POLITECNICO DI MILANO

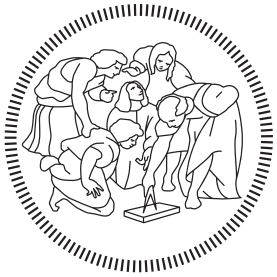
Facoltà di Ingegneria

Scuola di Ingegneria Industriale e dell'Informazione

Dipartimento di Elettronica, Informazione e Bioingegneria

Master of Science in

Computer Science and Engineering



A PointNet-based approach for neuromorphic vision in autonomous driving

Advisor: PROF. MATTEO MATTEUCCI

Co-advisor: DR. MARCO CANNICI

Master Graduation Thesis by:

YANNICK GIOVANAKIS

Student Id n. 883061

Academic Year 2020-2021

POLITECNICO DI MILANO

Facoltà di Ingegneria
Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria

Corso di Laurea Magistrale in
Computer Science and Engineering

Un approccio basato su PointNet per la visione neuromorfica nella guida autonoma

Relatore: PROF. MATTEO MATTEUCCI

Correlatore: DR. MARCO CANNICI

Tesi di Laurea Magistrale di:

YANNICK GIOVANAKIS
Matricola n. 883061

Anno Accademico 2020-2021

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

This template has been adapted by Emanuele Mason, Andrea Cominola and Daniela Anghileri: *A template for master thesis at DEIB*, June 2015. This version of the paper has been later adapted by Marco Cannici, March 2018.

"And I knew exactly what to do. But in a much more real sense, I had no idea what to do."

Micheal Scott

ACKNOWLEDGMENTS

My first thanks go to Prof. Matteo Matteucci, Marco Cannici and Simone Mentasti who not only gave me this great opportunity but were patient and always available during this time spent together (thank you Microsoft Teams!). A big thank you to my friends at *Owowiwowa* with whom I have shared these university years, my passions and various coloured drinks. Last but not least, a big thank you to my family: my parents *Marion* and *Georg* and my other half *Francesca*, who all have supported and helped me through these years. You have been an inspiration for me and are the reason I have become who I am.

CONTENTS

Abstract xiii

1 INTRODUCTION 1
- 1.1 Main contributions . . . 3
- 1.2 Thesis structure . . . 3

2 AUTONOMOUS DRIVING SYSTEMS 5
- 2.1 Introduction . . . 5
- 2.2 Classification of Autonomous Vehicles . . . 6
- 2.3 Sensors of autonomous vehicles . . . 7
  - 2.3.1 Ultrasonic sensors . . . 8
  - 2.3.2 RADAR Sensors . . . 9
  - 2.3.3 LiDAR Sensors . . . 9
  - 2.3.4 Positional sensors . . . 10
  - 2.3.5 Vision Sensors . . . 11

3 NEUROMORPHIC VISION AND EVENT BASED CAMERAS 13
- 3.1 Neuromorphic systems . . . 13
- 3.2 Neuromorphic Vision . . . 14
  - 3.2.1 Address-Event Representation AER . . . 17
- 3.3 Event-based sensors and cameras . . . 18
  - 3.3.1 The Dynamic Vision Sensor (DVS) . . . 18
  - 3.3.2 The Asynchronous Time-based Image Sensor (ATIS) . . . 20

4 EVENT-BASED VISION FOR AUTONOMOUS DRIVING 23
- 4.1 Event-Based Datasets for Autonomous Driving . . . 23
  - 4.1.1 N-CARS Dataset . . . 24
  - 4.1.2 Prophesee's GEN1 Automotive Detection Dataset . . . 24
  - 4.1.3 DAVIS Driving Dataset 2017 (DDD17) . . . 24
  - 4.1.4 MVSEC Dataset . . . 25
- 4.2 Representation of event-based data . . . 25
  - 4.2.1 Event Frames and Event Count Frames . . . 25
  - 4.2.2 Surface of active events . . . 27
  - 4.2.3 Leaky integrate-and-fire (LIF) neurons . . . 27
  - 4.2.4 Voxel Grid . . . 27
  - 4.2.5 Event Spike Tensor (EST) . . . 28
- 4.3 Handcrafted features: time surfaces . . . 29
- 4.4 Spiking Neural Networks (SNN) . . . 30
- 4.5 Convolutional Neural Networks (CNN) . . . 31
  - 4.5.1 Convolutional layer . . . 32
  - 4.5.2 Pooling layer . . . 33
  - 4.5.3 Optical flow, depth and egomotion . . . 33

4.5.4 Object detection . . . 35
4.5.5 Semantic segmentation . . . 36
4.5.6 Active perception . . . 37
4.6 The future for event-based cameras in autonomous driving . . . 38
5 EVENT-BASED LEARNING APPROACH WITH POINTNET 41
5.1 Point clouds . . . 41
5.2 PointNet architecture . . . 42
5.2.1 Symmetry Function for Unordered Input . . . 43
5.2.2 Local and Global Information Aggregation . . . 44
5.2.3 Joint Alignment Network . . . 44
5.2.4 Implementation and hyperparameters . . . 45
5.3 PointNet++ Architecture . . . 46
5.3.1 Set Abstraction level . . . 47
5.3.2 Feature Propagation Layer . . . 48
5.3.3 Robust Feature Learning under Non-Uniform Sampling Density . . . 48
5.3.4 Implementation and hyperparameters . . . 50
5.4 Performance analysis of PointNets . . . 51
5.4.1 Evaluation on ModelNet40 . . . 52
5.4.2 Evaluation on ScanNet . . . 52
5.5 Event-based data as PointNet input . . . 53
5.5.1 Sliding time-windows . . . 55
5.5.2 Noise processing . . . 56
5.6 Event-based classification on PointNet-based architectures . . . 57
5.6.1 Configuration of experiments . . . 58
5.6.2 Evaluation metrics . . . 59
5.6.3 Experimental results . . . 59
5.7 Event-based semantic segmentation on PointNet-based architectures . . . 62
5.7.1 Configuration of experiments . . . 63
5.7.2 Evaluation metrics . . . 64
5.7.3 Experimental results . . . 65
5.8 Transfer Learning . . . 68
5.9 Detection Pipeline proposal . . . 69
5.10 Future developments . . . 70

BIBLIOGRAPHY 73

LIST OF FIGURES

Figure 2.1 Autonomous vehicle classification according to SAE . . 7
Figure 2.2 Sensors of autonomous vehicles 8
Figure 2.3 LiDAR Pointcloud 9
Figure 3.1 Schematic drawing of biological retina 15
Figure 3.2 Circuit layout of an artificial retina 16
Figure 3.3 AER communication protocol 17
Figure 3.4 The DVS pixel 18
Figure 3.5 DVS output example 19
Figure 3.6 DAVIS output example 20
Figure 3.7 The ATIS pixel and its functioning 21
Figure 4.1 Samples taken from the NCARS and Prophesee's GEN1 dataset. 24
Figure 4.2 Event stream generated from a sample of the DAVIS Driving Dataset 26
Figure 4.3 Constant time frame and constant event frame comparison 26
Figure 4.4 EST Framework 29
Figure 4.5 Functioning of the convolution layer in CNNs 32
Figure 4.6 Functioning of the pooling layer in CNNs 33
Figure 4.7 Example of the ECN architecture 34
Figure 4.8 Example of object detection for event-based vision . . . 35
Figure 4.9 DDD17 labels generated by Ev-SegNet 37
Figure 5.1 PointNet pipeline 43
Figure 5.2 PointNet++ architecture 48
Figure 5.3 Schema of MSG and MSR in PointNet++ 49
Figure 5.4 NCars sample comparison: image reconstruction vs. point cloud 54
Figure 5.5 Sliding time window mechanism 55
Figure 5.6 Noise pre-processing 57
Figure 5.7 Majority voter 59
Figure 5.8 DDD17 Semantic segmentation labels 63
Figure 5.9 Semantic segmentation of PointNet 65
Figure 5.10 Semantic segmentation of PointNet++ 68
Figure 5.11 Example of transfer learning output 68

Figure 5.12 DBSCAN as object detector 70

LIST OF TABLES

Table 4.1 Summary of the most common type of event data representations 25
Table 5.1 Effect of input and feature transformations on the accuracy on the ModelNet40 dataset. 45
Table 5.2 3D object classification on ModelNet40 52
Table 5.3 Segmentation Performance on ScanNet 53
Table 5.4 PointNet performance on NCars. All samples use temporal difference and no polarity. 60
Table 5.5 PointNet performance on NCars. Additional features or transformations do not affect the performance in a significant way. 61
Table 5.6 Performance of PointNet with time window compared to state-of-the art models. 62
Table 5.7 PointNet performances on DDD17 66
Table 5.8 PointNet++ performances on DDD17. 67

ABSTRACT

The rapid improvement of machine learning and computer vision systems has fueled the development of self-driving vehicles. These systems rely on their underlying hardware, such as automotive image sensors, which are being developed with great intensity to effectively tackle the challenges imposed by real-world scenarios. In this context, bio-inspired silicon retinas or *event-based cameras* offer a wide range of characteristics that allow them to be a valid candidate for artificial vision: very high temporal resolution and low latency (both in the order of microseconds), very high dynamic range (140 dB vs. 60 dB of standard cameras), and low power consumption [3]. Unlike traditional cameras, which generate frames at a constant rate, event-based cameras respond to brightness changes in the scene *asynchronously* and *independently* for each pixel on the silicon retina, generating a variable data-rate stream of *events*. However, because event cameras work in a fundamentally different way from standard cameras, novel methods are required to process their output and unlock their potential.

In this thesis, we explore and develop deep learning architectures capable of successfully processing streams of events generated by neuromorphic image sensors for image classification and semantic segmentation tasks in automotive contexts. The main goal of our work was to exploit the sparse asynchronous nature of event-driven data without resorting to costly frame reconstruction techniques. To achieve this feature, we based our efforts on *PointNet* [1] and *PoinetNet++* [2] which are state-of-the-art architectures for *point clouds*, an important type of geometric data structure that shares all characteristics with the data generated by event cameras. Our architectures are trained and validated on different event-based automotive datasets. The results are then compared to state-of-the-art models.

SOMMARIO

Il secolo scorso è stato segnato da incredibili progressi scientifici e sviluppi tecnologici. Tra questi, forse quello che più ha cambiato la vita di tutti è stato lo sviluppo dei primi calcolatori elettronici. La crescita esponenziale dei progressi tecnologici nel campo del calcolo digitale sta dando i suoi frutti nella vita di tutti i giorni: oggi quasi ogni dispositivo tecnologico contiene una piccola unità di calcolo capace di eseguire intensi calcoli in una frazione di tempo. Non c'era da stupirsi che presto i computer sarebbero stati una parte fondamentale della vita umana. Fin dall'alba dei primi computer, la questione è stata fino a dove può arrivare la potenza di calcolo. Questa questione tecnologica ma anche filosofica è stata affrontata da molti ricercatori negli ultimi decenni in modo tale che la coesistenza tra uomo e macchina è diventata simbiotica e il miglioramento dei computer elettronici è stato spesso ispirato dall'organismo che conosciamo meglio: il corpo umano.

Basti dire che lo sviluppo più importante nell'area della tecnologia dell'informazione degli ultimi anni, *l'apprendimento automatico*, trae i suoi prodotti dal concetto di apprendimento come lo conosciamo negli esseri viventi. Un chiaro esempio di tecnologia bio-ispirata è rappresentato dalle architetture di reti neurali di apprendimento profondo, che basano il loro funzionamento sul concetto di "neurone artificiale", il gemello digitale del neurone come lo conosciamo negli animali. Queste potenti architetture possono essere adattate per funzionare in una grande varietà di situazioni quotidiane, che vanno dai sistemi di elaborazione del linguaggio naturale alla predizione del comportamento umano, come l'analisi del sentimento. Una delle aree più interessanti dei modelli di deep learning è nel campo della visione artificiale, dove vengono costruiti modelli robusti che permettono alle macchine di estrarre informazioni utili dalle immagini facendo uso di rappresentazioni astratte dei dati di input. Il riconoscimento facciale, che si trova su ogni smartphone moderno, è solo una delle tante applicazioni dei modelli di deep learning per la visione artificiale, ma il suo funzionamento è tutt'altro che banale. Gli esseri umani hanno una capacità innata di riconoscere diversi tipi di volti sulla base di caratteristiche chiaramente visibili. Una rete neurale di apprendimento profondo, d'altra parte, deve imparare a riconoscere il proprio set di caratteristiche per identificare correttamente i diversi volti. Una rete di questo tipo è costruita in strati, ognuno dei quali è progettato per estrarre le caratteristiche in modo gerarchico e combinando caratteristiche più semplici per crearne altre complesse. In questo contesto, le architetture leader e più performanti sono basate su un tipo speciale di modelli di deep learning chiamati *Reti Neurali Convoluzionali* che basano il loro modus operandi sulla

nozione matematica di convoluzione per generare un insieme gerarchico di modelli generali.

Gli sviluppi bio-ispirati non si limitano al regno del software della tecnologia dell'informazione, ma trovano anche la loro strada nei recenti sviluppi dell'hardware come i sensori neuromorfici, dispositivi biologicamente ispirati che trovano le loro grandi applicazioni nelle telecamere a eventi. I sensori visivi tradizionali acquisiscono immagini complete ad un determinato frame-rate, specificato da un clock interno. Al contrario, le telecamere basate sugli eventi, come i sensori di visione dinamica (DVS) o i sensori di visione dinamica e a pixel attivi (DAVIS), usano sensori asincroni che campionano la luce in base alla dinamica dell'ambiente circostante, generando un flusso di dati variabile di "eventi" o "picchi" digitali, dove ogni evento rappresenta un cambiamento di luminosità. In modo analogo, i sensori audio basati sul silicio, o sensori audio dinamici (DAS), emulano il funzionamento di una coclea binoculare producendo eventi ogni volta che viene rilevata un'attività uditiva. I sensori di visione dinamica hanno recentemente attirato l'interesse di molti ingegneri dell'industria automobilistica, che sono costantemente alla ricerca di sensori più efficienti e avanzati. Tra questi, i sensori di percezione sono di grande interesse data la recente tendenza a sviluppare veicoli in grado di interagire con l'ambiente circostante. Questi sensori di visione devono avere caratteristiche molto specifiche: basso costo, alta gamma dinamica, resistenza alle luci tremolanti e tolleranza alle cattive condizioni atmosferiche. In questo contesto le telecamere basate su eventi sono il candidato perfetto per gli scenari di guida autonoma, dove la latenza, il basso consumo energetico e un'alta risoluzione temporale sono elementi chiave. Infatti, le telecamere basate su eventi generano eventi solo quando un pixel percepisce un cambiamento di luminosità. L'alta risoluzione temporale permette di rilevare questi cambiamenti nella scena molto frequentemente, una caratteristica desiderabile per i sensori automobilistici. Inoltre, i dati prodotti non sono ridondanti in quanto solo le informazioni variabili vengono percepite, mentre ciò che rimane statico nella scena dal punto di vista dell'auto e quindi di scarsa rilevanza non viene percepito dalla telecamera ad eventi. Posch et al. sono stati tra i primi a proporre sensori basati sugli eventi per i sistemi di assistenza automatica alla guida (ADAS).

Questo nuovo cambio di paradigma imposto dalle telecamere basate sugli eventi richiede architetture progettate in modo specifico che possano gestire i flussi di eventi ed elaborarli. Le reti neurali spike (SSN) sono architetture che cercano di sfruttare ulteriormente gli aspetti neurobiologici della computazione neurale artificiale. I neuroni "spiking" comunicano tra loro tramite sequenze di picchi, chiamati "treni di picchi", che codificano informazioni spaziali e temporali. Una volta che tale neurone accumula abbastanza informazioni rilevanti, emette uno spike. Questo permette alla rete di rilevare gli stimoli in diverse istanze temporali. Poiché la natura degli eventi generati

dalle telecamere basate sugli eventi è simile a quella degli spike, le architetture SNN sono state utilizzate nelle prime fasi della ricerca nel campo della computer vision basata sugli eventi. Tuttavia, la performance delle Spiking Neural Networks è limitata dalla natura discreta degli eventi, che li rende modelli non differenziabili che sono molto difficili da addestrare con approcci tradizionali di discesa del gradiente. Soluzioni di ripiego sono state proposte da alcune ricerche, utilizzando i filtri Gabor come pesi nelle reti o addestrando prima una CNN e convertendo i pesi in una SNN. Le soluzioni ottenute, tuttavia, sono sub-ottimali e tipicamente le prestazioni sono inferiori alle CNN convenzionali sui frame. Altri approcci integrano eventi in specifici intervalli di tempo per generare fotogrammi ai quali vengono applicate le CNN tradizionali. Questo metodo produce prestazioni robuste, date le CNN potenti e ottimizzate disponibili oggi. Tuttavia, questi approcci non sfruttano pienamente la natura sparsa e asincrona dei flussi di eventi.

Il nostro approccio parte dall'analisi dei dati generati da un sensore basato sugli eventi. Una telecamera basata sugli eventi con una griglia di pixel di dimensioni MxN genera un flusso di eventi I:

$$\Psi = \{e_i\}_{i=1}^{I}, \text{ with } e_i = (x_i, y_i t_i, p_i) \tag{0.1}$$

dove $(x_i, y_i) \in [1, ..., M]x[1, ..., N]$ sono le coordinate del pixel che genera l'evento, $t_i \geqslant 0$ è l'istante temporale in cui l'evento è generato e $p_i \in \{-1, 1\}$ o $p_i \in \{0, 1\}$ (a seconda del tipo di sensore di eventi usato) è la *polarità* dell'evento. Questo flusso Ψ di eventi può essere visto come una nuvola di punti tridimensionale dove per ogni evento e_i le prime due dimensioni sono rappresentate dalle coordinate (x_i, y_i) e la terza dimensione è il tempo t_i. Questa analisi è stata il punto di partenza del nostro lavoro, che concentra i suoi sforzi nell'adattare e migliorare le architetture esistenti di elaborazione delle nuvole di punti, *Pointnet* e *PointNet++*, per gestire flussi basati su eventi generati da sensori di visione dinamica in scenari automobilistici.

1 INTRODUCTION

The last century has been marked by incredible scientific progress and technological developments. Among them, perhaps the one that changed everyone's life the most was the development of the first electronic calculators. The exponential growth of technological advances in the field of digital computation is bearing fruit in everyday life: nowadays almost every technological device contains a small computational unit capable of performing intense calculations in a fraction of time. It was to no surprise that soon, computers would have been a fundamental part of human life. Ever since the dawn of the first computers, the question has been how far computing power can go. This technological but also philosophical matter has been approached by many researchers in the last decades such that the coexistence between man and machine became symbiotic and the improvement of electronic computers were often inspired by the organism we know best: the human body.

Suffice it to say that the most prominent development in the area of information technology of recent years, *machine learning*, draws its products from the concept of *learning* as we know it in living beings. A clear example of bio-inspired technology is represented by deep learning neural network architectures, which base their functioning on the concept of the "artificial neuron", the digital twin of the neuron as we know it in animals. These powerful architectures can be adapted to function in a vast variety of everyday situations, ranging from *Natural Language Processing* systems to human behavioral prediction, in the likes of *Sentiment Analysis*. One of the most interesting areas of deep learning models is in the field of *Computer Vision* where robust models are build that allows machines to extract useful information from images making use of abstract representations of the input data. Facial recognition, found on every modern smartphone, is just one of many applications of deep learning models for computer vision but its functioning is far from trivial. Humans have an innate ability to recognize different types of faces based on clearly visible features. A deep learning neural network, on the other hand, must learn to recognize its own set of features in order to correctly identify different faces. Such a network is built in *layers*, each designed to extract features in a hierarchical manner and combining simpler features to create complex ones. In this context, the leading and best-performing architectures are based on a special kind of deep learning models called *Convolutional Neural Networks* which base their modus operandi on the mathematical notion of convolution to generate a hierarchical set of general patterns.

The bio-inspired developments are not limited to the software realm of information technology but also find their way in recent hardware developments like neuromorphic sensors, biologically-inspired devices that find their great applications in *event-based cameras* [3, 5–9] and *audio sensors* [8–10]. Traditional visual sensors acquire full images at a given frame rate, specified by an internal clock. By contrast, event-based cameras, such as *Dynamic Vision Sensors (DVS)* or *Dynamic and active-pixel vision sensors (DAVIS)*, use asynchronous sensors that sample light based on the dynamics of the surrounding environment, generating a variable data-rate stream of digital "events" or "spikes", with each event representing a change of brightness. In an analogous way, silicon-based audio sensors, or *Dynamic Audio Sensors (DAS)*, emulate the functioning of a binocular cochlea by producing events whenever an auditory activity is detected.

Dynamic vision sensors have recently attracted the interest of many engineers in the automotive industry, who are constantly looking for more efficient and advanced sensors. Among these, perception sensors are of great interest given the recent trend to develop vehicles capable of interacting with their surroundings. These vision sensors must have very specific characteristics: low cost, high dynamic range, resistance to flickering lights, and tolerance to bad weather conditions. In this context, event-based cameras are the perfect candidate for autonomous driving scenarios, where latency, low power consumption, and a high temporal resolution are key elements. Event-based cameras generate events only when a pixel perceives a change in brightness. The high temporal resolution allows these changes in the scene to be detected very frequently, a desirable feature for automotive sensors. In addition, the data produced is not redundant as only the variable information is perceived, while what remains static in the scene from the car's point of view and therefore of little relevance is not perceived by the event camera. Posch et al. [6] were among the first to propose event-based sensors for automatic driver assistance systems (ADAS).

