel feature (here we dropped the superscript $^{(x,y)}$ from T for readability). We build the final representation $\mathcal{R}_\mathcal{E}$ by collecting all LSTMs' final outputs $s_T^{(x,y)}$ into a dense tensor of shape $H \times W \times C$. Since event-based camera's pixels produce an output only if a

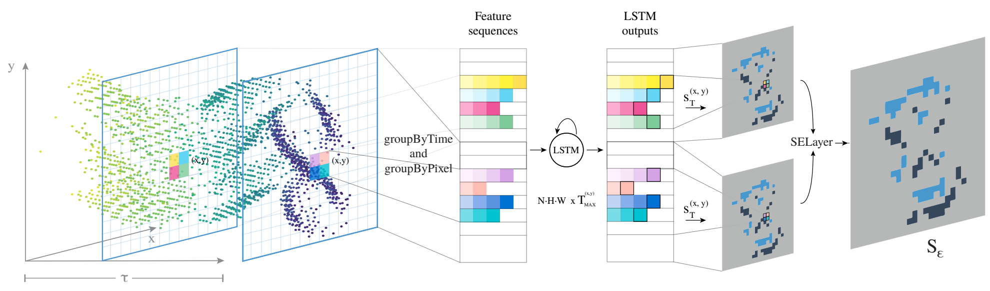


Figure 4.2: Matrix-LSTM with temporal bin splitting. The temporal domain is split in non-overlapping windows from which independent Matrix-LSTM representations are extracted. A Squeeze-and-Excitation [271] layer is then optionally applied to correlate features in time.

change is detected, a fixed all-zeros output is used whenever the set of events $\mathcal{E}^{(x,y)}$ is empty for a given (x, y) location.

RECEPTIVE FIELD SIZE. Matrix-LSTM, as for a kernel in a standard convolution, may be applied over the event stream using a variable stride and kernel dimension. In particular, given a receptive field of size $K_H \times K_W$, each LSTM cell can be extended to process the local neighborhood of asynchronous events around its pixel location $\mathcal{E}^{(x,y)} = \{(x_i, y_i, t_i, p_i) \mid t_i \in \tau, |x - x_i| \leq \lfloor K_W/2 \rfloor, |y - y_i| \leq \lfloor K_H/2 \rfloor\}$ instead of a single-pixel sequence. In this formulation, events' features are computed as in the original formulation. However, an additional coordinate feature (p_x, p_y) is also added, specifying the relative position of each event within the receptive field. Coordinate features are range-normalized so that an event occurring in the top-left pixel of the receptive field has feature $(0, 0)$, while one occurring in the bottom-right position has feature $(1, 1)$. Features are processed as in the previous formulation, with each LSTM cell consuming its event sequence to produce the overall pixel vector $s_T^{(x,y)}$. In this case, the resolution of the output representation $\mathcal{R}_\mathcal{E}$ may vary depending on the stride and kernel dimension, similarly to a convolutional layer. Events belonging to multiple LSTMs' receptive fields (e.g., when using a 1×1 stride and receptive field greater than 1×1) are processed multiple times by each of these LSTMs, independently, each time with a different coordinate feature. This general formulation is depicted in Figure 4.1 on page 76.

TEMPORAL BINS. Taking inspiration from previous methods [55, 92, 126] that discretize time into temporal bins, we further extend Matrix-LSTM to optionally operate on successive time windows. Given a fixed number of bins B, the original event sequence is split into B consecutive windows $\mathcal{E}_{\tau_1}, \mathcal{E}_{\tau_2}, ..., \mathcal{E}_{\tau_B}$. Each sequence is processed independently by taking the output of each LSTM at the end of each interval to build a representation $\mathcal{R}_{\mathcal{E}_b}$ and then re-initializing the LSTMs' states before the next sub-sequence starts. This gives rise of B different representations $\mathcal{R}_{\mathcal{E}_b}$ that are then concatenated to form the final representation $\mathcal{R}_{\mathcal{E}} \in \mathbb{R}^{H \times W \times B \cdot C}$. In this formulation, the LSTM's input features $f_i^{(x,y)}$ usually contain both global temporal features (i.e., referring to the original uncut sequence) and relative features (i.e., relative to the sub-sequence). Although LSTMs should be able to retain memory over very long periods, we found that discretizing time into intervals helps the Matrix-LSTM layer in maintaining event information, especially in tasks requiring precise time information such as optical flow estimation (see experimental results in Section 4.5.2 on page 88).

TEMPORAL CORRELATION. When computing MatrixLSTM over multiple bins, each bin's representation is extracted independently from the others. While this improves convergence, the LSTM temporal reasoning becomes limited to operate only within each temporal window. To overcome this issue, we propose to make use of the Squeeze-and-Excitation (SE) self-attention module proposed by Hu et al. [271]. SE is applied over $\mathcal{R}_{\mathcal{E}} \in \mathbb{R}^{H \times W \times B \cdot C}$ to re-modulate channel features by modeling correlations within and between different bins. Given $\mathcal{R}_{\mathcal{E}}$, the SE layer first *squeezes* the spacial information into a channels descriptor $\mathbf{z_{sq}} \in \mathbb{R}^{B \cdot C}$ through non-parametric average pooling. Then, an *excitation* operator is applied to correlate channel information. An activation vector $\mathbf{s} \in \mathbb{R}^{B \cdot C}$ is computed from $\mathbf{z_{sq}}$ through a learnable transformation implemented as a two-layer MLP with a bottleneck that downsizes $B \cdot C$ to $B \cdot C/r$. The final representation is computed as $\hat{\mathcal{R}}_{\mathcal{E}} = \mathcal{R}_{\mathcal{E}} \cdot \mathbf{s}$, where the product $\cdot$ implements the product between the scalar s_i and the i-th channel of $\mathcal{R}_{\mathcal{E}}$. As a result, $\hat{\mathcal{R}}_{\mathcal{E}}$ has the same dimensions of the original representation, but it provides channel information that is not confined to a single bin but that expands instead across all the event stream. This variant of Matrix-LSTM is depicted in Figure 4.2 on the facing page.

PARAMETERS SHARING. Inspired by the convolution operation defined on images, we designed Matrix-LSTM to enjoy translation invariance. This property is achieved by sharing the parameters across all the LSTM cells, as in a convolutional kernel. Events in each location are processed sequentially and independently using the LSTM memory to accumulate values and perform conditioned integration. Similar to a convolution operation, we force the learned transformation to be invariant on the position from which features are extracted, learning a general integration procedure as a result. Taking advantage of the LSTM gating mechanism, Matrix-LSTM learns an optimal integration strategy given the current state by deciding each time how to encode the current event based on the previous events' history (e.g., using the timing information to dynamically adapt to different event rates). Sharing parameters not only drastically reduces the number of parameters in the network, but it also enables transferring a learned transformation to higher or lower resolutions as in fully-convolutional networks [219].

We highlight that such an interpretation of the Matrix-LSTM functioning also fits the framework proposed by Gehrig et al. [92]. In their formulation, popular event densification mechanisms are rephrased as kernel convolutions on the *event field*, i.e., a discretized four-dimensional manifold encompassing the spatial, temporal, and polarity dimensions. We finally report that the simplest formulation given at the beginning of this section is equivalent to a 1×1 ConvLSTM [192] applied on a dense tensor obtained by stacking events in pixel locations by arrival order. Despite this equivalence, our formulation has better space and time performance on sparse event sequences as reported in Section 4.5.4 on page 97. Furthermore, the extension to larger receptive fields provided in Figure 4.1 on page 76 has superior accuracy performance on asynchronous event data compared to ConvLSTM.

4.4 IMPLEMENTATION DETAILS

The convolution-like operation described in the previous section can be implemented efficiently by means of two carefully designed event grouping operations. Rather than replicating the LSTM unit multiple times on each spatial location, a single recurrent unit is applied over different $\mathcal{E}^{(x,y)}$ sequences in parallel. These two operations, namely *groupByPixel* and *groupByTime*, allow event streams to be split based on the pixel location and temporal bin. After being reshaped, the input is ready to be processed by a single LSTM network, implementing parameter sharing across all pixel locations and temporal windows. We implement these operations as custom CUDA kernels to allow for fast event processing. These operations are not specific to Matrix-LSTM, since grouping events by pixel index is a common operation in event-based processing and could thus benefit other implementations making use of GPUs. We give a detailed overview of the two reshape operators in the following.

4.4.1 GroupByPixel

This operation translates from event-sequences to pixel-sequences. Let X be a tensor of shape $N \times T_{max} \times F$, representing the features $f_{n,i}^{(x,y)}$ of a batch of N samples, where T_{max} is the length of the longest sequence in the batch. We define the *groupByPixel* mapping on X as an order-aware reshape operation that rearranges the events into a tensor of pixel-sequences of shape $P \times T_{max}^{(x,y)} \times F$. Here $T_{max}^{(x,y)}$ is the length of the longest pixel sequence $\mathcal{E}_n^{(x,y)}$, and P is the number of active pixels (i.e., having at least one event) in the batch, which equals $N \cdot H \cdot W$ only in the worst case. Pixel-sequences shorter than $T_{max}^{(x,y)}$ are padded with all-zeros events to enable parallel processing

The tensor thus obtained is then processed by the LSTM cell, which treats samples in the first dimension independently, effectively implementing parameter sharing and applying the transformation in parallel over all the pixels. The LSTM output tensor, which has the same shape as the input one,

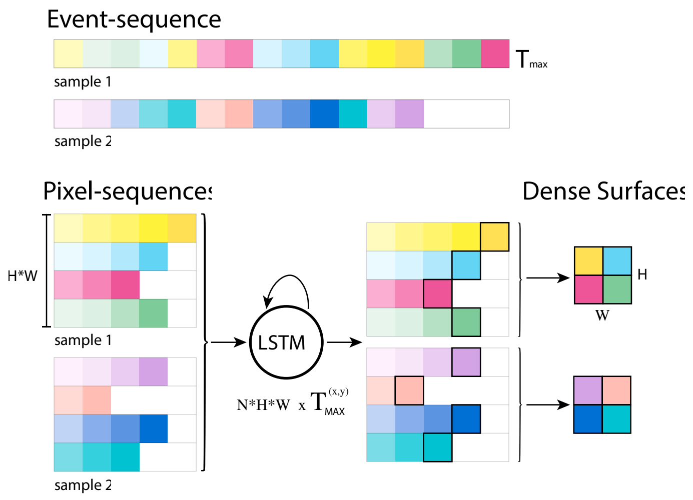


Figure 4.3: An example of the *groupByPixel* operation on a batch of $N = 2$ samples and a 2×2 pixel resolution. Different colors refer to different pixel locations while intensity indicates time. For clarity, the features dimension is not shown in the figure.

is then sampled by taking the output corresponding to the last event in each pixel-sequence $\mathcal{E}_n^{(x,y)}$, ignoring values computed on padded locations. Values thus obtained are then used to populate Matrix-LSTM representation. To improve efficiency, for each pixel-sequence $\mathcal{E}_n^{(x,y)}$, *groupByPixel* also keeps track of the original spatial position (x, y), the index of the sample inside the batch, and the length of the pixel-sequence $T_n^{(x,y)}$, namely the index of the last event before padding. Given this set of indexes, the densification step can be performed as a simple slicing operation. Refer to Figure 4.3 for visual representation of this process. *groupByPixel* is implemented as a custom CUDA kernel that processes each sample in parallel and places each event feature in the output tensor maintaining the original temporal order.

4.4.2 GroupByTime

The Matrix-LSTM variant that operates on temporal bins performs a similar pre-processing step. Each sample in the batch is divided into a fixed set of intervals. The *groupByTime* CUDA kernel pre-processes the input events generating an $N * B \times T_{max}^b \times F$ tensor where the B bins are grouped in the first dimension, taking care of properly padding intervals (T_{max}^b is the length of the longest bin in the batch). The Matrix-LSTM mechanism is then applied as usual, and the resulting $N * B \times H \times W \times C$ tensor is finally transposed and reshaped into an $N \times H \times W \times B * C$ event representation.

4.5 EVALUATION

Matrix-LSTM is a layer capable of learning an effective event aggregation process regardless of the task or the objective function to optimize. We test the proposed mechanism on three different tasks: object classification (Section 4.5.1), optical flow estimation (Section 4.5.2), and object detection (Section 4.5.3). With these three tasks, we test Matrix-LSTM's ability to generalize both to high-level and low-level tasks. We evaluate the goodness of Matrix-LSTM features indirectly: we take a state-of-the-art architecture as a reference and evaluate the proposed learnable layer in terms of the gain in performance obtained by replacing the network's representation with a Matrix-LSTM.

4.5.1 Object classification

We evaluate the model on the classification task using four publicly available event-based collections, namely N-Cars [85], N-Caltech101 [116], N-MNIST [116], and ASL-DVS [89]. The N-Caltech101 and N-MNIST collections are event-based conversions of the popular Caltech-101 [142] and MNIST [272] datasets, respectively, whilst N-Cars and ASL-DVS are recordings of real-world scenes. A detailed description and comparison of these datasets, along with visual representations of some of their samples, is provided in Section 2.2.3 on page 26.

Network Architectures

We use two network configurations to test Matrix-LSTM, namely the classifier used in Events-to-Video [126], and the one used to evaluate the EST [92] representation. Both are based on ResNet [220] backbones and pre-trained on images (i.e., ImageNet [146] or grayscales versions of the original images). Events-to-Video [126] uses a ResNet18 configuration, it leaves the first convolutional layer as in the pretraining and adds an extra fully-connected layer at the end to account for the different number of classes in both N-Calthech101 and N-Cars (we refer to this configuration as *ResNet–E2Vid*). EST [92] instead uses a ResNet34 backbone and replaces both the first and last layers, respectively, with a convolution matching the number of input features and a fully-connected layer with the correct number of output classes (we refer to this configuration as *ResNet–EST*). When testing on ASL-DVS [89] we follow a similar procedure but use ResNet50 [220] as the backbone, following the evaluation protocol of Bi et al. [89].

To perform a fair comparison, we replicated the two settings using the same number of channels in the event representation (although we also tried different channel values) and data augmentation procedures (random horizontal flips and crops of 224×224 pixels). We perform early stopping on a validation set in all experiments, using 20% of the training on N-Cars

Table 4.1: Accuracy (%) on N-Cars using the ResNet18–E2Vid network as backbone with variable time encoding and normalization.

ResNet Norm	ts absolute	ts relative	delay relative
✓	$95.22 \pm 0.41\%$	$94.77 \pm 1.01\%$	$95.40 \pm 0.59\%$
	$\mathbf{95.75 \pm 0.27}\%$	$95.32 \pm 0.85\%$	$\mathbf{95.80 \pm 0.53}\%$

Table 4.2: Accuracy (%) on N-Cars using the ResNet18–EST network as backbone with variable time encoding and number of bins.

		1 bin	2 bins	9 bins
delay	glob+loc	-	$92.68 \pm 1.23\%$	$92.32 \pm 1.02\%$
	local	$92.64 \pm 1.21\%$	$92.35 \pm 0.83\%$	$92.67 \pm 0.90\%$
ts	ts glob+loc	-	$\mathbf{93.46 \pm 0.84}\%$	$\mathbf{93.21 \pm 0.49}\%$
	local	$92.65 \pm 0.78\%$	$92.75 \pm 1.38\%$	$93.12 \pm 0.68\%$

and using the splits provided by the EST official code repository [92] for N-Caltech101. ADAM [257] is used as the optimizer for all experiments with a learning rate of 10^{-4}, except for ASL-DVS, where we used 10^{-3}. We use a batch size of 64 and a constant learning rate on N-Cars, ASL-DVS, and N-MNIST. On N-Caltech101, instead, we use a batch size of 16 while decaying the learning rate by a factor of 0.8 after each epoch when testing on *ResNet–E2Vid*, and a batch size of 100 with no decay with the *ResNet–EST* setup. Finally, to provide a robust evaluation, we compute the mean and standard deviation values by running evaluations with five different seeds in all the experiments.

Results

The empirical evaluation is organized as follows. We search for hyper-parameters using ResNet18 on N-Cars, since this configuration provides an optimal tradeoff between complexity and training time, thus allowing us to explore a larger number of parameters. We then select the best configuration resulting from this analysis and train the remaining architectures on the N-Caltech101, N-MNIST, and ASL-DVS datasets. We focus the analysis primarily on N-Caltech101 and N-Cars since they provide challenging scenarios, but we also evaluate Matrix-LSTM's performance on N-MNIST and ASL-DVS to analyze how the layer behaves when learning in more accessible settings.

MATRIX-LSTM + RESNET-E2VID. We start by identifying the optimal time feature to provide as input to Matrix-LSTM using the *ResNet–E2Vid* baseline, as reported in Table 4.1. Here we focus only on simple temporal encodings, as we want the layer to automatically learn to encode events from raw features, alleviating the need to hand-engineer ad-hoc feature extractors. We distin-

Table 4.3: Accuracy (%) on N-Cars with ResNet18–EST as a backbone, using *polarity + global ts + local ts* as features, optional SELayer and variable number of bins.

SE	2 bins	4 bins	9 bins	16 bins
	$93.46 \pm 0.84\%$	$92.68 \pm 0.62\%$	$93.21 \pm 0,49\%$	$92.01 \pm 0.45\%$
✓	$\mathbf{93.71} \pm \mathbf{0.93}\%$	$\mathbf{92.90} \pm \mathbf{0.62}\%$	$\mathbf{93.30} \pm \mathbf{0,47}\%$	$\mathbf{92.44} \pm \mathbf{0.43}\%$

Table 4.4: Accuracy (%) on N-Cars with ResNet18–EST as a backbone, using *polarity + global ts + local ts* as features, SELayer and variable number of channels.

	Channels		
bins	4	8	16
1	$93.88 \pm 0.87\%$	$93.60 \pm 0.30\%$	$\mathbf{94.37} \pm \mathbf{0.40}\%$
2	$93.05 \pm 0.92\%$	$93.97 \pm 0.52\%$	$\mathbf{94.09} \pm \mathbf{0.29}\%$
bins	4	7	8
9	$92.42 \pm 0.65\%$	$93.56 \pm 0.46\%$	$93.49 \pm 0.84\%$

guish between *ts* and *delay* features and between *absolute* and *relative* scopes. The first distinction refers to the type of time encoding, i.e., the timestamp of each event in the case of *ts* feature, or the delay between an event and the previous one in case of *delay*. Time features are always range-normalized between 0 and 1, with the scope determining whether the normalization occurs before (*absolute* feature) or after (*relative* feature) breaking events into pixel-sequences. In the case of *ts*, *absolute* means that the first and last events in the overall sequence $\mathcal{E}$ have time feature 0 and 1, respectively, regardless of their position, whereas *relative* means that the previous condition holds for every pixel-sequence $\mathcal{E}^{(x,y)}$. When utilizing delays, we only consider a relative scope because an absolute one would result in uncorrelated features after grouping them into pixel sequences, as the delay may refer to events in two separate positions when computed with a global scope. Finally, we always add the polarity, obtaining a 2-value feature $f_i^{(x,y)}$ as input to Matrix-LSTM. *Delay relative* and *ts absolute* are those providing the best results, with *ts relative* having higher variance. We select *delay relative* as the best configuration. Table 4.1 on the preceding page shows the effect of applying to Matrix-LSTM's output representations the same frame normalization used to pre-train the ResNet backbone on ImageNet. While applying normalization makes sense when the representations are similar to those used in pretraining, as in Events-to-Video [126], we discovered that in our case, where no constraint is imposed on representations' appearance, this does not improve the performance.

MATRIX-LSTM + RESNET-EST. We continue the experiments on N-Cars by exploring the effect of using bins on the quality of Matrix-LSTM features. We use the *ResNet–EST* baseline and start by identifying the best time encoding for this configuration. Since multiple intervals are involved, we distinguish between *global* and *local* temporal features. The first type is computed on the original sequence $\mathcal{E}$, before splitting events into intervals, whereas the latter locally, within the interval scope $\mathcal{E}_{\tau_b}$. For local features we consider the best options we identified on ResNet-E2Vid, namely *delay relative* and *ts absolute*. We only consider *ts* as a global feature since a global delay loses meaning after interval splitting, similarly to the scope. Results are reported in Table 4.2 on page 83 where values for single bin are missing since in that case there is no distinction between *global* and *local* features. Adding a global feature consistently improves performance. This additional feature can indeed help the LSTM network perform integration conditioned on a global timescale, thus enabling the extraction of temporal-consistent features. We use *global ts + local ts* features in the next experiments since this provides better performance and reduced variance, and we always add the polarity feature.

Following the analysis on the EST representation [92], we then search for the best number of bins, namely $B \in \{2, 4, 9, 16\}$, while using a fixed *polarity + global ts + local ts* configuration. In these experiments, we also make use of the Squeeze-and-Excitation (SE) [271] extension of Matrix-LSTM to correlate representations extracted from consecutive bins. Being the number of channels limited, we always use a reduction factor $r = 1$. As reported in Table 4.3 on the facing page, adding the SE layer consistently improves performance. We explain this by noticing that event-frames computed on successive intervals are naturally correlated and, thus, explicitly modeling their interdependency helps in extracting richer features.

Finally, we perform the last set of experiments to select the Matrix-LSTM hidden size, which also controls the number of output channels. Results are reported in Table 4.4 on the preceding page. Note that we only consider $4, 7, 8$ channels with 9 bins to limit the total number of channels after concatenation. We found that when the number of features is high, using fewer bins provides better performance, despite a higher complexity in terms of the number of parameters.

DISCUSSION. Classification results on N-Cars and N-Caltech101 datasets for the top performing configurations of both *ResNet-E2Vid* and *ResNet-EST* variants are reported in Table 4.5 on the following page. We use *relative delay* with *ResNet-E2Vid* and *global ts + local ts* with *ResNet-EST*. Through an extensive evaluation, we show that using the Matrix-LSTM representation as input to the baseline networks and training them jointly improves performance by a good margin. Indeed, under the ResNet34-E2Vid setup, our solution sets a new state-of-the-art on N-Cars, surpassing the Events-to-Video model. This is a remarkable result, as Events-to-Video was trained to extract realistic reconstructions and could, therefore, take full advantage of the ResNet pretraining, contrary to our model. The same does not happen on N-Caltech101, whose performance typically greatly depends on pretraining, even on the original image-based version, and where Events-to-Video has, therefore, an edge. Despite this, our model only performs 0.9% worse than the baseline. Under the

Table 4.5: Classification accuracy (%) on N-Cars [85] and N-Caltech101 [116].

Method	Classifier	Channels (bins)	N-Cars	N-Caltech101
H-First [140]	spike-based	-	56.1	0.54
HOTS [84]	histogram similarity	-	62.4	21.0
Gabor-SNN [85]	SVM	-	78.9	19.6
HATS [85]	SVM	-	90.2	64.2
	ResNet34–EST [92]	-	90.9	69.1
	ResNet18–E2Vid [126]	-	90.4	70.0
E2Vid [126]	ResNet18–E2Vid	1	91.0	**86.6**
Matrix-LSTM	ResNet18–E2Vid	3 (1)	**95.80 ± 0.53**	84.12 ± 0.84
(Ours)	ResNet34–E2Vid	3 (1)	**95.65 ± 0.46**	85.72 ± 0.37
EST [92]	ResNet34–EST	2 (9)	92.5	81.7
	ResNet34–EST	2 (16)	92.3	83.7
Matrix-LSTM (Ours)	ResNet18–EST	16 (1)	**94.37 ± 0.40**	81.24 ± 1.31
	ResNet34–EST	16 (1)	**94.31 ± 0.43**	78.98 ± 0.54
	ResNet18–EST	16 (2)	**94.09 ± 0.29**	83.42 ± 0.80
	ResNet34–EST	16 (2)	**94.31 ± 0.44**	80.45 ± 0.55
	ResNet18–EST	2 (16)	**92.58 ± 0.68**	**84.31 ± 0.59**
	ResNet34–EST	2 (16)	92.15 ± 0.73	83.50 ± 1.24

Table 4.6: Classification accuracy (%) on N-MNIST [116] and ASL-DVS [89].

Method	Classifier	Channels (bins)	N-MNIST	ASL-DVS
H-First [140]	spike-based	-	71.2	-
HOTS [84]	histogram similarity	-	80.8	-
HATS [85]	SVM	-	**99.1**	-
G-CNN [89]	Graph CNN	-	98.5	87.5
RG-CNN [89]	Graph CNN	-	99.0	90.1
Events Count [89]	ResNet50-EST	2 (1)	98.4	88.6
E2Vid [126]	ResNet18-E2Vid	1 (1)	98.3	-
EST [92]	ResNet50-EST	2 (1)	-	99.57
Matrix-LSTM	ResNet18-E2Vid	1 (1)	**98.9 ± 0.21**	
(Ours)	ResNet50-EST	2 (1)		**99.73 ± 0.04**

ResNet-EST setup, the model performs consistently better on N-Cars, while slightly worse on N-Caltech101. It should be noted, however, that the search for the best configuration was carried out using N-Cars as a dataset. As a result, we expect the performance to improve if a full hyper-parameter search is conducted directly on N-Caltech101. Finally, performance on N-MNIST and ALS-DVS is reported on Table 4.6 on the preceding page. In both cases, Matrix-LSTM performs better than other event-frame mechanisms. In particular, it outperforms alternative event-driven and graph-based classification architectures, showing its ability to generalize even in simple scenarios where overfitting on low-level features may be an issue.

4.5.2 Optical flow prediction

To assess Matrix-LSTM's ability to extract motion-rich features for low-level tasks, we use the optical flow extension [54] of the MVSEC [136] benchmark. The dataset features great variability, as it was collected both indoor and outdoor on a variety of different vehicles and lighting conditions. A detailed description of the dataset, along with visualizations of few selected samples are provided in Section 2.2.3 on page 26.

NETWORK ARCHITECTURE. We use the EV-FlowNet [54] architecture as a reference. To perform a fair comparison between Matrix-LSTM and the original hand-crafted features, we build our model on top of the publicly available codebase [54]. The code contains a few minor upgrades over the paper version, which we made sure to undo in our experiments. These consist of removing the batch normalization layers, setting to 2 the output channels of the layer preceding the optical flow prediction layer, and disabling random rotations during training. For completeness, we report the results we obtained by training the baseline from scratch with these modifications, along with the original paper's results in the following evaluation.

The original network uses a 4-channel event-frame, collecting in each pixel and for each polarity, the number of events received and the timestamp of the most recent one. We replace this representation with a Matrix-LSTM layer producing a 4-channel representation. Following Zhu et al. [54], we train the model on the *outdoor_day1* and *outdoor_day2* sequences for $300,000$ iterations. We use Adam [257] as the optimizer with batch size 8 and an initial learning rate of 10^{-5}, exponentially decayed every 4 epochs by a factor of 0.8. We noticed that EV-FlowNet might be unstable at higher learning rates, especially at the beginning, while Matrix-LSTM could benefit from larger rates. To make training more effective, we use a learning rate of 10^{-4} for Matrix-LSTM's parameters by multiplying their gradients by a factor of 10 during training.

Test is performed on a separate set of recordings, namely *indoor_flying1*, *indoor_flying2* and *indoor_flying3*, which are visually different from training data. The network performance is measured in terms of average endpoint error (AEE), computed as the distance between the endpoints of the

Table 4.7: Optical flow estimation on MVSEC when input frames and corresponding events are one frame apart ($dt = 1$).

Method		*indoor_flying1*		*indoor_flying2*		*indoor_flying3*	
		AEE	%Outlier	AEE	%Outlier	AEE	%Outlier
Two-Channel [92, 109]	events count	1.21	4.49	2.03	22.8	1.84	17.7
Voxel Grid [55, 92]	weighted polarity	0.96	1.47	1.65	14.6	1.45	11.4
Ev-FlowNet [54]	counts & time	1.03	2.2	2.12	15.1	1.53	11.9
Ev-FlowNet (ours)	counts & time	1.015	2.736	1.606	12.089	1.548	11.937
EST [92]	exp. kernel	0.96	1.27	1.58	10.5	1.40	9.44
	learnt kernel	0.97	0.91	1.38	8.20	1.43	6.47
Matrix-LSTM (Ours)	1 bin	1.017	2.071	1.642	13.89	1.432	10.44
	2 bins	0.829	**0.471**	1.194	**5.341**	1.083	**4.390**
	2 bins + SELayer	**0.821**	0.534	**1.191**	5.590	**1.077**	4.805
	4 bins	0.969	1.781	1.505	11.63	1.507	12.97
	4 bins + SELayer	**0.844**	**0.634**	**1.213**	**6.057**	**1.070**	**4.625**
	8 bins	**0.881**	**0.672**	1.292	**6.594**	1.181	5.389
	8 bins + SELayer	0.905	0.885	**1.286**	6.761	**1.177**	**5.318**

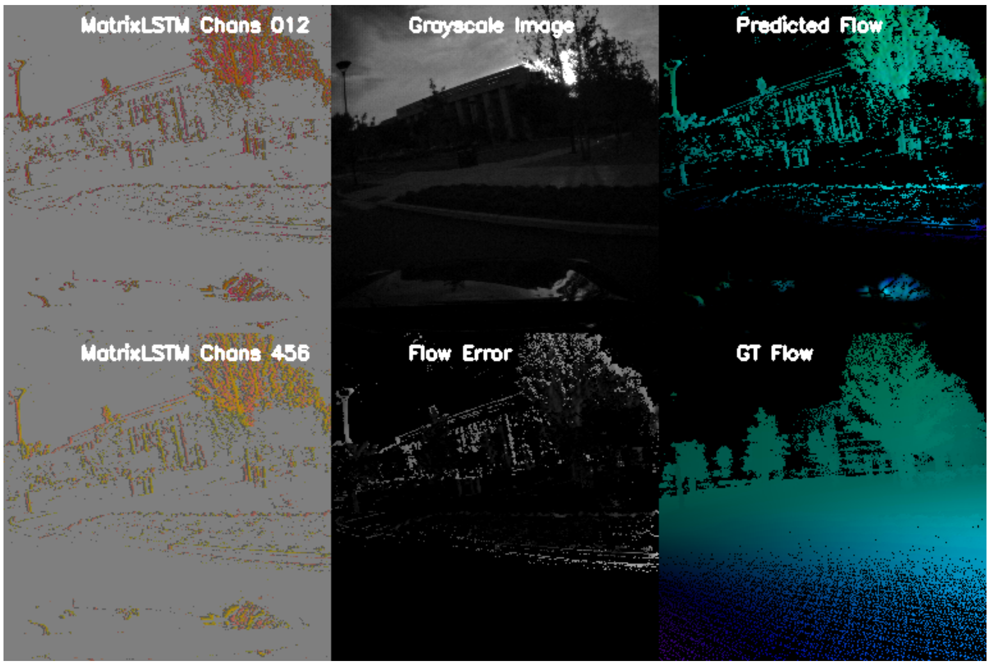
Table 4.8: Optical flow estimation on MVSEC when input frames and corresponding events are four frames apart ($dt = 4$).

Method		*indoor_flying1*		*indoor_flying2*		*indoor_flying3*	
		AEE	%Outlier	AEE	%Outlier	AEE	%Outlier
Ev-FlowNet [54]	counts & time	2.25	24.7	4.05	45.3	3.45	39.7
Ev-FlowNet (ours)	counts & time	3.432	48.685	5.957	63.226	5.247	57.662
Matrix-LSTM (Ours)	1 bin	3.366	42.022	5.870	65.379	5.015	57.094
	2 bins	**2.269**	**23.558**	**3.946**	**42.450**	**3.172**	**31.975**
	2 bins + SELayer	2.378	25.995	4.333	45.396	3.549	36.822
	4 bins	3.023	36.085	4.870	49.077	4.652	43.267
	4 bins + SELayer	**2.330**	**24.777**	**4.322**	**44.769**	**3.588**	**36.442**
	8 bins	**2.290**	**24.203**	**3.978**	**42.230**	**3.346**	**33.951**
	8 bins + SELayer	2.308	24.597	4.046	44.366	3.391	35.452

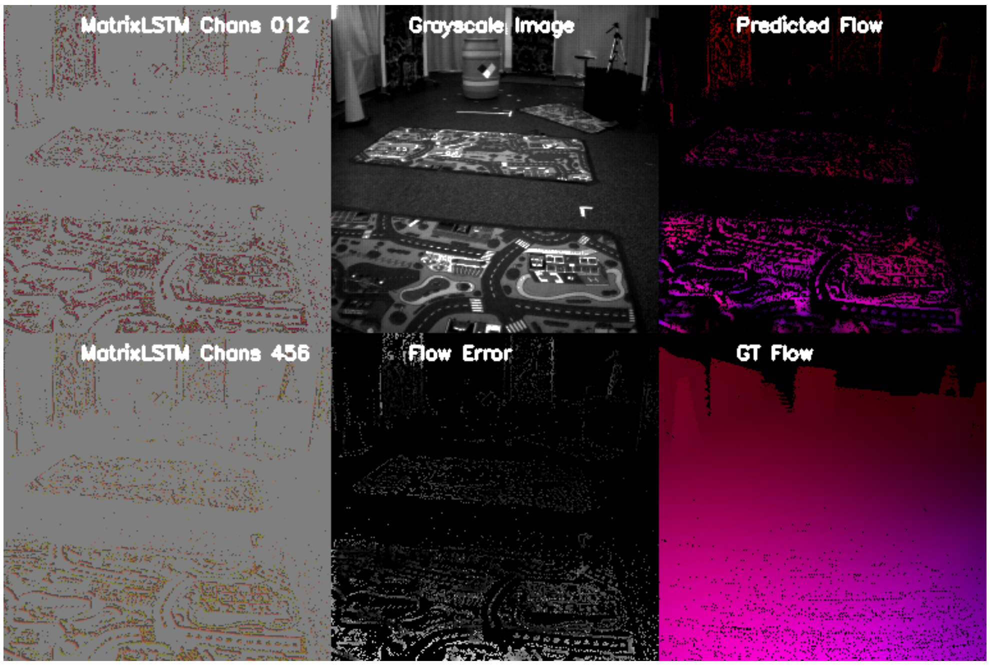
predicted and ground truth flow vectors. In addition, as proposed in the KITTI benchmark [273] and done by Zhu et al. [54], we report the percentage of outliers, defined as the points with endpoint error greater than 3 pixels and 5% of the magnitude ground truth vector. Finally, in accordance with Zhu et al. [54] evaluation protocol, we only compute the error in spatial locations where at least one event is received. Qualitative results showing the network's predictions on test sequences are accessible at `https://marcocannici.github.io/matrixlstm`. Few frame predictions are also provided in Figure 4.4 on the next page.

Results

In the previous classification experiments, we observed that the type of temporal features and the number of bins play an important role in extracting



(a) Sample prediciton on the outdoor_day1 sequence.



(b) Sample prediciton on the indoor_flying1 sequence.

Figure 4.4: Qualitative results on MVSEC's outdoor_day1 and indoor_flying1 sequences [136] using the best performing Matrix-LSTM configuration (2 bins and 4 channels). Some channels produced by Matrix-LSTM are visualized here as RGB images, along with the network's prediction. Notice that no visual constraints are enforced on Matrix-LSTM's representation, leaving the network the freedom to encode features (or colors in this case) that best fit the subsequent CNN.

effective representations. We expect time resolution to be a key factor of performance in optical flow. Hence, we focus here on measuring how different interval choices impact the flow prediction. Rather than exploring different types of time features, as done in classification experiments, we decide to always use the *polarity + global ts + local ts* configuration, which worked well on N-Cars while considering different bin setups.

STATE-OF-THE-ART COMPARISON. Zhu et al. [54] test Ev-FlowNet on two evaluation settings for each test sequence to benchmark how the network performs on different flow magnitudes: with input frames and events spanning two consecutive frames (denoted as *dt=1*), and with frames and events spanning five consecutive frames (denoted as *dt=4*). We report results on both settings in Tables 4.7 and 4.8 on page 88, respectively. While we were able to closely replicate the results of the first configuration (*dt=1*), with a minor improvement in the *indoor_flying2* sequence, the performance we obtain on the *dt=4* setup is instead worse on all sequences, as reported on the first two rows of Table 4.8.

Despite this discrepancy, which prevents the Matrix-LSTM performance on *dt=4* settings from being directly compared with the results reported on the Ev-FlowNet paper, we can still evaluate the benefits of our representation on larger flow magnitudes. Indeed, in this chapter, we evaluate the Matrix-LSTM layer based on the relative performance improvement obtained by substituting the original features with our layer. Using our Ev-FlowNet results as a baseline, we show that Matrix-LSTM is able to improve the optical flow quality on both settings, highlighting the capability of the layer to adapt to different sequence lengths and movement conditions. We report a relative improvement of up to 30.43% on *dt=1* and up to 39.55% on *dt=4* settings using our results as baseline. As expected, varying the number of bins greatly impacts performance. Indeed, when using two bins instead of just one, the AEE error decreases significantly. Interestingly, we achieve the best performance by considering only 2 intervals, as adding more bins hurts performance. We believe this behavior resides in the nature of optical flow prediction, where the network is implicitly asked to compare two distinct temporal instants. This configuration consistently improves the baseline by up to 30.76% on *indoor_flying2*, demonstrating the ability of Matrix-LSTM to adapt to low-level tasks where both spatial and temporal resolutions are critical for performance.

EFFECT OF ADDING THE SE LAYER. Optical flow prediction is a challenging task that requires deep neural networks to extract accurate features that precisely describe the scene's dynamics. Therefore, an event aggregation mechanism encoding rich temporal features is required to enable accurate predictions. In the previous experiments, we registered that time resolution is key for extracting effective features with Matrix-LSTM. In particular, increasing the number of bins significantly impacts the predicted flow and allows the network to retain temporal information over long sequences. Here we focus, instead, on correlating temporal features by adding an SELayer to the Matrix-LSTM output. Tables 4.7 and 4.8 on page 88 report the performance obtained using this additional layer on the MVSEC [136] task. The results we obtain show that adding the SELayer only improves performance on the 4 bins configuration for the *dt=4* benchmark, while the SELayer consistently helps in reducing the *AEE* metric on the *dt=1* setting. By comparing features obtained from subsequent intervals, the SELayer adaptively recalibrates features and helps in modeling interdependencies between time instants, which is crucial for predicting optical flow. We believe that a similar approach may be used to improve the performance of other event aggregation mechanisms based on voxel-grids of temporal bins, especially those employing data-driven optimization mechanisms [92].

4.5.3 Object Detection

As an additional task, we also test the performance of Matrix-LSTM on object detection using the Gen1 Automotive Detection Dataset [135]. The dataset features real-world recordings captured in urban environments using a QVGA 304×240 event camera [47], the same of N-Cars [85], and it constitutes, at the time of this research, the largest detection dataset for event cameras. Details are provided in Section 2.2.3 on page 26.

NETWORK ARCHITECTURE. We start by using the YOLOv3 [274] architecture, an improved version of the YOLO [19] network used in the previous chapter. YOLOv3 is an incremental improvement of its fully convolutional predecessor [247], introducing a refined prediction scheme featuring anchor boxes and multi-scale predictions. We extend a custom implementation [275] of YOLOv3, written in PyTorch [267], by adding a Matrix-LSTM layer before the network. Matrix-LSTM is configured to produce 416×416 representations for compatibility with YOLOv3's input resolution. Bounding boxes are first predicted on this input and then cropped to the 304×240 event camera resolution. The network is initialized with weights pre-trained on COCO [259], and then trained using SGD as optimizer with a learning rate of 0.001, a momentum of 0.9, weight decay with 0.0005 strength, and batch size 16. Similarly to semantic segmentation experiments, we increase gradients of Matrix-LSTM parameters by a factor of 2 during training. Since the dataset is unbalanced, with few *pedestrian* boxes compared to *cars*, we resample training samples after each epoch to favor those containing more boxes of the class misclassified the most at the previous validation step.

Following a subsequent work [104] to Matrix-LSTM, we also extend the detection analysis to the RetinaNet [276] architecture to compare Matrix-LSTM's performance with other baselines. We use a ResNet50 [220] as backbone with a pyramid feature extraction structure [277]. We integrate Matrix-LSTM within the detectron2 [278] framework for object detection and use it as the

Table 4.9: Detection results (mAP) on the Gen1 Automotive Detection dataset [135].

Representation	Backbone	Params (M)	mAP
Histograms [109]	VGG13-SparseConv [87]	-	0.15
Voxel Grids [55]	RetinaNet [276] + FPN [277]	32.8	0.20
E2Vid [126]	RetinaNet [276] + FPN [277]	43.5	0.30
MatrixLSTM (Ours)	YOLOv3 [274]	61.5	0.31
MatrixLSTM (Ours)	RetinaNet [276] + FPN [277]	37.9	**0.31**
Voxel Grids [55]	RED [104]	24.1	**0.40**
ATIS Grayscales	RetinaNet [276] + FPN [277]	32.8	**0.44**

input of the RetinaNet backbone, as for YOLOv3. We start from a RetinaNet pre-trained on COCO [259] with a 320×256 input resolution. We train it alongside Matrix-LSTM for $180,000$ iterations using SDG and the same hyperparameters as before, except for the batch size, which is set to 4, and the resampling strategy, which is not used in this case.

Results

The Gen1 Automotive dataset [135] exhibits similar environments to that of the N-Cars [85] dataset we used for classification experiments. They have been recorded with the same camera and under similar driving conditions. For this reason, we opt to set up Matrix-LSTM with the hyperparameters that delivered the best performance on N-Cars. In particular, we do not split the sequence into bins and set up Matrix-LSTM to produce 3 output channels for easier integration with the backbone network, using *delays* as temporal features.

STATE-OF-THE-ART COMPARISON. We compare Matrix-LSTM performance against other baselines in Table 4.9. We report the upper bound performance computed by Perot et al. [104] by training a RetinaNet directly on ATIS's grayscale images, as well as the performance obtained with voxel grids and E2Vid's reconstructed grayscales. We train these two baselines with the same hyperparameters and training process we used with Matrix-LSTM. Matrix-LSTM obtains the same performance regardless of the backbone used. It outperforms voxel grids by a good margin and performs on par with E2Vid grayscales [127] when compared under the same backbone architecture. The second of these results demonstrates Matrix-LSTM's ability to provide high-quality event representations and optimally interface with the processing backbone. Indeed, E2Vid's grayscales should be favored in this setting since they are significantly more similar in appearance to pictures used while pre-training the RetinaNet, and thus, they should be able to fully exploit pre-trained features extractors. Nevertheless, Matrix-LSTM performs slightly better than E2Vid, yet using significantly fewer parameters.

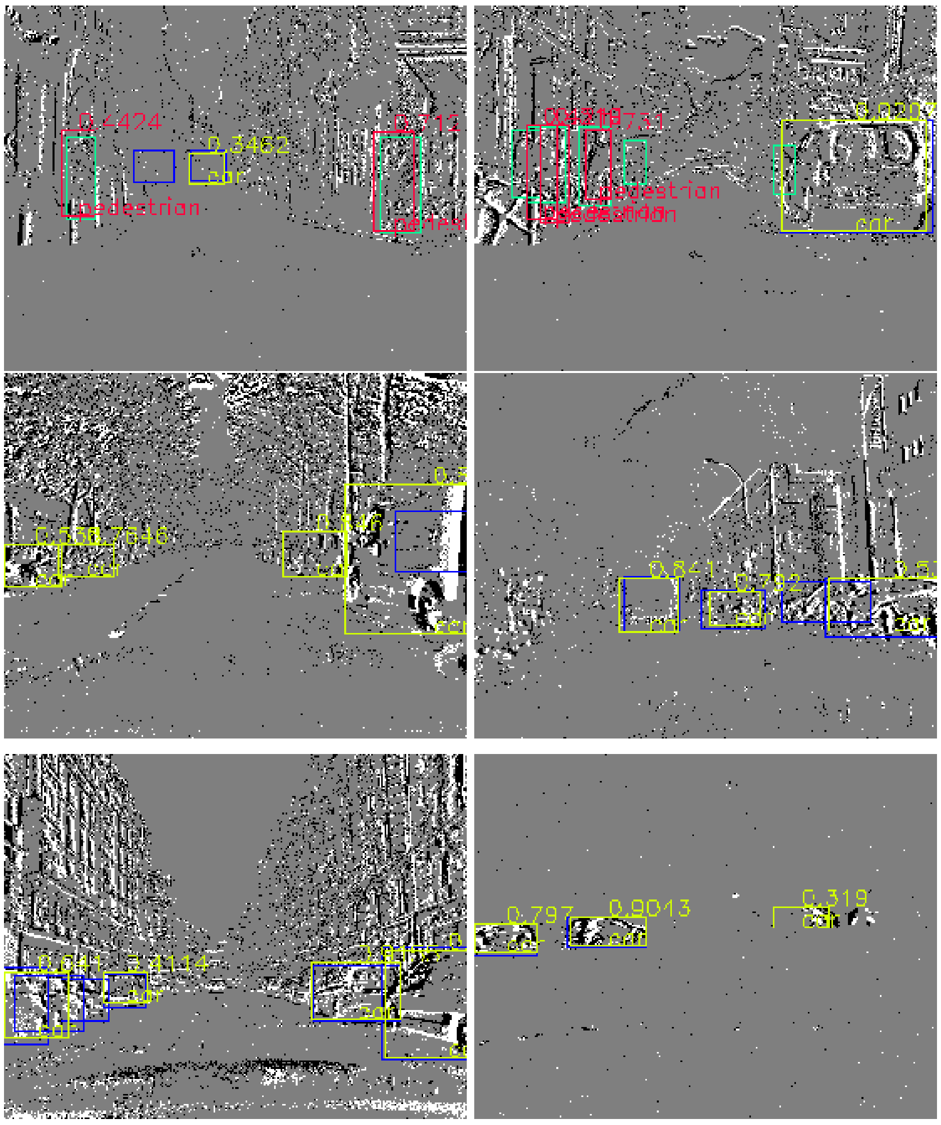


Figure 4.5: Predictions of the YOLOv3 + Matrix-LSTM architecture on few samples taken from the Gen1 Automotive Dataset's test set [135]. Red and yellow boxes refer to *pedestrians* and *cars* predictions, respectively, while blue and green boxes indicate ground truth annotations of the same classes. We report the predicted class probability, along with the label, on top of every predicted box.

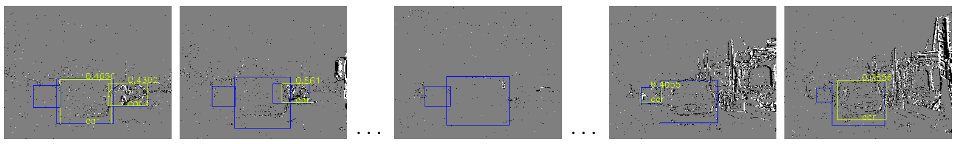


Figure 4.6: A failure case of Matrix-LSTM detection. The picture shows a continuous sequence in which both the ego-vehicle and the car in front come to a stop. Detection of the preceding car is interrupted when it stops triggering new events but restarts as soon as motion from either cars resumes.

We train Matrix-LSTM end-to-end to perform predictions on 50ms of data, following Perot et al. [104]. This gives the flexibility to learn high-quality representations for short-term encodings without incurring in memory issues caused by training from longer sequences, which is often impossible due to their size. Nevertheless, this configuration may sometimes prevent Matrix-LSTM from learning long-term behaviors, which are critical for performance. Figure 4.6 shows an example of these failure cases in which the layer, having been trained on relatively brief sequences, has not learned to preserve its internal state for long enough to handle cars that have stopped and thus ceased triggering new events. State-of-the-art performance is attained by the RED [104] architecture, a subsequent work to Matrix-LSTM, which is capable of predicting objects even in these challenging scenarios. The approach is comparable to that of Matrix-LSTM. However, recurrence is applied in deeper layers of the network rather than at the input, and a sequence of voxel grids is used instead on a learned representation, thus reducing memory usage. Despite a less expressive input representation, RED performance is higher, as it can better handle these conditions by training with longer sequences.

This chapter proposes to train Matrix-LSTM end-to-end without any additional modification to the backbone network for evaluating the quality of its features. We show that this approach is enough to increase performance, demonstrating Matrix-LSTM's advantage as an input representation. Although simple end-to-end training may not be sufficient to induce the network in learning complex behaviors from short sequences, we believe that adding unsupervised losses to discourage information loss or using different training schemes can significantly improve performance even in these cases. Along this line, we discuss possible extensions and future research directions in the conclusive chapter of this thesis.

4.5.4 Space and time complexity analysis

This section analyzes Matrix-LSTM's efficiency in terms of time and space complexity. We conduct experiments comparing our layer against both previous event-based representations as well as other traditional deep networks'

Table 4.10: Average sample computation time on N-Cars [85] and number of events processed per second.

Method	Bins	Channels	Asynch.	Time [ms]	Speed [kEV/s]
Gabor-SNN [85]	-	-	Yes	285.95	14.15
HOTS [84]	-	-	Yes	157.57	25.68
HATS [85]	-	-	Yes	7.28	555.74
EST [92]	9	2	No	**6.26**	**632.9**
Matrix-LSTM	1	3	No	10.89	385.7
(Ours)	9	2	No	8.25	468.36

Table 4.11: Average time to reconstruct the event surface in MVSEC [136] test sequences.

	Mean reconstruction time (on GPU) [ms]		
Bins	Ev-FlowNet [54] surf.	EST [92]	Matrix-LSTM
1	2.53 ± 2.74	3.62 ± 2.35	3.20 ± 0.97
2	2.01 ± 1.22	3.94 ± 1.47	5.18 ± 1.68
9	2.04 ± 1.20	9.09 ± 1.96	4.92 ± 1.47

layers. Indeed, since Matrix-LSTM is structurally similar to a ConvLSTM [192] layer, with some crucial differences that make it more suitable to process raw event streams, we also compare the two under the efficiency-vs-accuracy tradeoff, both in training and prediction settings. All evaluations, unless explicitly specified, are performed in PyTorch on a GeForce GTX 1080Ti GPU.

Time Efficiency

TIME COMPLEXITY. We compare the time performance of Matrix-LSTM with other event representations following EST [92] and HATS [85] evaluation procedures. In Table 4.10 we report the time required to compute features from a single sample, averaged over the whole N-Cars training dataset, for both ResNet–E2Vid and ResNet–EST configurations. Our representation achieves similar time performance than HATS and EST, performing only ∼2ms slower than EST on the same setting (9 bins and 2 channels). Similarly, in Table 4.11, we compute the mean representation's computing time for MVSEC *indoor_flying* test sequences. While EST can exploit parallel batch computation within each sample, as it processes every event independently, Matrix-LSTM relies on sequential computation to reconstruct the surface. Despite this difference, the custom CUDA kernels we designed enable bins and pixel sequences to be processed in parallel, drastically reducing the processing time.

The performance reported in Table 4.10 is computed on each sample independently to enable a fair comparison with the other methods. In Figures

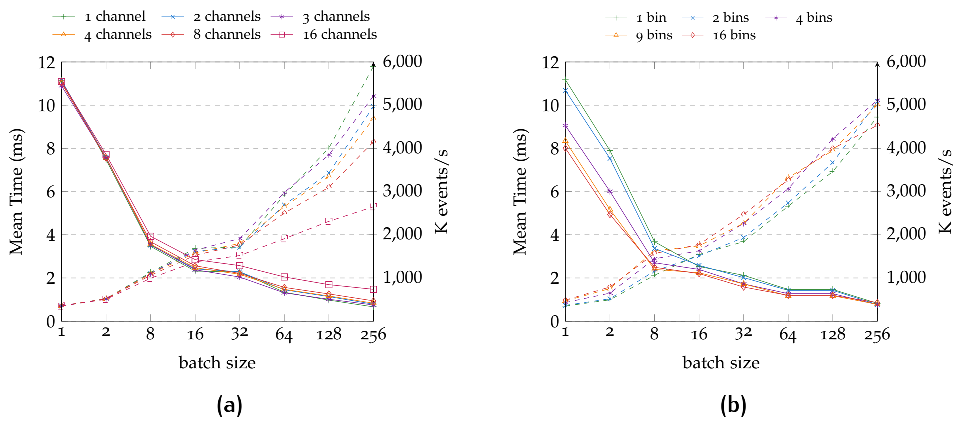


Figure 4.7: Number of processed events per second (dashed lines) and timing (solid lines) with varying number of channels **(a)**, and bins **(b)**.

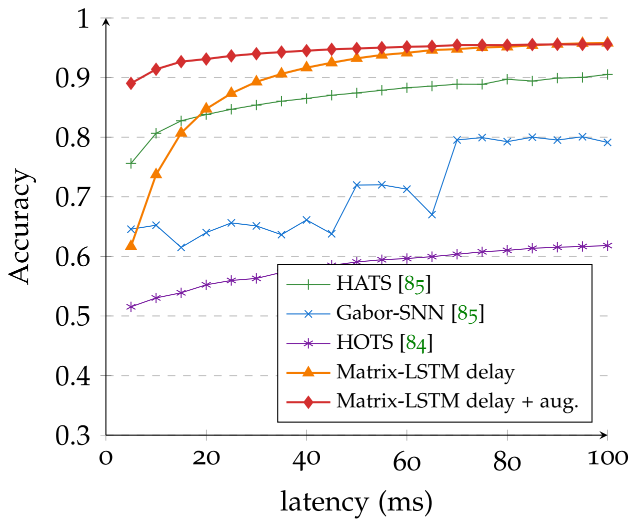


Figure 4.8: Accuracy as a function of latency (adapted from [85]).

4.7a and 4.7b we study instead how the mean time required to process a sample, averaged over all the N-Cars training dataset, changes as a function of the batch size. Performance dramatically increases when multiple samples are processed simultaneously in a batch. This is especially important during training as optimization techniques greatly benefit from batch-based computation.

Furthermore, while increasing the number of output channels, given the same choice of batch size, increases the time required to process each sample (since the resulting Matrix-LSTM operates on a larger hidden state), increasing the number of bins has the opposite behavior. Indeed, in this configuration, Matrix-LSTM splits sequences across multiple intervals, besides the pixel locations, enabling further parallelization, as intervals and pixel-sequences are treated independently.

Table 4.12: Accuracy (%) comparison on N-Cars [85] between Matrix-LSTM and ConvLSTM on the ResNet18–E2Vid configuration (3 channels and 1 bin).

	delay relative		ts absolute	
	3×3	5×5	3×3	5×5
Matrix-LSTM (ours)	**95.05 ± 0.96%**	**93.38 ± 0.64%**	**94.92 ± 0.74%**	**94.34 ± 0.94%**
ConvLSTM [192]	92.33 ± 0.41%	92.65 ± 0.78%	93.97 ± 1.30%	93.61 ± 1.59%

Table 4.13: Accuracy (%) comparison on N-Cars [85] between Matrix-LSTM and ConvLSTM on both ResNet18–EST configuration (16 channels and 1 bin).

	delay relative		ts absolute	
	3×3	5×5	3×3	5×5
Matrix-LSTM (ours)	**93.14 ± 0.77%**	**92.18 ± 0.28%**	**92.83 ± 1.32%**	**92.15 ± 0.67%**
ConvLSTM [192]	90.39 ± 0.94%	90.73 ± 1.05%	92.52 ± 1.26%	92.05 ± 0.56%

ACCURACY-VS-LATENCY TRADE-OFF. Following the analysis conducted by Sironi et al. [85], in Figure 4.8 on the preceding page we analyze the accuracy-vs-latency trade-off of Matrix-LSTM on the N-Cars dataset under the ResNet18-E2Vid setup. We first test Matrix-LSTM's ability to incrementally aggregate events over a variable time window. We train a classification network on N-Cars using 100ms sequences (i.e., full samples) and then vary the length of the sequence used for testing. While performance significantly drops when very few milliseconds of events are considered, the proposed method still shows good generalization, achieving better performance than the baselines when more than 20ms of events are used. Fixing the performance loss on small latencies is just a matter of training augmentation: by randomly cropping sequences to variable lengths (from 5ms to 100ms) during training, our method consistently improves the baselines, dynamically adapting to sequences of different lengths, even for very short latencies.

Matrix-LSTM vs. ConvLSTM

RECEPTIVE FIELD SIZE. In Tables 4.12 and 4.13 we compare Matrix-LSTM with ConvLSTM [192] for different choices of kernel size on the N-Cars [85] dataset using both ResNet18–E2Vid and ResNet18–EST backbones. When using ConvLSTM, events are densified in a volume $\tilde{\mathcal{E}}_{dense}$ of shape $N \times T_{max}^{(x,y)} \times H \times W \times F$. Matrix-LSTM performs better on all configurations, despite achieving lower performance than its best configuration with 1×1 kernels reported in Table 4.1 on page 83. Event surfaces produced by the Matrix-LSTM layer are indeed more blurry with larger receptive fields and this may prevent the subsequent ResNet backbone from extracting effective features. On the other hand, using a 1×1 kernel allows Matrix-LSTM to only focus on temporal information, leaving the subsequent convolutional layers in charge to deal with spatial correlation. A visual representation of the effect of

Receptive Field Size		
1×1	3×3	5×5
$95.80 \pm 0.53\%$	$95.05 \pm 0.96\%$	$94.38 \pm 0.64\%$

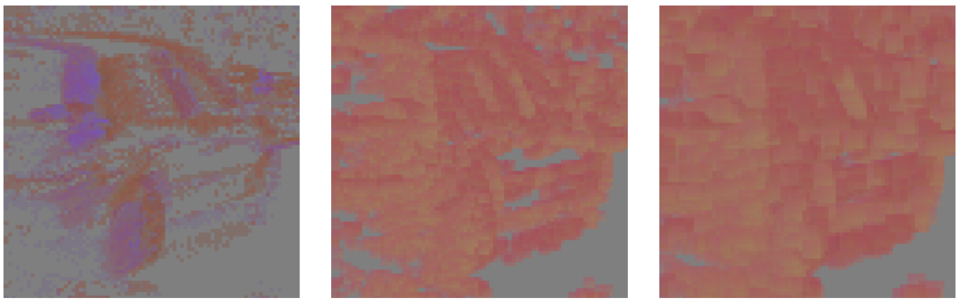


Figure 4.9: Matrix-LSTM Resnet18-E2Vid performance on N-Cars with varying receptive field size and stride 1×1.

increasing MatrixLSTM's receptive field size, along with the corresponding accuracy performance, is provided in Figure 4.9.

ConvLSTM, instead, does not properly handle asynchronous data when large receptive fields are employed, which may explain the performance discrepancy with Matrix-LSTM. To make events compatible with ConvLSTM, they need to be densified first. A dense event volume $\tilde{\mathcal{E}}_{dense}$ is first computed by assigning to each pixel location a vector containing the pixel's event sequence, padded with zeros at the end to match the longest sequence, i.e., $\tilde{\mathcal{E}}_{dense}(x, y) = pad(\mathcal{E}^{(x,y)}, T^{(x,y)}_{MAX})$. Because pixels at different locations often fire at different times and with varying frequencies, the $\tilde{\mathcal{E}}^{t}_{dense}$ slice processed by the ConvLSTM in each iteration t may not contain all simultaneous events. Using a large ConvLSTM receptive field means to compare a neighborhood of events that occurred at different timestamps and, therefore, not necessarily correlated. We also highlight that the performance improvement is greater when using *delay relative* temporal features than with *ts absolute* ones. Indeed, while delays are always consistent within each pixel sequence, they are not within the ConvLSTM kernel receptive field. Using an absolute temporal encoding alleviates this issue but still provides worse performance than Matrix-LSTM. In contrast, Matrix-LSTM provides more flexibility when large receptive fields are considered. Indeed, events do not need to be densified beforehand, and they are always processed by maintaining the original arrival order, even when large receptive fields are used. Note that we do not compare the two LSTMs on the 1×1 configuration since, when using $\tilde{\mathcal{E}}_{dense}$ as input to ConvLSTM, the two configurations compute the same transformation, despite ConvLSTM having to process more padded values. The two settings are indeed computationally equivalent only in the worst case in which all pixels in the batch happen to receive at least one event (i.e., $P = N \cdot H \cdot W$),

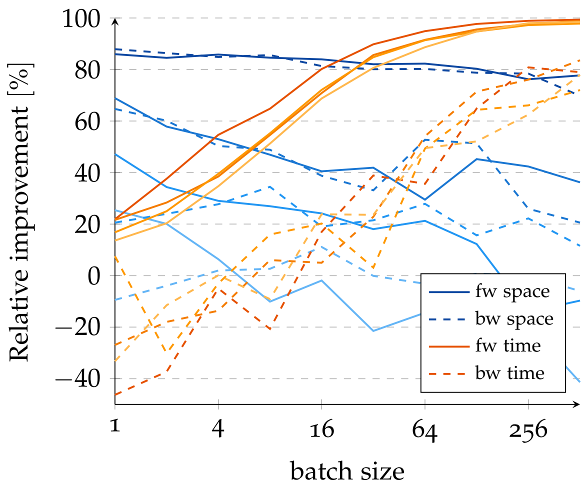


Figure 4.10: Space and time relative improvements of Matrix-LSTM over ConvLSTM as a function of the input density (from 10% to 100% with 30% steps). Colors refer to different density, from low (dark colors) to high (light colors).

which is however particularly unlikely, favoring Matrix-LSTM also in this configuration.