This new shift of paradigm imposed by event-based cameras requires specifically designed architectures that can handle event streams and process them. *Spiking Neural Networks (SSN)* [11–14] are architectures that try to exploit even further the neurobiological aspects of artificial neural computation. Spiking neurons communicate between each other by sequences of spikes, called *spike trains*, which encode spatial and temporal information. Once such a neuron accumulates enough relevant information it emits a spike. This allows the network to detect stimuli at different temporal instances. Since the nature of events generated by event-based cameras is similar to that of spikes, SNN architectures were used in the early stages of research in the field of event-based computer vision. However, the performance of Spiking Neural Networks is limited by the discrete nature of the events, which renders them non-differentiable models that are very hard to train with traditional gradient

descent approaches. Work-around solutions have been proposed by some researchers, using Gabor filters as weights in networks [15] or training on a CNN first and converting weights to a SNN [16]. The obtained solutions, however, are sub-optimal, and typically the performance is lower than conventional CNNs on frames. Other approaches integrate events at specific time intervals in order to generate frames to which traditional CNNs are applied [68, 18]. This method yields robust performances given the powerful and optimized CNNs available today. Nevertheless, these approaches do not fully exploit the sparse and asynchronous nature of event streams.

Our approach start with the analysis of data generated by an event-based sensor. An event-based camera with a pixel-grid size MxN generates a stream of I events:

$$\Psi = \{e_i\}_{i=1}^{I}, \text{ with } e_i = (x_i, y_i t_i, p_i) \tag{1.1}$$

where $(x_i, y_i) \in [1, ..., M]x[1, ..., N]$ are the coordinates of the pixel generating the event, $t_i \geqslant 0$ the timestamp at which the event is generated and $p_i \in \{-1, 1\}$ or $p_i \in \{0, 1\}$ (depending on the kind of event sensor used) being the *polarity* of the event. This stream Ψ of events can be seen as a three-dimensional *point cloud* where for each event e_i the first two dimensions are represented by the coordinates (x_i, y_i) and the third dimension being the time t_i. This analysis was the starting point of our work, which focuses its effort on adapting and improving existing point cloud processing architectures, *Pointnet* [1] and *PointNet++* [2], to handle event-based streams generated by dynamic vision sensors in automotive scenarios.

1.1 MAIN CONTRIBUTIONS

In this thesis, we have adapted PointNet and PointNet++ to process event streams generated by neuromorphic vision sensors. In Chapter 5 we first explain the parallelism that exists between point clouds, which constitute the typical PointNet input, and event streams. We then introduce techniques for adapting PointNet-based architectures so that they can process and generate deep learning models on event-driven data. Several experiments have been carried out on different datasets generated by event-based vision sensors in the automotive domain. The experiments include mainly classification and segmentation tasks. In the last Sections, we explore the transfer learning and object detection capabilities of our network.

1.2 THESIS STRUCTURE

The thesis is structured so that it is complete and comprehensible in every aspect. Below is a short overview of the topics covered:

- In Chapter 2 we first introduce the field of autonomous driving, the central interest of our neural network application. The chapter gives an overview of what is meant by autonomous driving and how this field is evolving. This is followed by a description of the most common perceptual sensors found in a modern vehicle along with their applications.
- Chapter 3 focuses on the history of neuromorphic systems with particular emphasis on the neuromorphic vision sensors that form the basis of the event cameras used in this thesis. The different types of event cameras available and their applications will then be analyzed.
- Chapter 4 gives a broad overview of the literature and the state-of-the-art for neuromorphic vision in the field of autonomous driving. Datasets recorded with event cameras will be explored along with various data representation models. Finally, a description of how neural networks are able to process event streams and how the networks can be used in various tasks related to autonomous driving is given.
- Chapter 5 we introduce our contribution by presenting the architectures used and conclude with the evaluation of the experiments.

2

AUTONOMOUS DRIVING SYSTEMS

2.1 INTRODUCTION

The past decade has been a turning point for the automotive industry worldwide. Changes in our society and environment have required car manufacturers to address the new needs that have emerged. One of the main and best-known factors is environmental pollution, which has led car manufacturers to develop increasingly efficient combustion engines, culminating in the development of electrically powered vehicles that can compete with combustion models. But one of the big trends in the automotive sector is autonomous driving, a car that is capable of fulfilling the operational functions of a traditional car without a human operator. The potential of autonomous driving systems in conjunction with recent technological developments has led to a rapid rise in this sector. An autonomous driving system placed in a surrounding environment may be able to drastically reduce the number of fatalities caused by road accidents due to human carelessness or error. In addition, such a system could have great benefits in terms of efficiency, increasing the viability of the road network by reducing traffic and the need for cities to create parking spaces. Finally, the success of recent car-sharing models could exploit autonomous driving to create fleets of autonomous cars, leaving the human component as a mere passenger.

Vehicles with autonomous or assisted driving systems must be equipped with sensors capable of collecting stimuli from the surrounding environment, which will be processed by sophisticated computer models designed to recognize useful elements in the scene, and finally by actuators that can interact with the state of the car in response to need. This complex of systems must work in unison even in the most adverse situations to ensure predictable vehicle behavior. For example, a car with advanced driver-assistance systems (ADAS) that rely on recognizing road signs will have difficulty if the signs are not visible due to damage to the road surface or unfavorable weather conditions. Another example is uncommon driving situations, such as avoiding unexpected obstacles on the road or reacting to traffic policemen. It is clear that the underlying components of autonomous vehicles must be designed in such a way that all possible situations can be assessed without incurring prohibitive costs. Recently the term *Vehicle-to-Everything (V2X)* has been coined in response to the ever complexity of communication skills required by autonomous systems. Cars of these types are able to communicate with the surrounding environment, vehicles, and even pedestrians.

Through its instant communication V2X allows road safety applications such as forward collision warning, lane change warning, roadworks warning o emergency vehicle approaching. Other benefits will be brought to various sectors, such as the telecommunication industry who will greatly profit from the increasing data traffic required by autonomous systems. Cities will be able to re-organize their urban spaces as autonomous cars will be able to park themselves outside of the city center. On the other hand, other sectors will be negatively impacted by autonomous driving. The car insurance sector will be revolutionized, as it will have to deal with increasingly safe vehicles and the possibility of better analyzing events in the event of accidents. Premiums will decrease and the liability will shift from drivers towards manufacturers. [19] Vehicles with more advanced driving systems will make the human driver component obsolete, affecting an entire sector of people who make their living from driving. Researchers forecast [21] that by 2025 we'll see approximately 8 million autonomous or semi-autonomous vehicles on the road. There is no doubt that this decade will be crucial for our society to adapt to the new demands and risks imposed by autonomous vehicles.

2.2 CLASSIFICATION OF AUTONOMOUS VEHICLES

The Society of Automotive Engineers (SAE) defines 6 levels of driving automation ranging from 0 (fully manual) to 5 (fully autonomous) [20]:

- L0: vehicles of this category do not possess any autonomous capabilities as the human driver is continuously in control of speed and direction.
- L1: the driver continuously performs the longitudinal or lateral dynamic driving task while the other task is performed by the system. An example of L1 system is the parking assistant.
- L2: the driver must monitor the dynamic driving task and the driving environment at all times, however, the system performs the longitudinal and lateral driving task in a *defined* use case. The traffic jam assist is an example of L2 system.
- L3: the driver does not need to monitor the dynamic driving task nor the driving environment, but he must always be in a position to resume control. Systems of such kind perform the driving task in a defined use case and are able to recognize its performance and limits by requesting the action of the driver at need with sufficient margin. An example of L3 are highway patrol systems.
- L4: the driver is not required during the defined use case. Autonomous driving in urban environments is an example of L4 system.

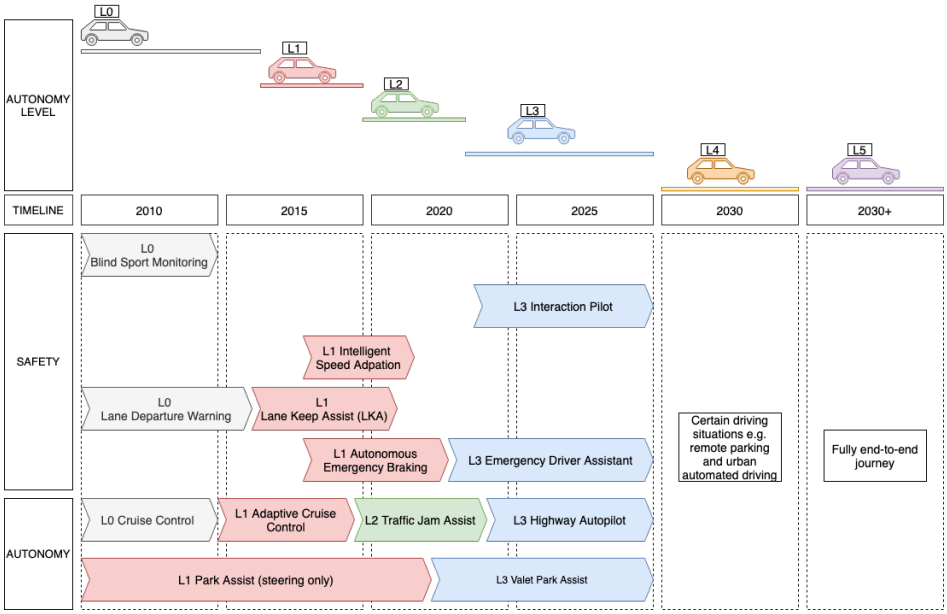


Figure 2.1: The autonomous vehicle classification and their timeline according the the KPMG study. Vehicles of kind L3 are current state-of-art.

- L5: the driver is not required during the entire journey as the system is able to perform the driving task in all situations. Full end-to-end journeys are L5 systems.

These levels have been adopted by the U.S. Department of Transportation.

A 2015 study published by KPMG [19] provided also an estimated development timeline as seen in Figure 2.1 for the above-mentioned systems. In the current state of affairs, major car manufacturers consider L3-type systems as their main production targets, as they are in line with available technology. Other companies such as Google are already looking ahead by working on fully autonomous vehicles at the L5 level. "The randomness of the environment such as children or wildlife can not be dealt with by today's technology", as stated by Markus Rothoff, Director of Autonomous Driving at Volvo, resonates with the majority of researchers who agree that the technology available today is not enough to make a step further than L3 models.

2.3 SENSORS OF AUTONOMOUS VEHICLES

Achieving automotive autonomy requires artificial intelligence to process and integrate data generated by a set of sensors that must allow the vehicle to get a useful and complete picture of the surrounding environment. The complexity of scenarios in everyday driving requires sensors specialized

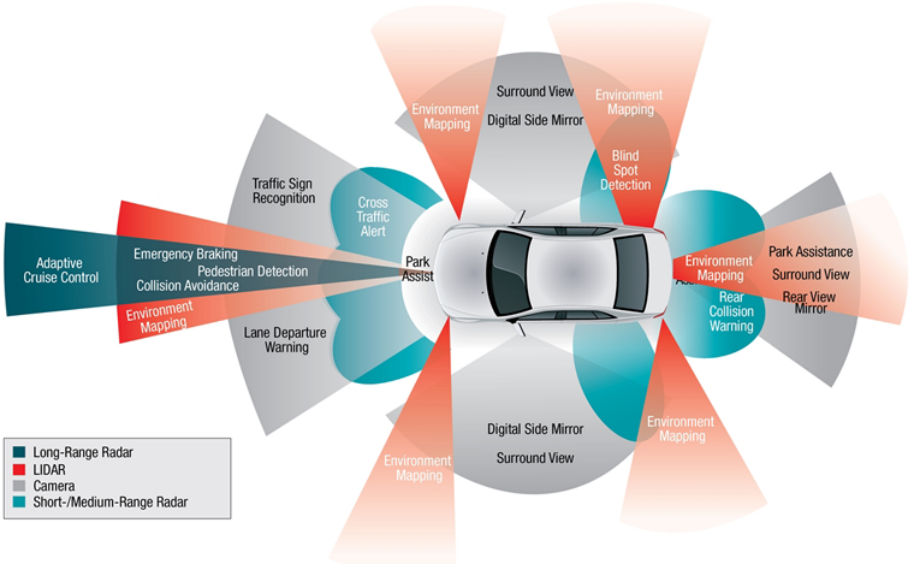


Figure 2.2: Modern vehicles are equipped with a suit of different kind of sensors, each specialized in a particulate task.
Source: SAE International

in different sensory tasks. Each sensor in the suite has its strengths and weaknesses. In the next Sections, we will analyze the most common sensors available on modern autonomous vehicles.

2.3.1 *Ultrasonic sensors*

Ultrasonic sensors are usually mounted in pairs or series of 4 on the front and rear of the machine. Their low cost makes this sensor an essential piece of equipment for modern vehicles for short-distance and low-speed detection. As showing in Figure 2.2 their main application is to detect obstacles in the immediate vicinity of the vehicle, imitating the navigation process of bats by generating ultrasonic impulses, i.e sound waves with frequencies above 20kHz, which are reflected by barriers. Using this information, the sensors are able to identify how far away the objects are and notify the vehicle in time. These sensors are used in parking space detection [22], which can be incorporated in self-parking systems, low-speed lateral collision avoidance [23] or blind spot detection. By their nature, the impulses are affected by the interference caused by climatic conditions as well as by the particular texture of the reflected object. Although they have a lower propagation speed than light or radio waves, ultrasonic sensors have found also applications in long-range scene evaluation [24].

Figure 2.3: PointPoint Cloud created by Velodyne Lidar's Alpha Prime sensor
Source: Velodyne

2.3.2 *RADAR Sensors*

Radar, or radio detection and ranging, uses reflected radio waves to sense surrounding objects, similar to LiDAR systems. These sensors are not a recent discovery, in fact, they have been used since the early 1900s when the potential of electromagnetic waves was used to detect the presence of ships in fog. With the advent of the first aircraft, radar became the dominant technology for exploring the surrounding airspace. Over the years radars able to detect various ranges of objects, depending on the emitted wavelengths. Radar sensors pick up metal objects best, seeing humans as partially translucent, and seeing plastic or wood as nearly transparent. Despite its drawbacks, this sensing method does give cars long-range sensing abilities that can see through dust, fog, rain, and snow. As shown in Figure 2.2 radars on are commonly used for adaptive cruise control [25, 26], emergency braking and collision avoidance [27–29]. Although less powerful than LiDAR, radars offer a wide range of benefits at a relatively low cost, making them one of the main sensors fitted to modern machinery.

2.3.3 *LiDAR Sensors*

Light Detection and Ranging or *LiDAR*, is a sensing method that measures the reflection of ultraviolet, visible, or near-infrared light off of objects as it scans in 360°. Multiple beams are emitted at angles to each other as the sensor array physically spins inside the device's housing to produce a three-dimensional picture of its surroundings. Repeating this process millions of times per second creates a precise, real-time 3D map of the environment as shown in Figure 2.3. An onboard computer can utilize this map for safe navigation.

LiDARs offer a wide range of sensing capabilities that unfortunately come currently at a very high cost. Recent studies [31] have focused on creating solid-state LiDARs, which perform a similar task and are significantly more cost-effective. Because they do not have a rotating component, solid-state LiDARs cannot create a 360° image without the aid of additional sensors.

Because of their powerful characteristics, LiDARs have attracted the focus of many researches. Navarro-Serment et al. used several SICK laser line scanners to build a LIDAR array for pedestrian detection and tracking in an indoor environment [32]. Schlosser et al. explored several aspects in LIDAR and RGB image fusion for CNNs for pedestrian detection [33]. The method proposed by Kun Zhou et al. [30] not only focuses on object detection and tracking but also recognizes lane markings and road features.

2.3.4 *Positional sensors*

Nowadays almost every modern vehicle is equipped with the *Global Positioning Sensor (GPS)*, a satellite-based radio navigation system owned by the United States government and operated by the United States Air Force. It is a global navigation satellite system (GNSS) that provides geolocation and time information to a GPS receiver anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites. Through GPS it is possible to determine the longitude and latitude (the position) within a precision of a couple of meters, as well as speed and course. Although widely used, GPS technology is not always sufficient to determine the exact position or speed of a vehicle. Several factors can affect the quality of the accuracy of GPS signals, such as the geometry of the satellites that make up the system, atmospheric conditions, the quality of signal receivers mounted on cars, radio interference, or signals reflected off buildings or walls ("multipath signals") or even strong solar storms. Everyone is familiar with the situation of losing GPS signals in tunnels, for example. Knowing the exact position and speed of a vehicle is, therefore, no trivial task.

The *Interial measurement unit (IMU)* is an electronic device that measures and reports a body's specific force, angular rate, and orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers. These measurements can then be shared with GPS systems to improve their accuracy in low-signal situations.

Another important category of positional sensors is defined by wheel speed sensors that record the speed of the wheels by measuring acceleration in both the longitudinal and vertical axis and communicate this data to the driving safety system. The steering angle sensor is used to determine where the front wheels are pointed. The wheel speed and angle sensor, combined with IMU and GPS system allow for a precise and robust system that can continuously

estimate the position, speed, and curse of AVs using well know controlling techniques like the *Kalman Filter* [34, 35].

2.3.5 *Vision Sensors*

Sight is certainly one of the fundamental tools for animals to evaluate the world around them. Not only can we see the visible spectrum of electromagnetic waves that are reflected in our eyes, but we make binocular vision our tool for assessing the distance and movement of objects. Vision sensors (or cameras, as they are also called) offer a good spatial resolution, but cannot directly measure distance or velocity. They rely on external light, so they see traffic signals and daytime scenes, but at night can miss pedestrians or wildlife not illuminated by headlights or street lights. Cameras have the advantage of being able to record and encode colors, which often yields valuable information (for example recognizing the color of the traffic lights) but analysis of color data is time-consuming. Infrared vision enhances that ability by picking up a thermal or heat signature to differentiate between humans and objects. Essential for noticing a difference in temperature on the road surface, alerting the AV to black ice. They can, however, be fooled in situations that human eyes would normally be able to handle, such as a brightly colored object against a bright sky or even painted objects that depict a different situation than reality. To enhance safety, existing implementations often mount eight or more 1,080-pixel cameras around the car running at 60 Hz amounting to a staggering 1.8 Gbytes of raw data per second generated.

The powerful and useful information carried by vision sensors combined with already well-defined image processing techniques for machine vision makes these sensors a must-have for AVs. Indeed, many are applications and researches involving vision sensors since images and videos can be processed by Deep Neural Networking. The state-of-the-art methods for object detection can be categorized into two main types: one-stage methods and two stage-methods. One-stage methods prioritize inference speed, and example models include YOLO [38], SSD [40] and RetinaNet [41]. Two-stage methods prioritize detection accuracy, and example models include Faster R-CNN [43] or Mask R-CNN [45]. An example is a method used by Jiwoong Choi et al. [42], where `YoloV3` is combined with Gaussian parameters and novelty loss function to increase bounding-box mean average precision (mAP). Combined with other sensors for speed and position estimation, raw camera data can be used to create depth-estimation of objections. [36].

In this study, the focus will be on a particular type of visual sensor called an event-based cameras [3, 5–9] which has a different working principle compared to the standard frame-based cameras, which leads to promising properties of low energy consumption, low latency, high dynamic range (HDR), and high temporal resolution. In the next Chapter, we will discuss

how these types of cameras work and how they can be used for vision in autonomous driving.

3 NEUROMORPHIC VISION AND EVENT BASED CAMERAS

Nature still outperforms the most powerful computers in tasks involving perception, sensing, and actuation like vision, audition, and motion control. Moreover, these tasks are performed by animals in such an energy-efficient way that cannot be matched by their artificial counterparts. It is, therefore, no surprise that many researchers have focused their efforts on studying, learning, and creating technology inspired by nature.

In this Chapter we will focus on bio-inspired technology, with particular attention to event-based cameras which are the main source for the data we will use in our study. Finally, we will introduce different types of Neural Networks that have been developed over the years that can process and operate on event-based data.

3.1 NEUROMORPHIC SYSTEMS

Engineers have taken inspiration from nature since the dawn of time. The solution to many problems requires a perfect understanding of how certain natural mechanisms work. The idea of applying concepts derived from nature to artificial information processing systems is older than you might think: the first work on artificial neurons was proposed in the 1940s and showed that they were capable of performing calculations and being used for machine learning [46, 47]. The first digital electronic circuits that mimic the functioning of neurons in neural systems were developed in the 1980s by Carver Med at CalTech [48–50], coining the term "neuromorphic" for systems that adopt the form, or *morph*, neural systems. In "Neuromorphic Electronic Systems"[50], Meads states that the advantage of biological information processing systems can be attributed principally to the use of elementary physical phenomena as computational primitives, and to the representation of information by the relative values of analog signals, rather than by the absolute values of digital signals. This approach requires adaptive techniques to mitigate the effects of component differences which naturally leads to systems that *learn* about their environment. Large-scale adaptive analog systems are more robust to component degradation and failure than are more conventional systems, and they use far less power. In his publication, Meads argues that the brain is 10 million times more efficient than the best technology we can imagine. During his research Meads found that transistors, or sets of transistors, share many interesting features with the nervous system. Both systems make use of electrical charge as analog state variable. To build more

complex systems, connections between components must be exclusive to certain links, while other components should be left out. This characteristic requires isolation by means of *energy barriers* that will not allow charges to leak where they should not. In the nervous system, such barriers are built by the difference in the dielectric constant between fat and aqueous solutions. In electronics, they are built by the difference in the band-gap between silicon and silicon dioxide, which is used as a gate dielectric in MOS technology. Furthermore a single transistor, in addition to providing amplification and gain to signals, computes a complex nonlinear function of its control and channel voltages. This function is not directly comparable to the functions that synapses evaluate using their pre-synaptic and post-synaptic potentials, but strategically arranged transistors are able to compute remarkably competent synaptic functions. Finally, Meads investigated the capacity of nervous systems to retain long-term memory. In microelectronics, electronic charges can be stored on a floating polysilicon node surrounded by silicon dioxide, an insulator. This charge can be stored *indefinitely* on the node. This technology has been used on commercial EEPROM, small non-volatile memory devices, since the 1970s. Meads, consequently, suggested that since elementary devices, transistors, are able to implement the basic functions of the nervous system, it should be feasible to build entire systems based on the organization of the nervous system.