SPACE AND TIME COMPLEXITY. We compare the 1×1 configurations in terms of space and time efficiency in Figure 4.10. We use the two layers to extract a 224×224 frame from artificially generated events with increasing density, defined as the ratio of pixels receiving at least one event. Benchmarking is performed under PyTorch [267] on a 12GB Titan Xp by varying the batch size, the LSTM hidden size, and the number of events in each active pixel. We start from one hidden feature and increase its size by a factor of 2, while we increase the number of events by a factor of 10, until allowed by GPU memory constraints. We compute the relative improvement of Matrix-LSTM in terms of representation's computation time and peak processing space (i.e., excluding the ResNet and input space) during both forward and backward stages. Finally, we aggregate the results by batch size computing the mean improvement over all the trials.

Matrix-LSTM performs better than ConvLSTM on prediction time, with the time efficiency increasing as the batch size increases. Its performance is worst than ConvLSTM on memory efficiency, but only on very dense surfaces ($>$ 70% density). However, this situation is quite uncommon in event-based cameras since they typically generate very sparse event streams. Indeed, uniform parts of the scene that remain at a constant brightness, despite the camera movement, do not appear in event-based camera recordings. For instance, the background sky and road in MVSEC [136] make *outdoor_day* sequences only have an average 10% of active pixels. Crucially, in deployment conditions, when the backward pass is not considered and the batch size is set to 1, Matrix-LSTM always outperforms ConvLSTM in both time and space complexity, further highlighting its advantage over ConvLSTM even outside training regimes.

4.6 CONCLUSIONS

This chapter proposed Matrix-LSTM, an effective method for learning to encode events in grid-like event representations. By modeling the event encoding process with a spatially shared LSTM, we obtain a fully differentiable procedure that can be trained end-to-end to extract the event representation that best fits the task at hand. Focusing on efficiently handling asynchronous data, Matrix-LSTM preserves sparsity during computation and surpasses other popular LSTM variants on space and time efficiency when processing sparse inputs. In this regard, we propose an efficient implementation of the method that exploits parallel batch-wise computation, showing efficiency comparable to other event-based solutions and outperforming traditional recurrent methods. We demonstrate the effectiveness of the Matrix-LSTM layer on multiple tasks. We improve the state-of-the-art of object classification on N-Cars by 3.3%, reach performance on par to that of grayscale reconstraction methods on the Gen1 Automotive Detection dataset [135], and obtain a relative improvement over previous differentiable techniques [92] by up to 23.07% on MVSEC's optical flow prediction. Although we mostly aggregate fixed windows of events, the proposed mechanism can be used to process continuous streams thanks to the LSTM memory, which is able to update its representation as soon as a new event arrives. We show this on the N-Cars dataset, where Matrix-LSTM generalizes to different window lengths and enables reliable predictions even at low latencies. As a future line of research, we plan to explore the use of Matrix-LSTM for more complex tasks such as grayscale frame reconstruction [126], ego-motion, and depth estimation [55, 56].

5 LEARNING DOMAIN-INVARIANT NETWORKS WITH EVENT SIMULATION

The previous two chapters focused on designing general methods for computing and extracting event representations, whether at the input or feature level, to maximize efficiency and performance. However, it becomes apparent that accuracy and robustness in such learning-based methods can only be achieved when large training sets featuring high-quality annotations are available. As a result, in many challenging applications, event-based vision is increasingly relying on simulation to compensate for the absence of annotated data. While this approach to vision has contributed to advancing the field in many applications, it also comes with a few open research questions: how well can simulated data generalize to real environments? Furthermore, which event representations are better suited to dealing with simulated events?

This chapter continues the analysis of event representations of the previous two chapters but moves the focus from the efficiency-vs-accuracy tradeoff towards studying their robustness when training conditions do not perfectly match real-world settings. We study this scenario under the lens of Unsupervised Domain Adaptation (UDA) and show that traditional adaptation approaches can also be utilized on events to encourage the extraction of domain-invariant features. Moreover, we also prove the effectiveness of UDA methods when dealing with synthetic scenes and propose an extension of the RGB-D Object Dataset (ROD) [22] to foster future research on this analysis.

5.1 INTRODUCTION

Learning-based methods have played a significant role in establishing the efficacy of event-based cameras in a variety of challenging conditions where standard cameras typically struggle. Several works [104, 207] exploit the high

This chapter is based on:
- Mirco Planamente, Chiara Plizzari, Marco Cannici, Marco Ciccone, Francesco Strada, Andrea Bottino, Matteo Matteucci, and Barbara Caputo. "DA4Event: Towards Bridging the Sim-to-Real Gap for Event Cameras Using Domain Adaptation." In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 6616–6623. DOI: 10.1109/lra.2021.3093870. ©2021 IEEE.
- Marco Cannici, Chiara Plizzari, Mirco Planamente, Marco Ciccone, Andrea Bottino, Barbara Caputo, and Matteo Matteucci. "N-ROD: a Neuromorphic Dataset for Synthetic-to-Real Domain Adaptation." In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2021. DOI: 10.1109/cvprw53098.2021.00148. ©2021 IEEE.

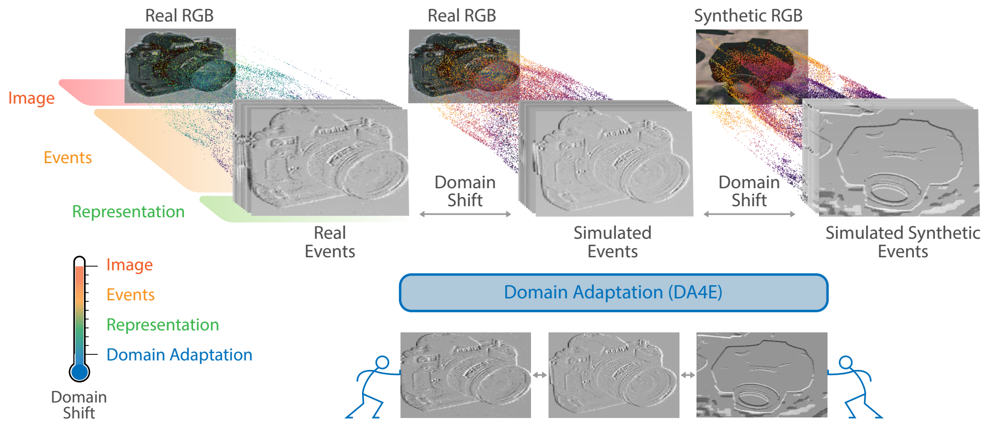


Figure 5.1: *How can we bridge domain shifts in event-based cameras?* DA4Events exploits unsupervised domain adaptation techniques to solve this problem by acting at the feature level. *How else can simulated events be used?* We propose to use events in a real context, exploiting the complementarity with RGB data to improve networks robustness.

dynamic range and temporal resolution of the sensor to provide reliable predictions even in the presence of fast motion and abrupt brightness changes, while others [58, 108, 279, 280] use them in conjunction with standard devices to exploit the benefits of both event-driven and traditional approaches. However, the scarcity of annotated training data is still hampering the full potential of learning-based approaches in many fields of event-based vision, especially in low-level tasks where acquiring high-quality annotations is particularly challenging.

Some works [55, 56, 204] propose to exploit unsupervised learning procedures to train event-based neural networks even in the absence of proper annotations. However, aside from the effort needed to design and adapt such learning tools, this possibility is only applicable in a limited number of applications and only provides sub-optimal performance when compared to supervised training [207]. A more general, task-independent solution involves exploiting event-based simulators [127, 165, 168, 170] to generate simulated event streams for training. Being able to generate data from both static images and traditional RGB videos, these simulators enable reusing any publicly available RGB dataset, thus taking full advantage of their annotations. Moreover, when paired with synthetic renderers [57], they reduce the effort of collecting event data even further and enable training with pixel-level accurate annotations.

Nevertheless, despite ongoing efforts in increasing the realism of event simulation [170], current simulators are still incapable of perfectly replicating the output of a genuine sensor. Since the operating principles of event-based cameras are complex and difficult to characterize fully [25], event simulators typically resort to simplified event generation models [165, 168] that only partially simulate sources of noise and non-idealities. Moreover, hyperparameters used during simulation, most notably the contrast threshold, might be set differently from the values used in the real camera, moving the two data distributions even further away. These differences often impact the performance of deep neural networks trained solely on simulation, which, by specializing in recognizing specific features of simulated events, may not obtain full performance when used on actual event streams.

These disparities between training and testing data distributions are typically referred to in the literature as the *Sim-to-Real* gap, or shift [281]. In event-based vision, this issue has primarily been studied in relation to *simulator parameters*, i.e., the contrast threshold, to generate more realistic recording by operating at the input level during data simulation. Gehrig et al. [127] propose to randomly sample the threshold C during training to make the network more robust to different hyperparameters, while Stoffregen et al. [281] show how to estimate a threshold that matches the one used on a real sensor. However, these approaches only partially address the issue, as they do not account for non-idealities in simulation other than potentially incorrect hyperparameter selections. Moreover, to date, no previous work has proposed ways to deal with extra distribution shifts caused by the use of rendering. Indeed, when event simulation is performed on synthetic scenes, discrepancies in event generation add to the differences in RGB distributions resulting in a double shift that impacts network performance even more. We refer to this combined shift as the *RGBE-Synth-to-Real* gap, while we call *E-Sim-to-Real* the simpler shift where RGB images come from the same distribution. A general procedure capable of dealing with all these inconsistencies and different tasks is still lacking in event-based vision.

5.1.1 Main Contributions

In this chapter, we propose to address these issues by leveraging Unsupervised Domain Adaptation (UDA) techniques [282–286]. Indeed, although these methods have been extensively used in computer vision to compensate for domain shifts, researchers have to date overlooked their potential in event-based vision. Our insight is that reducing this gap by operating *at feature level* leads to more transferable representations than traditional methods operating at the input level. Acting on network features rather than directly on input streams simplifies the problem since deeper representations, being more abstract, are simpler to align than low-level ones.

This chapter focuses on both the E-Sim-to-Real and the RGBE-Synth-to-Real shifts and analyzes different traditional domain adaptation techniques in these contexts. Inspired by the frame-based computer vision literature, we also analyze the impact of pretraining on network robustness and design *MV-DA4E*, a simple approach to encourage preserving robust pre-trained feature extractors during training[1]. Finally, we also study domain shifts in

1 Code available at https://github.com/DA4EVENT/home.

multimodal settings, showing that event data can increase robustness in neural networks by leveraging its complementarity with appearance-only modalities such as RGB images.

In summary, the contributions of this chapter are the following:

- We propose to bridge both *E-Sim-to-Real* and *RGBE-Synth-to-Real* gaps for event cameras using UDA techniques, which so far are still under-explored in the event-based field, reducing the issue to a domain shift problem. We show how the domain shift affects in different ways various event representations and to what extent different UDA approaches can soften these issues.

- We demonstrate the importance of exploiting robust pre-trained feature extractors when dealing with domain shifts in event-based vision. To this end, we propose a simple yet general approach, which we called *MV-DA4E*, that enables preserving low-level features regardless of the number of temporal bins in the input event representation of choice.

- Through extensive experiments on the object classification task using N-Caltech101 [116] and its simulated version Sim-N-Caltech101 [127], we demonstrate the effectiveness of the proposed approach. In particular, we show that UDA methods are able to fill the gap between the simulated and real event domains, obtaining performance comparable to a model trained on real data. Moreover, we show that the proposed approach also generalizes to semantic segmentation on DDD17 [138], obtaining better performance than other methods acting only on simulation parameters.

- Finally, as no existing dataset enables the analysis of even-based *RGBE-Synth-to-Real* shifts, we propose to extend the RGB-D Object Dataset (ROD) [22] and its synthetic counterpart [284] with simulated and real events. The novel N-ROD dataset provides RGB, depth, and event data. Thus, it does not only enable the study of the RGBE-Synth-to-Real gap, but it also unlocks multimodal analysis, encouraging further research in this direction.

5.2 RELATED WORKS

When training a deep neural network for real-world applications, we often are in a situation where the available training data does not perfectly match the conditions in which the network will be deployed. Although real-world data may typically be obtained with little effort, the work required in extracting accurate annotations often prohibits it from being utilized for supervised training. UDA proposes to address this problem by jointly exploiting both out-of-domain labeled data as well as unlabeled target data during training. The goal is to use labeled data, usually termed as the *source* domain, to learn the task at hand and then exploit unlabeled *target* data to induce an alignment between the distribution of features extracted from the two domains. Suppose features extracted from the target domain resemble those of the source domain, i.e., as if they were drawn from the same distribution. In that case, a predictor trained with supervision on source, out-of-domain data will likely perform well also on target, real-world data, even without finetuning.

SINGLEMODAL UDA. UDA methods can be categorized according to the adaptation strategy used. The first group comprises *discrepancy-based* methods, such as the maximum mean discrepancy (MMD) [283], which explicitly minimize a distance metric between source and target distributions [285, 287]. This constraint is often imposed at *feature level* and in deep layers of the network. Alternatively, *adversarial-based* methods [288, 289] promote domain-invariant features either leveraging adversarial losses or a gradient reversal layer (GRL) [282]. Another possibility is to use self-supervised *pretext tasks* [284, 290–292], whose losses act as an adaptation regularizer of the main loss. Finally, other works exploit batch normalization layers to align source and target statistics [293–295], while other use *generative-based* methods to perform style-transfer directly on input data [296, 297].

UDA FOR SYNTHETIC-TO-REAL SHIFTS. Some of the methods discussed earlier tackle the problem of the *Synthetic-to-Real* transfer for object classification tasks [282, 284, 285], intended as the domain shift between RGB images rendered through simulation and real RGB ones [298–300]. While these methods are general, several works adapt these procedures to the task at hand. To that end, a lot of effort has been put on semantic segmentation, where unsupervised adversarial approaches are widely used [297, 301–303]. Hoffman et al. [301] are the first to propose an unsupervised adversarial approach on semantic segmentation. Following the same setup, Zhang et al. [304] propose a curriculum-based adaptation framework, while Tsai et al. [302] use a multi-level adversarial network to perform adaptation at different feature levels. Departing from the adversarial setting, Vu et al. [303] extend the traditional entropy minimization framework to pixel-level predictions, while Hoffman et al. [297] propose to use a CycleGAN [297] to generate target images for training, based on synthetic ones. In this chapter, we take advantage of this literature and propose to utilize these methods to tackle multimodal domain shifts on event data for semantic segmentation.

MULTIMODAL UDA. The above methods have been specifically designed to deal with data coming from a single modality. Very few works address the problem of multimodal domain adaptation (MDA), but the topic has been attracting more and more attention over the last few years. Most of the existing MDA methods consist of simple extensions of single-modal domain adaptation methods. In particular, Wang and Zhang [305] apply a standard adversarial domain adaptation strategy on RGB and depth data (RGB-D) without relying on any relationship between the two modalities. Similarly, Li et al. [306] propose to exploit depth data only for adaptation, ignoring the potential benefits of using multimodal data to increase robustness on the target domain. Instead, Loghmani et al. [284] introduce a novel self-supervised pretext task that relates the two modalities during training, thus making the network more robust to domain shift. Along these lines, this chapter proposes tackling the MDA problem by leveraging depth and event modalities jointly.

5.3 UNSUPERVISED DOMAIN ADAPTATION

Given the previous high-level overview of Unsupervised Domain Adaptation (UDA) approaches in computer vision, we now focus on describing a few selected methods in greater depth. These methods are typically used to perform domain adaptation on RGB images, but have also demonstrated good performance on different modalities. As detailed in later sections, this chapter proposes to integrate them into a unified framework for unsupervised domain adaptation on events. Since the use of UDA techniques is completely novel in event-based vision, no prior knowledge is available on which approach works best and how these methods combine with different input representations. To perform a thorough analysis of unsupervised methods on events, we integrate UDA approaches utilizing different adaptation strategies (discrepancy-based, adversarial-based, and semisupervised ones) and investigate their effectiveness both in single modal and multi-modal settings, combining event data with RGBs. In the following, we give a detailed description of each of these methods individually, while the following section describes how these approaches can be incorporated into a broader framework for event-based adaptation.

5.3.1 Gradient Reversal Layer (GRL)

Ganin and Lempitsky [282] were amongst the first to propose addressing unsupervised domain adaptation with adversarial training. They focus on object classification and consider the case of a general feedforward neural network composed of a feature extractor $\mathcal{F}$ with learnable parameters θ_f followed by a label predictor $\mathcal{G}_y$ with parameters θ_y. In other words, classification is performed as $y = \mathcal{G}_y(\mathcal{F}(\mathbf{x};\theta_f);\theta_y)$. In this setting, domain adaptation is achieved when features $\mathbf{f} = \mathcal{F}(\mathbf{x};\theta_f)$ produced by the feature extractor follow the same distribution even if the input $\mathbf{x}$ comes from two different domains, i.e., source and target data. Assessing and minimizing the differences between these two distributions is not trivial in deep learning settings since features $\mathbf{f}$ are typically high-dimensional, and they constantly change during the learning process. The authors propose to achieve this by looking at the classification performance of an additional domain classifier $\mathcal{G}_d$ that, given features $\mathbf{f}$ as input, is trained to recognize the domain to which x belongs. Its output is given by $d = \mathcal{G}_d(\mathcal{F}(\mathbf{x};\theta_f);\theta_d)$. When the domain classifier cannot complete this task reliably, then the alignment of the two distributions is obtained. They propose to achieve this by jointly optimizing the classification accuracy of the label predictor $\mathcal{G}_y$ as well as that of the domain classifier $\mathcal{G}_d$ through the following loss:

$$\mathcal{L}_{GRL} = \mathcal{L}(\theta_f, \theta_y, \theta_d) = \mathcal{L}_y(\theta_f, \theta_y) + \lambda \mathcal{L}_d(\theta_f, \theta_d), \tag{5.1}$$

where $\mathcal{L}_y(\cdot,\cdot)$ is the label prediction loss on the source domain, typically a cross-entropy loss, and $\mathcal{L}_d(\cdot,\cdot)$ is the binary cross-entropy loss of the domain classifier. The parameters θ_y and θ_d are trained to maximize the performance of the two classifiers, while θ_f is trained to achieve the joint objective of maximizing task performance but also fooling the domain classifier.

To achieve this, a *gradient reversal layer* is put in between the feature extractor and the domain classifier (i.e., $d = \mathcal{G}_d(GRL(\mathcal{F}(\mathbf{x};\theta_f));\theta_d)$) to invert the sign of gradients backpropagated through $\mathcal{F}$ when minimizing $\mathcal{L}_d(\theta_f, \theta_d)$. The feature extractor is thus trained not to highlight domain-specific features that would make the task of discriminating the two domains easier. As a result, features produced by $\mathcal{F}$ are domain-invariant, and $\mathcal{G}_y$ can leverage this property to achieve reliable predictions even on target data.

5.3.2 Rotation (ROT) and Relative Rotation (RR) Tasks

Xu et al. [290] propose to perform unsupervised domain adaptation using an additional self-supervised task as a regularizer. The idea behind this kind of approach is to solve an additional task during training, along with the main classification, to incentivize the extraction of domain invariant features. These tasks, usually called *pretext* tasks, do not require any manually annotated labels but instead consist in solving relatively simple problems involving geometric transformations. Some known transformation is applied to available data, and the model is trained with supervision to recognize these transformations. The intuition is that, by jointly optimizing the main and pretext tasks together, the optimization procedure is asked to find a solution in which extracted features must be valid both for classification as well as for solving the auxiliary task. As a result, if the pretext task is designed to encourage the extraction of general features, the main task can leverage these features to make more reliable predictions even on target data.

The authors propose to achieve this through a self-supervised rotational task. During training, along with the main objective, samples from both the source and the target domain are randomly rotated of an angle $\Theta \in \{0°, 90°, 180°, 270°\}$. An additional classifier is trained to classify to what extent samples have been rotated based on features $\mathbf{f}$ extracted from the shared feature extractor. In order to solve this task, the feature extractor is encouraged to focus more on geometrical information and less on any peculiar characteristic of the source domain, resulting in domain-invariant features that benefit performance on target data.

Loghmani et al. [284] extend this task to multi-modal settings in which a network is trained to perform classification based on data from the same scene but captured with two different sensors. As before, data from the two modalities is rotated by two random angles θ_1 and θ_2. However, instead of predicting the absolute rotation of each modality, the authors propose to predict the relative rotation $\hat{\theta} \in |\theta_1 - \theta_2|$. This approach has the advantage of relating the two modalities during training, encouraging the extraction of complementary features that may help robustness when predicting out-of-domain samples.

5.3.3 Maximum Mean Discrepancy (MK-MMD)

The method proposed by Long et al. [283] falls onto discrepancy-based approaches, in which the distance between feature distributions is directly minimized through a loss. They propose to achieve this using a multi-kernel extension [307] of the Maximum Mean Discrepancy (MMD) [308], often referred to as MK-MMD. The key idea is to measure the difference between the two distributions by computing the mean embedding difference of the two through a reproducing kernel Hilbert space (RKHS), where this distance can be explicitly minimized. Given two distributions p and q, the MK-MMD distance is defined as the RKHS distance between the mean embeddings of p and q as:

$$d_k^2(p,q) \triangleq \left\| \mathbf{E}_p \left[\phi \left(\mathbf{x}^s \right) \right] - \mathbf{E}_q \left[\phi \left(\mathrm{x}^t \right) \right] \right\|_{\mathcal{H}_k}^2, \tag{5.2}$$

where $\phi(\cdot)$ is a multi kernel mapping the features in the Hilbert space.

During training, the following loss is directly minimized:

$$\mathcal{L}_{MMD} = \mathcal{L}_y(\theta_f, \theta_y) + \lambda \sum_{\ell=l_1}^{l_2} d_k^2 \left(\mathcal{D}_s^\ell, \mathcal{D}_t^\ell \right) \tag{5.3}$$

where $\mathcal{L}_y(\theta_f, \theta_y)$ is the classifier loss, as in GRL in Equation 5.1 on the preceding page, $\mathcal{D}_s^\ell$ and $\mathcal{D}_t^\ell$ are the hidden representations at the ℓ-th layer of the network computed from source and target samples, respectively, and d_k^2 computes the MK-MMD distance between the two features distributions. This loss is applied over multiple layers $\ell = \ell_1, \ldots, \ell_2$ to facilitate the alignment of feature spaces in deeper layers of the network. By relying on the property that $p = q$ iff $d_k^2(p,q) = 0$ [307], this approach directly minimizes the distance between the two distributions, guaranteeing the extraction of domain-invariant features when convergence is reached.

5.3.4 Adaptive Feature Norm (AFN)

Xu et al. [285] propose another discrepancy-based approach to address unsupervised domain adaptation. They first analyzed the cause underlying a difficult classification on out-of-domain samples and discovered that features computed on target data are typically characterized by much smaller norms if compared to that of the source domain. These features convey much less information and thus increase the uncertainty of the classifier that cannot rely on discriminative features to perform prediction. To tackle this issue, the authors propose to achieve a feature alignment indirectly by seeking to extract features with larger norms even when processing target samples. In order not to interfere with the learning process, they propose to reach this condition iteratively, as learning progresses. As in the previous case, the loss is composed of a supervised and an adaptation part:

$$\mathcal{L}_{AFN} = \mathcal{L}_y(\theta_f, \theta_y) + \frac{\lambda}{n_s + n_t} \sum_{x_i \in D_s \cup D_t} L_d \left(h\left(x_i; \theta_f^0\right) + \Delta r, h\left(x_i; \theta_f\right) \right), \quad (5.4)$$

where $h\left(x_i; \cdot\right)$ computes the mean L2-norm of features computed from samples x_i through the feature extractor and L_d implements an MSE loss. θ_f^0 and θ_f represent the feature extractor parameters treated as constants and learnable parameters respectively. The second term of the loss gradually increments features' norms by a step Δr at each learning step. As a result, the feature extractor is encouraged to produce features with higher norms, thus more informative, making the task of classifying objects easier for the label predictor even on the target domain.

5.3.5 Entropy Minimization (ENT)

Domain adaptation through entropy minimization relies on considerations similar to that of AFN. When a classifier trained on one distribution is utilized to classify samples from another, the uncertainty is often larger than when prediction is made on source samples. This uncertainty can easily be measured by analyzing how the probability is spread among predicted classes. When a prediction is made on the source domain, the predicted class typically receives most of the probability, while very little is given to other classes. On the contrary, the probability is often spread almost evenly over all classes on target data, even in the case of a correct prediction.

Grandvalet and Bengio [286] propose to address this issue by minimizing the entropy of predicted class distributions as a regularizer on the main loss. More formally, the following loss is minimized during training:

$$\mathcal{L}_{ENT} = \mathcal{L}_y(\theta_f, \theta_y) - \frac{1}{|\mathcal{T}|} \sum_{x_t \in \mathcal{T}} G_y(F(x_t; \theta_f)) \cdot \log G_y(F(x_t; \theta_f)), \quad (5.5)$$

where F and G_y are respectively the feature extractor and the label predictor, and x_t is a sample from the target distribution. The network is jointly trained to perform classification on the source domain through $\mathcal{L}_y$, as well as to commit to a single class prediction, even on target samples, by minimizing the additional entropy term. Through this minimization, the feature extractor is asked to learn a generic mapping that allows extracting discriminative features from both source and target samples, thus making sure that the classifier understands the structure of unlabeled data even without supervision.

5.3.6 Adversarial-based entropy minimization (ADVENT)

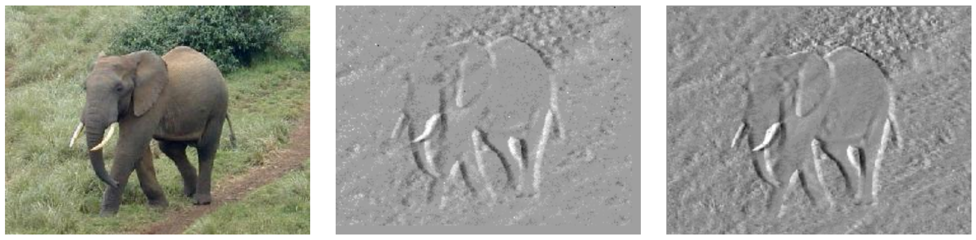
Vu et al. [303] extend the concept of entropy minimization to the task of semantic segmentation, where, instead of a global class prediction, each pixel predicts a class probability distribution $\boldsymbol{P}_{\boldsymbol{x}}^{(h,w,c)}$. The same considerations about uncertainty in predictions also apply to this case: over-confident (low-entropy) predictions are usually produced on source-like images, with the degree of uncertainty (entropy) growing as images move away from the source distribution. However, high-entropy regions are common in semantic segmentation, even in source-like images. These typically occur in correspondence of objects' boundaries, where the predicted class could potentially transition, while they are less common inside the objects, giving uncertainty maps the appearance of edge detectors.

Rather than globally minimizing entropy, Vu et al. [303] propose to take advantage of this characteristic and use an adversarial approach to encourage source-like entropy patterns even on the target domain's predictions. Uncertainty masks $\boldsymbol{I}_{\boldsymbol{x}}^{(h,w)} = -\boldsymbol{P}_{\boldsymbol{x}}^{(h,w)} \cdot \log \boldsymbol{P}_{\boldsymbol{x}}^{(h,w)}$ are first computed on both source and target domain samples, and then a domain discriminator is used to classify the original domain starting from these masks. Training is performed by alternately optimizing the domain classifier to recognize domains, as well as the feature extractor to fool the discriminator, through the following two objectives:

$$\min_{\theta_f} \frac{1}{|\mathcal{X}_s|} \sum_{\boldsymbol{x}_s} \mathcal{L}_y\left(x_s, y_s\right) + \frac{\lambda_{adv}}{|\mathcal{X}_t|} \sum_{\boldsymbol{x}_t} \mathcal{L}_D\left(\boldsymbol{I}_{\boldsymbol{x}_t}, 1\right), \tag{5.6}$$

$$\min_{\theta_d} \frac{1}{|\mathcal{X}_s|} \sum_{\boldsymbol{x}_s} \mathcal{L}_D\left(\boldsymbol{I}_{\boldsymbol{x}_s}, 1\right) + \frac{1}{|\mathcal{X}_t|} \sum_{\boldsymbol{x}_t} \mathcal{L}_D\left(\boldsymbol{I}_{\boldsymbol{x}_t}, 0\right), \tag{5.7}$$

where $\mathcal{L}_D$ is the binary cross-entropy loss of the discriminator. Notice that the expected label for target samples $\boldsymbol{I}_{\boldsymbol{x}_t}$ is the opposite in the two objective, since the first trains the discriminator to maximize performance (i.e., predict the correct label 0), while the second asks the feature extractor to produce entropy patterns making the discriminator mispredict. A simpler version of this approach has been proposed by Tsai et al. [302], who perform adversarial training directly on semantic segmentation logits produced on the two domains, instead of passing through uncertainty maps.



(a) RGB image (b) Real events (c) Simulated events

Figure 5.2: Real and simulated events (voxel grid [55]) on a Caltech101 sample.