Meads' studies have been greeted with great enthusiasm by researchers and engineers in the field of bio-engineering and have brought to light interesting new studies for the development of neuromorphic devices. The first neuromorphic electronic devices are implemented as *Very Large Scale Integration (VSLI)* integrated circuits or *systems-on-chips (SoCs)* on silicon sheets, which uses the transistor as is primary computational primitive, to model voltage-controlled neurons and synapses [53]. Further works focused on realizing biological computational primitives such as phototransduction, multiplication, inhibition, correlation, thresholding, or winner-take-all selection [48–50]. Over the following years, the greatest success of neuromorphic systems has been in the emulation of sensory signal acquisition and transduction, most notably in vision.

3.2 NEUROMORPHIC VISION

Traditional optical sensors base the acquisition of visual information on creating a sequence of digital images, called *frames*, which are recorded at discrete moments in time and therefore time-quantized at a given frame rate. Even state-of-the-art sensors are vulnerable to loss of information between frames. Generally speaking, recordings made by traditional optical sensors are suitable for most applications involving humans, as the loss of information due to a limited frame rate is tolerable. However, visual feedback loops, high-

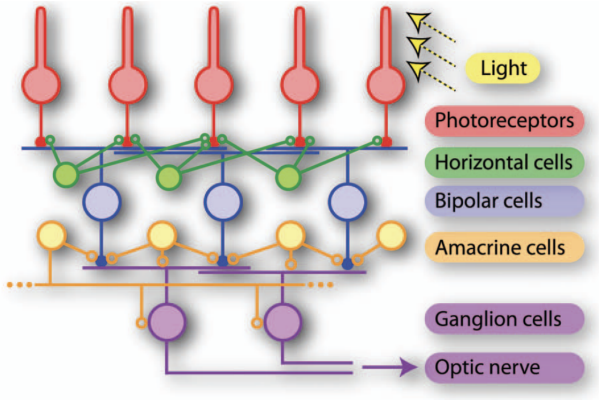


Figure 3.1: Simplified schematic drawing of the retina layers that are used to model the silicon retina. The lower layers encode information into spike patterns of ganglion cells and transmitted along their axons, which form the optic nerve, to different brain regions.
Source [55]

speed motor control, or autonomous navigation could be affected by this shortcoming.

Animals perceive visual information in a fundamentally different way. Visual systems in nature are driven and controlled by events happening in the surrounding environment that are captured asynchronously and in continuous time. As the frame-less model of biological vision is applied to artificial imaging systems, it means that control over the acquisition of visual information is no longer exerted externally on a collection of pixels, but rather that decision-making is delegated to each pixel. In frame-based models regardless of whether the information — or a portion of it — has changed since the previous frame was collected, each recorded frame conveys the information from all pixels. As a result of this process, the acquired image data has a higher degree of redundancy. Since in many applications the data generated must be consumed by other devices (e.g. Neural Networks) large data volumes increase memory size and processing power demands.

The artificial retina was one of the first neuromorphic electronic devices to be introduced [54]. The retina is a thin sheet of tissue that lines the orb of the eye and converts raw light into the nerve signals that the brain interprets as visual images. By closer inspection, the retina is made up of five layers of cells, each of which transmits information both vertically (from one layer to the next) and horizontally (between adjacent cells in the same layer). The top three layers are the best understood: photoreceptors (rods and cones), horizontal cells, and bipolar cells. The rods and cones transform light into electrical signals; the horizontal cells, meanwhile, respond to the average light intensity in their neighborhood. Bipolar cells transmit a signal corresponding

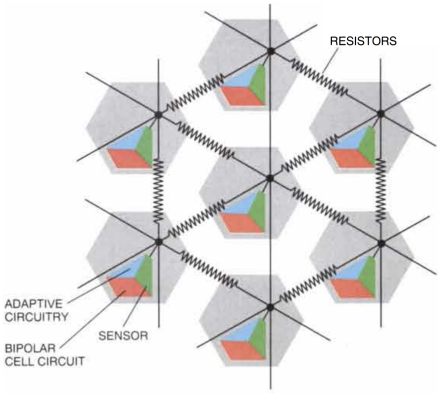


Figure 3.2: The hexagonal pattern of artificial retina cells. The voltage at each node in the horizontal cell network presents a spatially weighted average of the photoreceptor inputs to the network. Changing the values of the resistors modulates the effective area over which signals are averaged. The final output of each pixel in the silicon retina comes from an amplifier that senses the voltage difference between the output of a photoreceptor unit and the corresponding node in the horizontal cell network.
Source [54]

to the ratio of the signals from rods and horizontal cells through the ganglion cells, where it is further processed before being delivered to the brain. Figure 3.1 illustrates a simplified version of the retina schematic.

Silicon retinas are structured devices that try to model the three top layers of the retina. This artificial retina uses silicon area doped with impurities as the basis for transistors and photosensors, polysilicon is used to form wires and resistors and metal lines act as low-resistance wires. The single silicon retinal cell is then connected through resistances and capacitors in a hexagonal pattern to recreate the horizontal layer as seen in Figure 3.2. This prototype of the first artificial retinas yield results closely related to its biological counterparts in terms of response to changes in light intensities. The shape of the response curve is similar to that of biological bipolar cells. Abrupt changes in light cause a large jump in output voltage, equal to the difference between the input and the previous average voltage stored in the resistive network. The response then settles to a plateau as the network computes the new average [54]. Retina chips with 100 times more pixels, as well as additional circuits that replicate the movement-sensitive and edge-enhancing functions of lower-level retina layers, would be needed for real vision. Successive studies led to the development of more complex artificial retinas that include lower-levels of the retina as well [56, 57].

The abstraction of two main forms of retinal ganglion cells, X- and Y-cells, as well as their related retina-brain pathways, appears to be extremely significant in the production of useful bioinspired artificial vision systems.

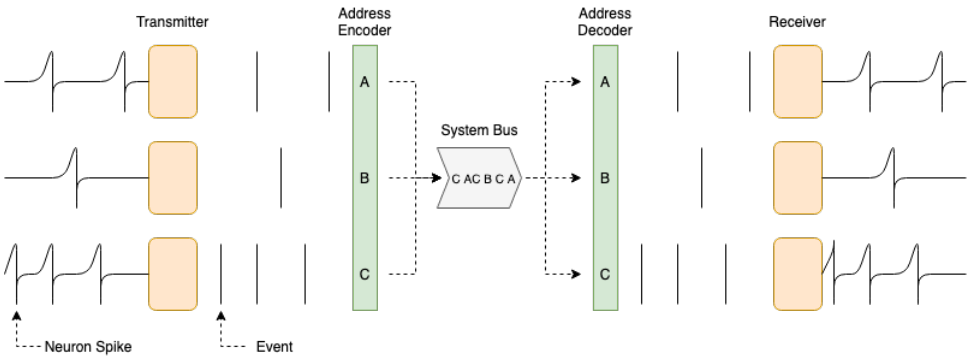


Figure 3.3: The AER communication protocol. Three neurons on the sending chip generate spikes which are interpreted as binary events. A binary address is generated by the AE and transmitted to the receiver chip by the bus line; the binary address is decoded to the binary event by the AD and spikes are emitted on the corresponding neurons of the receiver chip where the positions of the neurons are determined by the AD.

The Y-cells, also known as *Magno-cells*, are the base of the transient channel or *Magno-cellular pathway*. Y-Cells have short latency and rapidly conducting axons combined with a large receptive field and a transient response to changes. The X-cells are the cornerstone of the Parvo-cellular pathway, also known as the sustained channel. The axons of X-cells conduct more slowly and have longer latency. They have narrower receptive fields and react for longer periods. The transportation of detailed pattern, texture, and color details are most likely carried out by X-cells. Given their nature, Y-cells perform mainly general recognition and alerting functions, especially in peripheral vision. More detailed information, such as spatial features and color, is processed by the X-cells of the Parvo system. The implementation of Y-cells in neuromorphic systems led to the development of the *Dynamic Vision Sensors (DVS)* [58–60] and later to the *Asynchronous, Time-based Image Sensor (ATIS)* [61–63], which in addition to the abstraction of Y-cells implemented additional features taken from X-cells. In Section 3.3 we will provide an in-depth look at these two categories of sensors.

3.2.1 *Address-Event Representation AER*

The mechanism that helps neuromorphic systems' internal parts to interact is an essential feature of their architecture. Biological systems are normally made up of millions of neural cells that interact with one another through point-to-point connections in three dimensions. Such a complex layout is currently not possible for Very Large Scale Integration (VLSI) technologies due to wire complexity and spatial limits. Usually, a neuron produces bursts at frequencies between 10 and 1000 Hz, whereas the bandwidth of a current

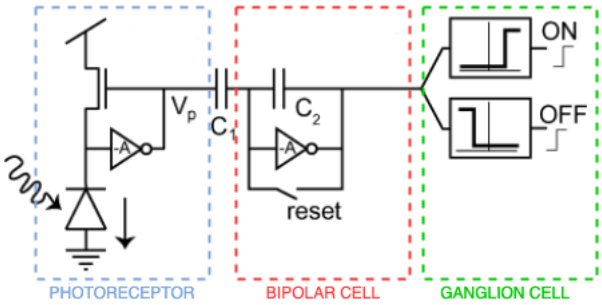


Figure 3.4: Circuit layout of the DVS pixel. The photoreceptor circuit has the desirable properties that it automatically controls individual pixel gain (by its logarithmic response) while at the same time responding quickly to changes. The differencing circuit then amplifies the changes with high precision whose output is fed to comparators. If the input of a comparator overcomes its threshold, an ON or OFF event is generated. *Source* [59]

bus is typically in the order of MHz. This allows several units to communicate at the same time by replacing the point-to-point architecture with a few connections and a packet-based, or *event-based*, protocol. The *Address-Event-Represention (AER)* is a protocol developed by the CalTech institute [64, 65] and became rapidly the most commonly used communication protocol for neuromorphic devices. Each computation unit is assigned to a unique address that identifies the source information, which is shared with other neurons on a physical bus through time-multiplexing (see Figure 3.3). Events consist of its unique address and additional information required. A timestamp is included only if events are processed on non-event-based architectures.

Because of its ability to simulate a point-to-point connection effectively, the AER protocol is the most widely used technology in neuromorphic electronics, especially in the field of artificial retinas.

3.3 EVENT-BASED SENSORS AND CAMERAS

As anticipated in Section 3.2, DVS and ATIS are powerful vision sensors based on silicon retinas. The sensors are "event-driven" instead of clock-driven and, like its biological model, respond to "natural" events happening in the scene they observed. *Event-based camera* are novelty cameras that use the DVS or ATIS as their building stone. Although they share common features, there is also some difference between cameras mounting DVS or ATIS.

3.3.1 *The Dynamic Vision Sensor (DVS)*

The Dynamic Vision Sensor models a 3-layer retina as shown in Figure 3.4. This sensor belongs to the category of *Termporal Difference (TD)* devices and

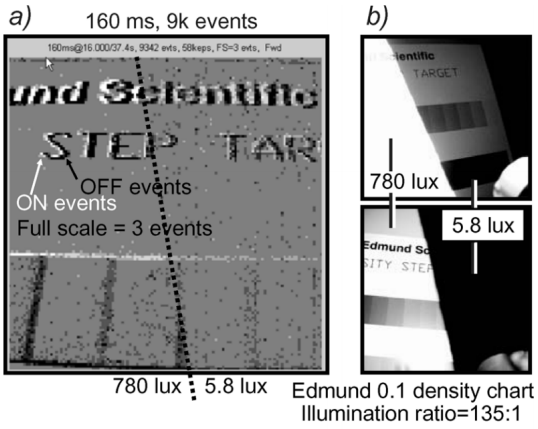


Figure 3.5: Histogrammed output from the vision sensor viewing with an illumination ratio of 135:1 (a shadow was cast to create this illumination step). The events are converted in gray-scale by using an integration interval of 160ms. (b) The same scene is photographed by a Nikon 995 digital camera to expose the two halves of the scene. *Source* [59]

is composed of an array of independent pixels. Every time a pixel detects a significant change in the illumination level - independently from the other pixels - an event is generated, containing the physical location of the pixel and a single bit to indicate if the illumination has increased or decreased. The event can be summarized as:

$$\mathbf{e} = [x, y, t, p] \tag{3.1}$$

where the event **e** indicates that a change in illumination has been detected by the pixel in position (x, y) at time t, with $p \in [0, 1]$ being the *polarity*. Conventionally $p = 1$ indicates an *ON event* (increase in in illumination) whereas $p = 0$ indicates a *OFF event* if a decrease in illumination has been detected. The information is the process by the AER protocol where the (x, y) location is encoded in the address of the emitted packets. DVS pixels capture its motion by emitting a continuous stream of events at a microsecond resolution that encodes changes in pixel log intensity. This type of sensor is very well suited for applications involving high-speed motion detection and analysis with a much higher time resolution compared to standard frame-based image sensors. The asynchronous stream of events, on the other hand, only contains change information and not absolute intensity information. This method of visual data acquisition and processing produces a pure dynamic vision device that closely resembles its paradigm, the human retina's transient pathway, without providing gray-level information in the scene. Examples of the dynamic range can be seen in Figure 3.5

Traditional computer vision algorithms cannot be readily applied as no static scene information is provided. As a solution to this problem was pro-

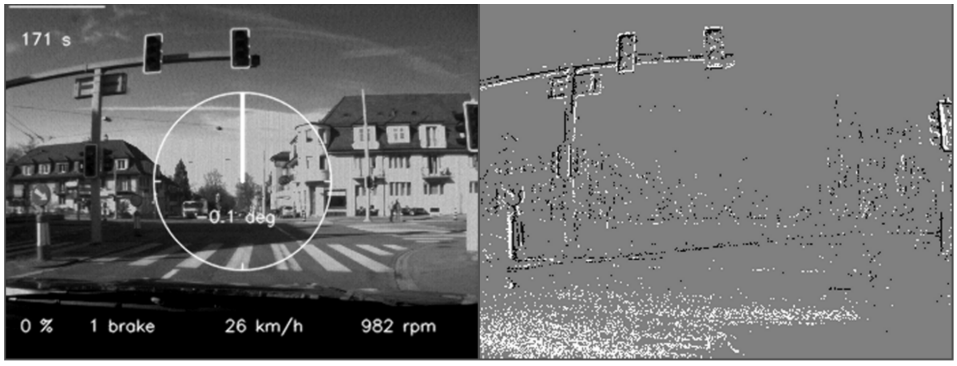


Figure 3.6: Urban scenario recorded with the DAVIS sensors. Frame-based images on the left (with additional driving measurements) and corresponding events integrated over the corresponding frame-rate based interval. *Source* [67]

posed in 2014 with the *Dynamic and Active Pixel Vision Sensor (DAVIS)* [66] which fuses the dynamic capabilities of the DVS pixel and the static, synchronous, and frame-rate based abilities of the *Active Pixel Sensor (APS)*. This dual readout is achieved through a shared photodiode and by adding five transistors to the original DVS pixel, increasing the DVS area by only about 5%. The combined static and dynamic output of the DAVIS makes it promising in a range of applications: The DVS output can be used to track and segment fast-moving objects, while the APS output allows for the recognition and classification of these objects using established machine vision techniques. Because tracking is done using only DVS events, the frame rate of the APS output can be set arbitrarily low. The combined advantage of the dual outputs makes the DAVIS sensor well-suited for mobile applications or distributed sensor networks with a tight power budget because it allows low latency at low system-level power consumption. In this context, the *"DDD17: End-to-End Driving Dataset"* [67] is one of the most complete datasets available for automotive applications that features recordings from several driving scenarios. The registrations were performed by a DAVIS346B prototype, containing a DAVIS APS+DVS camera, such that event-based and traditional frame-based data could be recorded at the same time, through the same optics. The camera resolution is 346×260 pixels. An example of the recording in the DDD17 dataset is shown in Figure 3.6.

3.3.2 *The Asynchronous Time-based Image Sensor (ATIS)*

The second type of neuromorphic devices, called *Exposure Measurement* devices, measure the *absolute* intensity of the pixel is whenever a pixel detects a change in the illumination. Its value is represented in the form of AER events by emitting two events whose relative inter-spike interval is propor-

tional to the measured intensity. This principle is also called asynchronous *pulse-width-modulation (PWM)* imaging. This type of devices allows also for the measurement of the actual intensity of the pixels while preserving an event-based approach, as opposed to DVS sensors that communicate only something that has changed in a particular location of the field of view.

The ATIS sensors *combine* the technology of the TD and EM devices [61]. The ATIS pixels convey transition and grayscale events to the readout periphery on their own when the events are arbitrated. An address encoder provides the pixel's array address, which is sent out on an asynchronous bit-parallel AER bus (Figure 3.7).

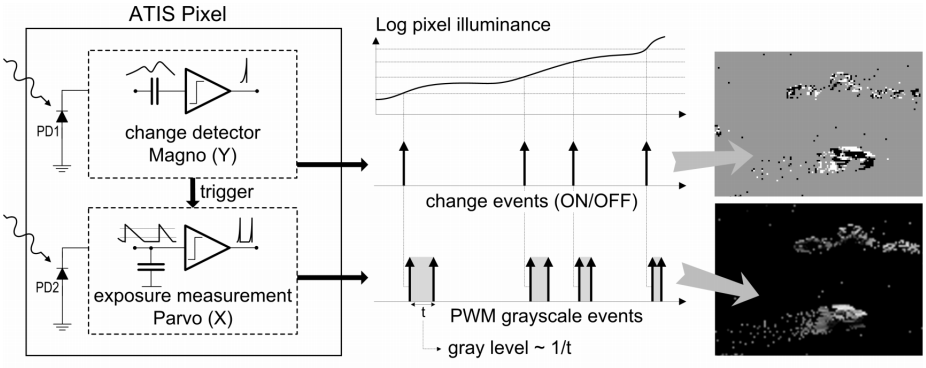


Figure 3.7: The ATIS pixel is composed of the DVS pixel that provides the change detector functionality and the Exposure Measurement (EM) module. *Source* [63]

4

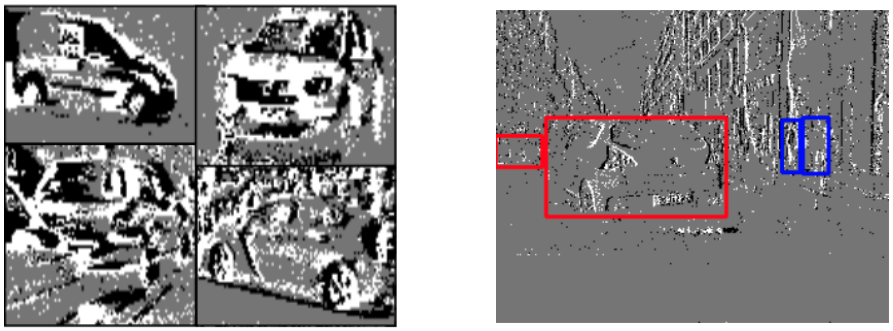
EVENT-BASED VISION FOR AUTONOMOUS DRIVING

Bio-inspired features make event cameras very attractive for the emerging field of autonomous driving. The event-based neuromorphic vision sensor such as DVS has an *High Dynamic Range (HDR)* (120 dB), compared to the 60 dB frame-based camera sensors. This allows artificial neuromorphic vision sensors to adapt to very dark and bright stimuli ensuring a highly robust perception system even in a light-changing scene such as an autonomous vehicle driving through a tunnel. The brightness changes can be captured quickly in analog circuity with a 1-MHz clock, detecting and generating timestamps with microsecond resolution. Considering the fast response requirement of the controller in autonomous vehicles in emergency driving scenes, this feature could lead to useful implementations. In a high-speed driving situation, the motion blur problem can occur if the motion of moving objects exceeds the frame-based camera's sampling frequency, causing the vision mechanism to struggle. A neuromorphic vision sensor based on events can capture dynamic motion precisely and without motion blur. Ultimately, power consumption is reduced as only active pixels of neuromorphic vision sensor transmit events, filtering redundant data autonomously. For the onboard computers and devices in autonomous vehicles, energy-efficient sensors are as important as advanced algorithms.

In the next Sections we will provide an in-depth overview of the existing technologies for event-based perception for autonomous driving.

4.1 EVENT-BASED DATASETS FOR AUTONOMOUS DRIVING

In the past years, many efforts have been made to create useful event-based datasets for autonomous driving using DVS, DAVIS, or ATIS systems. The data has then been processed to create useful data representations, ready to be consumed by ad-hoc created algorithms for computer vision tasks, such as semantic scene segmentation, object detection, or flow estimation. Labeling the asynchronous event data is always a challenging problem because almost all of the annotation tools are developed for frame-based cameras. Additionally, there is not a standard format for the annotations. Many attempts have been made to generate useful labels. Some techniques involve manually labeling other resort to pre-trained CNNs on gray-scale images to produce ground-truth data.



(a) NCars sample

(b) Prophesee's GEN1 sample

Figure 4.1: a) Example of the positive ("car") class of the NCARS dataset. b) Grayscale example from Prophesee's GEN1, with bounding boxes for cars (in red) and pedestrians (in blue)

4.1.1 *N-CARS Dataset*

The N-CARS data set introduced by Prophesee [68] provides recording cars in urban environments with a DVS. The data was recorded using an ATIS camera mounted behind the windshield of a car. The dataset is split into 7940 car and 7482 background training samples, 4396 car, and 4211 background testing samples. Each example lasts 100 milliseconds. The dataset is very useful to build event-based classification models. An extract from the dataset can be seen in Figure 4.1a.

4.1.2 *Prophesee's GEN1 Automotive Detection Dataset*

The Prophesee's GEN1 Automotive Detection Dataset was recorded using a PROPHESEE GEN1 sensor [69] with a resolution of 304×240 pixels, mounted on a car dashboard. The labels were obtained using the gray level estimation feature of the ATIS camera by labeling manually. It contains 39 hours of open road and various driving scenarios ranging from urban, highway, suburbs, and countryside scenes. Each file consists of 60 seconds recordings that were cut from longer recording sessions. Furthermore, manual bounding box annotations are available for two classes are present: pedestrians and cars (see Figure 4.1b).