5.4 METHOD

As pointed out by Gehrig et al. [127] and Stoffregen et al. [281], differences between simulated and real events can cause a drop in performance in several applications, independently by the representation used to encode events. As discussed in the previous section, this issue is typically caused by the network's tendency to overfit on peculiar features of the source domain, which, not being present in the target domain, cause the network to generalize poorly. An example is provided in Figure 5.2, where the same RGB image from the Caltech101 dataset [253] is converted into event streams by moving an event camera in front of it [116] or a virtual camera through simulation [127, 168]. These differences are typically caused by a wrong choice of simulator's hyperparameters or by the inability of event simulators to perfectly replicate the event generation process of a real camera, which might be affected by sensor noise and non-idealities. Despite often subtle, these discrepancies are significant enough to have an influence on network performance.

While Gehrig et al. [127] and Stoffregen et al. [281] suggest solving the problem by acting on the events' generation, our insight is to see the problem as a general *domain-shift* issue and employ unsupervised domain adaptation as a solution. We focus here on deep learning approaches in event-based vision and take advantage of deep neural networks' ability to extract abstract representations to address the problem at the *feature level*, where enforcing a distribution alignment is easier, rather than directly on events. We show that this approach provides better performance than previous methods as it does not focus on a specific discrepancy but instead attempts to reduce them all indistinguishably.

5.4.1 Formulation

Before describing the general framework and network architectures we designed, let us first formalize the UDA problem under our setting. Our goal is to learn, on a source domain $\mathcal{S} = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$ with N_s labeled samples associated with a known label space $\mathcal{Y}^s$, a neural network able to perform well on a target domain $\mathcal{T} = \{x_i^t\}_{i=1}^{N_t}$ with N_t unlabeled samples and label

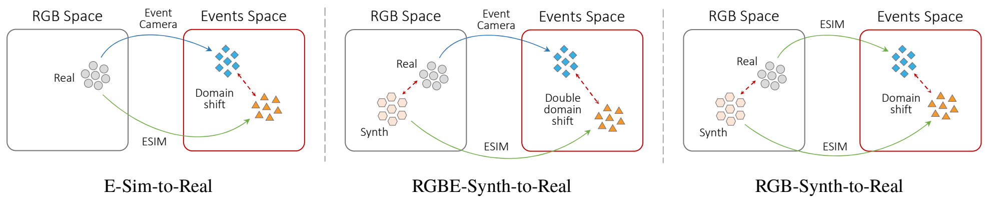


Figure 5.3: Visual representation of the three shifts studied in this chapter. Groups of symbols represent datasets being converted from the RGB space into the event space.

Table 5.1: A comparison between the settings studied in this chapter. We indicate as ESIM($\cdot$) the events obtained through simulation [168] from either synthetic or real RGB images, and with EvCamera($\cdot$) those obtained using a real event camera. We indicate Sim-to-Real and Synth-to-Real in different colors, and highlight the corresponding shift in the right side of the table using the same hue.

	Source		Target	
Setting	RGB	Event	RGB	Event
E-Sim-to-Real	Real	ESIM(RGB_{real})	Real	EvCamera(RGB_{real})
RGB-Synth-to-Real	Synth	ESIM(RGB_{synth})	Real	ESIM(RGB_{real})
RGBE-Synth-to-Real	Synth	ESIM(RGB_{synth})	Real	EvCamera(RGB_{real})

space $\mathcal{Y}^t$. We assume that (i) the two domains have different distributions, i.e., $\mathcal{D}_s \neq \mathcal{D}_t$, and (ii) they share the same label space, i.e., $\mathcal{Y}_s = \mathcal{Y}_t$. In single modal settings the input of the network is only composed of events (formally, $x^s = \mathcal{E}^s$ and $x^t = \mathcal{E}^t$), while in multimodal settings events are paired with images, and thus $x^s = (\mathcal{E}^s, RGB^s)$ and $x^t = (\mathcal{E}^t, RGB^t)$.

To prove the proposed approach is general, we focus on studying different domain gaps affecting events. These settings impact the difference between the two domain distributions $\mathcal{D}_s$ and $\mathcal{D}_t$ in different ways. We start by analyzing the *E-Sim-to-Real* and *RGBE-Synth-to-Real* shifts. While the first only considers differences in event generation, the second is a double shift that includes differences also in the RGB space. A simple variation of this second scenario is also considered, which we call *RGB-Synth-to-Real*, where the event generation process is shared across the two domains, and the only difference comes from a shift in the RGB space. An overview is presented in Table 5.1, while a visual representation is given in Figure 5.3. In the following, we provide a description of these shifts in greater depth.

E-SIM-TO-REAL SHIFT. With this shift, we only focus on differences in the event generation process. Simulated events in the source domain, e_{sim}, are generated from an RGB dataset using an event simulator, i.e., $\mathcal{E}^s_{sim} =$ ESIM(RGB^s_{real}), and paired with real events in the target domain recorded from the same RGB images using an actual event camera device, i.e., $\mathcal{E}^t_{real} =$

EvCamera(RGB^t_{real}). This shift is not novel in the literature and has been partially addressed by improving hyperparameter selection in simulation [127, 281]. We use this setting to investigate the ability of UDA methods to handle differences in the event generation process and how these affect different event representations.

RGBE-SYNTH-TO-REAL. This shift analyzes the combined effect of RGB rendering and event simulation. As in the previous case, the target domain consists in event streams captured with a real camera, namely $e^t_{real} =$ EvCamera(RGB^t_{real}). However, in this case, the source domain is composed of simulated events obtained from synthetic renderings, $\mathcal{E}^s_{sim} =$ ESIM(RGB^s_{synth}). This particular setting describes a double shift since the shift on event generation (ESIM $\rightarrow$ EvCamera) is combined with that on RGB images $(RGB^s_{synth} \rightarrow RGB^t_{real})$. We show that our approach can be used to tackle even this shift with no modification on the general framework.

RGB-SYNTH-TO-REAL. Finally, we consider a simplified version of the previous setting where the shift in event generation is removed by simulating events both on the source and target domains. Source events are thus defined as $\mathcal{E}^s_{sim} =$ ESIM(RGB^s_{synth}), while target ones as $\mathcal{E}^t_{sim} =$ ESIM(RGB^t_{real}). We use this simplified setting to analyze how differences in the RGB space propagate through the event generation affecting performance.

5.4.2 DA4Event: Single and Multi Modal Event-based DA

Unsupervised Domain Adaptation (UDA) methods presented in Section 5.3 on page 106 have all in common a similar underlying structure for performing adaptation. This chapter takes advantage of these similarities to build a general framework for unsupervised adaptation of events. To motivate our design choices, we first detail these common characteristics, which have also been exploited in previous works [284] with similar aims, and then show how they can be exploited to design a general adaptation pipeline.

- *Regularized task loss*: Training in UDA settings is typically realized by minimizing a loss composed of two terms. The first, $\mathcal{L}_y$, is responsible for maximizing task performance on the source domain. The second, $\mathcal{L}_{DA}$, acts as a regularizer on the primary loss and promotes adaptation by forcing feature spaces produced from the two input domains to be similar, without any label. Although the two terms are typically independent, meaning that a UDA technique may be utilized to accomplish adaptation on any task, many methods adjust the regularization specifically for the task at hand. We provide an overview of such regularization terms, i.e., UDA techniques, in Section 5.3 on page 106.

- *Network decomposition*: Domain adaptation is typically accomplished in deep neural networks by aligning feature spaces produced by the network's deep layers, as they are often more domain-specific than the initial ones [309]. To this end, the network is usually partitioned into a feature extractor $\mathcal{F}$ of arbitrary depth, followed by a task-specific network $\mathcal{G}$. Domain adaptation, i.e., the regularization terms, is applied over the final layers of the feature extractor, to promote domain-invariant features that the task predictor can use to perform inference regardless of the domain.

- *Data flow*: Apart from a few cases directly acting on the output of the task classifier, such as entropy (see Section 5.3.5 on page 109), the task predictor $\mathcal{G}$ is typically reserved to source domain data. The feature extractor $\mathcal{F}$, instead, is shared and used to extract features from both domains. These features are used to compute the domain adaptive regularization loss $\mathcal{L}_{DA}$, and, when coming from the source domain, they are paired with labels to compute the primary task loss $\mathcal{L}_y$. During backpropagation, gradient flows through $\mathcal{F}$ from both $\mathcal{L}_y$ and $\mathcal{L}_{DA}$ feedbacks to reach the dual objective of maximizing task performance and producing domain-invariant features.
- *Unpaired training*: All methods presented in Section 5.3 on page 106 perform adaptation without the need to have each sample in both source and target appearance. In other words, the alignment is not enforced by taking the same sample in both source and target renditions and then forcing feature maps obtained from the two to look the same. By contrast, these methods abstract from the specific content of the samples and focus instead on general differences between the *style* used to represent concepts in the two domains with the goal of making these differences not appear in latent spaces. Such methods are typically called *unpaired*. A random batch of source and target samples is taken during training, potentially containing different objects or classes, and then used to learn mapping these samples within the same feature space.

DA4EVENT FRAMEWORK. Techniques presented in Section 5.3 on page 106 are all designed to operate on convolutional neural networks processing dense input representations. In order to utilize them to perform adaptation on event data, we follow the same procedure used in previous chapters and convert events into a frame-like representation before processing. We then exploit the traditional design choices discussed before to design a common framework for UDA on event streams. We organize the proposed architecture into three blocks as depicted in Figure 5.4 on the next page.

Domain shifts discussed in this chapter deal with multimodal RGB+E settings in which both domains provide paired images and events, i.e., $(RGB^s, \mathcal{E}^s)$ on the source domain and $(RGB^t, \mathcal{E}^t)$ on the target domain. Images can either be captured by a real camera or obtained through rendering

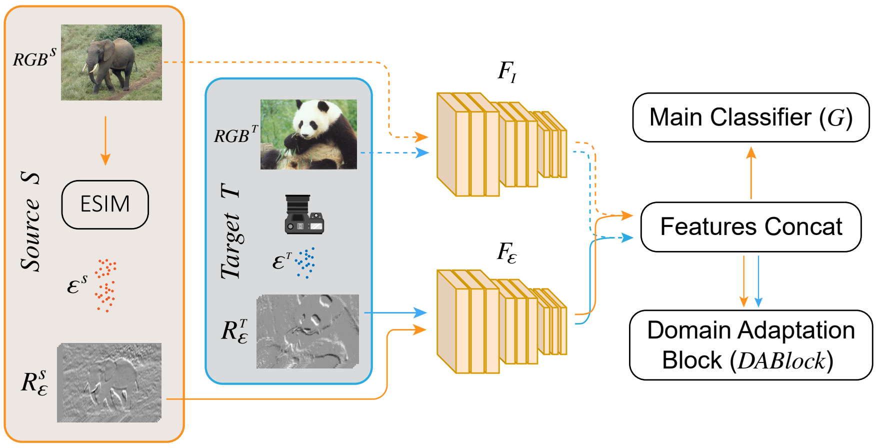


Figure 5.4: The proposed DA4E architecture. The event generation refers, in this case, to the E-Sim-to-Real shift, although the same pipeline applies unchanged even in other settings. Data coming from the source and target domains are processed separately during training. Source, labelled, data is used for supervised classification in G, while both target and source data are fed to the *DABlock*. In multimodal settings, features are extracted from each modality using different extractors F_I and F_ϵ, shared across domains, and then concatenated before prediction. The dashed data path is instead removed, along with features concatenation, when just the event modality is used.

[167], while events can be produced from a real event-based camera or with an event simulation [168] starting from any of the two image domains. We first convert event streams $\mathcal{E}^s$ and $\mathcal{E}^t$ into multi-channel event representations $\mathcal{R}^S_\mathcal{E}$ and $\mathcal{R}^T_\mathcal{E}$ using one of the multiple techniques available in the literature. Whenever possible, we make use of window-based computation by splitting the event stream into consecutive bins and extracting a representation from each of them, as this technique proved to provide better performance in previous works across different representations. As a result, we obtain representations $\mathcal{R}^{\{S,T\}}_\mathcal{E} \in \mathbb{R}^{H\times W\times F}$ with a variable number of features F depending on the representation itself and the number of bins used.

These representations are then fed into a feature extractor $\mathcal{F}_\mathcal{E}$ which, following traditional UDA methods as discussed before, is shared between the source and target domains. Features extracted from samples in the source domain are processed by the classifier $\mathcal{G}$ and by an additional *domain adaptation block* (*DABlock*) which, together with features from the target domain, performs adaptation. The *DABlock* implements all the techniques discussed in Section 5.3 on page 106. It may contain an additional classifier, as in GRL or in the case of a pretext task, or directly minimize a discrepancy between the feature distributions, as for MMD, AFN, and entropy minimization. During training, the goal of the *DABlock* is to regularize the primary loss $\mathcal{L}_y$ computed on $\mathcal{G}$'s predictions through an additional loss term $\mathcal{L}_{DA}$ that en-

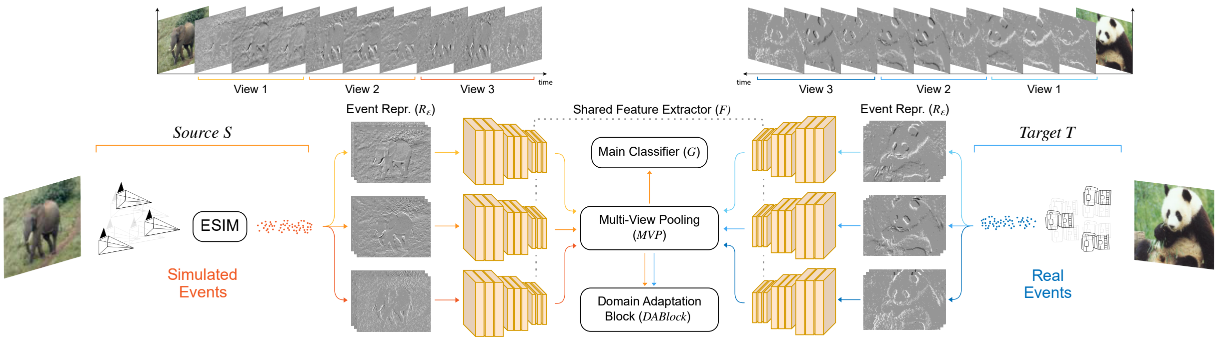

View 1
View 2
View 3
time
Source S
Target T
Event Repr. (R_ε)
Shared Feature Extractor (F)
Main Classifier (G)
Multi-View Pooling (MVP)
Domain Adaptation Block ($DABlock$)
ESIM
Simulated Events
Real Events

Figure 5.5: **Top** shows the process of extracting an event representation, taking voxel grids [55] and three views as an example, while **bottom** details the proposed multi-view architecture (MV-DA4E) under the E-Sim-to-Real shift. Two unpaired random batches from source and target domains are sampled and processed separately during training. When the multi-view approach is not used (DA4E), event representations are fed as a single multi-channel tensor to the feature extractor F, and multi-view pooling is removed. Notice that only source (labelled) data are fed to the classifier G, while both target and source data are fed to the DABlock.

courages extraction of domain-invariant features. After training, the *DABlock* is removed, and prediction is performed through $\mathcal{F}_\mathcal{E}$ and then $\mathcal{G}$, but this time on samples taken from the target domain.

This architecture can easily be extended to multimodal settings involving both RGB and events by adding an additional feature extractor $\mathcal{F}_\mathcal{I}$ processing images from both domains. As for events, $\mathcal{F}_\mathcal{I}$ is shared between domains, and its output is merged with that of $\mathcal{F}_\mathcal{E}$ with a simple channel-wise concatenation before being forwarded to $\mathcal{G}$ and the *DABlock*, under the same conditions as before. This particular variant is depicted in Figure 5.4 on the previous page.

5.4.3 MV-DA4Event: a Multi-View Approach

In the basic framework described before, we aggregate the event stream $\mathcal{E}^{\{s,t\}} = \{e_i = (x_i, y_i, t_i, p_i)\}_{i=1}^N$ describing the spatio-temporal content of the scene over a temporal period T, into a frame-based representation $\mathcal{R}_\mathcal{E}^{\{S,T\}} \in \mathbb{R}^{H\times W\times F}$. While standard RGB images encode spatial (static) information only (R, G, B channels), these frame-based representations also carry temporal information, often producing a variable number of temporal channels as the event sequence is split into several intervals (or bins) to retain temporal resolution, as in a video sequence. For instance, in saccadic motion, commonly used to gather event data from still planar images [116], these channels correspond to the camera response to different motion directions. As a consequence, each temporal channel represents a different observation of the recorded object highlighting different edges (or features) of the same.

In DA4E, we initialize the feature extractor with weights pre-trained on ImageNet. As commonly done in event-based vision when reusing frame-based CNNs, and discussed in the previous chapter, the first convolutional kernel is substituted with a new one matching the number of channels F of the input representation. This operation is typically done since $F \neq 3$ in most cases. However, while this approach enables an easy adaptation of the architecture to a particular representation and choice of bins, pre-trained feature extractors following the first convolution may be negatively affected, vanishing all the benefits of transfer learning from large data sets. This effect is particularly harmful in cross-domain settings. Indeed, we know from the literature that the first layers of the network are usually the most affected by the domain shift [309]. Training them from scratch may lead the network to specialize in low-level source domain features, thus poorly generalizing on target ones.

MV-DA4E FRAMEWORK. Motivated by these considerations, we propose to follow a *multi-view* approach to retain the first pre-trained convolutional layer. This consists in aggregating the multi-channel event representation into three-channels tensors, or *views*, obtaining representations $\widetilde{\mathcal{R}}_{\mathcal{E}} \in \mathbb{R}^{H \times W \times \lceil F/3 \rceil \times 3}$. The *MV-DA4E* architecture extends the previous DA4E approach by processing each of these three-channels views separately through the common feature extractor $\mathcal{F}_{\mathcal{E}}$. The set of features thus obtained is combined with a late-fusion approach within the multi-view pooling module *MVP*, which performs average pooling both spatially and across views on the same sample. The f_{out} feature vector thus produced is then used through the remaining parts of the network, and training takes place as in DA4E. This improved version of the framework is depicted in Figure 5.5 on the facing page, where only the single modal version is considered for clarity of presentation. Given that the very early layers of the network are more domain-specific, while the final ones contain more task-specific information, we believe that fusing different views at the final layers of the network, rather than the earliest, could allow better generalization in cross-domain settings.

5.5 N-ROD DATASET FOR SYNTHETIC-TO-REAL DOMAIN ADAPTATION

Very few works in the literature address domain shift's issues between simulated and real events [127, 281]. These works focus exclusively on the *E-Sim-to-Real* setting and use the N-Caltech101 [116] dataset for this type of analysis. N-Caltech101, combined with the simulated version proposed by Gehrig et al. [127], is excellent from this point of view since simulated events were obtained by moving a virtual camera with saccadic motion in front of RGB images, thus reproducing the same acquisition conditions used for recording events with the real camera. As a result, the same set of samples is given in both the source and target domains but acquired with different procedures. This particular condition aids in narrowing down any performance loss exclusively to the *E-Sim-to-Real* shift under consideration, excluding any drop caused by a different set of visual features in the training set.

However, simulating events from real RGB images inherits all the complexities and costs associated with standard data collection and becomes inexpensive only if the RGB dataset is already available. Indeed, collecting data with precise annotation is a hard problem even with standard vision devices. In the literature, a common solution is to use synthetic data generation, as it provides free access to precise annotations. Nevertheless, differences between synthetic training data and real test one, commonly referred to as the Synth-to-Real domain shift, severely undermine the model's performance on the actual data. Although the practice of simulating events from synthetic scenes has lately gained interest in the event-based vision field [57, 58, 310–312], the impact of simulation on the network robustness under these settings, the *RGBE-Synth-to-Real* shift, is yet to be clearly analyzed.

Designing a dataset for *RGBE-Synth-to-Real* settings is not trivial because two separate sets are needed, each of which must contain objects from the same classes but acquired under different conditions (i.e., synthetic and realistic). We propose to tackle this challenge by exploiting the RGB-D Object Dataset (ROD) [22] for object recognition. ROD contains 41,877 samples of 300 everyday items grouped in 51 categories, acquired with an RGB-D camera. Each object is acquired on a turntable at approximately 1 meter and at three different angles (30°, 45°, and 60°) above the horizon. SynROD [284], a recent extension of the dataset designed to study the Synth-to-Real domain shift in RGB+D multimodal settings, integrates the dataset by adding synthetic renderings of 3D models from the same categories as ROD in realistic lighting conditions.

These two datasets combined provide all it is needed to design a dataset for *Synth-to-Real* settings. We extend both versions of the dataset by introducing real event recordings obtained from ROD samples, as well as simulated events extracted from both ROD and SynROD images. The resulting extended dataset, N-ROD, is the first to support *RGBE-Synth-to-Real* analyses[2]. Moreover, as it inherits both real and synthetic depth data from ROD and SynROD, it also enables a comparison with the depth modality and potential uses of event data in multimodal settings. We provide examples in Figure 5.6 on the next page and a detailed description of the recording procedure in the following.

2 The dataset can be downloaded at `https://n-rod-dataset.github.io/home/`.

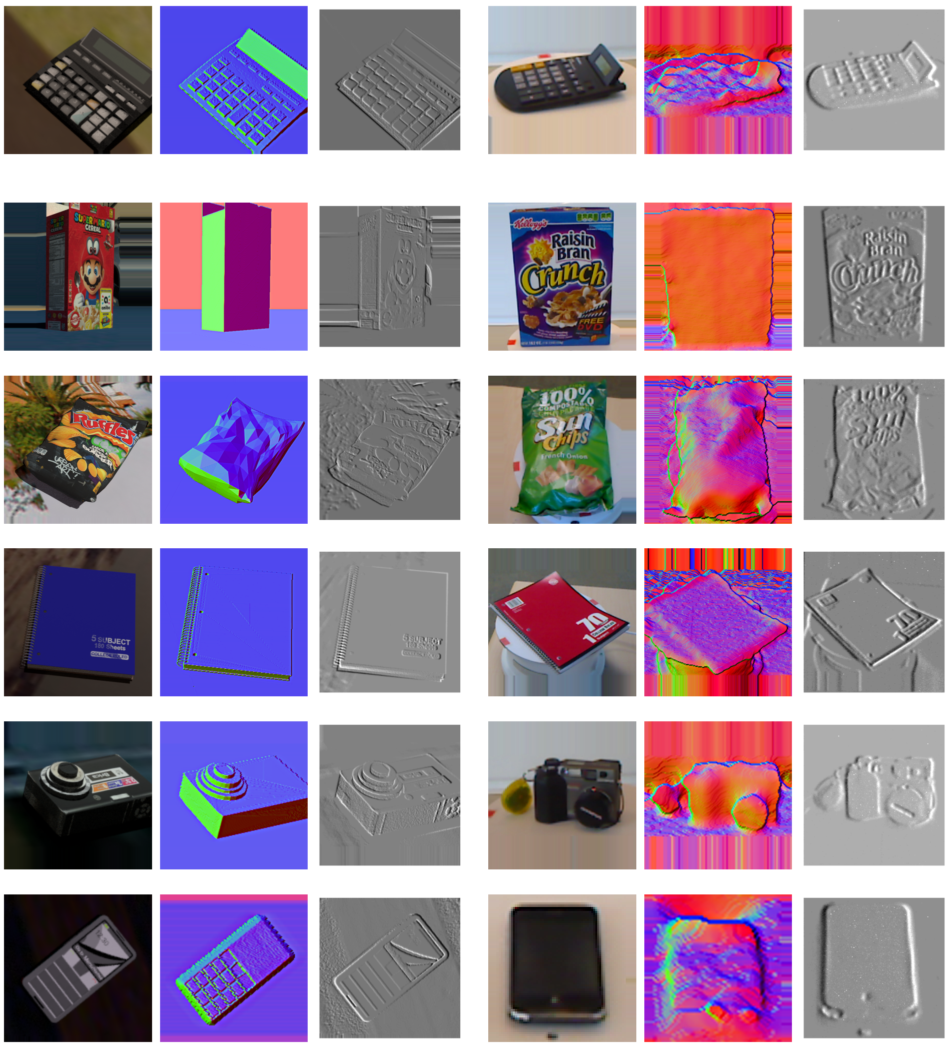


Figure 5.6: Synthetic (left) and real (right) samples from the N-ROD dataset. Depth images are colorized with surface normal encoding and event sequences are represented using voxelgrid [55].

5.5.1 Recording Setup

Objects from the ROD and SynROD datasets are provided as crops of variable size and aspect ratio. Samples are pre-processed by extending the smallest side of the image with padding that replicates the border to obtain squared images regardless of the original aspect ratio. We resize these pre-processed images to a fixed 256×256 resolution, and extract event recordings from each of them obtaining three collections: *RealROD*, *SimRealROD*, and *SimSynROD*. We combine SimSynROD and RealROD to obtain the *RGBE-Synth-to-Real* shift, where events in the source domain are obtained through simulation on synthetic images and that of the target domain with a real camera from realistic images. The *RGB-Synth-to-Real* is created instead by pairing SimSynROD and SimRealROD, obtained by performing event simulation in both source and target domains.

REAL-ROD. We replicate the setting proposed by Orchard et al. [116] for converting RGB images to event-based recordings. A Prophesee's HVGA Gen3 (CD+EM) [76] Asynchronous Time Based Image Sensor (ATIS) configured with default bias settings and mounting a Computar M0814-MP2 8mm[3] lens is placed on a pan-tilt and positioned at approximately 23 centimeters from an LCD monitor. We used a 2560×1440 76Hz IPS monitor with a 4ms minimum response time (Lenovo^{TM} ThinkVision® P27h-10), and set its brightness and contrast settings to their highest values as proposed by Hu et al. [130]. The pan-tilt[4], analogous to the one used by Orchard et al. [116], is composed of two Dynamixel MX-28 servo motors connected with each other, and an ArbotiX-M Robocontroller board controls their speed and target positions through serial communication.

We display still images from the original ROD dataset in a loop and record each sample while performing the same saccades motion pattern described by Orchard et al. [116] (i.e., three saccades motions of 100ms each in a triangular pattern). A 300ms waiting time was added after transitioning to the next image to ensure the image was correctly updated on the monitor, and the event camera has settled after detecting the visual changes incurred by changing the image. A 256×256 region of interest was set on the event camera to restrict recorded events to a squared resolution. Grayscale images from Exposure Measurement (EM) events were used, through an interactive GUI, to fit the size of displayed images to the camera's field of view before recording.

SIM+REAL-ROD AND SIM+SYN-ROD. We then use the ESIM [168] simulator to extract simulated events from both ROD and SynROD datasets obtaining the *SimRealROD* and *SimSynROD* collections respectively. We follow the same procedure used by Gehrig et al. [127] to obtain the simulated version of

3 https://computar.com/resources/files_v2/161/M0814-MP2.pdf
4 https://trossenrobotics.com/widowx-MX-28-pan-tilt

N-Caltech101. In particular, we replicate the same setting used to record real samples within a renderer [167], mapping RGB images on a plane and then moving the virtual event camera with the same saccadic motion to obtain event streams. Notice that we do not use the adaptive frame interpolation procedure proposed by Gehrig et al. [127] to upsample a low frame rate video for later conversion, but instead directly set the renderer to produce a high frame rate video. We generate 162 frames from each sample across the 300ms of recordings, equivalent to 540 fps, and then use ESIM [168] to obtain event recordings.

5.6 EXPERIMENTS

We conduct experiments over two different tasks, namely object classification and semantic segmentation. We focus primarily on object classification and use the N-Caltech101 dataset to compare with state-of-the-art approaches under the E-Sim-to-Real shift. We extend this analysis to semantic segmentation using the extension of the DAVIS Driving Dataset (DDD17) [138] proposed by Alonso and Murillo [105]. We then use the proposed N-ROD dataset to evaluate the DA4E framework under the RGBE-Synth-to-Real and RGB-Synth-to-Real settings.

We evaluate the proposed DA4E method using several event representations. Since no prior work has studied the impact of domain shifts across different representations, we decide to perform experiments on some of the most common ones, both hand-engineered and learnable. We select the HATS [85] and Voxel Grid images [55] for the first category, while the EST [92] and the Matrix-LSTM [2] representation, introduced in the previous chapter, for the second category. We refer the reader to Section 2.2.2 on page 18 and Chapter 4 on page 71 for a detailed description.

In the following, we start by providing details about the datasets used and then discuss the experimental validation we conducted.

5.6.1 Datasets

Apart from the N-ROD dataset already discussed in the previous section, we conduct experiments with both the N-Caltech101 [116] and DDD17 [105, 138] datasets. In order to evaluate the proposed approach under the E-Sim-to-Real shift, we consider simulated extensions of both datasets as detailed in the following.

N-CALTECH101. The Neuromorphic Caltech101 (N-Caltech101) [116] is an event-based conversion of the popular image dataset Caltech-101 [142] generated by recording the original RGB images using a real event-based camera moving in front of a still monitor on which still images are projected.

An extension of N-Caltech101 has recently been proposed by Gehrig et al. [127], who obtained a simulated replica of the dataset by re-creating the same setup used for recording real samples with the ESIM simulator [168]. We follow Gehrig et al. [127] and use these recordings as simulated source data and that from N-Caltech101 as target real samples. We use the train and test splits provided in the EST's official codebase [92] and evaluate the proposed approach by computing the top-1 accuracy on the test set of the target real domain.

DDD17. The DAVIS Driving Dataset (DDD17) [138], recently extended to semantic segmentation by Alonso and Murillo [105], is a set of real-world recordings captured under different environmental, weather, and speed conditions. Being recorded with a DAVIS event camera, it provides both event streams and grayscale recordings. In recent work, Gehrig et al. [127] exploit the available grayscale images to pair real events with simulated event streams obtained with their pipeline for video to event conversion. We evaluate our framework in semantic segmentation under the E-Sim-to-Real shift by following Gehrig et al. [127] and utilizing simulated samples as source data and those captured with the real camera as target data.