4.1.3 *DAVIS Driving Dataset 2017 (DDD17)*

The DDD17 [67] is the first-ever public dataset of real automotive end-to-end training data recorded with an advanced 346x260 pixel DAVIS sensor. The recording includes various car data such as steering angle, speed, GPS, throttle, etc along with DVS and APS data. Given the wide range of information

included, the dataset can be used for many applications such as semantic scene segmentation, object detection, or flow estimation. An example of the data recorded by the DAVIS for the DDD17 dataset can be seen in Figure 3.6.

4.1.4 *MVSEC Dataset*

The *Multivehicle Stereo Event Camera Data Set (MVSEC)* for 3D perception was presented in [70]. It is the first dataset with a synchronized stereo event-based neuromorphic vision system. The ground-truth data are generated with the aid of calibrating LiDAR system, contributing to the stereo depth estimation with the event-based vision sensor. The dataset consists of many recordings of outdoor scenarios, at different locations and speeds. Its main application is for localization, odometry, and obstacle avoidance tasks.

4.2 REPRESENTATION OF EVENT-BASED DATA

The raw output of event-based sensors is an event stream of sparse and asynchronously generated data points (Figure 4.2). This representation is very different than the topologically structured data consumed by standard vision pipelines, like convolutional neural networks. Therefore architectures that process data generated from neuromorphic vision sensors usually require encoding methods to create synchronous images or grid-like representations that can be easily consumed by traditional neural networks. *Spatial encoding*makes only use of spatial features. It is used in event frame and event count data representation. Temporal information is included in *spatial-temporal* representations, like Surface of Active Events (SAE), Voxel Grids, and Leaky integrate-and-fire (LIF). Table 4.1 gives a short overview of the main data representation techniques along with a brief description.

Representation	Dimensions	Description	Weakness
Event Frame	HxW	Image of event polarities	No temporal and polarity information
Event Count	2xHxW	Image of event counts	No time stamps
SAE	2xHxW	Image of most recent time stamp	No temporal history
LIF	HxW	Image of event spikes generated by the neuron	No polarity information
Voxel Grid	BxHxW	Voxel grid summing event polarities	No polarity information

Table 4.1: Summary of the most common type of event data representations

4.2.1 *Event Frames and Event Count Frames*

Spatial encoding converts event data into *event frames* [74] by storing event data at pixel location (x_i, y_i). Information of time is not stored but used for example to aggregate information over a given fixed time interval (*constant time frame*). The value of a pixel is usually represented by the polarity of the

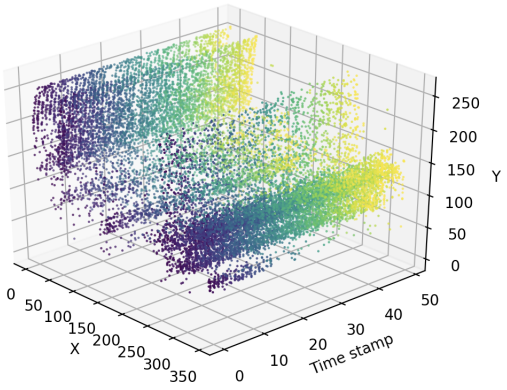
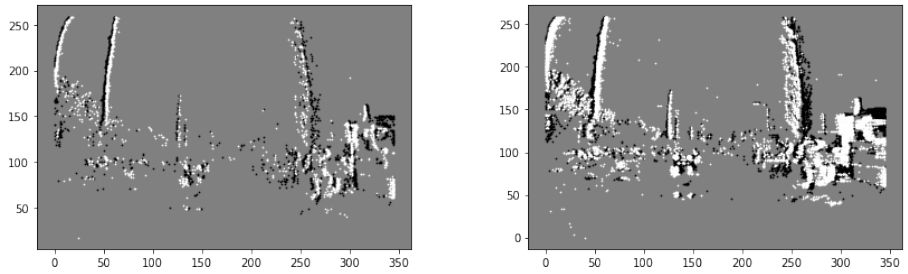


Figure 4.2: Visual representation of a 50ms data stream taken from a sample of the DAVIS Driving Dataset.



(a) Constant time frame T=10ms

(b) Constant event frame E=15000

Figure 4.3: Comparison of two spacial encoding approaches on the sample event stream in Figure 4.2

last event or statistical characteristics, such as the event count in a fixed time interval. An event-stream of $N-1$ events can be defined as:

$$\Psi = \{e_i\}_{i=1}^{I}, \text{ with } e_i = (x_i, y_i, t_i, p_i) \tag{4.1}$$

with x_i, y_i being the coordinates, p_i the polarity and t_i the time-stamp of event i.

A type of spatial encoding, called *Constant time frames*, aggregates events pixel-wise in a given fixed-time interval. This encoding can be formulated as

$$F_j^t = \mathbf{card}(e_i | T \cdot (j-1) \leqslant t_i \leqslant T \cdot j) \tag{4.2}$$

where F_j^t represents the jth frame of time interval T, $\mathbf{card}()$ being the cardinality of the set and e_i being the ith event of the stream.

In similar fashion, the *constant count frames* generates frames by aggregating a fixed number of events E:

$$F_j^t = \mathbf{card}(e_i | E \cdot (j-1) \leqslant i \leqslant E \cdot j) \tag{4.3}$$

with F_j^t being the jth frame with E events. Examples of constant time frame and constant event frame representations are shown in Figure 4.3

The last spatial encoding technique is called *event count frames* [75, 78], defined as

$$\text{Hist}^{\pm}(x, y) = \sum_{p_i=\pm, t_i \in T} \delta(x - x_i, y - y_i). \tag{4.4}$$

Two separate histograms, one for each polarity value, are generated in a fixed-time interval T using the Kronecker delta (δ) function as counting function. This methods yields a two-channel event frame.

4.2.2 *Surface of active events*

The *surface of active events (SAE)* [76, 77], uses timestamped values instead of intensity values to represent the pixel values, i.e. for each incoming event e_i:

$$\text{SAE} : t_i \rightarrow P(x_i, y_i) \tag{4.5}$$

where t_i is the timestamp of the most recent event at each pixel $P(x_i, y_i)$ whose value is determined by the occurrence time of the events. This is a useful method to incorporate the temporal information, however, it considers only the most recent event for each pixel, ignoring therefore previous events in that location.

4.2.3 *Leaky integrate-and-fire (LIF) neurons*

A more bio-inspired approach is represented by the *Leaky integrate and fire (LIF)*, which is an artificial neuron that receives input spikes (events) from neuromorphic sensors that are able to modify the potential of the neuron's membrane. If the potential exceeds a predefined threshold, a stimulus will be emitted. A LIF neuron can be modeled as

$$\tau \frac{dV}{dt} = -(V(t) - V_{reset}) + RI(t) \tag{4.6}$$

with $V(t)$ and R being the neuron's membrane potential and resistance, $I(t)$ the total synaptic current, and τ the membrane's time constant. LIF neurons are the building block of spiking neural networks, as explained in Section 4.4

4.2.4 *Voxel Grid*

A novel event representation was proposed with the *Voxel Grid* [78]. Given a stream of events $\Psi = \{e_i\}_{i=1}^{I}$ with $e_i = (x_i, y_i, t_i, p_i)$, a fixed number of *bins* B are used to split the stream in the time domain. The events in the time are then scaled in range $[0, B-1]$, generating an event volume:

$$t_i^* = (B-1)(t_i - t_0)/(t_N - t_1) \tag{4.7}$$

$$V(x,y,t) = \sum_i p_i k_b(x - x_i) k_b(y - y_i) k_b(t - t_i^*) \tag{4.8}$$

$$k_b(a) = max(0, 1 - |a|) \tag{4.9}$$

In the case where no events overlap between pixels, this representation allows us to reconstruct the exact set of events. When multiple events overlap on a voxel, the summation does cause some information to be lost, but the resulting volume still retains the distribution of the events across both the spatial and temporal dimensions within the window.

4.2.5 *Event Spike Tensor (EST)*

Event Spike Tensor (EST) [80] generalize the framework of the event data representing. The aim of EST is to find a mapping $M : \Psi \to T$ between the event stream Ψ and tensor T, a grid-like to be used in CNNs. An event stream can be seen as a point-set in four dimensions that can be summarized with the notion of *event field*:

$$S_\pm(x,y,t) = \sum_{e_k \in \Psi_\pm} \delta(x - x_k, y - y_k)\delta(t - t_k) \tag{4.10}$$

defined in continuous space and time, for event with positive polarity Ψ_+ and negative polarity Ψ_-. This representation replaces events with a Dirac pulse in space-time. Furthermore, Equation 4.10 can interpreted as successive measurements of a function $f_\pm$ in the event domain:

$$S_\pm(x,y,t) = \sum_{e_k \in \Psi_\pm} f_\pm(x,y,t)\delta(x - x_k, y - y_k)\delta(t - t_k). \tag{4.11}$$

Equation 4.11 is called *Event Measurement Field*. It assigns a measurement $f_\pm(x,y,t)$ to each event. By choosing the appropriate measurement function, the representations in the above mentioned section can be easily derived. If the measurements function considers the event polarity, $f_\pm(x,y,t) = \pm 1$, or the event count ,$f_\pm(x,y,t)$, Equation 4.11 reduces to the event frame and event count representation mentioned in Section 4.2.1.

Kernel convolutions are able to extract further information from the event measurement field, which retains high temporal resolution of the event but is still ill-defined due to the use of Dirac pulses. Using kernel convolutions the new representation becomes:

$$(k * S_\pm)(x,y,t) = \sum_{e_k \in \Psi_\pm} f_\pm(x,y,t)k(x - x_k, y - y_k, t - t_k). \tag{4.12}$$

An example of kernel is the *exponential kernel* $k(x,y,t) = \delta(x,y)\frac{1}{\tau}e^{\frac{-t}{\tau}}$ which is used to construct the hierarchy of time surfaces *(HOTS)* and histogram of average time surfaces *(HATS)*.

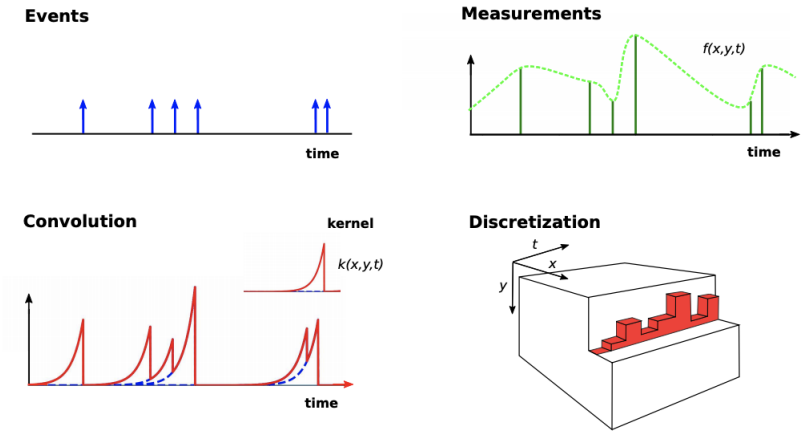


Figure 4.4: An overview of the EST framework. Each event is associated with a measurement (green) which is convolved with a (possibly learned) kernel. This convolved signal is then sampled on a regular grid. Finally, various representations can be instantiated by performing projections over the temporal axis or over polarities.

The convolved signal can then be sampled creating a grid-like representation

$$S_{\pm}[x_l, y_m, t_n] = \sum_{e_k \in \Psi_{\pm}} f_{\pm}(x, y, t) k(x_l - x_k, y_m, -y_k, t_n - t_k). \tag{4.13}$$

This generalized term in Equation 4.13 is called *Event Spike Tensor (EST)*. This general form retains all four dimensions, and therefore different than the representations mentioned in the previous sections. However, by adjusting the kernel k and measurement function f it is possible to derive SAE, HATS , HOTS or Voxel Grids by using projections.

In [80] an End-To-End Learned Representation is proposed exploiting the EST representation and replacing the handcrafted kernel with an MLP aimed at finding the best function for event streams. The proposed EST framework is illustrated in Figure 4.4.

4.3 HANDCRAFTED FEATURES: TIME SURFACES

The paradigm shift imposed by neuromorphic sensors requires new neural network architectures that can handle and process the generated event streams. In the past years fast, optimized, and highly performing architectures based con Convolutional Neural Networks have been developed for feature learning on frames generated by traditional cameras. Without proper

adjustments, these architectures fail with their intent on raw event-based data. A robust feature detection pipeline is essential in autonomous driving, where it is important to distinguish effectively the different elements in the scene. *Time surface* are effective features to track the activity of an object in an event-based scenario, developed due to the lack of low-level feature representation and descriptors. Time surfaces represent temporal characteristics and describe the spatial-temporal context around an event by using exponential decay that records the activity of past events in the neighborhood. For an event $e_i = (x_i, y_i, t_i, p_i)$ a time-surface S_i of dimension 2Rx2R is defined as

$$S_i = \begin{cases} e^{-\frac{t_i - T(C_i+R,P)}{\tau}}, & \text{if } p_i = P \\ 0, & \text{otherwise} \end{cases} \tag{4.14}$$

where $C_i = (x_i, y_i)$ is the pixel coordinates of the incoming event e_i, R the radius of the neighborhood around e_i, $T(C_i + R, P)$ is the time stamp of the *last* event with polarity P received from pixel $C_i + R$, and τ is a *constant decay factor*.

Hand crafted time surfaces have been used in a hierarchical structure for object recognition [71] that relies on a time-oriented approach to extract valuable spatial-temporal features from event. Another implementation was proposed by Sironi, Brambilla, Bourdis, Lagorce and Ryad Benosman with the *Histogram of averaged time-surfaces (HATS)* [68], where event streams are converted into local memory time surfaces which are used to compute histograms to create the final descriptor. After the features are extracted from the event stream, a *Support Vector Machine (SVM)* is used to classify objects in the N-CARS dataset.

Although time surfaces have shown good generalization performances, their usage is often limited to recognize simpler shapes in relatively uniform scenarios. More complex shapes generated by varying density event streams require the combination of different levels of feature representations to be effectively recognized by machines. For this purpose, it is necessary to use architectures that are able to learn and extract useful patterns in the data automatically.

4.4 SPIKING NEURAL NETWORKS (SNN)

A *Spiking Neural Network (SNN)* is a neural network architecture that leverages the functionality of biological neurons by creating a computational system that emulates the working principle of the receptive field in the primary visual cortex. Just as biological neurons, spiking neurons interact with one another through spike-trains, which encode temporal and spatial information. The basic principle of SNN is that a neuron will not emit any spike if it has not received any input spike from the preceding SNN layer. Furthermore, only

if the membrane voltage generated by obtained spikes reaches a predefined threshold can the resulting neuron produce spikes that are fed to the next layer. To abstract features, predefined network units such as the difference of Gaussians or Gabor filters are usually used in the first layer of SNN. The generated features are then processed in parallel and fed to deeper layers in the network [14–17]. Although proven to be useful for object detection scenarios [11–13], SNNs are not differentiable and therefore hard to train, as conventional training methods like gradient descent can not be applied.

J. Acharya, V. Padala, and A. Basu proposed a three-layer SNN architecture for autonomous driving [72]. The proposed architecture consists of refractory, convolution, and clustering layers designed with bio-realistic leaky integrate and fire(LIF) neurons and synapses. The proposed algorithm is tested on traffic scene recordings from a DAVIS sensor setup. Efforts led by J. H. Lee, T. Delbruck, and M. Pfeiffer result in an SSN architecture with backpropagation [73] using LIF neuron and *winner-takes-all (WTA)* circuits. The differentiable transfer functions are derived in the WTA configuration to make SNN trainable with backpropagation. However, trainable SNN is only tested on simple data sets (such as MNIST) and has not been applied in specific autonomous driving scenarios.

4.5 CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional neural networks (CNNs or ConvNets) are powerful feature extraction architectures, specialized in processing data that has a known grid-like topology. They are highly efficient and have been proved to be the most successful to handle 2-D image topology. Neurons of a layer are indeed organized into a two-dimensional surface and they receive inputs only from a small region in the previous layer, known as the receptive field. Each unit, therefore, computes a local feature of the previous layer which is then combined with the ones computed by adjacent neurons to form higher-order features in subsequent layers. This feature extraction topology is similar to the way in which visual information is processed in the biological brain. The main operation of CNNs are performed in two core layers called *convolutional layer* (Section 4.5.1) and *pooling layer* (Section 4.5.2).

The performance of CNNs has surpassed traditional machine learning methods in many vision tasks, relying on successful training algorithms and large amounts of data. In Sections 4.5.3 – 4.5.5, will provide examples of CNN architectures for autonomous driving with event-based cameras.

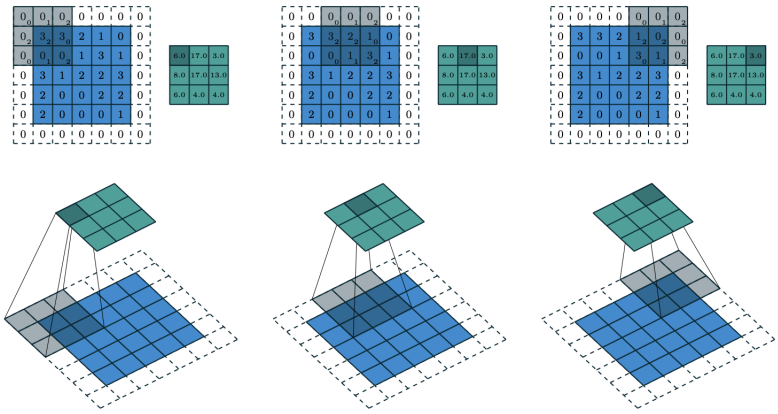


Figure 4.5: Convolution operation on a 5x5 pixel grid using a 3x3 kernel with zero-padding and stride 2. Each pixel in the input map and padding borders contain a certain value. This value is multiplied by the corresponding value in the sliding kernel (numbers in subscript). The multiplied values are then summed together, forming the value of that region in the output feature map.

4.5.1 *Convolutional layer*

The convolution is a mathematical operation on two functions (f and g) that produces a third function $f * g$ that expresses how the shape of one is modified by the other. In CNN the first argument f of the convolution is the input and the second argument g is the so-called *kernel* or *filters*. Every filter is usually small along the spatial dimensions (height and width), but it extends through the full depth of the input volume which represents the feature dimension. The choice of the filter dimension depends and their number depends on the particular objects to be recognized. Common kernel sizes are 3x3xd or 5x5xd, with d being the channels of the image. During the convolution operation, each filter is sliding with a given *stride* across the width and height of the image applying every time the filter transformation i.e at each location, the product between each element of the filter and the input element it overlaps is computed and the results are summed up to obtain the value of the current location. This procedure is applied for each dimension of the image in parallel. The spatial dimensions of the output feature map are equal to the number of steps made, plus one, accounting for the initial position of the kernel. This means that the size of the feature map is usually smaller than the size of the input grid. In certain cases it useful to maintain the original size of the input map. This can be achieved employing a technique called *zero-padding* which transforms the original image by adding borders of size P pixels, filled with zeros (Figure 4.5).

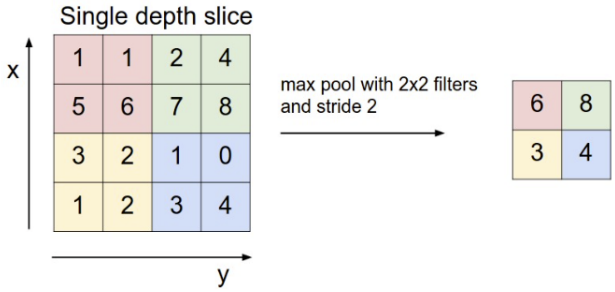


Figure 4.6: Example of pooling using a MAX aggregation function. A 2x2 filter with stride 2 slide over the feature map outputting for each region the maximum value.

On the other hand, certain applications require to produces *wider* receptive fields without interfering with the number of parameters in the model. This can be achieved with the *Dilated Convolution*. This operation includes the *dilation rate d*, which defines a spacing between kernel values. This operation is particularly useful for semantic segmentation where it is important to provide information of larger regions combined with fine details.

4.5.2 *Pooling layer*

The features generated by the convolutional layer can detect characteristic patterns in images, like edges or loops. However, it is improbable that two images representing the same object are inherently identical. Perspective, resolution, or lighting conditions are some of the effects that influence the outcome of a picture. Therefore a feature extractor must be able to cope with transformations or perturbation of the input. The pooling layer's goal is to make representations invariant to such transformation whilst also reducing the number of parameters in the model. The pooling operation reduces the spatial size of the previous representation, i.e. feature maps generated by convolutional layers, by using an *aggregation function* applied to every depth slice that results in downsampling of the feature map. Common aggregation functions are *MAX*, *average* or *L2-norm* implemented in 2x2`filters` with stride 2. Wider filters are usually avoided as too much information would be lost (Figure 4.6).

4.5.3 *Optical flow, depth and egomotion*

2-D motion estimation, also known as *optical flow* is defined as the distribution of apparent velocities of movements of brightness patterns between two images. By directly measuring the precise time at which each pixel changes, the event stream directly encodes fine-grained motion information, which

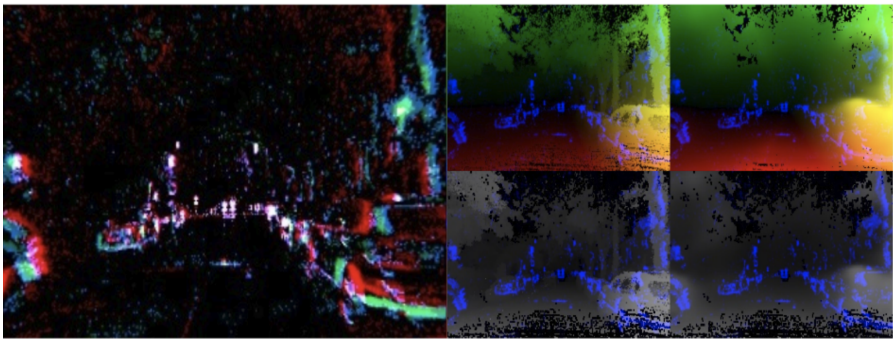


Figure 4.7: Example of the ECN network predicting the optical flow and depth on sparse event data in a night scene: event camera output (left), ground truth (middle column), network output (right) (top row - flow, bottom row - depth). The event data is overlaid on the ground truth and inference images in blue.

researchers have taken advantage of to perform optical flow estimation. Using the high temporal resolution of neuromorphic vision sensors, Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis proposed *Ev-FlowNet* [77], a self-supervised deep learning architecture for optical flow estimation. The architecture uses a four-channel event representation composed of a two-channel image using histograms (4.4) and SAE (4.5) of different polarity which is processed by four stride convolutional layers, two residual blocks, and four up-sampling convolutional layers. The architecture produces robust results estimating the flow of objects in the MVSEC dataset.