5.6.2 Implementation details

CLASSIFICATION. We implement the proposed method within the PyTorch [267] autodiff framework, using a ResNet34 [220] as the feature extractor $\mathcal{F}$ in N-Caltech101 experiments, and a ResNet18 [220] in ROD ones, both pre-trained on ImageNet. For a fair comparison with previous methods on the RGB+D version of ROD, we use the same network configurations of Loghmani et al. [284] for both the object recognition classifier $\mathcal{G}$ and the network used in the pretext rotation task. In particular, $\mathcal{G}$ is a two-layers MLP network with hidden size 1000, while the rotation classifier is a simple CNN composed of two convolutional layers, with 100 channels and kernel size 1 and 3 respectively, followed by a two-layers MLP. All classifiers' hidden layers use batch norm [313] and ReLU as activation functions. We use dropout [314] with 0.5 probability before the last fully connected layer and train them from scratch starting from a Xavier initialization [256].

Event representations and RGB images going through the main backbone $\mathcal{F}$ are preprocessed and augmented during training following the procedure of Loghmani et al. [284]. In particular, we first resize the smaller side of images to 256 pixels, keeping the original aspect ratio, and then randomly crop a 224×224 region of the image, followed by a random horizontal flip. We use the same resize procedure but a fixed center crop during testing. Input images are normalized with the same mean and variance used for the ImageNet pre-training, while we kept event representations un-normalized as this provided improved performance.

We use 9 bins for both voxel grids and EST representations, resulting respectively in 3 and 6 views in MV-DA4E. The number of output channels can be customized in MatrixLSTM. Therefore we set the layer to directly produce 3-channel output representations and set the number of bins to 3 as this configuration performed the best. Notice that, since HATS only provides 2 channels, without splitting the temporal frames into bins by default, we are not able to apply the proposed multi-view approach with this representation. We train all network configurations using SGD as the optimizer, batch size 32 and 64 for N-Caltech101 and ROD experiments, respectively, and weight decay 0.003. We fine-tune the DA losses' weights for each event representation and DA method, reporting the accuracy scores for the best configurations only, averaged over 3 runs with different random seeds.

SEMANTIC SEGMENTATION. We use the EV-SegNet [105] architecture as backbone in semantic segmentation experiments by re-implementing the original network in PyTorch [267] following the original Tensorflow [268] codebase [315]. EV-SegNet is an encoder-decoder architecture similar to a UNet [203], composed of an Xception [216] encoder and a lightweight decoder with skip connections interconnecting the two. The event representation used in the original EV-SegNet consists of a 6-channels 352×224 representation containing, for each polarity, the number of events received in each pixel, together with the mean and standard deviation of their timestamps. We use this representation in our experiments as well as a traditional 3-bins voxel grid for comparison. We use 50ms of events to build both representations as in the original EV-SegNet design.

We follow Alonso and Murillo [105] and train the neural network by performing augmentations during training. In particular, we randomly flip representations horizontally with a 50% probability, scale them by a factor $s \sim \mathcal{U}(0.50, 2)$, perform random rotations of $\pm 10^\circ$ and randomly translate them horizontally and vertically by $\pm 20\%$ of the original size. We train EV-SegNet for 15 epochs with Adam [257] as the optimizer, using a batch size of 8 samples and a learning rate of 0.001, decayed until $1e^{-9}$ with a polynomial scheduler of parameter 0.9. When performing adaptation with an adversarial approach, we use 0.0001 as the discriminator's learning rate and optimize the adaptation losses' weight for each representation and adaptation technique, as in classification experiments.

5.6.3 E-Sim-to-Real Analysis

This section focuses on analyzing how different UDA methods perform in reducing the *E-Sim-to-Real* shift under the proposed DA4E framework. We analyze the performance of the UDA algorithms presented in Section 5.3 on page 106 as well as their effect on different event representations. We perform most of the analysis on the classification task to compare with previous works and then test the proposed framework on semantic segmentation.

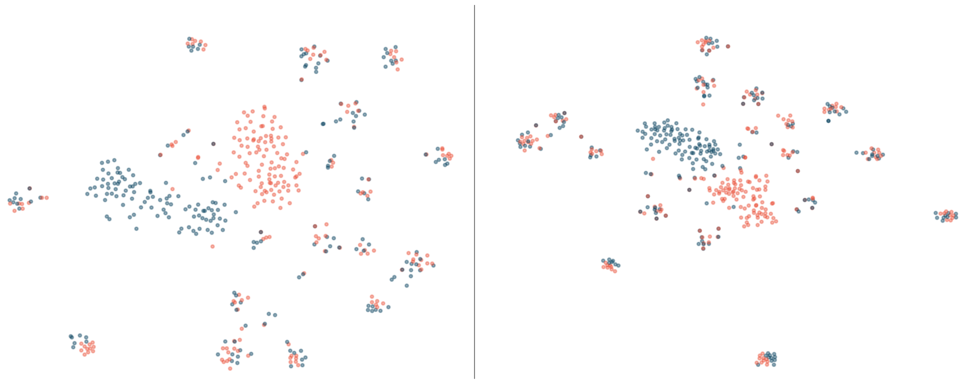
Table 5.2: Target Top-1 Test Accuracy (%) of UDA methods on N-Caltech101. Bold: representation's highest result.

N-Caltech101 $(ESIM(RGB_{real}) \implies EvCamera(RGB_{real}))$					
Method		Voxel Grid	HATS	EST	Matrix LSTM
Source Only	*baseline*	80.99	58.32	80.08	82.21
	MV-baseline	84.59	-	83.07	84.89
GRL [282]	DA4E	83.08	65.38	83.38	82.94
	MV-DA4E	86.77	-	84.03	85.75
MMD [283]	DA4E	86.37	69.86	83.61	84.04
	MV-DA4E	88.23	-	85.36	**88.05**
Rotation [290]	DA4E	79.13	61.52	80.69	83.57
	MV-DA4E	86.63	-	84.49	85.7
AFN [285]	DA4E	84.49	**69.96**	83.59	85.0
	MV-DA4E	88.3	-	85.92	87.59
Entropy [286]	DA4E	87.0	65.58	85.54	85.97
	MV-DA4E	**89.24**	-	**86.06**	86.09
Supervised	*RealEvent*	88.13	76.45	88.17	87.65
	MV-RealEvent	90.09	-	89.25	90.35

Object Classification

We assess the effectiveness of UDA algorithms in reducing the E-Sim-to-Real shift using N-Caltech101 as a benchmark. We first analyze to what extent the E-Sim-to-Real shift impacts performance by comparing the *Source Only* baseline against the *Supervised* upper bound. Source Only refers to the performance of a network trained only on simulated data (the source domain) and then tested directly on real events (the target domain) without any adaptation. Instead, the Supervised benchmark represents the performance obtained by directly training the network on the target domain with supervision. We consider this an upper bound since labels on the target domain are typically not available for training. UDA methods seek to reach this performance without supervision by only relying on improving the alignment between source and target features' distributions.

In Table 5.2, we show the performance of GRL [282], MMD [283], Rotation [290], AFN [285] and Entropy [286]. For each method, we report the results obtained with (*MV-DA4E*) and without (*DA4E*) the proposed multi-view approach. We consider the effect of UDA strategies on two non-learnable event representations (*VoxelGrid* and *HATS*), and two learnable ones (*EST*



(a) Source-only **(b)** DA4E

Figure 5.7: t-SNE visualization of N-Caltech101 [116] features from the last hidden layer of the main classifier. Red dots: source samples; blue dots: target samples. When adapting the two domains with the proposed DA4E (b), the two distributions align much better compared to the non-adapted case (a).

and *MatrixLSTM*). The empirical evaluation performed allows us to answer the following research questions.

IS UDA (DA4E) USEFUL IN REDUCING THE E-SIM-TO-REAL GAP? According to Table 5.2 on the facing page, UDA methods employed within DA4E outperform the Source Only baseline in almost all circumstances and for all event representations. They improve performance by up to 6% on VoxelGrid, 11% on HATS, 6% on EST, and 4% on MatrixLSTM. Combining voxel grids and the Rotation approach under the DA4E setting is the only instance where domain-adapted performance is below the Source Only baseline. We explain this behavior by noticing that Rotation gains most of the performance by encouraging the network to focus on the geometric structures of the objects, as this helps the network recognize orientation. As opposed to images, event streams already focus on the objects' structural and geometrical features, as they behave as edge detectors under motion. Although Rotation could help emphasize these structures even more in deep layers of the network, the event data bias towards these geometric structures could be the reason for Rotation's reduced gain. Moreover, since N-Caltech101 samples are captured with predefined motion matters, the original orientation of the sample could easily be recognized by looking at polarity patterns on the edges. This characteristic leaves the network the possibility of learning a trivial solution that may not encourage adaptation as much as it does on natural images.

Except for this only case, all UDA methods improve performance over the baselines. We provide a visual representation of how the features distribution change after adaptation in Figure 5.7. We compute samples' features from both source and target domains using the feature extractor of the source only baseline and that adapted using Entropy. Then we perform a t-Distributed

Stochastic Neighbor Embedding (t-SNE) [316] visualization of these features. When adapted, features overlap better, showing the ability of the feature extractor to compute domain-invariant features.

HOW MUCH DOES THE E-SIM-TO-REAL GAP AFFECT EVENT REPRESENTATIONS? Interestingly, we can see that not all representations suffer in the same way from the domain shift. For instance, HATS is the representation suffering the most from the E-Sim-to-Real shift, as performance decreases by up to 18% when testing directly on the target domain (Source Only) rather than on the source (Supervised). Intuitively, the reason is intrinsic in the representation itself. Indeed, when representing events with HATS the temporal resolution is lost (see Section 2.1), potentially causing a degradation in performance when testing on data belonging to a different distribution. By contrast, MatrixLSTM seems to be the representation less affected by the shift. Its performance decreases by only a 5%, compared to the 8% and 7% of EST and voxel grids, respectively, under the DA4E setting. Notice that, in these experiments, we were able to attenuate the difference between learnable and hand-engineered representations by performing a thorough hyperparameters search across all representations. Learnable representations were still easier to finetune, showing their advantage over hand-engineered ones.

IS THE PROPOSED MULTI-VIEW APPROACH (*MV-DA4E*) EFFECTIVE? Table 5.2 on page 124 shows that applying the multi-view approach *MV-DA4E* significantly improves over the *DA4E* configuration in all experiments, regardless of the representations and DA strategies used. Crucially, using a multi-view approach improves performance even in the Source Only baseline, where no adaptation is performed, showing that exploiting general low-level feature extractors is essential for better performance across domains. These results prove the validity of the proposed method, confirming the claims made in Section 5.4.3 on page 116. Interestingly, *MV-DA4E* not only provides an improvement in the cross-domain scenario but also improves the intra-domain (Supervised) one. Thus, we believe this approach could be regarded as a general way to handle event representations when pre-trained features are involved, regardless of potential domain shifts. Finally, we highlight that by employing MV-DA4E with Entropy using voxel grids as an event representation, we are able to almost completely close the E-Sim-to-Real shift on N-Caltech101, as our adapted network reduces the shift to less than 1% of performance loss against the supervised upper bound.

We show the effect of adapting featured with the MV-DA4E approach in Figure 5.8 on the facing page, where we compute the Gradient-weighted Class Activation Mapping (Grad-CAM [317]) on several N-Caltech101's target domain samples. Grad-CAMs show regions of the input event representation the network believes are the most representative of the class being predicted. When no adaptation is performed, the network often focuses on parts of the background or areas of the object that are clearly not representative of

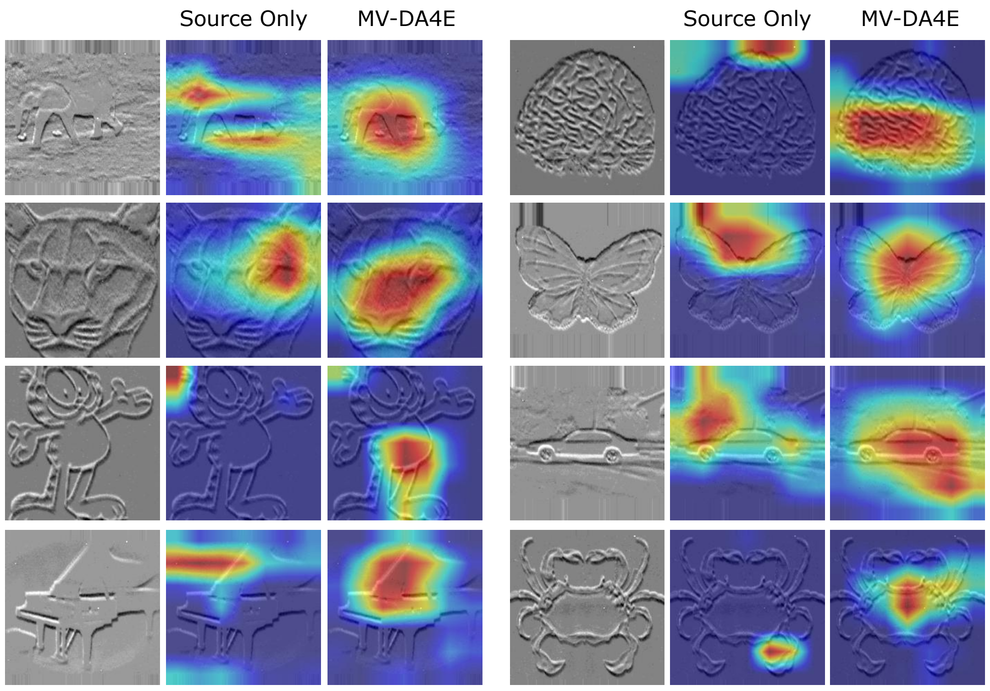


Figure 5.8: Grad-CAM [317] visualizations on several real N-Caltech101 samples. In each triplet we show the input event representations (voxel grid [55]), the activation maps when the network is trained on simulated data only, and those obtained by training with MV-DA4E.

the class. This condition significantly changes when MD-DA4E is applied. Indeed, although the network is used to classify target samples, it is still able to focus on the salient part of the event stream, suggesting its benefits in feature extraction capabilities.

HOW DOES OUR STRATEGY COMPARE TO APPROACHES ACTING ON THE CONTRAST THRESHOLD? Several methods in the literature [127, 281] address the *Sim-to-Real* problem by exclusively acting on the contrast threshold C used during event simulation. Since we operate with a fixed threshold, a possible question is whether our results merely derive from an optimal selection of C or stem from our choice to favor adaptation by working at the feature level. To answer this question, we run the DA4E frameworks with three choices of C, using voxel grid as representation and all UDA methods. We select $C = 0.06$ following the initial choice of Gehrig et al. [127], we find the optimal parameter $C = 0.15$ using the technique proposed by Stoffregen et al. [281], as well as randomly sample $C \sim \mathcal{U}(0.05, 0.5)$ using the solution of Gehrig et al. [127].

Without additional adaptation, the performance obtained training the networks with these three parameter choices constitutes the baselines. Results are reported in Table 5.3 on the next page. We run the baselines on both basic and multi-view network configurations (first two rows of the table) and

Table 5.3: Target Top-1 Test Accuracy (%) of UDA methods w.r.t. to methods that act on the contrast threshold C.

N-Caltech101 ($ESIM(RGB_{real}) \implies EvCamera(RGB_{real})$)				
Baselines		C=0.06	C=0.15 [281]	C$\sim \mathcal{U}$ [127]
Source only	*baseline*	76.81	80.99	82.29
	MV-baseline	83.12	84.59	84.93
Our approach	w/ C values:	C=0.06	C=0.15	C$\sim \mathcal{U}$
GRL [282]	DA4E	80.89	83.08	81.91
	MV-DA4E	84.93	86.77	86.45
MMD [283]	DA4E	83.84	86.37	84.38
	MV-DA4E	86.94	88.23	87.31
ROT [290]	DA4E	80.05	79.13	80.36
	MV-DA4E	86.31	86.63	87.08
AFN [285]	DA4E	84.38	84.49	84.3
	MV-DA4E	87.71	88.3	88.17
Entropy [286]	DA4E	85.26	87.0	85.16
	MV-DA4E	**88.38**	**89.24**	**88.61**

then combine these approaches with the DA4E framework. Our approach consistently and largely outperforms the baselines for every choice of C, highlighting the effectiveness of approaching DA at the feature level. Even if samples are simulated with a bad choice of contrast threshold ($C = 0.06$), our basic framework (DA4E) can still achieve an 85.26% accuracy, filling an 8.45% gap against the Source Only baseline, contrary to other approaches that only achieve a maximum accuracy of 82.29% (first row of the table). When the proposed framework is combined with previous methods (last two columns), performance is improved even further, demonstrating the complementarity of our technique with C-only based approaches. Similarly, the multi-view approach improves performance regardless of the initial choice of C, significantly reducing the network's sensitivity to C variations.

HOW MUCH TARGET DATA IS NEEDED TO PERFORM ADAPTATION? The framework proposed in this chapter still makes use of target data, captured with a real device, to perform adaptation. While this may seem a limitation, as one has to record data to train the system, the effort required is no more than that necessary to use the system in a real-world environment. Indeed, target data does not have to be labeled or share the same environments and the exact objects used during training to perform adaptation, being all methods

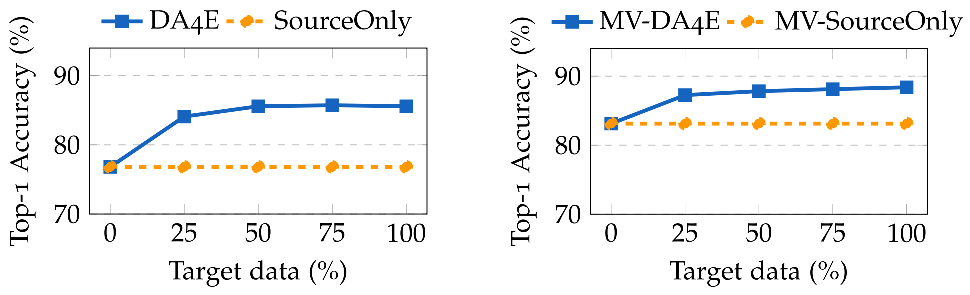


Figure 5.9: Difference in terms of performance based on the percentage (%) of target data used during training, obtained with constant threshold $C = 0.06$.

Table 5.4: Top-1 accuracy (%) of the proposed DA4E and MV-DA4E over the baseline (source only) on extended Partial DA, Domain Generalization, and $\mathcal{S} \cap \mathcal{T} = \emptyset$ on the N-Caltech101 dataset.

N-CALTECH101 $(ESIM(RGB_{real}) \implies EvCamera(RGB_{real}))$			
Method	Partial DA	Universal DA	*DG*
Baseline	79.59	68.46	76.81
MV-baseline	83.76	82.51	83.12
DA4E	84.84	72.58	79.06
MV-DA4E	**87.84**	**84.07**	**84.95**
RealEvent	87.85	85.73	88.13
MV-RealEvent	90.91	89.89	90.09

unsupervised and unpaired. Still, a legitimate question is how much target data the framework needs in order to perform adaptation successfully.

In Figure 5.9 we showcase the scalability of our approach when the access to target data is limited. We show how the performance of the proposed methods changes when only a percentage of target data is available during training (25%, 50%, 75%). It can be noticed that an improvement by up to 4% over the *Source Only* baseline (0% of training target data) is guaranteed, even when a very small percentage of target samples is available.

Moreover, we remark that the UDA framework can be further extended towards more realistic settings where the target label space is not the same as the source (as in Universal DA [318], Partial DA [319, 320] and Open Set DA [321, 322]), or when target data is not available at all during training (as in Domain Generalization [291]). A general overview of these settings is provided in [318]. To prove the proposed MV-DA4E can be easily extended to the above-mentioned settings, in Table 5.4 we provide some preliminary results showcasing how the system performs under:

- *Universal DA*: target data available during training is from a label space $\mathcal{T}$ disjoint from that of the source domain $\mathcal{S}$ ($\mathcal{S} \cap \mathcal{T} = \emptyset$). We split the set of classes in half and use, during training, simulated N-Caltech101 samples from the first set as the source domain and real ones from the second set as the target. Testing is performed using target samples from the first half instead, which were not available by the time of training. UDA is used to make features of the two domains look as if they were sampled from the same distribution, despite being from two different label spaces.

- *Partial DA*: the label space of the target domain $\mathcal{T}$ is a subset of that of the source domain $\mathcal{S}$ ($\mathcal{T} \subset \mathcal{S}$). We use the same target as in the previous setting (half of the label space) but use simulated N-Caltech101 samples from all classes as the source domain.
- *Domain generalization (DG)*: the target domain is not accessible during training. Similar to a Source Only baseline, we only use simulated samples [127] from N-Caltech101 during training and then directly test on real N-Caltech101 samples. During training, however, we adopt self-supervised adaptation methods on the source domain to encourage the extraction of general features that could potentially benefit an unseen target domain.

We use Entropy in the Partial DA setting, AFN in Universal DA, and Rotation for the domain generalization setting since pretext tasks are commonly used in this context [323]. In this last case, the pretext task is only applied to the source domain to promote the extraction of general features, as the target domain is not available. Noticeably, the proposed DA4E and MV-DA4E outperform the *Source Only* baseline in all settings, demonstrating the potential of this approach even in more complex scenarios. Results under the Universal DA setting ($\mathcal{S} \cap \mathcal{T} = \emptyset$) demonstrate that adaptation can be performed effectively even if real-world data, by the time of training, is not available for the set of classes the system must be able to predict. This is a typical circumstance since one may wish to train a system to recognize objects of an RGB dataset, converted through simulation, but real-world data captured with the specific camera the system must be deployed on is only available for a different set of classes or environments. Results on domain generalization further generalize this scenario to the case in which no real-world data is available during training. We show that using a self-supervised pretext task can be used to promote the extraction of domain-invariant representations that enable better performance even on an unknown target domain.

Semantic Segmentation

We designed the DA4E framework primarily around the object classification task. UDA methods are well-established in this field, and a variety of different methodologies have been offered over the years, allowing us to conduct a thorough analysis of these methods on event-based data. Despite our choice

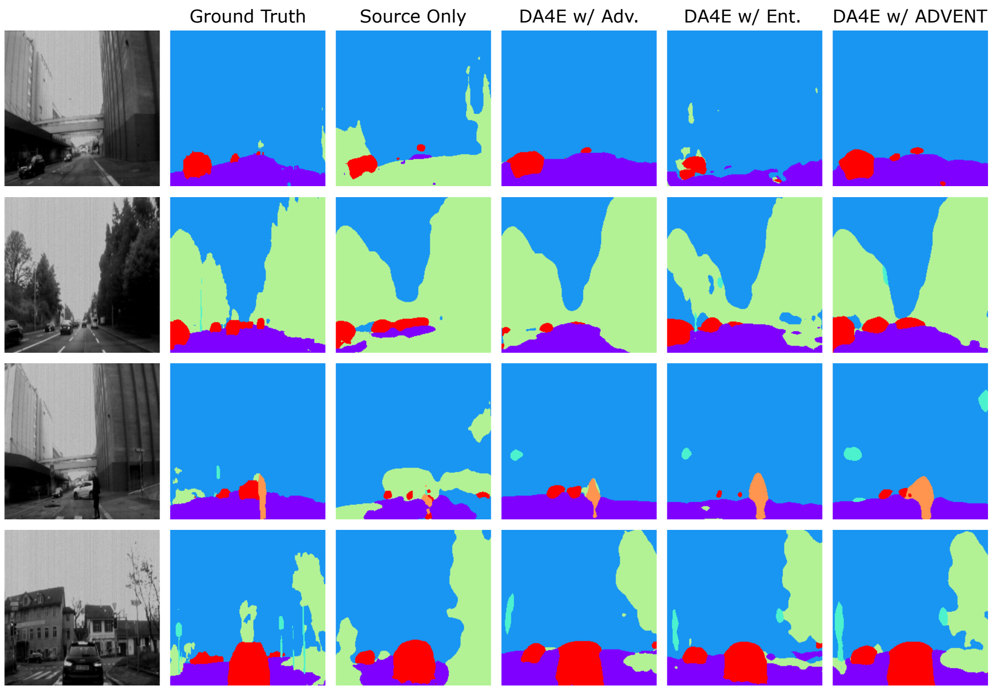


Figure 5.10: Qualitative semantic segmentation results on the DDD17 dataset. Each row displays the grayscale image of the scene, the ground truth labels, and the predictions obtained in the Source Only setting, as well as the results obtained using three different UDA techniques.

of focusing on classification, the proposed framework is still general and could potentially be extended to address even more challenging tasks. Indeed, unsupervised domain adaptation is typically achieved in more complex tasks by extending methods designed for classification while still maintaining the overall framework mentioned in Section 5.4.2 on page 113. That is the case of semantic segmentation, where multiple extensions of UDA methods have been proposed, such as the ADVENT [303] and adversarial-based [302] approaches discussed in Section 5.3.6 on page 110.

To demonstrate that the proposed approach is general, we provide preliminary results on semantic segmentation using the DAVIS Driving Dataset (DDD17) [138], with the extension proposed by Alonso and Murillo [105] serving as the target domain, and the simulated version used by Gehrig et al. [127] as the source. We follow the analysis of Gehrig et al. [127] and employ EvSegNet [105] as the semantic segmentation network and a temporal window of 500ms for prediction. We use the original EVSegNet representation, as done by Gehrig et al. [127], as well as voxel grids. We adapt DA4E to semantic segmentation by integrating, within the *DABlock*, entropy minimization [286], the approach of Tsai et al. [302], who use an adversarial discriminator on semantic segmentation logits, and the ADVENT [303] method combining the previous two.

Table 5.5: Top1 Accuracy (%) and mean Intersection over Union results on the DDD17 dataset.

DDD17 $(ESIM(RGB_{real}) \implies EvCamera(RGB_{real}))$				
	EvSegNet		VoxelGrid	
	mIOU	Top1 Acc	mIOU	Top1 Acc
Source Only	0.477	85.95	0.428	85.80
Adv. [302]	0.509	87.67	0.482	86.21
ADVENT [303]	**0.514**	**87.74**	0.490	86.72
Ent. [286]	0.498	86.95	**0.496**	**86.39**
Supervised	0.548	89.76	0.530	89.12
Vid2E [127]	0.482	86.03	-	-

As shown in Table 5.5, all the UDA methods improve over the Source Only baseline, which, we recall, consists of training on source data only (*Sim*), and then testing directly on unlabelled target data (*Real*), without performing any adaptation strategy. The proposed framework achieves improved results regardless of the event representation used, outperforming the method proposed by Gehrig et al. [127] when no finetuning on the target domain is applied. Qualitative results are provided in Figure 5.10 on the preceding page, where we compare predictions obtained by only training on simulated data against that obtained after adaptation. Unsupervised domain adaptation improves the network's prediction of foreground objects (i.e., cars, pedestrians, and street signs) and background areas, which are often misclassified when no adaptation is used (*Source Only*).

5.6.4 RGBE and RGB Synth-to-Real Analysis

This section focuses on studying the RGBE-Synth-to-Real shift under the proposed DA4E framework. We make use of the N-ROD dataset proposed in Section 5.5 on page 117, which is to date the only classification dataset for event-based cameras enabling this type of analysis. We use the DA4E framework with the same settings as in the previous experiments but only focus on the voxel grid representation, as it provided a good compromise between accuracy and training time as well as the best performance when combined with UDA methods on N-Caltech101. In the following, we focus first on a single modal evaluation and then study how multimodal (RGB+E) data can be used to improve performance on the RGBE-Synth-to-Real shift. We then analyze how the Synth-to-Real shift on RGBs impacts event-based neural networks' performance with the simplified RGB-Synth-to-Real setting. In this case, the shift only happens between image domains, as the event generation process is the same (i.e., simulated) for both the shifts. The results we obtained are reported in Table 5.7 on page 135.

Table 5.6: Top-1 accuracy (%) of UDA methods on the RGBE-Synth-to-Real shift. **Bold:** highest mean result, underline: highest single- and multi-modal results. ▲ indicates the improvement of the avg of UDA methods over the baseline Source Only.

N-ROD $\left(ESIM(RGB_{synth}) \implies EvCamera(RGB_{real})\right)$					
	Single-modal			Multi-modal	
Method	RGB	Depth	**Event**	RGB+D	**RGB+E**
Source Only	52.13	7.56	21.78	47.70	50.78
GRL [282]	57.12	26.11	33.09	59.51	57.15
MMD [283]	63.68	29.34	42.05	62.57	61.78
Rot [290][284]	63.21	6.70	31.26	66.68	68.54
AFN [285]	64.63	30.72	55.12	62.40	64.04
Entropy [286]	61.53	16.79	50.14	63.12	64.08
Avg	62.03 ▲+9.9	21.93 ▲+14.4	42.33 ▲+20.6	62.86 ▲+15.2	**63.12** ▲+12.3

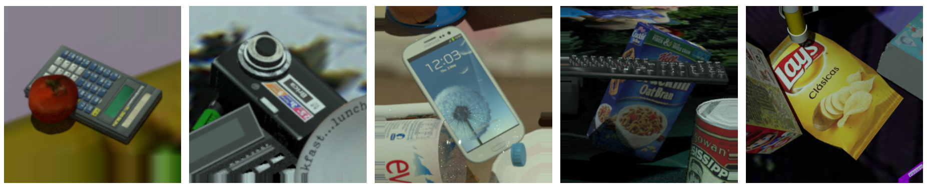


Figure 5.11: Sample images from SynROD [284] showing occlusions and multi object scenes. The label indicates the class associated to each sample.

IS THE SHIFT IMPACTING MODALITIES EQUALLY? We begin by comparing the event modality's performance against that of other modalities accessible in N-ROD, namely RGB and depth data. Contrary to the analysis performed on N-Caltech101, here we cannot assess the domain shift directly, since, following the setup of Loghmani et al. [284], the target domain does not provide training data to evaluate the performance of a supervised baseline. Nevertheless, we can see from Table 5.7 on page 135 that the event modality performs worse than RGBs, but better than the depth. The ROD setting, in fact, is particularly challenging for modalities other than RGB. In contrast to N-Caltech101 and DDD17, the environment and the objects present in the two domains are distinct. The synthetic set contains object occlusions, random background, and object instances that may not be present in the target domain, even if from the same category. Therefore, networks employing RGB data may better exploit the ImageNet pretraining, giving them an advantage over other modalities. Nonetheless, the event modality still outperforms the depth, which is frequently utilized as an alternative, or complementary, modality in cross-domain scenarios. Few selected samples showing these challenging conditions are provided in Figure 5.11 on the preceding page.