In [78] the same authors proposed an improved architecture for unsupervised learning used for optical flow, depth, and egomotion. The improved architecture uses Voxel Grid (4.8) data representation in the input of the pipeline. The data is then processed by an encoder-decoder architecture responsible for optical flow and depth. Besides, a pose model block has been added for estimating egomotion. Experimental results on the MVSEC data indicate that the architecture can learn various motion information of events.

Chengxi Ye, Anton Mitrokhin, Cornelia Fermuller, James A. Yorke, and Yiannis Aloimonos developed the *Evently-Cascaded Convolutional Network (ECN)* architecture for optical flow, depth, and egomotion using a monocular pipeline. The architecture has a depth prediction component, consisting of an encoder-decoder architecture, and a parallel pose component that uses consecutive frames to estimate the translational and rotational velocity with respect to the middle frame. The optical flow is calculated given the poses of the neighboring frame and the depth of the middle frame. The lightweight architecture allows processing 250 fps on a single NVIDIA 1080 titanium GPU. The performances are significantly improved compared to previous works. Figure 4.7 illustrates an output example of the ECN network.

Figure 4.8: Output of the detection pipeline proposed by Chen [81]. The left image of is the DVS image with bounding boxes (in red) produced by DVS-only detection, while the right image of each pair is the APS image with DVS-only bounding boxes (in red) copied over and the RRC detections (in yellow) for comparison.

4.5.4 *Object detection*

One of the most interesting applications of convolutional networks is object recognition, the aim of which is to efficiently identify targets in the analyzed scene. In autonomous driving, object recognition plays a significant role. It of fundamental importance for a vehicle to able to distinguish the different objects that can be found in its surrounding environment to correctly take an action.

In [81] Chen proposed a framework for generating and learning on labels and bounding boxes of the DDD17. The first step consists of producing pseudo-labels created by feeding a CNN the APS images from the DAVIS dataset. The pseudo-labels generated above a certain threshold are considered as ground-truth and can be processed along with the event data provided by the DVS sensors in more complex architectures for event-based object detection. The dynamic vision sensor data are converted to images by binning the dynamic vision sensor outputs in 10ms intervals, each pixel taken value $\sigma(x) = 255 * \frac{1}{1+e^{-\frac{x}{2}}}$ with x being the sum of polarities in the 10ms interval. The data is then processed by the tiny YOLO CNN architecture [37] achieving high-speed detection (100fps) in real outdoor scenarios. Detection was also performed on the APS grayscale images generated by the DAVIS using the Recurrent Rolling Convolution (RCC) architecture [39]. The DVS binned frame + YOLO CNN pipeline has an AP@0.5 $= 40,3\%$ being able to detect $60,1\%$ of the actual objects. The APS grayscales + RRC has a better average precision AP@0.5 $= 53,7\%$ and was able to detect $64,1\%$ of the actual objects. Nevertheless, the DVS sensor was able to detect $10,6\%$ of the objects that were not detected by the APS+RRC pipeline, reinforcing the fact that the DVS only detector learned general representations of cars, though it was trained on the knowledge from the RRC.

A similar approach was proposed in [82]. A convolutional SNN with LIF neurons [4.2.3] is utilized to generate visual attention maps based on the firing rate of output neurons. Two separate event-based and frame-based streams are incorporated into the YOLO V3 [38] object detector to obtain the detection output. With a joint decision model to post-process the output, the algorithm outperforms the state-of-the-art achieving a $AP = 0.908$ at 9FPS in day conditions and $AP = 0.833$ at 9FPS.

In [83] an architecture was proposed and tested for the detection of pedestrians. The event data is split into three separate streams, one for negative events, one for positive events, and one for both polarities combined. The three streams are then applied to a Frequency, SAE, and LIF encoding, each allowing different characteristics of the event stream to be exploited. This method is called multi-cue event information fusion. The method generates three sequences of frames, which can be merged as RGB channels of an image and processed by a YOLO-based architecture or can be processed individually by a YOLO detector obtaining three different detectors that are fused with a *Dynamic Belief Fusion* function. The performance improvement of DBF indicates that detection accuracy can be improved by investigating complementary information provided by each detector.

Recently, a cross-modal approach was presented in [84]; wormhole learning was utilized to pair red, green, blue (RGB) camera and event-based neuromorphic vision sensors to improve the object detection performance under the scenario of urban driving. The experimental results of wormhole learning reveal that there are many innovative approaches to combine data from different heterogeneous sensors, such as RGB cameras, infrared cameras, and neuromorphic vision sensors.

4.5.5 *Semantic segmentation*

In the sensing and perception system of autonomous driving, a comprehensive understanding of the surrounding environment is provided by semantic segmentation. The first CNN-based baseline for semantic segmentation with an event-based neuromorphic vision sensor is introduced [18], trained, and validated on the DDD17 event-based dataset. The ground-truth labels for semantic segmentation were generated by applying a pre-trained CNN to grayscale images provided in the DDD17 dataset (Figure 4.9).

The event data stream is encoded in a novelty 6-channel image representation. The first two-channels are classic histogram representations [4.2.1] for each polarity. Furthermore, the mean (M) and standard deviation (S) of the

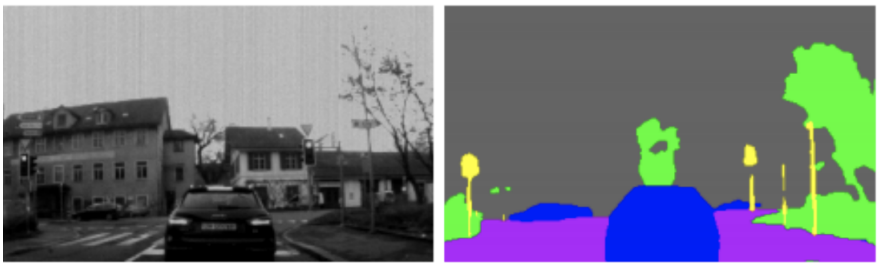


Figure 4.9: Grascale image taken from the DDD17 dataset (left) and the labels generated by a pre-trained CNN on the *Cityscales* dataset (right). The labels are then used on a 6 channel representation in the Ev-SegNet pipeline.

normalized timestamps of events in time interval W happening at each pixel (x_i, y_i) is computed separately for the positive and negative events:

$$M(x,y,p) = \frac{1}{\text{Hist}(x,y,p)} \sum_{i=1, t_i \in W}^{N} t_i \delta(x_i, x)\delta(y_i, y)\delta(p_i, p) \tag{4.15}$$

$$S(x,y,p) = \sqrt{\frac{\sum_{i=1, t_i \in W}^{N} (t_i \delta(x_i, x)\delta(y_i, y)\delta(p_i, p) - \text{Mean}(x,y,p))^2}{\text{Hist}(x,y,p) - 1}} \tag{4.16}$$

The 6-channel data representation is then fed to an Xception [85] state-of-the-art encoder and lightweight decoder. Lastly, the complementarity between the frame-based camera and event-based neuromorphic vision sensor is presented by comparing the semantic segmentation results produced from event data and corresponding gray-scale images. The EV-SegNet pipeline is able to achieve high accuracy (0.897) and *mean intersection over union (mIoU)* (0.548) on 6 different classes of labels.

4.5.6 *Active perception*

State-of-the-art self-driving vehicles have sophisticated active systems that can not only perceive their surroundings but can also make decisions about the state of the car. An example of this kind was presented in [86]. Raw data from event-based sensors are collected into histograms[4.2.1] within a time interval T, using separate channels for each polarity. The resulting synchronous event-frames are processed by a ResNet [44] inspired network that predicts a steering angle of the vehicle. The proposed method can accurately predict the steering angle of vehicles and performs better on DDD17 data sets than the state-of-the-art systems using gray-scale images.

4.6 THE FUTURE FOR EVENT-BASED CAMERAS IN AUTONOMOUS DRIVING

Event cameras, like the DVS, undoubtedly have interesting applications in the field of autonomous driving, but they are also a fairly recent technology that has not yet reached the maturity levels of other sensors such as LiDAR, radar, or traditional cameras. State-of-the-art CNNs that process images are heavily optimized achieving excellent performances in classification, segmentation, and object detection. By using different representations to create a grid-like topology, event-based streams are capable of working on such CNNs, but they still lack important characteristics (e.g. color or resolution) that are essential for achieving state-of-the-art performances. On the other hand, the features of neuromorphic vision sensors have incredible potential for autonomous vehicles with much researches being focused on event-based cameras, as the consensus is that there is substantial room for development and improvement.

As mentioned in Chapter 2, autonomous driving vehicles have a wide range of sensors, each with its advantages and disadvantages. A neuromorphic visual sensor fits perfectly into this scenario: it has features that few other sensors can offer, such as a very high temporal resolution or a high dynamic range, but it also has shortcomings, such as the lack of color perception, which is present in traditional cameras. Merging the capabilities of an event-based sensor with other sensors in the vehicle to create a complete sensor system seems like the ideal solution, but doing so reintroduces the disadvantage of redundant data. It remains to be seen whether the DVS output can be used to trigger frame captures of other sensors. If it is, the DVS and other sensors can operate together with mixed conventional machine vision, bio-inspired, and event-based neuromorphic vision-based approaches. Therefore, some of the limitations of a traditional sensor-based perception system may be overcome; moreover, new scenarios that were previously inaccessible in the visual sensing and perception of autonomous vehicles might be reached.

Active perception is perhaps the area where event cameras can bring the biggest improvements. Currently, self-driving vehicles distinguish between perception and active movement phases. This is because state-of-the-art perception sensors are frame-based, i.e. in discrete-time domains while vehicle movement takes place in continuous time. Event cameras can bridge this gap with their very high temporal resolution, being able to actually "see" the movement of the vehicle. This hypothesis supposes the creation of new types of event flow representation that are more suitable for the purpose. The creation of a fast and robust active perception system would open the door for near-perfect autonomous navigation systems capable of recognizing obstacles and making decisions accordingly.

Event-based sensors are also limited to their technology: the is no appearance feature such as color and texture because an event-based neuromorphic

vision sensor only transmits local pixel-level changes, making it perform poorly in some applications with high requirements for appearance features. Although researchers have used the method of IR (mentioned in the section "Spatial Encoding") to reconstruct image frames from event streams, the quality of reconstructed image frames is still not comparable to the output data produced by RGB cameras. The application of an event-based neuromorphic vision sensor is limited in some scenarios where energy, latency, and dynamic range are not important, especially in high-resolution complex scenarios.

5 EVENT-BASED LEARNING APPROACH WITH POINTNET

Almost all deep learning architectures that process event streams generated by neuromorphic sensors base their operation on mature and optimized convolutional neural networks, using intermediate grid-like topology representations. This stems from the fact that modern object recognition architectures are optimized to such an extent that they can process images at high frame rates, making them suitable for application in autonomous driving. Instead, our efforts have focused on maintaining an asynchronous and sparse representation of events, treating the event stream as a *point cloud*. Point clouds are important geometric data structures that have become popular over the past years thanks to the rise of more affordable 3D sensors such as LiDARS and RGB-D cameras. Consequently, a large part of technological research has also focused on how to exploit 3D data in the form of point clouds with modern Deep Learning architectures.

In the next Sections we will introduce point clouds and two deep learning networks that are able to process and learn directly on raw point clouds. In Section 5.5 we introduce our pipeline on point clouds generated from event-based sensors, evaluated on different automotive datasets.

5.1 POINT CLOUDS

With the rapid development of 3D acquisition technologies, sensors capable of capturing a 3D environment have become increasingly available and affordable such as LiDARSs and RGB-D cameras. 3D data acquired by these sensors can provide a rich set of geometric, shape, and scale information. There is also a variety of representations for 3D data including depth images, meshes, volumetric grids, and point clouds. As a commonly used format, point cloud representation preserves the original geometric information in 3D space, a very useful characteristic for scenes understanding related applications like robots or autonomous driving. Moreover, point clouds are simple and unified structures that avoid the combinatorial irregularities and complexities of meshes.

A point cloud is represented as a set of 3D points:

$$\mathbf{P} = \{p_i\}_{i=1}^{I}, \text{ with } p_i = (x_i, y_i, z_i) \tag{5.1}$$

where each point P_i is a vector of its coordinates (x, y, z) plus extra feature channels such as color, normal etc. Most deep learning architectures do not directly handle point clouds, but try to transform the input into topologies

that can be consumed by traditional convolutional neural networks. While the typical sparsity of point clouds is lost, the ability to process input with advanced CNN-based systems is gained. In the next two Sections we will provide two architectures that are able to learn directly on raw point clouds. Exploiting the sparsity of events is a desired feature especially in autonomous driving scenario, where data throughput is one of the key characteristics.

5.2 POINTNET ARCHITECTURE

PointNet [1] is a unified neural network architecture that directly takes unordered point clouds as input and outputs either class labels for the entire input or per point segment/part labels for each point of the input. The architecture is relatively simple but at the same time highly robust to small perturbations of input points as well as to corruption through point insertion (outliers) or deletion (missing data). The network learns a set of optimization functions/criteria that select interesting or informative points of the point cloud and encode the reason for their selection. The final fully connected layers of the network aggregate these learnt optimal values into the global descriptor for the entire shape as mentioned above (shape classification) or are used to predict per point labels (shape segmentation). The versatility of the neural network makes it suitable for a wide range of applications ranging from object classification, part segmentation, to scene semantic parsing.

As input, Pointnet expects an unordered set of 3D point $\{P_i i = 1, ..., n\}$ where each point P_i is a vector of its (x, y, z) coordinates plus extra feature channels such as color, normal etc. A point cloud in Euclidean space has three main properties:

- Unordered: a point cloud is a set of points without a specific order. A network working with N 3D point sets must be invariant to N! permutations of the input set in data feeding order. This is in contrast with traditional pixel arrays in images or volumetric grids.
- Interaction among points: points are within a certain metric from a space, which means that points are not isolated and neighboring points form a meaningful subset. The network must be able to capture local structures from close-by points and combinatorial interactions among local structures.
- Transformation invariance: a geometric object, a point cloud should be invariant to certain transformations. For example, rotating the whole point cloud should not affect the global point category or segmentation of the points.

The research of designing a point cloud-consuming neural network must bear in mind these three fundamental characteristics. PointNet proposes

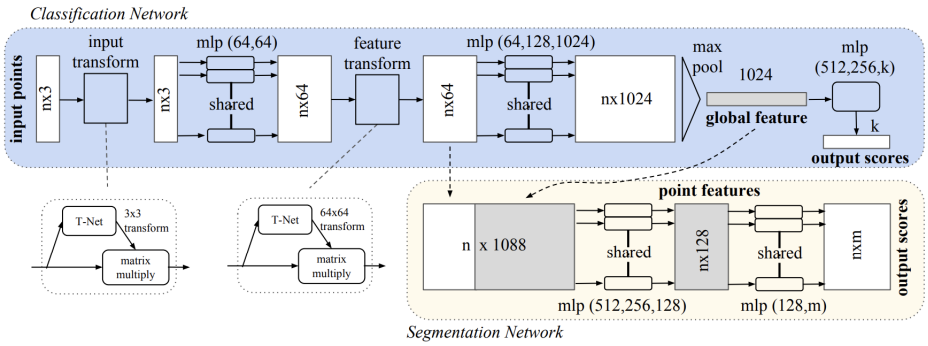


Figure 5.1: The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net

a pipeline consisting of three key modules, each tackling one of the above-mentioned properties: a *max pooling layer* as a symmetric function to aggregate information from all the points, a local and global information combination structure, and two joint alignment networks that align both input points and point features. In Figure 5.1 the general architecture is proposed. In the next Sections, we will explore the core modules of the architecture and their applications.

5.2.1 *Symmetry Function for Unordered Input*

The first property of a 3D point set is invariance to input permutation. Sorting the input into a canonical order seems like a trivial solution to the problem. Generally, in a high-dimensional space, it is not easy to find a stable ordering with respect to point perturbations. It is hard for a network to learn a consistent mapping from input to output as the ordering issue persists. Experimental results show that by ordering the input the proposed architecture performs poorly, even if slightly better than on a raw unordered set of data points. Another approach consists of implementing a Recurrent Neural Network (RNN) trained with randomly permuted sequences. Ideally, the network should be able to become invariant to input order. Unfortunately, this does not hold for RNNs, in a general sense [89]. A slight invariance can be handled by RNNs but the effect does not scale to the usual sizes of point clouds. The solution adopted by the authors of PoineNet is to use a *symmetric function* on transformed elements in the point set:

$$f(\{x_1, ..., x_n\}) \approx g(h(x_1), ..., h(x_n)), \tag{5.2}$$

where $f : 2^{\mathbb{R}^N} \rightarrow \mathbb{R}, h : \mathbb{R}^N \rightarrow \mathbb{R}^K$ and $g : \underbrace{\mathbb{R}^K \times ... \times \mathbb{R}^K}_{n} \rightarrow \mathbb{R}$ is a symmetric function. Function h is approximated by a multi-layer perceptron network and function g by a composition of a single variable function and a *max pooling function*. The key idea is that in the worst case the network can learn to convert a point cloud into a volumetric representation, by partitioning the space into equal-sized voxels. In practice, however, the network learns a much smarter strategy, by summarizing a shape by a sparse set of key points.

5.2.2 *Local and Global Information Aggregation*

The output of the Equation 5.2 forms a vector $[f_1, ..., f_K]$ which is a global signature of the input set. This vector can be used for classification purposes by applying a *Support Vector Machine (SVM)* or a multi-layer perceptron classifier. However, part and semantic segmentation require the combination of local and global features. As shown in Figure 5.1, the global feature vector is combined with per point features. The combined feature vector is now aware of local and global patterns.

5.2.3 *Joint Alignment Network*

Invariance under transformation is the third property of unordered 3D point sets. For example, in an urban driving scenario, an event-based vision sensor records an event stream corresponding to a moving car in front of the camera. This event stream is fed to the PointNet architecture which must be able to classify the point cloud correctly as 'car'. Should the point cloud be rotated or translated, the architecture must still be able to correctly classify the point set. This feature is achieved by PointNet with means an affine transformation matrix, predicted by a mini-network called *T-Net*. The T-Net emulates the whole PointNet pipeline, by processing the raw point cloud and regressing it to a 3x3 matrix. To further improve performance the T-Net can also be applied to features by regressing it to a 64x64 matrix. However, the transformation matrix in the feature space has a much higher dimension than the spatial transform matrix, which greatly increases the difficulty of optimization. A regularization term is therefore added to the softmax training loss, increasing the performance as backed by experimental results. The feature transformation matrix is constrained to be close to an orthogonal matrix:

$$L_{reg} = \|I - AA^T\|_F^2, \tag{5.3}$$

where A is the feature alignment matrix predicted by the mini-network. An orthogonal transformation does not lose information in the input, which is a desired feature. In Table 5.1 the effects of different transformations on the

Transform	Accuracy
none	87.1
input (3x3)	87.9
feature (64x64)	86.9
feature (64x64) + reg.	87.4
both	**89.2**

Table 5.1: Effect of input and feature transformations on the accuracy on the ModelNet40 dataset.

accuracy on the ModelNet40 datasets are compared. Experimental results show that the highest accuracy can be obtained by using both transformations on input and features.

5.2.4 *Implementation and hyperparameters*

The PointNet architecture is implemented in `Python 3.6` using `Pytorch v. 1.3` as deep learning framework. The PointNet pipeline is implemented as indicated in the official release [1].

At run-time different set of hyperparameters are set:

- *Number of epochs*: represents the number training iterations. This parameter usually has high value as it could take many iterations to find the best weights for the model. However, training for too many epoch may cause the network to overfit, decreasing the generalization ability of the network. In our experiments we decided to use early stopping, a technique which is commonly used to assist the training of neural networks. After each epoch, the validation set is used to evaluate the generalization capabilities of the network on samples never seen during training. When the network performance on the validation set begins to decrease (or, equally, when its validation error starts to increase), the training procedure is stopped. This prevents the network from overfit since the training process is interrupted as soon as the generalization capabilities of the network start to decrease. For all our experiments we set the number of epochs to 150 with early-stopping patience set to 10, meaning that if the validation error does not decrease for 10 consecutive epochs the training process is stopped. Testing is then performed by the model that according to validation has the highest overall accuracy.
- *Optimization method*: defines the algorithm used to optimize the loss function and to train the network to perform the desired task. The optimization strategy is the same as the basic PointNet implementation

that uses the *Adam* optimizer. The optimizer is configured using a learning rate, the parameter that controls the magnitude of every weights update, and beta-coefficients used for computing running averages of gradient and its square. In our experiments the learning rate is set to 0.001 with $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

- *Learning Rate Scheduler*: the learning rate scheduler is responsible for adjusting the learning rate during training by progressively decreasing the learning rate parameter in order to reduce the magnitude of each update in late stages of the learning procedure. In our experiments we adopt the `Step learning rate scheduler`, that decays the learning rate of each parameter group by gamma every step-size epochs. In all our experiments we use $step - size = 20$ and $\gamma = 0.5$.
- *Batch size*: determines how many samples are fed simultaneously to the network. The bigger the batch-size the. The choice of this parameter may influence the learning convergence in a similar ways as the learning rate does. Using a higher number of samples, indeed, usually results in a smoother decrease of the loss function since the training procedure can base its updates on a greater number of samples. In our experiments we used batch-size 4 or 8. Bigger sizes would not allow for the samples to fit inside the GPU memory. We adopted the *gradient accumulation* technique by collecting the gradients of different batches for n steps. Optimization is performed only after the accumulation has been concluded. This has the effect of simulating bigger batch sizes.