ARE UDA METHODS EFFECTIVE ON EVENTS? We follow the analysis of Loghmani et al. [284] of depth data adaptation and apply the proposed DA4E framework to adapt event data using all classification techniques discussed in Section 5.3 on page 106, as done for N-Caltech101. Results show that, among all modalities, the event modality is the one receiving the highest benefits from UDA methods. Indeed, performance improves, on average, on a 20.6% against the Source Only baseline, while RGB and depth only record an increment of 9.9% and 14.4%, respectively. The event modality, triggered by egocentric motion, focuses more on geometric components and object shapes, contrary to RGB data which is biased towards texture [324]. These intrinsic peculiarities make UDA techniques more effective on events, as shape information is per se more robust [324] in the transition from the synthetic to real domains and thus easier to be aligned than the information encoded in RGB. Moreover, despite the events' disadvantages compared to RGBs discussed before, they still reach remarkable results thanks to UDA techniques. Most notably, events show a consistent improvement when using all UDA techniques, in contrast to the depth modality, which not always benefits from them. This result proves that events can be a good alternative to RGB in contexts where the latter is limited.

HOW DO EVENTS COMBINE WITH OTHER MODALITIES? It is well known that the complementarity of different input modalities, such as RGB and depth, can be exploited to improve adaptation performance in cross-domain scenarios [284]. Since multimodal RGB-E adaptation is still unexplored in the literature, we propose a first approach for combining the two modalities by relying on methodologies commonly used for RGB-D data [325] (see Section 5.4.2 on page 113). We use the same adaptation methods as for the single modal analysis, making sure to use multimodal variants whenever available. In particular, we use Relative Rotation [284] instead of the simpler variant [290] in these experiments. The results in Table 5.7 on the next page show that the proposed DA4E approach performs well even in multimodal setting, as all methods consistently improve over the Source Only baseline, as in the single modal setting. The RGB-E approach performs better than RGB-D on average, reaching a maximum performance of 68.54% against the 66.68% of the RGB-D network and improving the RGB baseline by 4%.

Rotation [290] provides interesting results to be discussed. Indeed, when absolute rotation is applied to each modality individually, this method is the one achieving the lower performance gain if compared to all the others. Instead, when extended to the RGB-E context by applying the *Relative Rotation* [284] between the two modalities, it interestingly reveals to be the UDA method performing the best. This brings to light the importance of leveraging

Table 5.7: Top-1 accuracy (%) on events, in two different scenarios: *RGBE-Synth-to-Real* and *RGB-Synth-to-Real*. In **bold** the highest mean result.

N-ROD $(ESIM(RGB_{synth}) \implies (ESIM \text{ vs. } EvCamera)(RGB_{real}))$								
Source	Target	Source Only	GRL	MMD	Rot	AFN	Entropy	Avg
ESim	ESim	40.47	44.52	48.29	42.98	53.50	49.29	**47.68**
ESim	EvCamera	21.78	33.09	42.05	31.26	55.12	50.14	42.33

over the complementarity of multiple modalities, even in the event field. We believe this result emphasizes the need to further push research towards networks and adaptation strategies specifically designed to make the two modalities efficiently cooperate.

HOW MUCH IS THE RGB SHIFT IMPACTING EVENTS? Under the RGBE-Synth-to-Real shift, using simulation on the source domain and a real event camera on the target one indirectly introduces the E-Sim-to-Real shift over the pre-existing shift between synthetic and realistic RGB images. In order to understand how combining these two shifts affects performance, we compare RGBE-Synth-to-Real results with those obtained under the simplified RGB-Synth-to-Real shift, where the shift on events is removed by performing simulation on both domains. We can consider this experiment as a relaxed setting where the E-Sim-to-Real gap is not present. To this purpose, Table 5.7 compares single-modal results obtained under these two settings. Without any kind of adaptation (Source Only), performance decreases by 18.69% when passing from the RGB-Synth-to-Real to the RGBE-Synth-to-Real shift, which quantifies the E-Sim-to-Real gap. The proposed approach reveals once again the effectiveness of UDA techniques in the event context, which consistently improve performance, reducing this gap to only a 5%. This emphasizes the importance of developing broad approaches for event adaptation capable of reducing domain shifts regardless of the cause of the gap, as in the suggested approach.

5.7 CONCLUSIONS

In this chapter, we suggest an alternative solution to the recent research problem of bridging the Sim and Synth to Real gaps for event cameras. We propose to tackle this issue as a domain shift and demonstrate that addressing the problem at the feature level is more effective than operating on simulation hyperparameters. To this end, we propose DA4E, a comprehensive framework for even-based adaptation that integrates well-known UDA methods with popular event-based representations. We benchmark the framework on both classification and semantic segmentation tasks, using the N-Caltech101

[116] and DDD17 [105, 138] datasets and achieving better results than other approaches acting on simulation only. We build on these results and propose MV-DA4E, a simple yet effective variation of the framework that enhances resilience against domain shifts by improving the transferability of ImageNet pre-trained feature extractors. Finally, as synthetic to real domain shifts have not been extensively studied in event-based vision, we propose N-ROD as a novel dataset for this investigation and demonstrate that UDA approaches are once again successful in addressing these shifts. We believe that this work is a valuable starting point for the community, and we hope it could ignite further research towards studying novel approaches for event-based adaptation and multimodal event integration.

6 CONCLUSIONS AND FUTURE WORKS

In this thesis, we focused on deep learning approaches for event-based vision. We discussed the importance of event representations, both at the input and latent layers of a network, and showed that by concentrating on these aspects, significant advances toward more effective and efficient processing are possible.

Asynchronous Convolutional Neural Networks

In Chapter 3, we focused on efficiency during computation and proposed a novel approach for computing hidden layers' representations incrementally, exploiting the differential nature of event streams. Traditional deep learning systems for event-based vision trade off efficiency for accuracy, integrating events into frame-like representations and then utilizing conventional frame-based CNNs for synchronous computation. Although this strategy has allowed the event-based vision field to develop in a number of visual tasks, these results are still purely theoretical in the sense that they can only be achieved at far lower rates than an event camera may theoretically enable. Our technique constitutes a first step towards unlocking state-of-the-art performance even at low latencies, effectively exploiting event-based cameras' benefits without sacrificing accuracy.

Aiming at this result, we took inspiration from Spiking Neural Networks (SNNs) and proposed a mechanism to convert a deep neural network trained on frame-like event representations into a dynamical system capable of producing the same predictions but through incremental and asynchronous computation. We achieved this by enhancing traditional convolution and max-pooling layers with an internal memory storing the previous layer's output. Asynchronous computation is attained through a set of update rules that allow the network's state to be moved forward in time, asynchronously, and the layer's representations to be sparsely updated based on incoming events. Since these layers produce identical predictions, they can substitute conventional ones without any adaptation or additional training steps, thus enabling traditional approaches to enjoy asynchronous computation with the same accuracy.

We showed promising results when comparing our event-based layers against traditional ones under equally optimized code on the CPU. Our work inspired Messikommer et al. [87], who improved event-based layers by departing from the SNNs inspiration and employing recurrent event representations and submanifold sparse convolutions [263] to increase sparse computation and efficiency even further. However, as for our solution, these layers still perform slower than a traditional neural network when implemented with highly parallelized deep learning frameworks or when leveraging GPU computation. An exciting area of research is to investigate the potential of these layers when implemented on dedicated hardware, such as FPGAs, capable of exploiting the asynchronous and efficient processing promised by this approach. Toolchains able to automatically convert neural networks built using popular deep learning frameworks into optimized implementations compatible with hardware accelerators are already available in the literature [326–330]. A further step towards making these layers more accessible is to develop a similar solution for automated deployment of deep neural networks leveraging event-based layers onto optimized hardware designs, potentially unlocking the use of such neural networks in real-world applications.

Our work focused explicitly on optimizing convolutional and max-pooling layers, as they are the most fundamental yet widely used operators in deep architectures. However, the adoption of more advanced layers, such as ConvLSTMs [192] or other recurrent variants [20, 331], is steadily increasing in event-based networks [57, 58, 104, 108] to take advantage of their temporal modeling. These are related to our event-based layers in that they also incorporate a memory, despite not enabling sparse computation. Event-based versions of these units might broaden our framework's applicability to more advanced networks and potentially enable hybrid designs where memory accumulation is learned, such as that of Gehrig et al. [58] or our MatrixLSTM.

Learning Representations for Event-based Neural Networks

In Chapter 4, we focuses on events' encoding and proposed a novel fully-differentiable representation for event-based cameras. We aimed at improving the performance of existing event-based neural networks with a flexible solution compatible with most deep learning architectures. Event-based neural networks still lag behind conventional image-based systems when only task performance is considered, especially in tasks that largely rely on appearance. This gap is not caused by limits of the sensor, as it even conveys more visual information than a traditional device, but rather by the processing algorithms, which are often not optimized to fully exploit event data. Indeed, the majority of event-based networks still rely on hand-engineered event representations that may not encode all the information necessary for effectively solving a task. While computing efficiency is frequently the major focus in event processing, event-based systems must nevertheless strive for state-of-the-art performance to prove their competitiveness against traditional cameras in real-world applications.

With the goal of maximizing performance, we proposed to learn how to effectively interface events with a CNN system by convolving a Long Short-Term Memory (LSTM) cell over the stream of events. We exploited the memory mechanism of LSTM networks to incrementally and sparsely convert event streams into a 3D representation that can directly be used as the input of state-of-the-art frame-based architectures. We demonstrated improved performance in both high-level and low-level tasks, regardless of the architecture used for prediction. Exploiting the LSTM cells' natural ability to handle sequential data and ad-hoc optimized operations, the proposed system can compute event representations efficiently, almost matching the efficiency of simpler, hand-engineered solutions.

Despite its ability to learn from long-term event streams, aided by its windowing mechanism and by the LSTM operating principles, training MatrixLSTM could still become impractical when dealing with very long sequences. Indeed, since training is performed end-to-end, all event features must be kept in GPU memory together with the rest of the network to allow for effective backpropagation. This practically restricts the length of sequences used during training, potentially preventing complex dynamics from being learned. Annamalai et al. [332] partially solves this issue by training MatrixLSTM as a recurrent autoencoder, thus removing the need to learn from a downstream prediction network and enabling the extraction of task-independent representations. Along these lines, an exciting research direction is that of exploiting the similarities between event-based and Natural Language Processing (NLP), in their sequential nature, for the design of event representations that could better capture both short-term and long-term temporal relations. An option could be using multiscale recurrent neural layers [333–336] to model temporal dynamics at different time-scales, or taking inspiration from language models [337, 338] by learning to predict the future event dynamics to improve event understanding. Taking inspiration from this field, we are now exploring the use of Transformer [187] based attention mechanisms to model event sequences. These have indeed demonstrated outstanding performance even on data modalities different from text, as we also proved with our work on skeleton-based action recognition [5, 6].

Learning Domain-invariant Networks with Event-based Simulation

In Chapter 5, we focused again on event representations, both at the input and hidden layers, but this time we moved away from their design and focused instead on studying them from a learning perspective. Learning-based methods have played a significant role in establishing the efficacy of event-based cameras in a variety of challenging conditions where standard cameras typically struggle. However, this level of robustness can only be achieved if a sufficient amount of annotated data is accessible, hampering event-based cameras' potential in many applications. As a result, event-based vision is increasingly relying on event simulation to enable the use of event-based devices even in tasks where available data is still scarce. Nonetheless, non-idealities in event simulation often cause performance degradation when transitioning from simulated to real-world conditions. Given the growing interest in using event-based systems in several fields of robotics, studying the effect of these inconsistencies, as well as designing effective solutions to mitigate them, is becoming increasingly important in event-based vision.

We proposed to study these aspects with the tools of Unsupervised Domain Adaptation (UDA), reducing all these issues to the analysis of two specific domain shifts. We looked specifically at performance losses caused by training on simulated events when event simulation is used to convert existing image-based datasets as well as synthetic scenes acquired through rendering. We proposed to tackle these shifts using UDA methods. We designed DA4E, a general framework for event-based adaptation unifying several well-known techniques, and demonstrated its flexibility by reducing both shifts on several benchmarks as well as a novel dataset.

Our analysis shows that UDA methods are a valuable tool for tackling domain shifts induced by simulation and prove that methods designed for image-based networks can be applied to event representations indistinguishably, without any modification. Despite the encouraging results and the novel dataset released, however, our analysis is limited in studying simplified settings where motion patterns are shared between the source and target domains and where HDR and motion blur conditions are rare. Since simulation is often used to convert existing image-based datasets, where these conditions may exist, a future research direction is that of studying how simulation in the presence of these artifacts affects prediction performance. Along this line, we are currently studying [8] the use of simulated events in the context of first-person action recognition with the proposed E^2(GO)MOTION suite and a novel extension of the EPIC-Kitchens dataset [339]. We show that event data can help reduce image-based shifts, even if completely simulated, as it focuses on motion, which is typically more robust than appearance, and it inherently highlights shapes and structures of the objects over the texture, which is typically the primary cause of shifts in the image domain [324]. We plan to supplement the dataset with real-world recordings to broaden the analysis of the simulation's impact and include the challenging setting of first-person recordings, where fast camera motion typically affects traditional cameras and could, in turn, impact event simulation as well.

Finally, a future line of research could also be that of designing novel adaptation mechanisms specifically designed for events. Following the work of Loghmani et al. [284] on depth adaptation, one solution could be to design multimodal pretext tasks leveraging the relation between event streams and videos, for instance, through motion [202, 264] similarly to what has been proposed by Messikommer et al. [190]. Another possibility could be to focus on events alone and exploit intrinsic properties of event streams, e.g., the relation between polarity and motion direction through time, and design transformations for unsupervised tasks as typically done with images [290]. Moreover, while we focused specifically on representation-based networks, future efforts could be devoted to studying GCNN [102, 103] and PointNet [90, 188] based architectures which, by relying on raw event features, could be more affected by these shifts.

BIBLIOGRAPHY

[1] Marco Cannici et al. "Asynchronous Convolutional Networks for Object Detection in Neuromorphic Cameras." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2019. DOI: 10.1109/cvprw.2019.00209 (cit. on pp. v, 4, 16, 41, 74).

[2] Marco Cannici et al. "A Differentiable Recurrent Surface for Asynchronous Event-Based Data." In: *Computer Vision – ECCV 2020*. Springer International Publishing, 2020, pp. 136–152. DOI: 10.1007/978-3-030-58565-5_9 (cit. on pp. v, 5, 17, 18, 35, 71, 121).

[3] Mirco Planamente et al. "DA4Event: Towards Bridging the Sim-to-Real Gap for Event Cameras Using Domain Adaptation." In: *IEEE Robotics and Automation Letters* 6.4 (2021), pp. 6616–6623. DOI: 10.1109/lra.2021.3093870 (cit. on pp. v, 5, 72, 101).

[4] Marco Cannici et al. "N-ROD: a Neuromorphic Dataset for Synthetic-to-Real Domain Adaptation." In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2021. DOI: 10.1109/cvprw53098.2021.00148 (cit. on pp. v, 5, 27, 30, 34, 72, 101).

[5] Chiara Plizzari et al. "Spatial Temporal Transformer Network for Skeleton-Based Action Recognition." In: *Pattern Recognition. ICPR International Workshops and Challenges*. Cham: Springer International Publishing, 2021, pp. 694–701. DOI: 10.1007/978-3-030-68796-0_50 (cit. on pp. vi, 139).

[6] Chiara Plizzari et al. "Skeleton-based action recognition via spatial and temporal transformer networks." In: *Computer Vision and Image Understanding* 208-209 (July 2021), p. 103219. DOI: 10.1016/j.cviu.2021.103219 (cit. on pp. vi, 139).

[7] Alberto Archetti et al. "Neural Weighted A*: Learning Graph Costs and Heuristics with Differentiable Anytime A*." In: *Machine Learning, Optimization, and Data Science*. Cham: Springer International Publishing, 2022, pp. 596–610. DOI: 10.1007/978-3-030-95467-3_43 (cit. on p. vi).

[8] Chiara Plizzari et al. "E^2(GO)MOTION: Motion Augmented Event Stream for Egocentric Action Recognition." In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2022 (cit. on pp. vi, 140).

[9] Bhavin R. Sheth and Ryan Young. "Two Visual Pathways in Primates Based on Sampling of Space: Exploitation and Exploration of Visual Information." In: *Frontiers in Integrative Neuroscience* 10 (Nov. 2016). DOI: 10.3389/fnint.2016.00037 (cit. on p. 1).

[10] D Van Essen. "Organization of visual areas in macaque and human cerebral cortex." In: *The Visual Neurosciences*. Ed. by L. M. Chalupa and J. S. Werner. Cambridge, MA: MIT Press, 2004, pp. 507–521 (cit. on p. 1).

[11] D Hubel and T Wiesel. "Receptive fields of single neurones in the cat's striate cortex." In: *J Physiol* 148 (1959), pp. 574–591. DOI: 10.1113/jphysiol.1959.sp006308pmid:14403679 (cit. on p. 1).

[12] D Hubel. "Single unit activity in striate cortex of unrestrained cats." In: *J Physiol* 147 (1959), pp. 226–238. DOI: 10.1113/jphysiol.1959.sp006238pmid:14403678 (cit. on p. 1).

[13] D Hubel and T Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." In: *J Physiol* 160 (1962), pp. 106–154. DOI: 10.1113/jphysiol.1962.sp006837pmid:14449617 (cit. on p. 1).

[14] Maximilian Riesenhuber and Tomaso Poggio. "Hierarchical models of object recognition in cortex." In: *Nature Neuroscience* 2.11 (Nov. 1999), pp. 1019–1025. DOI: 10.1038/14819 (cit. on p. 1).

[15] Thomas Serre et al. "A quantitative theory of immediate visual recognition." In: *Progress in Brain Research*. Vol. 165. Elsevier, 2007, pp. 33–56. DOI: 10.1016/s0079-6123(06)65004-8 (cit. on p. 1).

[16] Cameron D. Danesh et al. "Synaptic Resistors for Concurrent Inference and Learning with High Energy Efficiency." In: *Advanced Materials* 31.18 (Mar. 2019), p. 1808032. DOI: 10.1002/adma.201808032 (cit. on p. 1).

[17] *TOP500 Supercomputers Ranking - November 2021*. [Online; Accessed Feb. 2022]. Feb. 2022. URL: https://www.top500.org/lists/top500/2021/11 (cit. on p. 1).

[18] William B Levy and Victoria G. Calvert. "Computation in the human cerebral cortex uses less than 0.2 watts yet this great expense is optimal when considering communication costs." In: *BioRxiv* (Apr. 2020). DOI: 10.1101/2020.04.23.057927 (cit. on p. 1).

[19] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016, pp. 779–788. DOI: 10.1109/cvpr.2016.91 (cit. on pp. 4, 42, 43, 53, 54, 56, 61, 62, 91).

[20] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cit. on pp. 4, 73, 75, 77, 138).

[21] Patrick Lichtsteiner et al. “A 128 $\times$ 128 120 dB 15μs Latency Asynchronous Temporal Contrast Vision Sensor.” In: *IEEE journal of solid-state circuits* 43.2 (2008), pp. 566–576 (cit. on pp. 5, 10, 11, 27).

[22] Kevin Lai et al. “A large-scale hierarchical multi-view RGB-D object dataset.” In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, May 2011, pp. 1817–1824. DOI: 10.1109/icra.2011.5980382 (cit. on pp. 5, 101, 104, 118).

[23] Christoph Posch et al. “Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output.” In: *Proceedings of the IEEE* 102.10 (Oct. 2014), pp. 1470–1484. DOI: 10.1109/jproc.2014.2346153 (cit. on pp. 7–13).

[24] Tobi Delbruck. “Neuromorophic vision sensing and processing.” In: *2016 46th European Solid-State Device Research Conference (ESSDERC)*. IEEE, Sept. 2016, pp. 7–14. DOI: 10.1109/essderc.2016.7599576 (cit. on p. 7).

[25] Guillermo Gallego et al. “Event-based Vision: A Survey.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1. DOI: 10.1109/tpami.2020.3008413 (cit. on pp. 7, 10, 13, 15, 27, 31, 41, 102).

[26] K.A. Boahen. “Point-to-point connectivity between neuromorphic chips using address events.” In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 47.5 (May 2000), pp. 416–434. DOI: 10.1109/82.842110 (cit. on p. 8).

[27] E. Echeverri. “Limits of Capacity for the Exchange of Information in the Human Nervous System.” In: *IEEE Transactions on Information Technology in Biomedicine* 10.4 (2006), pp. 803–808. DOI: 10.1109/titb.2006.879585 (cit. on p. 8).

[28] Christine A. Curcio et al. “Human photoreceptor topography.” In: *The Journal of Comparative Neurology* 292.4 (1990), pp. 497–523. DOI: 10.1002/cne.902920402 (cit. on p. 8).

[29] Vincent Chan et al. “AER EAR: A Matched Silicon Cochlea Pair With Address Event Representation Interface.” In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 54.1 (Jan. 2007), pp. 48–59. DOI: 10.1109/tcsi.2006.887979 (cit. on p. 9).

[30] Minhao Yang et al. “A 0.5 V 55 μW 64 $\times$ 2 Channel Binaural Silicon Cochlea for Event-Driven Stereo-Audio Sensing.” In: *IEEE Journal of Solid-State Circuits* 51.11 (Nov. 2016), pp. 2554–2569. DOI: 10.1109/jssc.2016.2604285 (cit. on p. 9).

[31] Shih-Chii Liu et al. “Asynchronous Binaural Spatial Audition Sensor With 2x64x4 Channel Output.” In: *IEEE Transactions on Biomedical Circuits and Systems* 8.4 (Aug. 2014), pp. 453–464. DOI: 10.1109/tbcas.2013.2281834 (cit. on p. 9).

[32] T.C. Pearce et al. "Silicon-based Neuromorphic Implementation of the Olfactory Pathway." In: *Conference Proceedings. 2nd International IEEE EMBS Conference on Neural Engineering, 2005.* IEEE, 2005. DOI: 10.1109/cne.2005.1419619 (cit. on p. 9).

[33] Thomas Jacob Koickal et al. "Analog VLSI Circuit Implementation of an Adaptive Neuromorphic Olfaction Chip." In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 54.1 (Jan. 2007), pp. 60–73. DOI: 10.1109/tcsi.2006.888677 (cit. on p. 9).

[34] T Finateu et al. "A 1280x720 backilluminated stacked temporal contrast event-based vision sensor with 4.86m pixels, 1.066geps readout, programmable event-rate controller and compressive data-formatting pipeline." In: *State Circuits Conf. (ISSCC)*, p. 2020 (cit. on pp. 9, 11, 27).

[35] Y Suh et al. "A 1280x960 Dynamic Vision Sensor with a 4.95-m pixel pitch and motion artifact minimization." In: *IEEE Int. Symp. Circuits Syst* (), p. 2020 (cit. on pp. 9, 11).

[36] Massimo Antonio Sivilotti. "Wiring considerations in analog VLSI systems, with application to field-programmable networks." en. PhD thesis. USA, 1991. DOI: 10.7907/STJ4-KH72 (cit. on p. 9).

[37] Misha Mahowald. "VLSI analogs of neuronal visual processing: a synthesis of form and function." en. PhD thesis. Pasadena, 1992. DOI: 10.7907/4BDW-FG34 (cit. on p. 9).

[38] Misha A. Mahowald and Carver Mead. "The Silicon Retina." In: *Scientific American* 264.5 (May 1991), pp. 76–82. DOI: 10.1038/scientificamerican0591-76 (cit. on p. 10).

[39] Kareem A Zaghloul and Kwabena Boahen. "A silicon retina that reproduces signals in the optic nerve." In: *Journal of Neural Engineering* 3.4 (Sept. 2006), pp. 257–267. DOI: 10.1088/1741-2560/3/4/002 (cit. on p. 10).

[40] Kwabena Boahen. "Neuromorphic Microchips." In: *Scientific American* 292.5 (2005), pp. 56–63. ISSN: 00368733, 19467087 (cit. on p. 10).

[41] Gregory Kevin Cohen. "Event-Based Feature Detection, Recognition and Classification." Theses. Université Pierre et Marie Curie - Paris VI, 2016. URL: https://tel.archives-ouvertes.fr/tel-01426001/ (cit. on pp. 10, 18–20, 22, 46, 47, 74).

[42] Tobi Delbruck et al. "Activity-driven, event-based vision sensors." In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, May 2010, pp. 2426–2429. DOI: 10.1109/iscas.2010.5537149 (cit. on p. 10).

[43] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. "A 128x128 1.5% Contrast Sensitivity 0.9% FPN 3 ¯s Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers." In: *IEEE Journal of Solid-State Circuits* 48.3 (Mar. 2013), pp. 827–838. ISSN: 0018-9200. DOI: 10.1109/jssc.2012.2230553 (cit. on pp. 10, 27).

[44] Biyin Wang et al. "An In-Pixel Gain Amplifier Based Event-Driven Physical Unclonable Function for CMOS Dynamic Vision Sensors." In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 2019. DOI: 10.1109/iscas.2019.8702472 (cit. on p. 11).

[45] Raphael Berner and Tobi Delbruck. "Event-based color change pixel in standard CMOS." In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE. IEEE, May 2010, pp. 349–352. DOI: 10.1109/iscas.2010.5537787 (cit. on p. 11).

[46] Raphael Berner and Tobi Delbruck. "Event-Based Pixel Sensitive to Changes of Color and Brightness." In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 58.7 (2011), pp. 1581–1590. DOI: 10.1109/TCSI.2011.2157770 (cit. on pp. 11, 34).

[47] Christoph Posch et al. "A QVGA 143 dB Dynamic Range Frame-Free PWM Image Sensor With Lossless Pixel-Level Video Compression and Time-Domain CDS." In: *IEEE Journal of Solid-State Circuits* 46.1 (Jan. 2011), pp. 259–275. ISSN: 0018-9200. DOI: 10.1109/jssc.2010.2085952 (cit. on pp. 11, 12, 27, 91).

[48] Ali Yoonessi and Ahmad Yoonessi. "Functional assessment of magno, parvo and konio-cellular pathways; current state and future clinical applications." In: *Journal of ophthalmic & vision research* 6.2 (2011), p. 119 (cit. on pp. 11, 12).

[49] Raphael Berner et al. "A 240× 180 10mW 12us latency sparse-output vision sensor for mobile applications." In: *2013 Symposium on VLSI Circuits*. IEEE. 2013, pp. C186–C187. ISBN: 978-1-4673-5531-5 (cit. on p. 13).

[50] Chenghan Li et al. "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)*. IEEE, May 2015, pp. 718–721. DOI: 10.1109/iscas.2015.7168734 (cit. on p. 13).

[51] Diederik Paul Moeys et al. "Color temporal contrast sensitivity in dynamic vision sensors." In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. IEEE, May 2017, pp. 1–4. DOI: 10.1109/iscas.2017.8050412 (cit. on p. 13).

[52] Hanme Kim. *Event-Based Camera vs Standard Camera*. [Online; accessed 5. Jan. 2022]. Apr. 2016. URL: https://www.youtube.com/watch?v=kPCZESVfHoQ (cit. on p. 14).

[53] Elias Mueggler et al. “Event-based, 6-DOF pose tracking for high-speed maneuvers.” In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sept. 2014, pp. 2761–2768. DOI: 10.1109/iros.2014.6942940 (cit. on p. 14).

[54] Alex Zhu et al. “EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras.” In: *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, June 2018. DOI: 10.15607/rss.2018.xiv.062 (cit. on pp. 15, 17, 19, 23, 24, 27, 32, 37, 38, 41, 42, 47, 75, 87, 88, 90, 95).

[55] Alex Zihao Zhu et al. “Unsupervised Event-Based Learning of Optical Flow, Depth, and Egomotion.” In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019, pp. 989–997. DOI: 10.1109/cvpr.2019.00108 (cit. on pp. 15, 17–19, 23, 24, 38, 42, 72, 75, 78, 88, 92, 100, 102, 111, 116, 119, 121, 127).

[56] Chengxi Ye et al. “Unsupervised Learning of Dense Optical Flow, Depth and Egomotion with Event-Based Sensors.” In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2020, pp. 5831–5838. DOI: 10.1109/iros45743.2020.9341224 (cit. on pp. 15, 17, 38, 42, 72, 75, 100, 102).

[57] Javier Hidalgo-Carrio et al. “Learning Monocular Dense Depth from Events.” In: *2020 International Conference on 3D Vision (3DV)*. IEEE, Nov. 2020. DOI: 10.1109/3dv50981.2020.00063 (cit. on pp. 15, 42, 102, 118, 138).

[58] Daniel Gehrig et al. “Combining Events and Frames Using Recurrent Asynchronous Multimodal Networks for Monocular Depth Prediction.” In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 2822–2829. DOI: 10.1109/lra.2021.3060707 (cit. on pp. 15, 42, 102, 118, 138).

[59] Gianluca Scarpellini et al. “Lifting Monocular Events to 3D Human Poses.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2021, pp. 1358–1368 (cit. on p. 15).

[60] R. Qi Charles et al. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space.” In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017 (cit. on pp. 15, 17, 36).

[61] R. Qi Charles et al. “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation.” In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017, pp. 652–660. DOI: 10.1109/cvpr.2017.16 (cit. on pp. 15, 17, 36).

[62] Ian Goodfellow et al. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 15).

[63] Wolfgang Maass. “Networks of spiking neurons: The third generation of neural network models.” In: *Neural Networks* 10.9 (Dec. 1997), pp. 1659–1671. DOI: 10.1016/s0893-6080(97)00011-7 (cit. on pp. 16, 42, 74).

[64] QingXiang Wu et al. “Edge Detection Based on Spiking Neural Network Model.” In: *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*. Springer Berlin Heidelberg, 2007, pp. 26–34. DOI: 10.1007/978-3-540-74205-0_4 (cit. on p. 16).

[65] B. Meftah et al. “Segmentation and Edge Detection Based on Spiking Neural Network Model.” In: *Neural Processing Letters* 32.2 (Aug. 2010), pp. 131–146. DOI: 10.1007/s11063-010-9149-6 (cit. on p. 16).

[66] Jun Haeng Lee et al. “Training Deep Spiking Neural Networks Using Backpropagation.” In: *Frontiers in Neuroscience* 10 (Nov. 2016). ISSN: 1662-453X. DOI: 10.3389/fnins.2016.00508 (cit. on pp. 16, 22, 35, 42, 45).

[67] Peter U. Diehl et al. “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing.” In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2015, pp. 1–8. DOI: 10.1109/ijcnn.2015.7280696 (cit. on p. 16).

[68] Janos Botzheim et al. “Human gesture recognition for robot partners by spiking neural network and classification learning.” In: *The 6th International Conference on Soft Computing and Intelligent Systems, and The 13th International Symposium on Advanced Intelligence Systems*. IEEE, Nov. 2012, pp. 1954–1958. DOI: 10.1109/scis-isis.2012.6505305 (cit. on p. 16).

[69] Nitin Rathi et al. “STDP-Based Pruning of Connections and Weight Quantization in Spiking Neural Networks for Energy-Efficient Recognition.” In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.4 (Apr. 2019), pp. 668–677. DOI: 10.1109/tcad.2018.2819366 (cit. on p. 16).

[70] Yunzhe Hao et al. “A biologically plausible supervised learning method for spiking neural networks using the symmetric STDP rule.” In: *Neural Networks* 121 (Jan. 2020), pp. 387–395. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.09.007 (cit. on p. 16).

[71] J. A. Perez-Carrasco et al. “Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing–Application to Feedforward ConvNets.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11 (Nov. 2013), pp. 2706–2719. ISSN: 1939-3539. DOI: 10.1109/tpami.2013.71 (cit. on pp. 16, 27, 28, 44, 45).

[72] Steven K. Esser et al. “Convolutional networks for fast, energy-efficient neuromorphic computing.” In: *Proceedings of the National Academy of Sciences* 113.41 (Sept. 2016), pp. 11441–11446. DOI: 10.1073/pnas.1604850113 (cit. on p. 16).

[73] Bodo Rueckauer et al. “Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification.” In: *Frontiers in Neuroscience* 11 (Dec. 2017). ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00682 (cit. on pp. 16, 35).

[74] Hanme Kim et al. “Simultaneous Mosaicing and Tracking with an Event Camera.” In: *Proceedings of the British Machine Vision Conference 2014*. British Machine Vision Association, 2014. DOI: 10.5244/c.28.26 (cit. on pp. 16, 72).

[75] Christian Reinbacher et al. “Real-time panoramic tracking for event cameras.” In: *2017 IEEE International Conference on Computational Photography (ICCP)*. IEEE, May 2017. DOI: 10.1109/iccphot.2017.7951488 (cit. on pp. 16, 72).

[76] Guillermo Gallego et al. “Event-Based, 6-DOF Camera Tracking from Photometric Depth Maps.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.10 (Oct. 2018), pp. 2402–2412. DOI: 10.1109/tpami.2017.2769655 (cit. on pp. 16, 72, 120).

[77] Alireza Khodamoradi and Ryan Kastner. “O(N)-Space Spatiotemporal Filter for Reducing Noise in Neuromorphic Vision Sensors.” In: *IEEE Transactions on Emerging Topics in Computing* (2018), pp. 1–1. DOI: 10.1109/tetc.2017.2788865 (cit. on pp. 16, 72).

[78] Daniel Czech and Garrick Orchard. “Evaluating noise filtering for event-based asynchronous change detection image sensors.” In: *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*. IEEE, June 2016, pp. 19–24. DOI: 10.1109/biorob.2016.7523452 (cit. on pp. 16, 72).

[79] Cedric Scheerlinck et al. “Continuous-Time Intensity Estimation Using Event Cameras.” In: *Computer Vision – ACCV 2018*. Springer International Publishing, 2019, pp. 308–324. DOI: 10.1007/978-3-030-20873-8_20 (cit. on p. 16).

[80] Christian Reinbacher et al. “Real-Time Intensity-Image Reconstruction for Event Cameras Using Manifold Regularisation.” In: *Procedings of the British Machine Vision Conference 2016*. British Machine Vision Association, 2016. DOI: 10.5244/c.30.9 (cit. on pp. 16, 72).

[81] Cedric Scheerlinck et al. “Asynchronous Spatial Image Convolutions for Event Cameras.” In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019), pp. 816–822. DOI: 10.1109/lra.2019.2893427 (cit. on p. 16).

[82] R. Serrano-Gotarredona et al. "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory–Processing–Learning–Actuating System for High-Speed Visual Object Recognition and Tracking." In: *IEEE Transactions on Neural Networks* 20.9 (Sept. 2009), pp. 1417–1438. DOI: 10.1109/tnn.2009.2023653 (cit. on p. 16).

[83] R. Serrano-Gotarredona et al. "On Real-Time AER 2-D Convolutions Hardware for Neuromorphic Spike-Based Cortical Processing." In: *IEEE Transactions on Neural Networks* 19.7 (July 2008), pp. 1196–1219. DOI: 10.1109/tnn.2008.2000163 (cit. on p. 16).

[84] Xavier Lagorce et al. "HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.7 (July 2017), pp. 1346–1359. DOI: 10.1109/tpami.2016.2574707 (cit. on pp. 16–18, 20, 22, 27, 35, 47, 61, 74, 86, 95, 96).

[85] Amos Sironi et al. "HATS: Histograms of Averaged Time Surfaces for Robust Event-Based Object Classification." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 1731–1740. DOI: 10.1109/cvpr.2018.00186 (cit. on pp. 16–19, 22, 23, 27, 29, 35, 47, 61, 62, 71, 73, 74, 82, 86, 91, 92, 95–97, 121).

[86] Xavier Glorot et al. "Deep Sparse Rectifier Neural Networks." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon et al. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 315–323 (cit. on p. 16).

[87] Nico Messikommer et al. "Event-Based Asynchronous Sparse Convolutional Networks." In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Cham: Springer International Publishing, 2020, pp. 415–431. ISBN: 978-3-030-58598-3 (cit. on pp. 16, 70, 92, 137).

[88] Yi Zhou et al. "Event-Based Motion Segmentation With Spatio-Temporal Graph Cuts." In: *IEEE Transactions on Neural Networks and Learning Systems* (2021), pp. 1–13. DOI: 10.1109/tnnls.2021.3124580 (cit. on pp. 17, 39, 72).

[89] Yin Bi et al. "Graph-Based Object Classification for Neuromorphic Vision Sensing." In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019, pp. 491–501. DOI: 10.1109/iccv.2019.00058 (cit. on pp. 17, 27, 29, 75, 82, 86).

[90] Yin Bi et al. "Graph-Based Spatio-Temporal Feature Learning for Neuromorphic Vision Sensing." In: *IEEE Transactions on Image Processing* 29 (2020), pp. 9084–9098. DOI: 10.1109/tip.2020.3023597 (cit. on pp. 17, 36, 140).

[91] Junming Chen et al. “Dynamic Graph CNN for Event-Camera Based Gesture Recognition.” In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, Oct. 2020, pp. 1–5. DOI: 10.1109/iscas45731.2020.9181247 (cit. on p. 17).

[92] Daniel Gehrig et al. “End-to-End Learning of Representations for Asynchronous Event-Based Data.” In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019. DOI: 10.1109/iccv.2019.00573 (cit. on pp. 17–19, 23–25, 35, 41, 72, 74, 75, 77–79, 82, 83, 85, 86, 88, 91, 95, 100, 121, 122).

[93] Simone Undri Innocenti et al. “Temporal Binary Representation for Event-Based Action Recognition.” In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, Jan. 2021. DOI: 10.1109/icpr48806.2021.9412991 (cit. on pp. 17–19, 24, 25, 72).

[94] Yongjian Deng et al. “AMAE: Adaptive Motion-Agnostic Encoder for Event-Based Object Classification.” In: *IEEE Robotics and Automation Letters* 5.3 (July 2020), pp. 4596–4603. DOI: 10.1109/lra.2020.3002480 (cit. on pp. 17, 18, 25, 26, 35, 72).

[95] Yi Zhou et al. “Semi-dense 3D Reconstruction with a Stereo Event Camera.” In: *Computer Vision - ECCV 2018*. Springer International Publishing, 2018, pp. 242–258. DOI: 10.1007/978-3-030-01246-5_15 (cit. on p. 17).

[96] Sio-Hoi Ieng et al. “Neuromorphic Event-Based Generalized Time-Based Stereovision.” In: *Frontiers in Neuroscience* 12 (July 2018). DOI: 10.3389/fnins.2018.00442 (cit. on p. 17).

[97] Zhen Xie et al. “Event-Based Stereo Depth Estimation Using Belief Propagation.” In: *Frontiers in Neuroscience* 11 (Oct. 2017). DOI: 10.3389/fnins.2017.00535 (cit. on p. 17).

[98] Yi Zhou et al. “Event-Based Stereo Visual Odometry.” In: *IEEE Transactions on Robotics* 37 (5 Oct. 2021), pp. 1433–1450. ISSN: 1941-0468. DOI: 10.1109/TRO.2021.3062252 (cit. on p. 17).

[99] Jianhao Jiao et al. “Comparing Representations in Tracking for Event Camera-based SLAM.” In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2021. DOI: 10.1109/cvprw53098.2021.00151 (cit. on p. 17).

[100] Ignacio Alzugaray and Margarita Chli. “Asynchronous Corner Detection and Tracking for Event Cameras in Real Time.” In: *IEEE Robotics and Automation Letters* 3.4 (Oct. 2018), pp. 3177–3184. DOI: 10.1109/lra.2018.2849882 (cit. on pp. 17–20).

[101] Elias Mueggler et al. “Fast Event-based Corner Detection.” In: *Procedings of the British Machine Vision Conference 2017*. British Machine Vision Association, 2017. DOI: 10.5244/c.31.33 (cit. on p. 17).

[102] Yusuke Sekikawa et al. “EventNet: Asynchronous Recursive Event Processing.” In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019, pp. 3887–3896. DOI: 10.1109/cvpr.2019.00401 (cit. on pp. 17, 36, 75, 140).

[103] Qinyi Wang et al. “Space-Time Event Clouds for Gesture Recognition: From RGB Cameras to Event Cameras.” In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Jan. 2019, pp. 1826–1835. DOI: 10.1109/wacv.2019.00199 (cit. on pp. 17, 36, 75, 140).

[104] Etienne Perot et al. “Learning to Detect Objects with a 1 Megapixel Event Camera.” In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 16639–16652 (cit. on pp. 17, 24, 27, 30, 31, 36, 53, 91, 92, 94, 101, 138).

[105] Inigo Alonso and Ana C. Murillo. “EV-SegNet: Semantic Segmentation for Event-Based Cameras.” In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2019. DOI: 10.1109/cvprw.2019.00205 (cit. on pp. 17, 20, 27, 32, 33, 39, 41, 42, 121–123, 131, 136).

[106] Lin Wang et al. “Event-Based High Dynamic Range Image and Very High Frame Rate Video Generation Using Conditional Generative Adversarial Networks.” In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019, pp. 10081–10090. DOI: 10.1109/cvpr.2019.01032 (cit. on pp. 17–20, 23, 24, 42, 72).

[107] Cedric Scheerlinck et al. “Fast Image Reconstruction with an Event Camera.” In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Mar. 2020, pp. 156–163. DOI: 10.1109/wacv45572.2020.9093366 (cit. on pp. 17, 23, 24).

[108] Henri Rebecq et al. “High Speed and High Dynamic Range Video with an Event Camera.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.6 (June 2021), pp. 1964–1980. DOI: 10.1109/tpami.2019.2963386 (cit. on pp. 17, 23, 24, 42, 102, 138).

[109] Ana I. Maqueda et al. “Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars.” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 5419–5427. DOI: 10.1109/cvpr.2018.00568 (cit. on pp. 18, 19, 47, 88, 92).

[110] Daniel Gehrig et al. “EKLT: Asynchronous Photometric Feature Tracking Using Events and Frames.” In: *International Journal of Computer Vision* 128.3 (Aug. 2019), pp. 601–618. DOI: 10.1007/s11263-019-01209-w (cit. on pp. 18–20, 37).

[111] Ryad Benosman et al. "Event-Based Visual Flow." In: *IEEE Transactions on Neural Networks and Learning Systems* 25.2 (Feb. 2014), pp. 407–417. DOI: 10.1109/tnnls.2013.2273537 (cit. on pp. 18, 20, 37).

[112] Junho Kim et al. "N-ImageNet: Towards Robust, Fine-Grained Object Recognition With Event Cameras." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 2146–2156 (cit. on pp. 18, 19, 21, 27, 29, 33, 35, 41, 63, 72).

[113] R. Wes Baldwin et al. "Inceptive Event Time-Surfaces for Object Classification Using Neuromorphic Cameras." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 395–403. DOI: 10.1007/978-3-030-27272-2_35 (cit. on pp. 18, 20).

[114] Anton Mitrokhin et al. "Event-Based Moving Object Detection and Tracking." In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2018. DOI: 10.1109/iros.2018.8593805 (cit. on pp. 18, 20, 39).

[115] Henri Rebecq et al. "Real-time Visual-Inertial Odometry for Event Cameras using Keyframe-based Nonlinear Optimization." In: *Proceedings of the British Machine Vision Conference 2017*. British Machine Vision Association, Sept. 2017. DOI: 10.5244/c.31.16 (cit. on pp. 18–20).

[116] Garrick Orchard et al. "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades." In: *Frontiers in Neuroscience* 9 (Nov. 2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00437 (cit. on pp. 19, 27, 28, 30, 57, 59–61, 63, 73, 82, 86, 104, 111, 116, 117, 120, 121, 125, 136).

[117] Antoni Rosinol Vidal et al. "Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High-Speed Scenarios." In: *IEEE Robotics and Automation Letters* 3.2 (2 Apr. 2018), pp. 994–1001. ISSN: 2377-3774. DOI: 10.1109/lra.2018.2793357 (cit. on p. 19).

[118] Mohammad Mostafavi et al. "Learning to Reconstruct HDR Images from Events, with Applications to Depth and Flow Prediction." In: *International Journal of Computer Vision* 129.4 (Jan. 2021), pp. 900–920. DOI: 10.1007/s11263-020-01410-2 (cit. on pp. 20, 23, 24).

[119] Timo Stoffregen and Lindsay Kleeman. "Simultaneous Optical Flow and Segmentation (SOFAS) using Dynamic Vision Sensor." In: *Australasian Conference on Robotics and Automation* (2017) (cit. on pp. 20, 39).

[120] Timo Stoffregen et al. "Event-Based Motion Segmentation by Motion Compensation." In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019. DOI: 10.1109/iccv.2019.00734 (cit. on pp. 20, 39, 72).

[121] Chethan M. Parameshwara et al. “0-MMS: Zero-Shot Multi-Motion Segmentation With A Monocular Event Camera.” In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2021, pp. 9594–9600. DOI: 10.1109/icra48506.2021.9561755 (cit. on pp. 20, 39, 72).

[122] Ammar Mohemmed et al. “SPAN: A Neuron for Precise-Time Spike Pattern Association.” In: *Neural Information Processing*. Springer Berlin Heidelberg, 2011, pp. 718–725. DOI: 10.1007/978-3-642-24958-7_83 (cit. on p. 22).

[123] Filip Ponulak and Andrzej Kasinski. “ReSuMe learning method for Spiking Neural Networks dedicated to neuroprostheses control.” In: *Proceedings of EPFL LATSIS Symposium 2006, Dynamical Principles for Neuroscience and Intelligent Biomimetic Devices*. Citeseer. 2006, pp. 119–120 (cit. on p. 22).

[124] Stepan Tulyakov et al. “Time Lens: Event-based Video Frame Interpolation.” In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2021, pp. 16155–16164. DOI: 10.1109/cvpr46437.2021.01589 (cit. on p. 23).

[125] Max Jaderberg et al. “Spatial Transformer Networks.” In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015 (cit. on p. 24).

[126] Henri Rebecq et al. “Events-To-Video: Bringing Modern Computer Vision to Event Cameras.” In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019, pp. 3857–3866. DOI: 10.1109/cvpr.2019.00398 (cit. on pp. 24, 42, 72, 75, 77, 78, 82, 84, 86, 92, 100).

[127] Daniel Gehrig et al. “Video to Events: Recycling Video Datasets for Event Cameras.” In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2020, pp. 3586–3595. DOI: 10.1109/cvpr42600.2020.00364 (cit. on pp. 25, 33, 34, 39, 40, 92, 102–104, 111, 113, 117, 120–122, 127, 128, 130–132).

[128] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. “Poker-DVS and MNIST-DVS. Their History, How They Were Made, and Other Details.” In: *Frontiers in Neuroscience* 9 (Dec. 2015). ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00481 (cit. on pp. 27, 28, 57, 59).

[129] Hongmin Li et al. “CIFAR10-DVS: An Event-Stream Dataset for Object Classification.” In: *Frontiers in Neuroscience* 11 (May 2017). ISSN: 1662-4548. DOI: 10.3389/fnins.2017.00309 (cit. on pp. 27, 29).

[130] Yuhuang Hu et al. “DVS Benchmark Datasets for Object Tracking, Action Recognition, and Object Recognition.” In: *Frontiers in Neuroscience* 10 (Aug. 2016). DOI: 10.3389/fnins.2016.00405 (cit. on pp. 27, 29, 30, 120).

[131] Christian Brandli et al. "A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor." In: *IEEE Journal of Solid-State Circuits* 49.10 (Oct. 2014), pp. 2333–2341. DOI: 10.1109/jssc.2014.2342715 (cit. on p. 27).

[132] Bongki Son 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)*. IEEE, Feb. 2017. DOI: 10.1109/isscc.2017.7870263 (cit. on p. 27).

[133] Arnon Amir et al. "A Low Power, Fully Event-Based Gesture Recognition System." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017. DOI: 10.1109/cvpr.2017.781 (cit. on pp. 27, 30).

[134] Shu Miao et al. "Neuromorphic Vision Datasets for Pedestrian Detection, Action Recognition, and Fall Detection." In: *Frontiers in Neurorobotics* 13 (June 2019). ISSN: 1662-5218. DOI: 10.3389/fnbot.2019.00038 (cit. on pp. 27, 30).

[135] Pierre de Tournemire et al. "A large scale event-based detection dataset for automotive." In: *arXiv preprint arXiv:2001.08499* (2020) (cit. on pp. 27, 30, 31, 53, 74, 91–93, 100).

[136] Alex Zihao Zhu et al. "The Multivehicle Stereo Event Camera Dataset: An Event Camera Dataset for 3D Perception." In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 2032–2039. DOI: 10.1109/lra.2018.2800793 (cit. on pp. 27, 32, 38, 74, 87, 89, 91, 95, 99).

[137] M Almatrafi et al. *Distance surface for event-based optical flow*. 2020. DOI: 10.1109/tpami.2020.2986748 (cit. on pp. 27, 32, 72).

[138] Jonathan Binas et al. "DDD17: End-to-end DAVIS driving dataset." In: *arXiv preprint arXiv:1711.01458* (2017) (cit. on pp. 27, 32, 33, 39, 104, 121, 122, 131, 136).

[139] Sifan Yang et al. "VESS: Variable Event Stream Structure for Event-based Instance Segmentation Benchmark." In: *Proceedings of the 2020 4th International Conference on Digital Signal Processing*. ACM, June 2020, pp. 112–116. DOI: 10.1145/3408127.3408178 (cit. on pp. 27, 33).

[140] Garrick Orchard et al. "HFirst: A Temporal Approach to Object Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.10 (Oct. 2015), pp. 2028–2040. DOI: 10.1109/tpami.2015.2392947 (cit. on pp. 27, 35, 61, 62, 86).

[141] Y. Lecun et al. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791 (cit. on pp. 28, 57, 61).

[142] Li Fei-Fei et al. "One-shot learning of object categories." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.4 (Apr. 2006), pp. 594–611. ISSN: 0162-8828. DOI: 10.1109/tpami.2006.79 (cit. on pp. 28, 82, 121).

[143] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images." In: (2009) (cit. on pp. 29, 57).

[144] G Griffin et al. *Caltech-256 Object Category Dataset*. Tech. rep. 2006 (cit. on p. 29).

[145] Kishore K. Reddy and Mubarak Shah. "Recognizing 50 human action categories of web videos." In: *Machine Vision and Applications* 24.5 (Nov. 2012), pp. 971–981. DOI: 10.1007/s00138-012-0450-4 (cit. on p. 29).

[146] Jia Deng et al. "ImageNet: A large-scale hierarchical image database." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2009, pp. 248–255. DOI: 10.1109/cvpr.2009.5206848 (cit. on pp. 29, 82).

[147] Zhong-Qiu Zhao et al. "Object Detection With Deep Learning: A Review." In: *IEEE Transactions on Neural Networks and Learning Systems* 30.11 (Nov. 2019), pp. 3212–3232. DOI: 10.1109/tnnls.2018.2876865 (cit. on pp. 30, 52, 55).

[148] Zhengxia Zou et al. "Object detection in 20 years: A survey." In: *arXiv preprint arXiv:1905.05055* (2019) (cit. on pp. 30, 52).

[149] Li Liu et al. "Deep Learning for Generic Object Detection: A Survey." In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 261–318. DOI: 10.1007/s11263-019-01247-4 (cit. on pp. 30, 52).

[150] Kai Kang et al. "T-CNN: Tubelets With Convolutional Neural Networks for Object Detection From Videos." In: *IEEE Transactions on Circuits and Systems for Video Technology* 28.10 (Oct. 2018), pp. 2896–2907. DOI: 10.1109/tcsvt.2017.2736553 (cit. on p. 30).

[151] Philipp Bergmann et al. "Tracking Without Bells and Whistles." In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019. DOI: 10.1109/iccv.2019.00103 (cit. on p. 30).

[152] Kaiming He et al. "Mask R-CNN." In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017, pp. 2980–2988. DOI: 10.1109/iccv.2017.322 (cit. on p. 30).

[153] Shu Liu et al. "Path Aggregation Network for Instance Segmentation." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 8759–8768. DOI: 10.1109/cvpr.2018.00913 (cit. on p. 30).

[154] Liang-Chieh Chen et al. "MaskLab: Instance Segmentation by Refining Object Detection with Semantic and Direction Features." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 4013–4022. DOI: 10.1109/cvpr.2018.00422 (cit. on p. 30).

[155] Qi Wu et al. "Image Captioning and Visual Question Answering Based on Attributes and External Knowledge." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.6 (June 2018), pp. 1367–1381. DOI: 10.1109/tpami.2017.2708709 (cit. on p. 30).

[156] Peter Anderson et al. "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018. DOI: 10.1109/cvpr.2018.00636 (cit. on p. 30).

[157] Matej Kristan et al. "A Novel Performance Evaluation Methodology for Single-Target Trackers." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.11 (Nov. 2016), pp. 2137–2155. DOI: 10.1109/tpami.2016.2516982 (cit. on p. 30).

[158] Mingliang Zhai et al. "Optical flow and scene flow estimation: A survey." In: *Pattern Recognition* 114 (June 2021), p. 107861. DOI: 10.1016/j.patcog.2021.107861 (cit. on p. 31).

[159] Shijie Hao et al. "A Brief Survey on Semantic Segmentation with Deep Learning." In: *Neurocomputing* 406 (Sept. 2020), pp. 302–321. DOI: 10.1016/j.neucom.2019.11.118 (cit. on p. 31).

[160] Fahad Lateef and Yassine Ruichek. "Survey on semantic segmentation using deep learning techniques." In: *Neurocomputing* 338 (Apr. 2019), pp. 321–348. DOI: 10.1016/j.neucom.2019.02.003 (cit. on p. 31).

[161] Bodo Rueckauer and Tobi Delbruck. "Evaluation of Event-Based Algorithms for Optical Flow with Ground-Truth from Inertial Measurement Sensor." In: *Frontiers in Neuroscience* 10 (Apr. 2016). DOI: 10.3389/fnins.2016.00176 (cit. on pp. 32, 37).

[162] Patrick Bardow et al. "Simultaneous Optical Flow and Intensity Estimation from an Event Camera." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2016. DOI: 10.1109/cvpr.2016.102 (cit. on pp. 32, 37, 72).

[163] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223. DOI: 10.1109/cvpr.2016.350 (cit. on p. 33).

[164] M. L. Katz et al. "Live demonstration: Behavioural emulation of event-based vision sensors." In: *2012 IEEE International Symposium on Circuits and Systems*. IEEE, May 2012, pp. 736–740. DOI: 10.1109/iscas.2012.6272143 (cit. on p. 33).

[165] Elias Mueggler et al. “The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and SLAM.” In: *The International Journal of Robotics Research* 36.2 (Feb. 2017), pp. 142–149. DOI: 10.1177/0278364917691115 (cit. on pp. 34, 59, 102).

[166] Wenbin Li et al. “InteriorNet: Mega-scale Multi-sensor Photo-realistic Indoor Scenes Dataset.” In: *British Machine Vision Conference (BMVC)*. 2018 (cit. on p. 34).

[167] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: http://www.blender.org (cit. on pp. 34, 60, 115, 121).

[168] Henri Rebecq et al. “ESIM: an open event camera simulator.” In: *Conference on Robot Learning*. PMLR. 2018, pp. 969–982 (cit. on pp. 34, 102, 111, 112, 115, 120–122).

[169] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator.” In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine et al. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1–16 (cit. on p. 34).

[170] Y Hu et al. “V2e: From video frames to realistic DVS events.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 1312–1321 (cit. on pp. 34, 102).

[171] Yuji Nozaki and Tobi Delbruck. “Temperature and Parasitic Photocurrent Effects in Dynamic Vision Sensors.” In: *IEEE Transactions on Electron Devices* 64.8 (Aug. 2017), pp. 3239–3245. DOI: 10.1109/ted.2017.2717848 (cit. on p. 34).

[172] David H. Hubel and Torsten N. Wiesel. “RECEPTIVE FIELDS AND FUNCTIONAL ARCHITECTURE IN TWO NONSTRIATE VISUAL AREAS (18 AND 19) OF THE CAT.” In: *Journal of Neurophysiology* 28.2 (Mar. 1965), pp. 229–289. DOI: 10.1152/jn.1965.28.2.229 (cit. on p. 35).

[173] Thomas Serre et al. *Object Recognition with Features Inspired by Visual Cortex*. Tech. rep. Jan. 2006. DOI: 10.21236/ada454604 (cit. on p. 35).

[174] Bo Zhao et al. “Feedforward Categorization on AER Motion Events Using Cortex-Like Features in a Spiking Neural Network.” In: *IEEE Transactions on Neural Networks and Learning Systems* 26.9 (Sept. 2015), pp. 1963–1978. DOI: 10.1109/tnnls.2014.2362542 (cit. on p. 35).

[175] Timothée Masquelier and Simon J Thorpe. “Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity.” In: *PLoS Computational Biology* 3.2 (Feb. 2007). Ed. by Karl J Friston, e31. DOI: 10.1371/journal.pcbi.0030031 (cit. on p. 35).

[176] Thomas Serre et al. “Robust Object Recognition with Cortex-Like Mechanisms.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.3 (Mar. 2007), pp. 411–426. DOI: 10.1109/tpami.2007.56 (cit. on p. 35).

[177] Plinio Moreno et al. “Gabor Parameter Selection for Local Feature Detection.” In: *Pattern Recognition and Image Analysis*. Berlin, Heidelberg; Berlin Heidelberg: Springer Berlin Heidelberg, 2005, pp. 11–19. DOI: 10.1007/11492429_2 (cit. on p. 35).

[178] Guo qiang Bi and Mu ming Poo. “Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type.” In: *The Journal of Neuroscience* 18.24 (Dec. 1998), pp. 10464–10472. DOI: 10.1523/jneurosci.18-24-10464.1998 (cit. on p. 35).

[179] Xin Jin et al. “Implementing spike-timing-dependent plasticity on SpiNNaker neuromorphic hardware.” In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2010. DOI: 10.1109/ijcnn.2010.5596372 (cit. on p. 35).

[180] Abhronil Sengupta et al. “Going Deeper in Spiking Neural Networks: VGG and Residual Architectures.” In: *Frontiers in Neuroscience* 13 (Mar. 2019). ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00095 (cit. on p. 35).

[181] Chankyu Lee et al. “Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures.” In: *Frontiers in Neuroscience* 14 (Feb. 2020). DOI: 10.3389/fnins.2020.00119 (cit. on p. 35).

[182] Emre O. Neftci et al. “Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks.” In: *IEEE Signal Processing Magazine* 36.6 (Nov. 2019), pp. 51–63. DOI: 10.1109/msp.2019.2931595 (cit. on pp. 35, 42, 45).

[183] Evangelos Stromatias et al. “An Event-Driven Classifier for Spiking Neural Networks Fed with Synthetic or Dynamic Vision Sensor Data.” In: *Frontiers in Neuroscience* 11 (June 2017). DOI: 10.3389/fnins.2017.00350. URL: https://www.frontiersin.org/article/10.3389/fnins.2017.00350 (cit. on pp. 35, 59).

[184] Youngeun Kim and Priyadarshini Panda. “Optimizing Deeper Spiking Neural Networks for Dynamic Vision Sensing.” In: *Neural Networks* 144 (Dec. 2021), pp. 686–698. DOI: 10.1016/j.neunet.2021.09.022 (cit. on pp. 35, 42, 45).

[185] Yongjian Deng et al. “MVF-Net: A Multi-view Fusion Network for Event-based Object Classification.” In: *IEEE Transactions on Circuits and Systems for Video Technology* (2021), pp. 1–1. DOI: 10.1109/tcsvt.2021.3073673 (cit. on p. 35).

[186] Jiancheng Yang et al. “Modeling Point Clouds With Self-Attention and Gumbel Subset Sampling.” In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019. DOI: 10.1109/cvpr.2019.00344 (cit. on p. 36).

[187] Ashish Vaswani et al. “Attention is All you Need.” In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017 (cit. on pp. 36, 139).