5.3 POINTNET++ ARCHITECTURE

Although PointNet shows great capabilities to perform on three-dimensional point sets, it lacks the ability to capture local structures induced by the metric space points live in, limiting its ability to recognize fine-grained patterns and generalizability to complex scenes. Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas propose a novelty architecture, PointNet++ [2], that applies PointNet recursively on a nested partition of the input point cloud. By exploiting metric space distances, the network is able to learn local features with increasing contextual scales. The idea derives from traditional CNNs: at lower levels, neurons have smaller receptive fields whereas at higher levels they have larger receptive fields. The ability to abstract local patterns along the hierarchy allows better generalizability to unseen cases. Furthermore, the improved architecture implements new learning layers that are able to combine features from multiple scales. This allows the network to generalize better on point sets that are generated with varying densities.

PointNet++ is designed in a hierarchical manner where the input space is partitioned into overlapping regions by the distance metric of the underlying

space. In each region, local features are extracted capturing fine geometric patterns from small neighborhoods. Such features are combined at increasing scales to produce higher-level features. This process is repeated until features for the whole point set are generated. The main functionalities of PointNet++ are within two core layers called *Set Abstraction (SA) level* and *Feature Propagation (FP) layer*.

5.3.1 *Set Abstraction level*

PointNet uses a single max-pooling operation to aggregate the whole point set. On the other hand, PointNet++ builds a hierarchical grouping of points and progressively abstracts larger and larger regions along the hierarchy. This hierarchical structure is composed of several *set abstraction levels*. At each level, a set of points is abstracted to produce a new set with *fewer* elements. Each set abstraction level is composed of three layers:

- *Sampling layer*: given input points $\{x_1, ..., x_n\}$ the *farthest point sampling (FPS)* algorithm chooses a subset of points $\{x_{i1}, ..., x_{im}\}$, such that x_i is the most distant point from the set $\{x_{i1}, ..., x_{ij-1}\}$ with regard to the rest of the points. This generates a receptive field in a data dependent manner.

- *Grouping layer*: the grouping layers find local region sets by defining neighboring points around the centroids found at the SA level. Given a point set $N \times (d + C)$ and a set of centroids $N' \times d$ the grouping layers find $N \times K \times (d + C)$ group of points, where each group corresponds to a local region and K is the number of points in the neighborhood of centroid points. PointNet++ focuses on *Ball Query* and *K-Nearest Neighbors (kNN)* algorithms to find local regions for each centroids. Compared with kNN, ball query's local neighborhood guarantees a fixed region scale thus making local region features more generalizable across space, which is preferred for tasks requiring local pattern recognition.

- *PointNet layer*: As stated in Section 5.2, PointNet is a valid and robust feature extractor for point sets. In this layer N' local regions of points with data size $N' \times K \times (d + C)$ is abstracted employing a PointNet architecture by its centroid and local features that encode the centroid's neighborhood, forming a $N' \times (d + C')$ output vector. The coordinates of each region a firstly translated into a local frame relative to the centroids point. By using relative coordinates together with point features the network is able to capture per-point relation in the local regions.

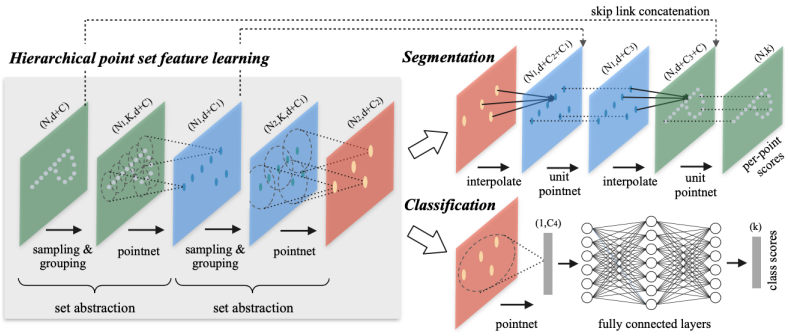


Figure 5.2: The PointNet++ pipeline. This particular layout represents the single scale point grouping.

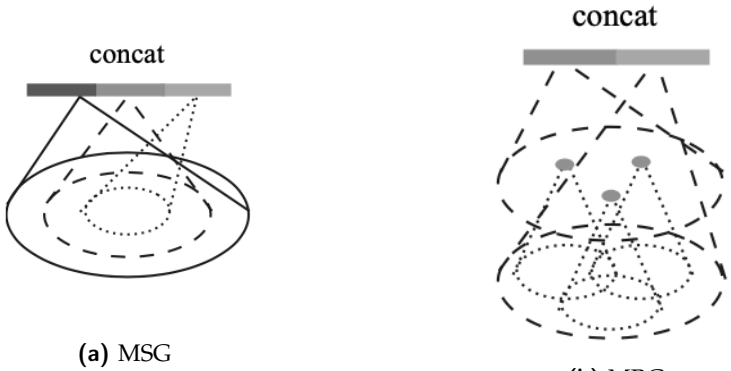
5.3.2 *Feature Propagation Layer*

In the set abstraction layer, the original point set is subsampled and features are created for local regions. In segmentation tasks however it is necessary to provide point features for *all* the original points. A solution is to sample *all* points as centroids, which would lead to very high computational costs. A better solution is to *propagate* the generated features from subsampled points to the original points. Just like the sampling layers, the feature propagation is structured in a hierarchical manner using distance-based interpolation and cross-level skip-links (Figure 5.2). In each feature propagation level, the point features are propagated from $N_l \times (d + C)$ point to $Nl - 1$ points where N_{l-1} and N_l (with $N_l \leqslant N_{l-1}$) are point sets of input and output of set abstraction level *l*. Since the two layers contain a different number of points, interpolation is used by means of inverse distance weighted average based on *k* nearest neighbors, as shown in Equation 5.4. Usual values are $k = 3$ nearest neighbors and $p = 2$. The interpolated features on N_{l-1} points are concatenated with *skip-link* point features from the set abstraction level. The resulting features are then processed by a unit PointNet, similar to a one-by-one convolution in CNNs. Finally, a few shared connected and ReLU layers are applied to update each point's feature vector. The process is repeated until all features have been propagated to the original points.

$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f_i^{(j)}}{\sum_{i=1}^{k} w_i(x)} \text{ where } w_i(x) = \frac{1}{d(x, x_i)^p}, j = 1, ..., C \tag{5.4}$$

5.3.3 *Robust Feature Learning under Non-Uniform Sampling Density*

An important analysis to make when processing point clouds is to understand the distribution of points within the data structure. PointNet has shown very



(a) MSG **(b)** MRG

Figure 5.3: a) Multi-scale grouping b) Multi-resolution grouping

good performance when it comes to recognizing objects within point clouds generated synthetically by sampling 3D meshes uniformly. This is different with point clouds generated by a sensor for example. In this scenario, the point set will often have more densely populated areas than others. The variability of density within the point cloud is certainly a factor to be taken into account. Features learned in dense data may not generalize to sparsely sampled regions. Consequently, models trained for sparse point clouds may not recognize fine-grained local structures. During the design of PointNet++, this aspect has been taken into account. Ideally, PointNet++ should be able to inspect as closely as possible into a point set to capture the finest details in densely sampled regions. However, such close inspection is prohibited in low-density areas because local patterns may be corrupted by the sampling deficiency. In this case, the architecture should look for larger-scale patterns in the greater vicinity. This problem is solved in PointNet++ by proposing density adaptive PointNet layers, that learn to combine features from regions of different scales when the input sampling density changes. Each abstraction level extracts multiple scales of local patterns and combines them intelligently according to local point densities. In terms of grouping local regions and combining features from different scales, PointNet++ proposes two types of adaptive layers (Figure 5.3):

- *Multi-scale grouping (MSG)*: a simple but effective way to capture multi-scale patterns is to apply grouping layers with different scales followed by according PointNets to extract features of each scale. Features of different scales are concatenated from a multi-scale feature. In training, the network learns an optimized strategy to combine the multi-scale features. This is done by randomly dropping out input points with a randomized probability for each instance, called random input dropout. Specifically, for each training point set, a dropout ratio θ uniformly sampled from $[0, p]$ where $p \leqslant 1$ is defined. For each point, the network

randomly drops a point with probability θ. In most settings, $p = 0.95$ is chosen to avoid generating empty point sets. In doing the network trains on point sets of various sparsity (induced by θ) and varying uniformity (induced by randomness in dropout). During the test phase, all available points are kept.

- *Multi-resolution grouping (MRG)*: as shown in Figure 5.3, features of a region at some level L_i are a concatenation of two vectors. One vector (left in figure) is obtained by summarizing the features at each subregion from the lower level L_{i-1} using the set abstraction level. The other vector (right) is the feature that is obtained by directly processing all raw points in the local region using a single PointNet. When the density of a local region is low, the first vector may be less reliable than the second vector, since the subregion in computing the first vector contains even sparser points and suffers more from sampling deficiency. In such a case, the second vector should be weighted higher. On the other hand, when the density of a local region is high, the first vector provides information of finer details since it possesses the ability to inspect at higher resolutions recursively in lower levels. Compared with MSG, this method is computationally more efficient since this method avoids feature extraction in large-scale neighborhoods at the lowest levels.

5.3.4 *Implementation and hyperparameters*

The PointNet++ architecture is implemented in `Python 3.7` using `PytorchLightning v. 1.3` as deep learning framework. The PointNet++ pipeline is implemented as indicated in the official release [2].

At run-time different set of hyperparameters are set:

- *Number of epochs*: represents the number training iterations. This parameter usually has high value as it could take many iterations to find the best weights for the model. However, training for too many epoch may cause the network to overfit, decreasing the generalization ability of the network. In our experiments we decided to use early stopping, a technique which is commonly used to assist the training of neural networks. After each epoch, the validation set is used to evaluate the generalization capabilities of the network on samples never seen during training. When the network performance on the validation set begins to decrease (or, equally, when its validation error starts to increase), the training procedure is stopped. This prevents the network from overfit since the training process is interrupted as soon as the generalization capabilities of the network start to decrease. For all our experiments we set the number of epochs to 150 with early-stopping patience set to 15,

meaning that if the validation error does not decrease for 10 consecutive epochs the training process is stopped. Testing is then performed by the model that according to validation has the highest overall accuracy.

- *Optimization method*: defines the algorithm used to optimize the loss function and to train the network to perform the desired task. The optimization strategy is the same as the basic PointNet implementation that uses the *Adam* optimizer. The optimizer is configured using a learning rate, the parameter that controls the magnitude of every weights update. In our experiments the learning rate is set to 0.001.

- *Learning Rate Scheduler*: the learning rate scheduler is responsible for adjusting the learning rate during training by progressively decreasing the learning rate parameter in order to reduce the magnitude of each update in late stages of the learning procedure. In our experiments we adopt the `Lambda learning rate scheduler`, that sets the learning rate of each parameter group to the initial learning-rate times a given function. The learning rate decay function is set to 0.5.

- *Batch size*: determines how many samples are fed simultaneously to the network. The bigger the batch-size the. The choice of this parameter may influence the learning convergence in a similar ways as the learning rate does. Using a higher number of samples, indeed, usually results in a smoother decrease of the loss function since the training procedure can base its updates on a greater number of samples. In our experiments we used batch-size 4 or 8. Bigger sizes would not allow for the samples to fit inside the GPU memory. We adopted the *gradient accumulation* technique by collecting the gradients of different batches for n steps. Optimization is performed only after the accumulation has been concluded. This has the effect of simulating bigger batch sizes.

- *Number of centroids*: this parameter determines the number of centroids used in each of the 4 set abstraction levels. The default number of centroids are (in order of set-abstraction layer they belong to) $[1024, 256, 64, 16]$. To achieve a better partition of the space we experimented with a larger size at lower levels using $[4096, 1024, 256]$ centroids.

5.4 PERFORMANCE ANALYSIS OF POINTNETS

PointNet and PointNet++ are robust feature learners for point clouds. Although the underlying architecture is relatively simple, PoinNet has excellent capabilities as a feature extractor in tasks such as classification or segmentation. More complex structures, where the density of points within the cloud

Method	Input	Accuracy (%)
VoxNet [91]	vox.	85.9
Subvolume [92]	vox.	89.2
MVCNN [93]	img.	90.1
PointNet (vanilla)	pc	87.2
PointNet (with transforms)	pc	89.1
PointNet++	pc	90.7
PointNett++ + norm.	pc	91,9

Table 5.2: 3D object classification on ModelNet40

is variable for example, need instead the hierarchical pipeline of PoineNet++ to generate more consistent predictions. In the next Section we will provide experimental results carried out on PointNet and PointNet++ by the authors. Each performance evaluation is compared with other state-of-the-art models.

5.4.1 *Evaluation on ModelNet40*

The two architectures were tested on the ModelNet40 dataset [90], a dataset comprising 40 classes of artificially generated CAD objects containing 9,843 samples for training and 2,468 samples for testing. From each CAD model, 1024 points were extracted uniformly based on the face area, which is then normalized into a unit sphere. Augmentation is performed on training samples by using rotation along an axis and random jitter of the points by Gaussian noise with zero mean and 0.02 standard deviation. The performances are compared in Table 5.2 to other state-of-the-art models and a baseline model built using MLP on traditional features extracted from point clouds (point density, D2, shape contour, etc.). There is still a small gap between basic PointNet and multi-view based method (MVCNN), which is due to the loss of fine geometry details that can be captured by rendered images. With PointNet++ more fine-grained patterns can be detected along the hierarchy, allowing for a performance on-par with state-of-the-art CNNs. Although the dataset is mostly made up of low-complex 3D objects, the performances of the two architectures show that the core layers of PointNet are indeed able to learn structures via raw point clouds.

5.4.2 *Evaluation on ScanNet*

To validate that PointNet and PointNet++ are suitable for large scale point cloud analysis, both architectures are evaluated on semantic scene labeling task. The performances in this task are evaluated on the ScanNet dataset [94],

Method	Input	Accuracy (%)
3DCNN [95]	vox.	73.0
PointNet (with transforms)	pc	73.9
PointNet++ SSG	pc	83.3
PointNet++ MSR	pc	83.4
PointNet++ MSG	pc	84.5

Table 5.3: Segmentation Performance on ScanNet

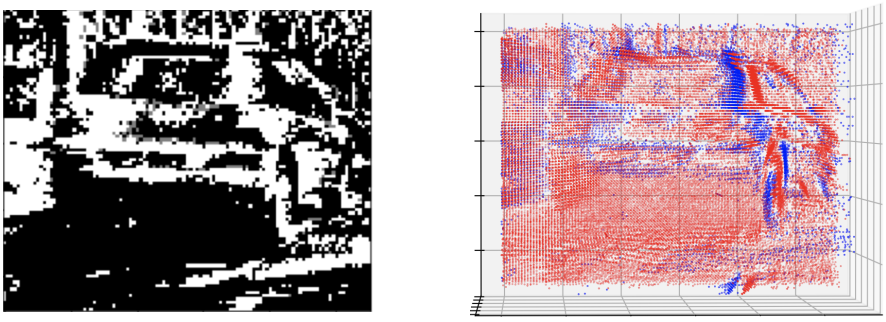
an RGB-D video dataset containing 2.5 million views in more than 1500 scans, annotated with 3D camera poses, surface reconstructions, and instance-level semantic segmentations. The data was recorded with an easy-to-use and scalable RGB-D capture system that includes automated surface reconstruction and crowd-sourced semantic annotation. The authors of ScanNet provided also a neural network to provide a baseline score by applying a fully convolutional neural network on voxelized scans based on [95]. RGB information contained in the dataset is not considered and only on the scanned geometry is used as input. To make a fair comparison, both PointNet architectures do not consider RGB information in all experiments and convert point cloud label prediction into voxel labeling following Dai et al.'s approach. The performances are compared in Table 5.3. Learning directly on point clouds, sampled in data depending manner, instead of voxelized scans reduces additional quantization error resulting more effective learning. Finally PointNet++ shows the best overall performances due to its hierarchical structure.

5.5 EVENT-BASED DATA AS POINTNET INPUT

Equation 5.1 formalizes the notion of the point cloud as a set of independent points in a 3D space, where each point is identified by its coordinates. This representation shares many similarities with an event stream generated by a dynamic vision sensor with a pixel-grid size MxN, as formalized by Equation 4.1:

$$\Psi = \{e_i\}_{i=1}^{I}, \text{ with } e_i = (x_i, y_i t_i, p_i)$$

where (x_i, y_i) are the coordinates of the pixel generating event, t_i is the time stamp and $p_i \in 0, 1$ or $p_i \in -1, 1$ the polarity of event i. It is easy to see that each event stream can be modeled as a 3D point cloud, with the third dimension being the time. The polarity of each event is an additional feature of each point in the point cloud, just like color for RGB-D cameras. In Figure 5.4 we show an event stream represented as a 2D image and how it compares to its 3D point cloud representation. It should be noted that these



(a) Image reconstruction

(b) Point cloud

Figure 5.4: Sample taken from the 'car' class of the NCars dataset. a) Image reconstruction of the sample. Positive polarity events have a white pixel, negative polarity events are represented by black pixels. b) Point cloud of the sample. Red dots are negative polarity events, blue dots are positive polarity events.

objects detected by event-based sensors do not represent the actual 3D shape of the object, but they represent how the object is perceived by the sensor over time. As time goes on, more changes i.e. more motion is detected by the sensor leading to a point cloud increasing in size. Intuitively, one might think that having more events in the point cloud is an advantage for neural networks: more points mean that the scene is better represented and it is, hence, easier to extract defined structures such as cars or pedestrians. This is generally not true for event-based sensors. In rapidly changing scenarios, such as urban driving, a huge amount of data is generated by the event sensor. By analyzing some random samples from the DDD17 dataset, around 350.000 events are generated each second on average by the DAVIS sensors. An event stream over 10s would produce 3.5 million events. However, extracting useful information from such a point cloud is far from easy. Firstly, the large number of points is very difficult for neural networks to manage. Furthermore, the cloud of points taken as a whole generates structures that are difficult to distinguish. Just think of a car that turns left in front of the sensor. Initially, the car generates events in the center of the sensor. As it turns, events are generated on the left side of the sensors. The point cloud thus obtained should contain an object, the car, which forms an 'L' structure. Taking a sufficiently long time interval, the car could assume arbitrary structures that are difficult for a neural network to learn. Conversely, if we consider a point cloud generated over a time span of 1ms only a small number of events will be generated. If the number of points is too small, it is impossible to extract meaningful objects. For object recognition tasks it is consequently important that the time domain (the z-axis in a point cloud) is set properly. For this purpose we adopt the notion of *time-windows* in order to learn efficient representation from data streams.

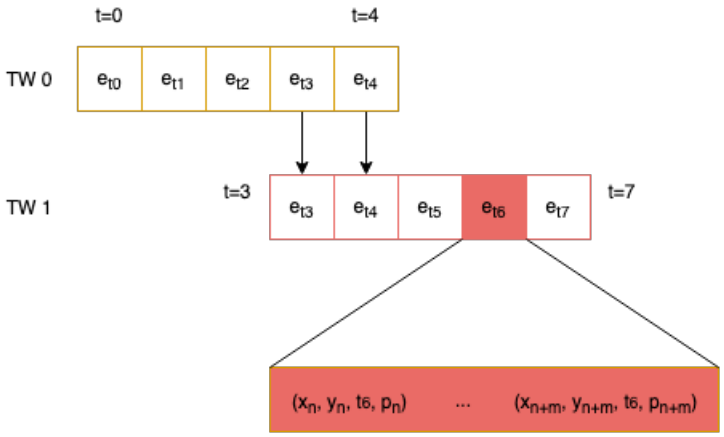


Figure 5.5: Example of the sliding time window mechanism. The time window length is $T = 5\mu s$ and the stride is $s = 3\mu s$. Each square e_{ti} represents all the events happening at time stamp t_i. In this case the set of events for time-window TW_1 is given by $\bigcup_{i=3}^{i=7} e_{ti}$.

5.5.1 *Sliding time-windows*

Finding the right cut-off time point is not trivial: point clouds generated in a smaller time span (e.g. 500µs) are less resource-intensive and can be processed faster, but often lack enough structural information to extract useful patterns. Conversely, point clouds generated in a larger time span (e.g. 5s) could contain information that is not relevant anymore and is more resource-intensive to handle. To try different sized point clouds, we choose to handle event-based data using a *sliding time-window* representation (Figure 5.5).

Given a time interval T and a stride s a time window can be described as :

$$TW_i = \{e_j = (x_j, t_j, t_j, p_j) | s * i \leqslant t_j \leqslant s * i + T\} \tag{5.5}$$

The sliding time window representation is easy to implement. At inference time this implementation can be modeled as a rolling buffer of length T containing an arbitrary number of events. The buffer is updated according to the stride s, which means that the last s events will be inserted into the buffer while the oldest s events will be discarded. This creates a new time window on which to perform inference. By finding the right buffer length and stride the sliding time window representation allows to dynamically handle the incoming stream of events with the benefit of fast response time without losing relevant information of past events. To improve performance, a decision buffer can be implemented to ignore unreliable predictions.

The sliding time window representation has been used in [88], a pipeline that combines a PointNet [1] architecture with sliding time windows on event streams to classify hand gestures generated by a neuromorphic sensor.

The paper proposed different time window lengths and strides to find the best configuration for the given task. The network is effectively able to learn and classify hand gestures with a fast response time (100ms). A similar architecture has been proposed in EventNet [87], a neural network designed for real-time processing of asynchronous event streams in a recursive and event-wise manner. The EventNet architecture proposes an optimized version of PointNet that focuses on fewer computational operations, allowing the network to achieve higher response times to incoming event streams without losing inference performance.

Our work has mainly focused on applying PointNet and PointNet++ to automotive datasets. This combination is an excellent candidate for meeting the demands of the autonomous driving sector. On the hardware side, event cameras have attractive features in terms of time resolution, energy efficiency, and dynamic range. On the software side, PointNet is a good candidate to efficiently process the data streams generated by the sensor.

5.5.2 *Noise processing*

The pre-processing of the raw data is essential for extracting meaningful information for sensor systems. An event-based neuromorphic vision sensor not only captures the change in the light intensity caused by moving objects, but also generates some noise activities due to the movements of background objects and the sensor noise such as temporal noise and junction leakage currents. To improve the quality of data we adopt a spatial-temporal correlation filter on our time windows. The filter searches for the most recent neighborhood (defined by distance d) event around each event's pixel location. So given an event $e^{j}_{t_i}$ in time-window tw_i, the neighborhood of pixel (x_j, y_j) is searched for events in the most recent time-window tw_{i-1}. If there are no events, then the event is considered as a noise point and discarded. Illustration 5.6 depicts the functioning of the filter using an example. This process effectively cleans out events located in isolated regions. A useful application of the filter is with data that is being processed by PointNet++. For the purposes of effective machine learning, the FPS algorithm adopted in the set abstraction levels of PointNet++ should select centroids that go on to define regions containing useful and interesting structures. If instead a noise point is selected as a centroid then the information contained in its neighborhood will be of little relevance as it will not contain useful structures.

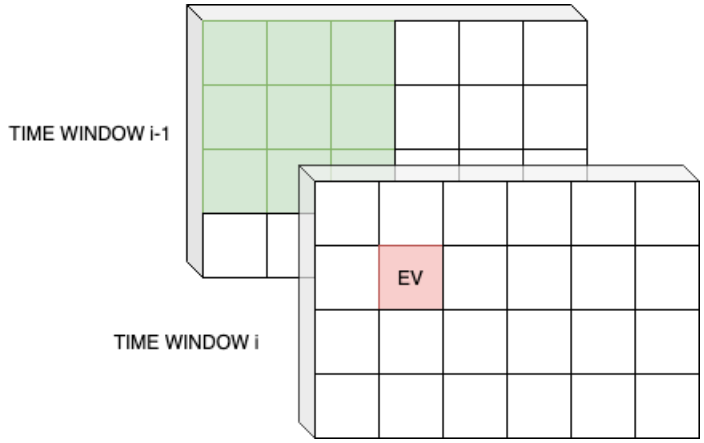


Figure 5.6: Example of how the noise filter works. For each event (in red) of time window i, the filter searches for events in the neighborhood (in green) defined by d in the previous time window $i-1$. In this case $d = 1$

5.6 EVENT-BASED CLASSIFICATION ON POINTNET-BASED ARCHITECTURES

In Section 5.4.1 we reported on the performance that PointNet and its successor have achieved in classifying three-dimensional objects, generated synthetically by sampling CAD meshes from the ModelNet40 Dataset. Our first approach in applying PointNet-based architectures in the automotive field is based on the classification of short data streams from the NCars dataset [68]. The dataset, briefly introduced in Section 4.1.1, was created as part of the HATS representation proposed by Sironi et al. The data was generated using an ATIS camera recording 80 minutes of video. The gray-scale measurements generate were used to generate gray-scale images. A state-of-the-art object detector was then used to extract bounding boxes of vehicles and background images. Subsequently, the data was cleaned manually to ensure that all objects were consistent with the predictions made by the detector. Since the gray-scale images have the same temporal resolutions as the change detection events, it is easy to extract event-based samples from the full stream corresponding to the predicted bounding boxes. The NCars dataset is composed of $12,336$ car samples and $11,693$ non- cars samples (background). The dataset was split into 7940 car and 7482 background training samples, and 4396 car and 4211 background testing samples. Each example lasts 100 milliseconds stored in generic `.dat` files. We generated a *validation dataset* by using a balanced $80-20$ split on the training set. The validation set is used to check the performance after each epoch and is used as a discriminator of the early stopping regularization.

5.6.1 *Configuration of experiments*

Each experiment is generated from a configuration file containing values that can be set dynamically. In addition to the network parameters defined in the previous section, various parameters allow the input data to be modified to improve performance. As already mentioned in Section 5.5.1, time windows are an effective tool for creating point clouds from a stream of events. Each sample is divided at run-time into several time windows, which are assigned the class of the source sample. For example, a 100ms sample of the class 'car' can be split into four disjointed time windows labeled 'car' by using a 25ms long window with a 25ms stride. Below is the complete list of parameters of the configuration file:

- *Time window length*: defines the length in µs of the time window. A time window of 5.000µs means that the difference of the most recent and the oldest timestamp in the time window cannot exceed 5.000µs. Different values have been chosen, ranging from 5.000µs to 100.000µs.
- *Time window step*: defines by how much the time window moves forward. For example given a time window with length $length = 5.000\mu s$, from $t_{first} = 0\mu s$ and $t_{last} = 4.999\mu s$, a time step of 2.500µs generates a new time window with events in range $t_{first} = 2.500\mu$ and $t_{last} = 7.500\mu s$. Different values have been chosen, ranging from 5.000µs to 50.000µs.
- *Number of events*: defines the minimum number of events a time window must contain. If the number is below a certain threshold then splitting the sample into smaller time windows would create point clouds containing an insufficient number of events. Experimental results on PointNet have shown that a point cloud should contain at least 512 points.
- *Polarity*: the PointNet architecture can handle additional information about points, such as normal or RGB color values. Event-based cameras provide the polarity feature, which can assume either value 0 (off event) or 1 (on event). IIn some experiments, the polarity of the events was an integral part of the point clouds in order to assess whether the presence or absence of this feature could influence the performance of the network.
- *Temporal difference*: is a transformation of the timestamps of a given point cloud. Timestamps are expressed in µs and can therefore assume large integer values. On the other hand. the x, y pixel coordinates are within a defined domain depending on the pixel array size of the camera. Larger numbers have the tendencies to dominate over smaller numbers inside neural networks. To avoid this, each timestamp is replaced with the

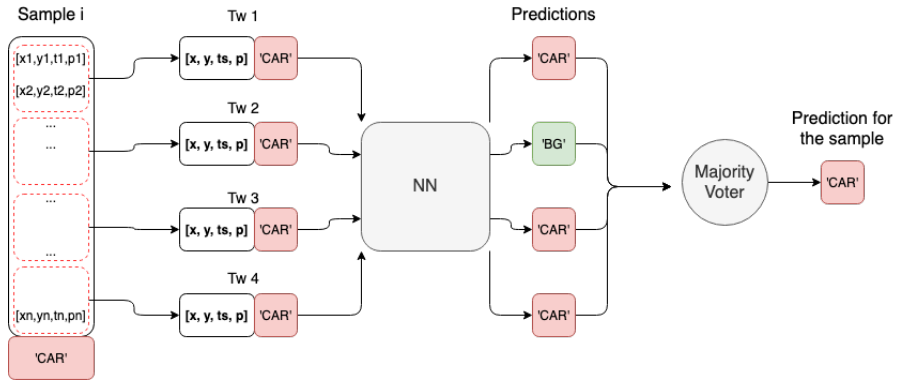


Figure 5.7: Sample i is split into 4 disjoint time windows, each containing a variable number of events. Each time window has the label of the original sample. The neural network predicts a class for each time window. The majority voter picks the most represented class and assigns it to the original sample.

temporal difference calculated as $t_{delta} = t_{current} - t_{previous}$. The events with the smallest time value (i.e. the first events of the time window) will have $t_{delta} = 0$.

- *Min-Max Normalization*: is a transformation that maps values in a certain range to values between 0 and 1. This has a regulatory effect since every time window contains values in $[0, 1]$ for each of its features.
- *Input and Feature transforms*: input and feature transforms are performed by the so-called *T-Nets* inside the PointNet pipeline, as mentioned in Section 5.2.3. Experimental results show that keeping both transformation matrices produces better performance in the network. In each of our experiments, both input and feature transform has been used.

5.6.2 *Evaluation metrics*

The performance of each model is evaluated using accuracy, calculated as the total number of correctly classified time windows divided by the total number of time windows in the dataset considered. To also provide a metric for each sample, the scores of each time window that make up the sample are accumulated. A majority voter finally chooses the most represented class. An example of how the majority voter works is shown in Figure 5.7.

5.6.3 *Experimental results*

We used different sets of configurations to find the most effective way of learning event-based point clouds using the PointNet framework. Parameters are loaded dynamically at run-time from configuration files.

T. Length	T. Step	Accuracy	MV Accuracy
5.000	5.000	0,898	0,901
15.000	5.000	0,891	0,910
50.000	5.000	0,847	0,851
5.000	15.000	0,829	0,848
15.000	15.000	0,874	0,88
50.000	15.000	0,857	0,866
100.000	-	0,697	0,697

Table 5.4: PointNet performance on NCars. All samples use temporal difference and no polarity.

Firstly we considered different time window lengths and time window steps to evaluate the performance. Each time window is composed of a point cloud containing $tw_i = [x, y, t_{delta}]$. Temporal difference values have been used instead of timestamps. The polarity feature has not been considered initially, as we wanted to treat the input data as a 3D point cloud without additional features. The results are shown in Table 5.4. Time windows with a length equal to the length of the whole sample, 100ms, perform worst. Smaller time windows in the order of 5ms to 15ms achieve the highest accuracy scores. Although the time windows are relatively small, they contain enough points for the network to effectively generalize the structures they contain. One of the biggest advantages of PointNet is the ability to learn structures through key points. On the other hand, larger time windows may contain structures that are difficult to distinguish within the point cloud. The time step parameter determines how much the time window advances over time. The time step parameter determines how far the time window moves forward. If the parameter coincides with the length of the time window, then each sample will be divided into disjointed time windows. Assigning smaller values to the time step parameter than the length has the effect of creating overlapping time windows. Thus, the smaller the step, the more time windows will be generated at run-time. This has a similar effect to data augmentation in that more different samples will be presented to the network. Experimental data confirm this hypothesis in that with the same window length, lower time step values generate better-performing models.

We then experimented to assess how polarity, number of events, and normalization can influence the results. To do this, we took as reference the length and step values of the time window of the best performing configuration. The polarity takes on a binary value for each event in the new point.

T. Length	T. Step	Feat/Trans	Accuracy	MV Accuracy
5.000	15.000	-	0,891	0,910
5.000	15.000	Polarity	0,894	0,904
5.000	15.000	Normalization	0,884	0,897
5.000	15.000	1024 min events	0,878	0,891

Table 5.5: PointNet performance on NCars. Additional features or transformations do not affect the performance in a significant way.

The results show that adding this feature does not affect the performance of the data. Using min-max normalization has the effect of standardizing data in a range between 0 and 1 (including timestamps).

The configuration with normalized data produces more robust performance and is less prone to fluctuations in both training and validation. This is a desirable effect as the network makes use of early-stopping, which in the event of validation loss fluctuations could lead to a premature halt in learning. Yet, it does not increase the performances of the models.

Finally, we tried different approaches regarding the minimum number of events within a time window. If a time window contains less than n events, then it is stretched beyond its standard length to accumulate at least n events. However, choosing values that are too high may produce time windows that are too long, which, as shown in Table 5.4, decreases performance. The effects of polarity, the minimum number of events, and normalization are compared in Table 5.5.

We then applied the best PointNet configuration to PointNet++ using both SSG and MSG. The performance of PointNet++ improves when using normalisation on the input data. This is due to the fact that PointNet++ is applied recursively on three dimensional sub-regions in space. The parameterisation of these sub-regions is more intuitive and quicker using values between 0 and 1. Experimental results a slight improvement of the basic version of PointNet. The hierarchical structure of PointNet++ extracts features on different levels and scales which are combined to create robust features for the global point cloud.

PointNet and PointNet++ have shown excellent capabilities in classifying objects generated datasets in the automotive field 5.6. Our best model performs better than HATS[68] which uses active surfaces instead of point clouds. Finally, the performance is slightly lower than MatrixLSTM [4] which processes images using LSTMs instead of asynchronous point clouds.

Method	Accuracy
PointNet	0,910
PointNet++ SSG	0,899
PointNet++ MSG	0,921
HATS [68]	0,902
Matrix-LSTM [4]	0,950

Table 5.6: Performance of PointNet with time window compared to state-of-the art models.

5.7 EVENT-BASED SEMANTIC SEGMENTATION ON POINTNET-BASED ARCHITECTURES

The process of semantic segmentation poses an interesting problem in the automotive sector. Considering a cloud of points generated by a flow of events of a neuromorphic sensor, semantic segmentation has the task of assigning a certain label to each point in the point cloud. The final objective is to divide the scene into meaningful structures. For example, in an automotive dataset, a neural network should effectively distinguish pedestrians, cars, roads, and other objects in the scene. Our semantic segmentation experiments are based on samples extracted from the DDD17 dataset published by Binas et al. and briefly introduced in Section 4.1.3. The dataset is composed of over 12h of recordings using a 346x260 pixel DAVIS sensor on urban and highway driving scenarios. The DVS and APS data is recorded along with vehicle speeds, GPS position, driver steering, throttle, and brake measurements. The DVS sensor generates the event streams that are of interest to us. However, the dataset does not include event labels. Alonso and Murillo propose the EvSegNet architecture [18] which exploits a subset of samples from the DDD17 dataset for semantic segmentation. The DDD17 dataset consists of 40 sequences of different driving set-ups. These sequences were recorded on different scenarios (e.g. motorways and urban scenarios), with very different illumination conditions: some of them have been recorded during day-time (where everything is clear and visible), but others have overexposure or have been recorded at night, making some of the grayscale images almost useless for standard visual recognition approaches. Therefore only a subset of DDD17 has been used for the label generation process. Ground truth labels were generated from frames generated by the APS sensor in a 50ms time interval using a state-of-the-art CNN pre-trained on *Cityscapes* [96] dataset. The final model obtains 83% categories mIoU on the Cityscapes validation data. This is still a bit far from the top results obtained on that dataset with RGB images (92% mIoU), but enough quality for the process. The network generates labels for a total of six different categories: road, background + buildings

Figure 5.8: A sample taken from the DDD17 dataset. On the right side is the gray-scale image generated by using 50ms of APS data. In the middle is the corresponding labels image generated using a pre-trained CNN. By integrating over the same time interval, we can obtain an image reconstruction using the events provided by the DVS sensor. Each event is assigned the value of the corresponding pixel of the labels reference image.

and constructions, objects (traffic lights, signs, etc.), nature, pedestrians, and vehicles. The labels are stored in `.png` format and were used as ground-truth during the training and evaluation of experiments. For each label to be assigned to the event at the right time instant, it is necessary to use the same integration interval used to generate the reference frames, i.e. 50ms. Thus, the configuration used in all our segmentation experiments uses time windows of 50ms length with 50ms time steps. This generally generates voluminous point clouds that are consistent with the labels found in [18].

5.7.1 *Configuration of experiments*

Below we propose different configuration and setup strategies that are aimed to improve the learning abilities of both PointNet architectures.

- *Sub-sampling*: the time windows generated contain a variable number of points depending on the scene perceived. Since these are mostly samples taken from urban driving scenarios with frequent changes in the scene, the point clouds within our time windows contain a large number of points. This could create memory problems during the training phase on the GPU. To overcome this problem, we adopted sub-sampling measures on point clouds that are too large, without losing any structural information contained in them. The first method considers only the n latest events inside the point cloud. The second method sub-samples n data points from 3D space evenly, by dropping random points at uniform intervals so that every time slice is equally represented by events. The value of n is set at 30.000 events, as bigger point clouds can lead to memory issues during training.
- *Weighted random sampler*: the recordings taken into account by EvSeg-Net for label generation concern urban driving scenarios. This makes it possible to include classes such as pedestrians, which would not

have been represented in highway driving. In any case, the dataset suffers from class imbalance. Most of the samples contain road, background+buildings, and vehicle labels. Nature, objects, and pedestrians are not so common. We implemented therefore a *weighted random sampler*, that picks time windows randomly based on a fitness score. The higher the score the more likely will the sample be picked during training. We created a fitness function that takes into account the number of different classes represented over the overall number of pixels in the point cloud. The value of each pixel is based on the total frequency of that class in the dataset. This means that background events have a low score and events labeled as a pedestrian will have the highest score. During training samples containing uncommon labels are more likely to be picked. This has the effect of increasing the performance of less represented classes.

- *Augmentation*: point clouds contained inside a time window are subject to random rotations between $(0, 30°)$. This avoids that certain labels are predicted in fixed regions of the scene resulting in overall better generalization performance.
- *Normalization*: every time window is processed with mix-max normalized input features. Experimental results have shown that non-normalized data generate very high loss values. This does not allow an objective evaluation of performance nor does it allow the correct use of early stopping.
- *Number of classes*: as specified above, the labels are generated by a pre-trained CNN. The labels, hence, have inaccuracies, especially in smaller objects such as cones, road signs, traffic lights. The segmentation of trees and plants is also often inaccurate. In addition to the segmentation on the 6 labels, we propose the segmentation made on 4 classes, grouping in single class background, nature and, objects.

5.7.2 *Evaluation metrics*

To evaluate performance in semantic segmentation, it is not enough to use accuracy as in the case of prediction. For example, a network that predicts for all the points the class "background" would be able to obtain a discreet score of accuracy since almost all the scenes contain a large number of background points. A valid metric for segmentation task is the *mean intersection over union (mIOU)* or *Jaccard index*. The IoU score is calculated for each class as the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground

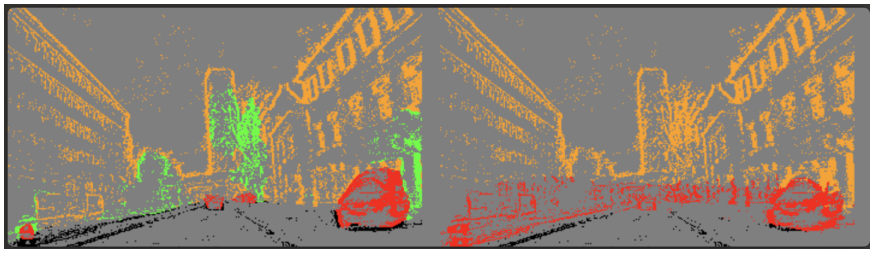


Figure 5.9: The left image shows events of the time window with ground-truth labels. The image on the right are events with predicted labels. PointNet struggles with highly detailed structures like trees (in green). PointNet also tends to predict a horizontal strip with the label 'car' (in red), being the region where vehicles are present in the training samples. This is an undesired outcome as the network was not able to capture structures within the point cloud.

truth. The scores of each class are summed together and divided by the total number of classes in order to obtain the average (mIOU).

5.7.3 *Experimental results*

Different configurations have been used to find the best segmentation strategy. The first experiments were performed using PointNet on all six classes, without using data augmentation. Performance with this configuration did not produce satisfactory results with an accuracy score of 0.68 and 0.39 mIOU. By visualizing the predicted labels we noticed that PointNet fails to capture local structures. PointNet learns to predict labels based on the particular location of the event and not based on effective structure. This can be seen in Fig. 5.9. To improve the performance we add augmentation and weighted sampling. Augmentation is used on time window during training to avoid the region-based bias of labels by rotating the point clouds randomly. Weighted sampling is used to improve the performance on less represented labels such as pedestrians. The same configuration was then applied to only 4 classes. The aim is to see whether reducing the number of classes and therefore of unrepresentative objects, will improve the overall performance. The general feeling is that PointNet learns a naive method of predicting semantic labels, based more on regions than on objects in the scene. Basically, it predicts labels there where it expects to find objects of that category, even if often there aren't any. This problem is due to PointNet's inability to capture local structures and patterns in the scene, especially in regions with varying densities. Experimental results on PointNet are shown in Table 5.7.

PointNet++ mitigates this problem by introducing hierarchical layers that are able to recognize local structures at different scales. This fact is reflected in the experimental data obtained on PointNet++. In addition to applying PointNet recursively and hierarchically, PointNet++ adds the functionality

Classes	Augmentation	Weighted sampler	Polarity	Accuracy	mIOU
6	No	No	Yes	0,684	0,291
6	Yes	Yes	No	0,689	0,278
6	Yes	Yes	Yes	0,709	0,299
4	No	No	Yes	0,848	0,395
4	Yes	Yes	Yes	0,871	0,421

Table 5.7: PointNet performances on DDD17

of multi-scale grouping that improves the networks learning ability by combining features from different scales. This has the effect of generating robust features even for point clouds with varying point densities. This is very useful in event streams generated by neuromorphic sensors. Some areas of the sensor are more subject to changes than others, depending on the scene. The PointNet++ experiments all feature augmentation, polarity, and weighted sampling which have proven to boost performances on the basic version of PointNet. The single-scale variant of PointNet++ has lower performances in accuracy and mIOU with respect to the multi-scale implementations as shown in Table 5.8. Multi-scale grouping versions have shown their effectiveness by aggregating features from different scales. To further exploit the MSG pipeline we implemented an MSG version that features the usual 4 SA layers but each layer has a larger number of centroids. This means that in each level more overlapping regions are defined. This is computationally more intensive but produces higher mIOU and accuracy results. Adding more centroids especially at lowers levels means that the network learns features with greater detail. This is useful for small objects, like traffic signs, branches, or objects further away from the camera. PointNet++ is able to extract structures within the scene reducing significantly the location-based bias that was typical of the basic version of PointNet.

PointNet++ with multi-scale grouping combined with an increased number of centroids grouping achieves the overall best results for our segmentation pipeline using 6 classes. By visualizing the results the pipeline can provide a good generalization of the scene as seen in Figure 5.10. Still, labels are not as consistent between consecutive frames as the ground-truth values. PointNet++ is a good feature extractor for static point clouds but lacks the memory element typically needed for sequences like time windows. We propose an addition to the PointNet++ architecture by adding two *Gated Recurrent Units*, a special kind of recurrent neural network that works as a memory element. The GRU units are used between the last Set Abstraction layer and its corresponding Feature Propagation layer. The aim is to track the position of the centroids (first GRU) and features generated (second GRU) in the last SA level, to generate labels that are more consistent across consecutive sequences.