[188] Yongjian Deng et al. “EV-VGCNN: A Voxel Graph CNN for Event-based Object Classification.” In: *arXiv preprint arXiv:2106.00216* (2021) (cit. on pp. 36, 140).

[189] Jia Li et al. “Adaptive Temporal Pooling for Object Detection using Dynamic Vision Sensor.” In: *Procedings of the British Machine Vision Conference 2017*. British Machine Vision Association, 2017. DOI: 10.5244/c.31.40 (cit. on pp. 36, 53, 57).

[190] Nico Messikommer et al. “Bridging the Gap between Events and Frames through Unsupervised Domain Adaptation.” In: *IEEE Robotics and Automation Letters* (2022), pp. 1–1. DOI: 10.1109/lra.2022.3145053 (cit. on pp. 36, 53, 72, 140).

[191] Shaoqing Ren et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (June 2017), pp. 1137–1149. DOI: 10.1109/tpami.2016.2577031 (cit. on pp. 36, 42, 52).

[192] Xingjian SHI et al. “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting.” In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 802–810 (cit. on pp. 36, 73, 74, 76, 80, 95, 97, 138).

[193] Guillermo Gallego et al. “Event-based camera pose tracking using a generative event model.” In: *arXiv preprint arXiv:1510.01972* (2015) (cit. on pp. 36, 72).

[194] Ryad Benosman et al. “Asynchronous frameless event-based optical flow.” In: *Neural Networks* 27 (Mar. 2012), pp. 32–37. DOI: 10.1016/j.neunet.2011.11.001 (cit. on p. 36).

[195] Bruce D. Lucas and Takeo Kanade. “An Iterative Image Registration Technique with an Application to Stereo Vision.” In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI ’81, Vancouver, BC, Canada, August 24-28, 1981*. Ed. by Patrick J. Hayes. William Kaufmann, 1981, pp. 674–679 (cit. on pp. 36, 38).

[196] Garrick Orchard et al. “A spiking neural network architecture for visual motion estimation.” In: *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, Oct. 2013. DOI: 10.1109/biocas.2013.6679698 (cit. on pp. 37, 38).

[197] Elias Mueggler et al. "Lifetime estimation of events from Dynamic Vision Sensors." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. IEEE, May 2015, pp. 4874–4881. DOI: 10.1109/icra.2015.7139876 (cit. on p. 37).

[198] Francisco Barranco et al. "Contour Motion Estimation for Asynchronous Event-Driven Cameras." In: *Proceedings of the IEEE* 102.10 (Oct. 2014), pp. 1537–1556. DOI: 10.1109/jproc.2014.2347207 (cit. on p. 37).

[199] Francisco Barranco et al. "Bio-inspired Motion Estimation with Event-Driven Sensors." In: *Advances in Computational Intelligence*. Springer International Publishing, 2015, pp. 309–321. DOI: 10.1007/978-3-319-19258-1_27 (cit. on p. 37).

[200] Tobias Brosch et al. "On event-based optical flow detection." In: *Frontiers in Neuroscience* 9 (Apr. 2015). DOI: 10.3389/fnins.2015.00137 (cit. on p. 37).

[201] Alex Zihao Zhu et al. "Event-based feature tracking with probabilistic data association." In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2017, pp. 4465–4470. DOI: 10.1109/icra.2017.7989517 (cit. on p. 37).

[202] Guillermo Gallego et al. "A Unifying Contrast Maximization Framework for Event Cameras, with Applications to Motion, Depth, and Optical Flow Estimation." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018. DOI: 10.1109/cvpr.2018.00407 (cit. on pp. 37, 72, 140).

[203] Olaf Ronneberger et al. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28 (cit. on pp. 37, 123).

[204] Jason J. Yu et al. "Back to Basics: Unsupervised Learning of Optical Flow via Brightness Constancy and Motion Smoothness." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2016, pp. 3–10. DOI: 10.1007/978-3-319-49409-8_1 (cit. on pp. 38, 102).

[205] Wei-Sheng Lai et al. "Semi-Supervised Learning for Optical Flow with Generative Adversarial Networks." In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 2017, pp. 354–364 (cit. on p. 38).

[206] Ziluo Ding et al. "Spatio-Temporal Recurrent Networks for Event-Based Optical Flow Estimation." In: *CoRR* abs/2109.04871 (2021). arXiv: 2109.04871 (cit. on p. 38).

[207] Mathias Gehrig et al. "E-RAFT: Dense Optical Flow from Event Cameras." In: *arXiv preprint arXiv:2108.10552* (Dec. 2021). DOI: 10.1109/3dv53792.2021.00030 (cit. on pp. 38, 101, 102).

[208] Federico Paredes-Valles et al. "Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow Estimation: From Events to Global Motion Perception." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.8 (Aug. 2020), pp. 2051–2064. DOI: 10.1109/tpami.2019.2903179 (cit. on p. 38).

[209] Germain Haessig et al. "Spiking Optical Flow for Event-Based Sensors Using IBM's TrueNorth Neurosynaptic System." In: *IEEE Transactions on Biomedical Circuits and Systems* 12.4 (Aug. 2018), pp. 860–870. DOI: 10.1109/tbcas.2018.2834558 (cit. on p. 38).

[210] Paul A. Merolla et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface." In: *Science* 345.6197 (Aug. 2014), pp. 668–673. DOI: 10.1126/science.1254642 (cit. on p. 38).

[211] Chankyu Lee et al. "Spike-FlowNet: Event-Based Optical Flow Estimation with Energy-Efficient Hybrid Neural Networks." In: *Computer Vision – ECCV 2020*. Springer International Publishing, 2020, pp. 366–382. DOI: 10.1007/978-3-030-58526-6_22 (cit. on p. 38).

[212] Arren Glover and Chiara Bartolozzi. "Event-driven ball detection and gaze fixation in clutter." In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Oct. 2016. DOI: 10.1109/iros.2016.7759345 (cit. on p. 39).

[213] Arren Glover and Chiara Bartolozzi. "Robust visual tracking with a freely-moving event camera." In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sept. 2017. DOI: 10.1109/iros.2017.8206226 (cit. on p. 39).

[214] V. Vasco et al. "Independent motion detection with event-driven cameras." In: *2017 18th International Conference on Advanced Robotics (ICAR)*. IEEE, July 2017. DOI: 10.1109/icar.2017.8023661 (cit. on p. 39).

[215] Anton Mitrokhin et al. "EV-IMO: Motion Segmentation Dataset and Learning Pipeline for Event Cameras." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Nov. 2019. DOI: 10.1109/iros40897.2019.8968520 (cit. on p. 39).

[216] Francois Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017. DOI: 10.1109/cvpr.2017.195 (cit. on pp. 39, 123).

[217] Jiaming Zhang et al. "ISSAFE: Improving semantic segmentation in accidents by fusing event-based data." In: *arXiv preprint arXiv:2008.08974* (2020) (cit. on pp. 39, 42).

[218] Youngeun Kim et al. "Beyond Classification: Directly Training Spiking Neural Networks for Semantic Segmentation." In: *arXiv preprint arXiv:2110.07742* (2021) (cit. on pp. 39, 42, 45).

[219] Jonathan Long et al. "Fully convolutional networks for semantic segmentation." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015, pp. 3431–3440. DOI: 10.1109/cvpr.2015.7298965 (cit. on pp. 40, 42, 56, 79).

[220] Kaiming He et al. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778. DOI: 10.1109/cvpr.2016.90 (cit. on pp. 40, 42, 82, 91, 122).

[221] Alex Krizhevsky et al. "ImageNet classification with deep convolutional neural networks." In: *Communications of the ACM* 60.6 (May 2017), pp. 84–90. DOI: 10.1145/3065386 (cit. on p. 42).

[222] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *CoRR* abs/1409.1556 (2014) (cit. on pp. 42, 61).

[223] Christian Szegedy et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." In: *AAAI*. Vol. 4. 2017, p. 12 (cit. on p. 42).

[224] Wei Liu et al. "SSD: Single Shot MultiBox Detector." In: *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2 (cit. on pp. 42, 53).

[225] Fisher Yu and Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions." In: *International Conference on Learning Representations (ICLR)*. 2016 (cit. on p. 42).

[226] Aman Raj et al. *Multi-scale convolutional architecture for semantic segmentation*. Tech. rep. 2015 (cit. on p. 42).

[227] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." In: *International Conference on Learning Representations (ICLR)*. 2017 (cit. on p. 42).

[228] Federico Monti et al. "Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017, pp. 5115–5124. DOI: 10.1109/cvpr.2017.576 (cit. on p. 42).

[229] Min Liu and Tobi Delbruck. *Adaptive Time-Slice Block-Matching Optical Flow Algorithm for Dynamic Vision Sensors*. Tech. rep. 2018 (cit. on p. 42).

[230] Amirhossein Tavanaei et al. "Deep learning in spiking neural networks." In: *Neural Networks* 111 (Mar. 2019), pp. 47–63. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2018.12.002 (cit. on p. 42).

[231] Filipp Akopyan et al. “TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip.” In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (10 Oct. 2015), pp. 1537–1557. ISSN: 1937-4151. DOI: 10.1109/tcad.2015.2474396 (cit. on p. 42).

[232] Mike Davies et al. “Loihi: A Neuromorphic Manycore Processor with On-Chip Learning.” In: *IEEE Micro* 38.1 (Jan. 2018), pp. 82–99. DOI: 10.1109/mm.2018.112130359 (cit. on p. 42).

[233] Steve B. Furber et al. “The SpiNNaker Project.” In: *Proceedings of the IEEE* 102.5 (May 2014), pp. 652–665. DOI: 10.1109/jproc.2014.2304638 (cit. on p. 42).

[234] Wulfram Gerstner and Werner M. Kistler. *Spiking Neuron Models*. Cambridge University Press, Aug. 2002. DOI: 10.1017/cbo9780511815706 (cit. on pp. 44, 45).

[235] A Grüning and S Bohte. “Spiking neural networks: Principles and challenges.” In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. Apr. 2014. ISBN: 978-287419095-7 (cit. on p. 44).

[236] Simon Thorpe and Jacques Gautrais. “Rank Order Coding.” In: *Computational Neuroscience*. Springer US, 1998, pp. 113–118. DOI: 10.1007/978-1-4615-4831-7_19 (cit. on p. 45).

[237] A Hodgkin and A Huxley. “A quantitative description of membrane current and its application to conduction and excitation in nerve.” In: *Bulletin of Mathematical Biology* 52.1-2 (1990), pp. 25–71. DOI: 10.1016/s0092-8240(05)80004-7 (cit. on p. 45).

[238] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007. ISBN: 9780262090438 (cit. on p. 45).

[239] Jesus L. Lobo et al. “Spiking Neural Networks and online learning: An overview and perspectives.” In: *Neural Networks* 121 (Jan. 2020), pp. 88–100. DOI: 10.1016/j.neunet.2019.09.004 (cit. on p. 45).

[240] Jilles Vreeken. *Spiking Neural Networks, an Introduction*. Tech. rep. 2003 (cit. on p. 45).

[241] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines.” In: *ICML*. 2010, pp. 807–814 (cit. on p. 48).

[242] Andrew L. Maas et al. “Rectifier nonlinearities improve neural network acoustic models.” In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013 (cit. on p. 48).

[243] Ross Girshick et al. “Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.” In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June 2014. DOI: 10.1109/cvpr.2014.81 (cit. on p. 52).

[244] Kaiming He et al. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (Sept. 2015), pp. 1904–1916. DOI: 10.1109/tpami.2015.2389824 (cit. on p. 52).

[245] Ross Girshick. “Fast R-CNN.” In: *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Dec. 2015, pp. 1440–1448. DOI: 10.1109/iccv.2015.169 (cit. on p. 52).

[246] Jifeng Dai et al. “R-FCN: Object Detection via Region-based Fully Convolutional Networks.” In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016 (cit. on p. 52).

[247] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger.” In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017, pp. 7263–7271. DOI: 10.1109/cvpr.2017.690 (cit. on pp. 53, 91).

[248] Zhishuai Zhang et al. “Single-Shot Object Detection with Enriched Semantics.” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 5813–5821. DOI: 10.1109/cvpr.2018.00609 (cit. on p. 53).

[249] Cheng-Yang Fu et al. “Dssd: Deconvolutional single shot detector.” In: *arXiv preprint arXiv:1701.06659* (2017) (cit. on p. 53).

[250] Chien-Yao Wang et al. “Scaled-YOLOv4: Scaling Cross Stage Partial Network.” In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2021, pp. 13029–13038. DOI: 10.1109/cvpr46437.2021.01283 (cit. on p. 53).

[251] Nicholas F. Y. Chen. “Pseudo-Labels for Supervised Learning on Dynamic Vision Sensor Data, Applied to Object Detection Under Ego-Motion.” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2018. DOI: 10.1109/cvprw.2018.00107 (cit. on p. 53).

[252] Bharath Ramesh et al. “DART: Distribution Aware Retinal Transform for Event-based Cameras.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019), pp. 1–1. DOI: 10.1109/tpami.2019.2919301 (cit. on p. 53).

[253] Li Fei-Fei et al. “Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories.” In: *2004 Conference on Computer Vision and Pattern Recognition Workshop*. IEEE, 2004. DOI: 10.1109/cvpr.2004.383 (cit. on pp. 57, 63, 111).

[254] Martin Ester et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.” In: *KDD*. AAAI Press, 1996, pp. 226–231 (cit. on p. 57).

[255] Volodymyr Mnih et al. "Recurrent models of visual attention." In: *Advances in neural information processing systems*. 2014, pp. 2204–2212 (cit. on p. 57).

[256] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *PMLR* (2010), pp. 249–256. ISSN: 1938-7228. URL: http://proceedings.mlr.press/v9/glorot10a.html (cit. on pp. 61, 122).

[257] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *Int. Conf. on Learning Representations (ICLR)* (2015). URL: https://arxiv.org/abs/1412.6980 (cit. on pp. 61, 83, 87, 123).

[258] Daniel Neil et al. "Phased LSTM: Accelerating recurrent network training for long or event-based sequences." In: *Advances in neural information processing systems*. 2016, pp. 3882–3890 (cit. on pp. 61, 75, 76).

[259] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge." In: *International Journal of Computer Vision* 88.2 (Sept. 2009), pp. 303–338. ISSN: 1573-1405. DOI: 10.1007/s11263-009-0275-4 (cit. on pp. 62, 91, 92).

[260] Yihan Lin et al. "ES-ImageNet: A Million Event-Stream Classification Dataset for Spiking Neural Networks." In: *Frontiers in Neuroscience* 15 (Nov. 2021). DOI: 10.3389/fnins.2021.726582 (cit. on p. 63).

[261] Wenjie Luo et al. "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016 (cit. on p. 65).

[262] Charles R. Harris et al. "Array programming with NumPy." In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2 (cit. on p. 65).

[263] Benjamin Graham et al. "3D Semantic Segmentation with Submanifold Sparse Convolutional Networks." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018. DOI: 10.1109/cvpr.2018.00961 (cit. on pp. 70, 137).

[264] Zelin Zhang et al. "Image Reconstruction from Events. Why learn it?" In: *arXiv* (Dec. 2021). eprint: 2112.06242 (cit. on pp. 72, 140).

[265] Lin Wang et al. "EventSR: From Asynchronous Events to Image Reconstruction, Restoration, and Super-Resolution via End-to-End Adversarial Learning." In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2020, pp. 8315–8325. DOI: 10.1109/cvpr42600.2020.00834 (cit. on p. 72).

[266] Fang Xu et al. "Matching Neuromorphic Events and Color Images via Adversarial Learning." In: *arXiv preprint arXiv:2003.00636* (2020). eprint: 2003.00636 (cit. on p. 72).

[267] Benoit Steiner et al. “PyTorch: An imperative style, high-performance deep learning library.” In: *Advances in Neural Information Processing Systems* 32 (2019) (cit. on pp. 74, 91, 99, 122, 123).

[268] Martin Abadi et al. “TensorFlow: A system for large-scale machine learning.” In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283 (cit. on pp. 74, 123).

[269] Marco Cannici et al. “Attention Mechanisms for Object Recognition With Event-Based Cameras.” In: *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Jan. 2019, pp. 1127–1136. DOI: 10.1109/wacv.2019.00125 (cit. on pp. 74, 75).

[270] Cedric Scheerlinck et al. “CED: Color Event Camera Dataset.” In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2019. DOI: 10.1109/cvprw.2019.00215 (cit. on p. 77).

[271] Jie Hu et al. “Squeeze-and-Excitation Networks.” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 7132–7141. DOI: 10.1109/cvpr.2018.00745 (cit. on pp. 78, 79, 85).

[272] Li Deng. “The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web].” In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477 (cit. on p. 82).

[273] Moritz Menze and Andreas Geiger. “Object scene flow for autonomous vehicles.” In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. DOI: 10.1109/cvpr.2015.7298925 (cit. on p. 88).

[274] Joseph Redmon and Ali Farhadi. “Yolov3: An incremental improvement.” In: *arXiv preprint arXiv:1804.02767* (2018) (cit. on pp. 91, 92).

[275] Glenn Jocher et al. *ultralytics/yolov3: v9.5.0 - YOLOv5 v5.0 release compatibility update for YOLOv3*. 2021. DOI: 10.5281/ZENODO.4681234 (cit. on p. 91).

[276] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection.” In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017, pp. 2980–2988. DOI: 10.1109/iccv.2017.324 (cit. on pp. 91, 92).

[277] Tsung-Yi Lin et al. “Feature Pyramid Networks for Object Detection.” In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, July 2017, pp. 2117–2125. DOI: 10.1109/cvpr.2017.106 (cit. on pp. 91, 92).

[278] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019 (cit. on p. 91).

[279] Stefano Pini et al. "Video Synthesis from Intensity and Event Frames." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2019, pp. 313–323. DOI: 10.1007/978-3-030-30642-7_28 (cit. on p. 102).

[280] Yuhuang Hu et al. "DDD20 End-to-End Event Camera Driving Dataset: Fusing Frames and Events with Deep Learning for Improved Steering Prediction." In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, Sept. 2020, pp. 1–6. ISBN: 978-1-7281-4150-3. DOI: 10.1109/itsc45102.2020.9294515 (cit. on p. 102).

[281] Timo Stoffregen et al. "Reducing the Sim-to-Real Gap for Event Cameras." In: *Computer Vision – ECCV 2020*. Springer International Publishing, 2020, pp. 534–549. DOI: 10.1007/978-3-030-58583-9_32 (cit. on pp. 103, 111, 113, 117, 127, 128).

[282] Yaroslav Ganin and Victor Lempitsky. "Unsupervised Domain Adaptation by Backpropagation." In: ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1180–1189 (cit. on pp. 103, 105, 106, 124, 128, 133).

[283] Mingsheng Long et al. "Learning transferable features with deep adaptation networks." In: *International conference on machine learning*. PMLR. 2015, pp. 97–105 (cit. on pp. 103, 105, 108, 124, 128, 133).

[284] Mohammad Reza Loghmani et al. "Unsupervised Domain Adaptation Through Inter-Modal Rotation for RGB-D Object Recognition." In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020), pp. 6631–6638. DOI: 10.1109/lra.2020.3007092 (cit. on pp. 103–106, 108, 113, 118, 122, 133, 134, 140).

[285] Ruijia Xu et al. "Larger Norm More Transferable: An Adaptive Feature Norm Approach for Unsupervised Domain Adaptation." In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019, pp. 1426–1435. DOI: 10.1109/iccv.2019.00151 (cit. on pp. 103, 105, 109, 124, 128, 133).

[286] Yves Grandvalet and Y. Bengio. "Semi-supervised Learning by Entropy Minimization." In: *Adv. Neural Inform. Process. Syst.* Vol. 367. Jan. 2004, pp. 281–296 (cit. on pp. 103, 109, 124, 128, 131–133).

[287] Kuniaki Saito et al. "Maximum Classifier Discrepancy for Unsupervised Domain Adaptation." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 3723–3732. DOI: 10.1109/cvpr.2018.00392 (cit. on p. 105).

[288] Zhijie Deng et al. "Cluster Alignment With a Teacher for Unsupervised Domain Adaptation." In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2019, pp. 9944–9953. DOI: 10.1109/iccv.2019.01004 (cit. on p. 105).

[289] Hui Tang and Kui Jia. "Discriminative Adversarial Domain Adaptation." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.04 (Apr. 2020), pp. 5940–5947. DOI: 10.1609/aaai.v34i04.6054 (cit. on p. 105).

[290] Jiaolong Xu et al. "Self-Supervised Domain Adaptation for Computer Vision Tasks." In: *IEEE Access* 7 (2019), pp. 156694–156706. DOI: 10.1109/access.2019.2949697 (cit. on pp. 105, 107, 124, 128, 133, 134, 140).

[291] Fabio M. Carlucci et al. "Domain Generalization by Solving Jigsaw Puzzles." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019, pp. 2229–2238. DOI: 10.1109/cvpr.2019.00233 (cit. on pp. 105, 129).

[292] Konstantinos Bousmalis et al. "Domain separation networks." In: *Advances in neural information processing systems* 29 (2016), pp. 343–351 (cit. on p. 105).

[293] Yanghao Li et al. "Adaptive Batch Normalization for practical domain adaptation." In: *Pattern Recognition* 80 (Aug. 2018), pp. 109–117. DOI: 10.1016/j.patcog.2018.03.005 (cit. on p. 105).

[294] Yanghao Li et al. "Revisiting Batch Normalization For Practical Domain Adaptation." In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. 2017 (cit. on p. 105).

[295] Woong-Gi Chang et al. "Domain-Specific Batch Normalization for Unsupervised Domain Adaptation." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019, pp. 7354–7362. DOI: 10.1109/cvpr.2019.00753 (cit. on p. 105).

[296] Rui Gong et al. "DLOW: Domain Flow for Adaptation and Generalization." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019, pp. 2477–2486. DOI: 10.1109/cvpr.2019.00258 (cit. on p. 105).

[297] Judy Hoffman et al. "Cycada: Cycle-consistent adversarial domain adaptation." In: *International conference on machine learning*. PMLR. 2018, pp. 1989–1998 (cit. on p. 105).

[298] German Ros et al. "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3234–3243. DOI: 10.1109/cvpr.2016.352 (cit. on p. 105).

[299] Stephan R Richter et al. "Playing for data: Ground truth from computer games." In: *European conference on computer vision*. Springer. 2016, pp. 102–118. DOI: 10.1007/978-3-319-46475-6_7 (cit. on p. 105).

[300] Emanuele Alberti et al. "IDDA: A Large-Scale Multi-Domain Dataset for Autonomous Driving." In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020), pp. 5526–5533. DOI: 10.1109/lra.2020.3009075 (cit. on p. 105).

[301] Judy Hoffman et al. "Fcns in the wild: Pixel-level adversarial and constraint-based adaptation." In: *arXiv preprint arXiv:1612.02649* (2016) (cit. on p. 105).

[302] Yi-Hsuan Tsai et al. "Learning to Adapt Structured Output Space for Semantic Segmentation." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 7472–7481. DOI: 10.1109/cvpr.2018.00780 (cit. on pp. 105, 110, 131, 132).

[303] Tuan-Hung Vu et al. "ADVENT: Adversarial Entropy Minimization for Domain Adaptation in Semantic Segmentation." In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019, pp. 2517–2526. DOI: 10.1109/cvpr.2019.00262 (cit. on pp. 105, 110, 131, 132).

[304] Yang Zhang et al. "Curriculum Domain Adaptation for Semantic Segmentation of Urban Scenes." In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017, pp. 2020–2030. DOI: 10.1109/iccv.2017.223 (cit. on p. 105).

[305] Jing Wang and Kuangen Zhang. "Unsupervised Domain Adaptation Learning Algorithm for RGB-D Stairway Recognition." In: *Instrumentation* 6(2) (2019), pp. 21–29 (cit. on p. 105).

[306] Xiao Li et al. "Domain adaptation from RGB-D to RGB images." In: *Signal Processing* 131 (Feb. 2017), pp. 27–35. DOI: 10.1016/j.sigpro.2016.07.018 (cit. on p. 106).

[307] Arthur Gretton et al. "Optimal kernel choice for large-scale two-sample tests." In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012 (cit. on p. 108).

[308] Dino Sejdinovic et al. "Equivalence of distance-based and RKHS-based statistics in hypothesis testing." In: *The Annals of Statistics* 41.5 (Oct. 2013). DOI: 10.1214/13-aos1140 (cit. on p. 108).

[309] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *arXiv preprint arXiv:1411.1792* (2014) (cit. on pp. 114, 117).

[310] Mohammad Mostafavi et al. "Event-Intensity Stereo: Estimating Depth by the Best of Both Worlds." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 4258–4267 (cit. on p. 118).

[311] Farzeen Munir et al. "LDNet: End-to-End Lane Marking Detection Approach Using a Dynamic Vision Sensor." In: *IEEE Transactions on Intelligent Transportation Systems* (2021) (cit. on p. 118).

[312] Manasi Muglikar et al. “Event Guided Depth Sensing.” In: *arXiv preprint arXiv:2110.10505* (2021) (cit. on p. 118).

[313] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift.” In: *International conference on machine learning*. 2015, pp. 448–456 (cit. on p. 122).

[314] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: *The journal of machine learning research* (2014) (cit. on p. 122).

[315] Inigo Alonso and Ana C. Murillo. *Ev-SegNet*. [Online; accessed 7. Jan. 2022]. Jan. 2022. URL: https://github.com/Shathe/Ev-SegNet (cit. on p. 123).

[316] L Van Der Maaten and G Hinton. “Visualizing High-Dimensional Data Using t-SNE.” In: *Journal of Machine Learning Research* 9 (Nov. 2008), pp. 2579–2605 (cit. on p. 126).

[317] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.” In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017, pp. 618–626. DOI: 10.1109/iccv.2017.74 (cit. on pp. 126, 127).

[318] Kaichao You et al. “Universal Domain Adaptation.” In: *Domain Adaptation in Computer Vision with Deep Learning*. Springer International Publishing, 2020, pp. 195–211. DOI: 10.1007/978-3-030-45529-3_11 (cit. on p. 129).

[319] Zhangjie Cao et al. “Partial Transfer Learning with Selective Adversarial Networks.” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 2724–2732. DOI: 10.1109/cvpr.2018.00288 (cit. on p. 129).

[320] Jing Zhang et al. “Importance Weighted Adversarial Nets for Partial Domain Adaptation.” In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 8156–8164. DOI: 10.1109/cvpr.2018.00851 (cit. on p. 129).

[321] Pau Panareda Busto and Juergen Gall. “Open Set Domain Adaptation.” In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017, pp. 754–763. DOI: 10.1109/iccv.2017.88 (cit. on p. 129).

[322] Kuniaki Saito et al. “Open Set Domain Adaptation by Backpropagation.” In: *Computer Vision – ECCV 2018*. Springer International Publishing, 2018, pp. 156–171. DOI: 10.1007/978-3-030-01228-1_10 (cit. on p. 129).

[323] Silvia Bucci et al. “Self-Supervised Learning Across Domains.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP (99 2021), pp. 1–1. ISSN: 1939-3539. DOI: 10.1109/tpami.2021.3070791 (cit. on p. 130).

[324] Robert Geirhos et al. “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.” In: *arXiv preprint arXiv:1811.12231* (2018) (cit. on pp. 134, 140).

[325] Andreas Aakerberg et al. “Depth Value Pre-Processing for Accurate Transfer Learning based RGB-D Object Recognition.” In: *Proceedings of the 9th International Joint Conference on Computational Intelligence*. SCITEPRESS - Science and Technology Publications, 2017, pp. 121–128. DOI: 10.5220/0006511501210128 (cit. on p. 134).

[326] El Hadrami Cheikh Tourad and Mohsine Eleuldj. “Survey of Deep Learning Neural Networks Implementation on FPGAs.” In: *2020 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*. IEEE, Nov. 2020. DOI: 10.1109/cloudtech49835.2020.9365911 (cit. on p. 138).

[327] Maciej Wielgosz and Michał Karwatowski. “Mapping Neural Networks to FPGA-Based IoT Devices for Ultra-Low Latency Processing.” In: *Sensors* 19.13 (July 2019), p. 2981. DOI: 10.3390/s19132981 (cit. on p. 138).

[328] Daniel H. Noronha et al. “LeFlow: Enabling Flexible FPGA High-Level Synthesis of Tensorflow Deep Neural Networks.” In: *FSP Workshop 2018; Fifth International Workshop on FPGAs for Software Programmers*. 2018, pp. 1–8 (cit. on p. 138).

[329] Kamran Khan. *Xilinx DNN Processor (xDNN), Accelerating AI in Datacenters*. Tech. rep. 2018 (cit. on p. 138).

[330] Yijin Guan et al. “FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates.” In: *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, Apr. 2017. DOI: 10.1109/fccm.2017.25 (cit. on p. 138).

[331] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.” In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014. DOI: 10.3115/v1/d14-1179 (cit. on p. 138).

[332] Lakshmi Annamalai et al. “Event-LSTM: An Unsupervised and Asynchronous Learning-based Representation for Event-based Data.” In: *arXiv preprint arXiv:2105.04216* (2021) (cit. on p. 139).

[333] Asier Mujika et al. “Fast-Slow Recurrent Neural Networks.” In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017 (cit. on p. 139).

[334] Junyoung Chung et al. “Hierarchical Multiscale Recurrent Neural Networks.” In: *ICLR (Poster)*. OpenReview.net, 2017 (cit. on p. 139).

[335] Jan Koutnik et al. “A Clockwork RNN.” In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, 2014, pp. 1863–1871 (cit. on p. 139).

[336] Salah Hihi and Yoshua Bengio. “Hierarchical Recurrent Neural Networks for Long-Term Dependencies.” In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky et al. Vol. 8. MIT Press, 1996 (cit. on p. 139).

[337] Tom Brown et al. “Language Models are Few-Shot Learners.” In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901 (cit. on p. 139).

[338] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on p. 139).

[339] Dima Damen et al. “Scaling Egocentric Vision: The EPIC-KITCHENS Dataset.” In: *European Conference on Computer Vision (ECCV)*. 2018 (cit. on p. 140).