Mode	Classes	Accuracy	mIOU
SSG	6	0,692	0,299
MSG	6	0,723	0,330
MSG + centroids	6	0,728	0,332
MSG + GRU	6	0,759	0,336
EvSegNet [18]	6	0,897	0,548
EvSegNet (events only) [18]	6	0,890	0,520
SSG	4	0,893	0,433
MSG	4	0,893	0,446
MSG + centroids	4	0,884	0,374
MSG + GRU	4	0,919	0,431

Table 5.8: PointNet++ performances on DDD17. The scores are compared to the baseline EvSegNet. The first baseline score refers to the performance on images. The second score takes into account only the pixel locations where events have occurred. This was done to create a score that is more comparable to our evaluation.

For this purpose, each gated unit processes 4 consecutive sequences at a time. The addition of GRU adds computational complexity but brings also advantages in terms of performance. Experiments on the DDD17 dataset have shown that models trained with both GRUs achieve smoother and more consistent predictions among consecutive frames. Yet, this approach is prone to generating inconsistent predictions in case of rapid changes in the scene. As an example, time windows that contain event streams of the car taking a turn are subject to a bigger volume mispredictions. This effect is alleviated after a few milliseconds. The approach is, consequently, more suitable in scenarios with little variation in the scene, for example on motorway driving.

In conclusion, the basic version of PointNet shows good aptitudes for segmenting macro-regions within a point cloud generated by event sensors. It lacks finesse when it comes to more detailed objects in the scene. The naive way of interpreting the scene is not sufficient for the automotive field. While PointNet++ succeeds in covering the gaps highlighted by its basic model, it lacks an element of consistency between consecutive time windows. Adding a memory element such as GRUs only improves performance in certain scenarios. The results are not comparable with those obtained by EvSegNet, which achieves a high accuracy of 0.89 and 0.54 mIOU. It must be said that EvSegNet does make use of events, but generates 6-channel histogram representations from them, which are processed by highly optimized state-of-the-art CNNs.

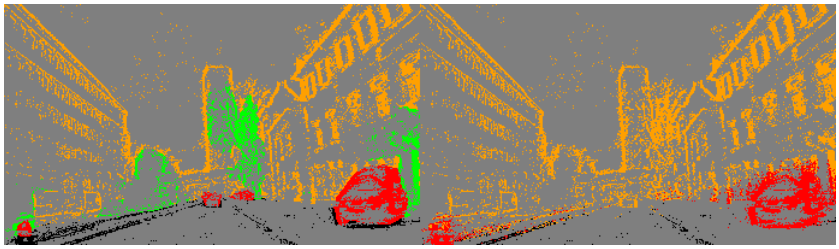


Figure 5.10: The left image shows events of the time window with ground-truth labels. The image on the right are events with predicted labels. PointNet++ is able to locate separate objects that are well defined and closer to the vision sensor. Objects that are further away and close to other objects are harder to detect, like the tree in the background.

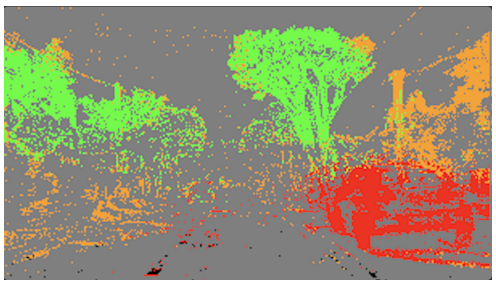


Figure 5.11: A good example of transfer learning. We applied PointNet++ pre-trained on the DDD17 to generate labels on time windows generated from the GEN1 Auatomotive Detection Dataset from Prophesee. The network is able to recognize different elements in the scene correctly.

5.8 TRANSFER LEARNING

Transfer learning is an interesting branch of machine learning that focuses on storing knowledge acquired while solving one problem and applying it to a related but separate problem. In our use case, it is interesting to see if networks trained on the DDD17 dataset can perform on other datasets in the automotive field. This is an interesting aspect to explore because it could provide a basic architecture to generate labels for each event on datasets that do not have them. Instead, our networks were trained with samples from the DDD17 dataset with a 346x260 DAVIS sensor, which were then pre-processed to eliminate the lower portion of events at the windshield and bonnet. Ground-truth labels were provided by EvSegNet. To test the transfer learning capabilities, we chose Prophesee's GEN1 Automotive Detection Dataset [69], which is one of the largest automotive event datasets. It contains 39 hours of recordings made with an ATIS event camera with 304x240 resolution. The different resolution is irrelevant as PointNet++ uses point clouds normalized between 0 and 1, as well as making use of T-Nets that make the network tolerant to initial input transformations. To generate the

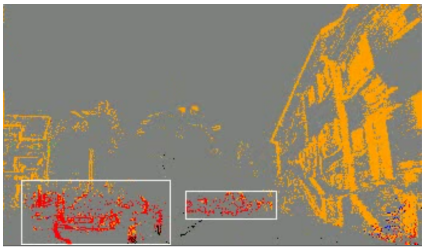
time windows we used the same settings as the pre-trained network at 50ms intervals. The generated samples were then tested on the best-performing model of PointNet++. Visual feedback shows that the network is able to extract meaningful information, but in limited use cases (see Figure 5.11).

Generally speaking, the predictions are not as consistent as on the DDD17 dataset. The difference between the two sensors is a factor to be considered: the DAVIS camera combines a DVS sensor and an APS sensor, whereas the ATIS camera has pixels that contain a DVS subpixel (called change detection CD) that triggers another subpixel to read out the absolute intensity (exposure measurement EM). The ATIS achieves large static dynamic range (> 120dB). However, the ATIS has the disadvantage that pixels are at least double the area of DVS pixels. Resolution can be a determining factor as a higher number of pixels allows the scene to be captured in greater detail. This is important for a self-driving event camera, where objects in the distance appear smaller and are difficult to distinguish.

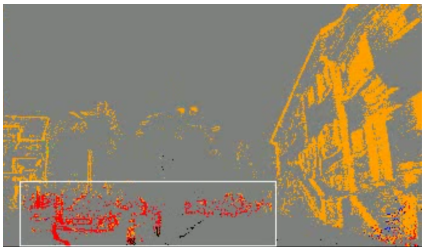
5.9 DETECTION PIPELINE PROPOSAL

Object detection is one of the most sought-after fields in the field of autonomous driving. It consists of recognizing objects within the scene and assigning them a bounding box, i.e. a region in which the neural network is confident of finding that object. Among the best-known object detection architectures is YoloV3 [38], which represents the state-of-the-art thanks to its ability to recognize a wide range of objects, even in real-time. It is easy to understand how to assign bounding-boxes, once the object has been recognized, in a two-dimensional space. The task becomes more complicated when considering 3D point clouds. A good proposal has been made in [97] which is capable of generating bounding boxes in 3D. The algorithm is based on data collected from a LiDAR that allows the depth of objects in the scene to be estimated. In this way, objects can be placed in three-dimensional space. With data collected by event sensors, no component gives information about the depth of the field of view. Generating bounding boxes with event data is, consequently, anything but simple. In our work, we propose a simple variant of creating bounding boxes on event-driven data streams. The algorithm is based on the clustering of segmented scenes. The points belonging to each class are processed by a clustering algorithm. Depending on its configuration, the algorithm proposes clusters to which bounding boxes can be assigned. It is clear that the goodness of the clustering is highly dependent on the effectiveness of the segmentation algorithm.

For our experiments, we use DBSCAN, a fast, non-parametric clustering algorithm. Given a set of points in some space, DBSCAN groups together points that are closely packed together (points with many nearby neighbors), marking them as outliers points that lie alone in low-density regions (whose



(a) Eps = 10, Min points = 10



(b) Eps = 50, Min points = 20

Figure 5.12: DBSCAN on a sample of the DDD17 dataset. The labels are predicted from our best performing model a) With this setup the algorithm is able to tell different cars a part better. b) This configuration is unable to detect the different vehicles in the scene, however it is more robust to small mispredicted regions.

nearest neighbors are too far away). DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means, and shows excellent performance with variable-density data. The algorithm can be set up using two parameters:

- *Min points*: represents the number points in a neighborhood for a point to be considered as a core point.
- *Eps*: represents the maximum distance between two samples for one to be considered as in the neighborhood of the other.

The choice of the parameters affects considerably the outcome of the clustering performance. Low eps and min points create very small clusters that are not very robust to noise. High values for both parameters produce few clusters containing many points. A comparison is shown in Figure 5.12. This naive approach is fairly simple and too naive for complex scenarios as autonomous driving. A limited use case could be scenes that are not subject to any sudden changes, for example, the detection of obstacles in autonomous agricultural vehicles.

5.10 FUTURE DEVELOPMENTS

The results obtained highlight the advantages and limitations of PointNet on event-driven data. Both architectures can process point clouds directly without the need for intermediate representations, thus maintaining the sparse and asynchronous nature of events. The PointNet architecture certainly leaves room for improvement: its application is designed more for static environments, although it shows great potential to be exploited also in more dynamic scenarios. A combination of more robust memory elements could increase the generalization ability. Second, all mathematical models are highly dependent on the quality of their input data. PointNet-based architectures

are no exception here. Neuromorphic sensors are still at an early stage of their development, but ongoing developments are bringing new interesting features to this area. Recent event cameras have higher resolutions in terms of the pixel matrix [98]. This certainly has advantages in the generation of point clouds as each object in the scene can be sampled with a larger number of points, generating more defined structures. Nevertheless, the problem of generating ground-truth labels that are consistent with the nature of the events remains. In this field, the work done by EvSegNet has laid stable foundations for a valid approach to this problem. Ultimately, an event camera is not the only protagonist inside a self-driving vehicle, but rather part of a heterogeneous ecosystem of sensors, each with its strengths and weaknesses. The fusion of data from different sensors has often shown great advantages in the field of autonomous driving. Integrating an event camera, which by nature works asynchronously, into this scenario is not so obvious, but it could open the way for a wide range of data to be explored.

BIBLIOGRAPHY

[1] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation." In: *CVPR 2017* (2017) (cit. on pp. xiii, 3, 42, 45, 55).

[2] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space." In: *2017* (2017) (cit. on pp. xiii, 3, 46, 50).

[3] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, S. Censi A.and Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza. "Event-based Vision: A Survey." In: *IEEE Trans. Pattern Anal. Machine Intell. (TPAMI)* (2020) (cit. on pp. xiii, 2, 11).

[4] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. "A Differentiable Recurrent Surface for Asynchronous Event-Based Data." In: *2020* (2020) (cit. on pp. 61, 62).

[5] P. Lichtsteiner and T. Delbruck. "64x64 Event-Driven Logarithmic Temporal Derivative Silicon Retina." In: *2005* (2005) (cit. on pp. 2, 11).

[6] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck. "Retinomorphic Event-Based Vision Sensors: Bioinspired Cameras With Spiking Output." In: *Proc. IEEE 102.10* (2014), pp. 1470–1484 (cit. on pp. 2, 11).

[7] C. Posch, D. Matolin, and R. Wohlgenannt. "A QVGA 143 dB Dy- namic Range Frame-Free PWM Image Sensor With Lossless Pixel- Level Video Compression and Time-Domain CDS." In: *IEEE J. Solid- State Circuits* (2011), pp. 259–275. ISSN: 0018-9200 (cit. on pp. 2, 11).

[8] Anup Vanarse, Adam Osseiran, and Alexander Rassau. "A Review of Current Neuromorphic Approaches for Vision, Auditory, and Olfactory Sensors." In: *Frontiers in Neuroscience, section Neuromorphic Engineering* (2016). ISSN: 0018-9200 (cit. on pp. 2, 11).

[9] Shih-Chii Liu (Editor), Tobi Delbruck (Co-Editor), Giacomo Indiveri (Co-Editor), Adrian Whatley (Co-Editor), and Rodney Douglas (Co-Editor). *Event-Based Neuromorphic Systems*. Wiley, 2014. ISBN: 978-1-118-92762-5 (cit. on pp. 2, 11).

[10] S. C. Liu, A. van Schaik, B. A. Minch, and T. Delbruck. "Asynchronous Binaural Spatial Audition Sensor With 2 × 64 × 4 Channel Output." In: *IEEE Trans. Biomed. Circuits Systs. 8.4* (2014), pp. 453–464. ISSN: 1932-4545 (cit. on p. 2).

[11] Y. Cao, Y. Chen, and D. Khosla. "Spiking deep convolutional neural networks for energy-efficient object recognition." In: *IJCV* (2015) (cit. on pp. 2, 31).

[12] N. Kasabov, K. Dhoble, N. Nuntalid, and G. Indiveri. "Dynamic evolving spiking neural networks for on-line spatio and spectro-temporal pattern recognition." In: *Networks* (2013) (cit. on pp. 2, 31).

[13] A. Russell, G. Orchard, Y. Dong, S¸. Mihalas, E. Niebur, J. Tapson, and R. Etienne-Cummings. "Optimization methods for spiking neurons and networks." In: *IEEE transactions on neural networks* (2010) (cit. on pp. 2, 31).

[14] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang. "Feedforward categorization on aer motion events using cortex-like features in a spiking neural network." In: *IEEE transactions on neural networks and learning systems* (2015) (cit. on pp. 2, 31).

[15] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman. "HFirst: A Temporal Approach to Object Recognition." In: *IEEE Trans. Pattern Anal. Mach. Intell. 37.10* (2015) (cit. on pp. 3, 31).

[16] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer. "Theory and tools for the conversion of analog to spiking convolutional neural networks." In: *arXiv [Accessed 03/03/2021]* (2016) (cit. on pp. 3, 31).

[17] F. Folowosele, R. J. Vogelstein, and R. Etienne-Cummings. " "Towards a cortical prosthesis: Implementing a spike-based HMAX model of visual object recognition in silico." In: *"IEEE J. Emerg. Select. Topics Circuits Syst., vol.1 no.4* (2011), pp. 516–525 (cit. on p. 31).

[18] Inigo Alonso and Ana C. Murillo. "EV-SegNet: Semantic Segmentation for Event-based Cameras." In: *CVPR 2019* (2019) (cit. on pp. 3, 36, 62, 63, 67).

[19] KPMG. *Connected and Autonomous Vehicles – The UK Ecnomic Opportunity*. 2015. URL: https://assets.kpmg/content/dam/kpmg/images/2015/05/connected-and-autonomous-vehicles.pdf (cit. on pp. 6, 7).

[20] SAE International. "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles." In: *Online* (2018) (cit. on p. 6).

[21] ABI Research. *ABI Research Forecasts 8 Million Vehicles to Ship with SAE Level 3, 4 and 5 Autonomous Technology in 2025*. 2018. URL: https://www.abiresearch.com/press/abi-research-forecasts-8-million-vehicles-ship-sae-level-3-4-and-5-autonomous-technology-2025/ (cit. on p. 6).

[22] Wan-Joo Park, Byung-Sung Kim, Dong-Eun Seo, Dong-Suk Kim, and Kwae-Hi Lee. "Parking Space Detection Using Ultrasonic Sensor in Parking Assistance System." In: *2008 IEEE Intelligent Vehicles Symposium* (2008) (cit. on p. 8).

[23] Kai-Tai Song, Chih-Hao Chen, and Cheng-Hsien Chiu Huang. "Design and Experimental Study of an Ultrasonic Sensor System for Lateral Collision Avoidance at Low Speeds." In: *2004 IEEE Intelligent Vehicles Symposium* (2004) (cit. on p. 8).

[24] Jamie Condliffe MIT Technology Review. *Tesla Announces New Sensors and Puts the Brakes on Autopilot*. 2016. URL: https://www.technologyreview.com/2016/10/20/156529/tesla-announces-new-sensors-and-puts-the-brakes-on-autopilot/ (cit. on p. 8).

[25] Petros A. Ioannou, ZEEE Member, and C. C. Chien. "Autonomous Intelligent Cruise Control." In: *IEEE transaction on vehicular technology, Vol. 42, No.4* (1993) (cit. on p. 9).

[26] Greg Marsden and Mike McDonald andMark Brackstone. "Towards an understanding of adaptive cruise control." In: *Transportation Research Part C: Emerging Technologies* (2001) (cit. on p. 9).

[27] Erik Coelingh, Andreas Eidehall, and Mattias Bengtsson. "Collision Warning with Full Auto Brake and Pedestrian Detection - a practical example of Automatic Emergency Braking." In: *Annual Conference on Intelligent Transportation Systems 2010* (2010) (cit. on p. 9).

[28] Emilia Moldovan, Serioja-Ovidiu Tatu, Tamara Gaman, Ke Wu, IEEE Fellow, and IEEE Renato G. Bosisio Fellow. "A New 94-GHz Six-Port Collision-Avoidance Radar Sensor." In: *IEEE Transaction on microwave theory and techniques, Vol. 52, No.3* (2004) (cit. on p. 9).

[29] ZHUANG Jia-yuan, ZHANG Lei, ZHAO Shi-qi, CAO Jian, WANG Bo, and SUN Han-bing. "A New 94-GHz Six-Port Collision-Avoidance Radar Sensor." In: *China Ocean Eng., Vol. 30, No. 6* (2016), pp. 867–883 (cit. on p. 9).

[30] Kun Zhou, Zhou Kun, Wang Xiqin, Xiqin Wang, M. Tomizuka, Wei Bin Zhang, Wei-Bin Zhang, and Ching-Yao Chan. "A new maneuvering target tracking algorithm with input estimation." In: *Proceedings of the 2002 American Control Conference* (2002) (cit. on p. 10).

[31] Philip E. Ross for IEEE Spectrum. *Velodyne Announces a Solid-State Lidar*. 2018. URL: https://spectrum.ieee.org/cars-that-think/transportation/sensors/velodyne-announces-a-solidstate-lidar (cit. on p. 10).

[32] L. E. Navarro-Serment, C. Mertz, and M. Hebert. "Pedestrian detection and tracking using three-dimensional LADAR data." In: *Int. J. Robot. Res., Vol. 29, No. 12* (2010), pp. 1516–1528 (cit. on p. 10).

[33] J. Schlosser, C. K. Chow, and Z. Kira. "Fusing LIDAR and images for pedestrian detection using convolutional neural networks." In: *Proc. 2016 IEEE Int. Conf. Robot. Autom.* (2016), pp. 2198–2205 (cit. on p. 10).

[34] M. Bersani, M. Vignati, S. Mentasti, S. Arrigoni, and F. Cheli. "Vehicle state estimation based on Kalman filters." In: *Poltitecnico di Milano* (2019) (cit. on p. 11).

[35] Jihan Ryu and J. Christian Gerdes. "Integrating Inertial Sensors With Global Positioning System (GPS) for Vehicle Dynamics Control." In: *Journal of Dyn. Sys., Meas., Control.* (2004), pp. 243–254 (cit. on p. 11).

[36] Christian Hane, Torsten Sattler, and Marc Pollefeys. "Obstacle Detection for Self-Driving Cars Using Only Monocular Cameras and Wheel Odometry." In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015) (cit. on p. 11).

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. "You only look once: Unified, real-time object detection." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016) (cit. on p. 35).

[38] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement." In: *arXiv* (2018) (cit. on pp. 11, 36, 69).

[39] J. Ren, X. Chen, J. Liu, W. Sun, J. Pang, Q. Yan, Y.-W. Tai, and L. Xu. "Accurate single stage detector using recurrent rolling convolution." In: *CVPR* (2017) (cit. on p. 35).

[40] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single Shot MultiBox Detector." In: *ArXiv* (2016) (cit. on p. 11).

[41] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal Loss for Dense Object Detection." In: *ArXiv* (2018) (cit. on p. 11).

[42] Jiwoong Choi, Dayoung Chun, Hyun Kim, and Hyuk-Jae Lee. "Gaussian YOLOv3: An Accurate and Fast Object Detector Using Localization Uncertainty for Autonomous Driving." In: *ArXiv* (2019) (cit. on p. 11).

[43] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." In: *ArXiv* (2015) (cit. on p. 11).

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *IEEE Int. Conf. Comput. Vis. Pattern Recog. (CVPR)* (2016) (cit. on p. 37).

[45] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN." In: *ArXiv* (2017) (cit. on p. 11).

[46] W.McCulloch and W.Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *Bulletin ofMath. Bio* (1943) (cit. on p. 13).

[47] D.Hebb. *The organization of behaviour*. Wiley, 1949 (cit. on p. 13).

[48] C. Mead. *Analog VLSI and neural systems*. Addison-Wesley, 1989 (cit. on pp. 13, 14).

[49] M.A.C. Maher, S.P. Deweerth, M.A. Mahowald, and C.A. Mead. "Implementing neural architectures using analog VLSI circuits." In: *IEEE Trans. Circuits Syst* (1989) (cit. on pp. 13, 14).

[50] C. Mead. "Neuromorphic electronic systems." In: *Proceedings of the IEEE* (1990) (cit. on pp. 13, 14).

[51] K.A. Boahen. "Neuromorphic microchips." In: *Sci.Am* (2005).

[52] R. Sarpeshkar. "Brain power — borrowing from biology makes for low power computing (bionic ear)." In: *IEEE Spectr. 43* (2006).

[53] A.L. Hodgkin and A.F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve." In: *J. Physiol.117* (1952) (cit. on p. 14).

[54] M.A. Mahowald and C.A. Mead. "The Silicon Retina." In: *Sci. Am* (1991) (cit. on pp. 15, 16).

[55] C. Posh. "Bio-insired vision." In: *JINST the 7 C01054* (2012) (cit. on p. 15).

[56] K.A. Boahen. "Neuromorphic microchips." In: *Sci. Am. 292* (2005) (cit. on p. 16).

[57] K.A. Zaghloul and K.A. Boahen. "A silicon retina that reproduces signals in the optic nerve." In: *J. Neural Eng. 3* (2006) (cit. on p. 16).

[58] P. Lichtsteiner and T. Delbruck. "A 64×64 AER logarithmic temporal derivative silicon retina." In: *Research in Microelectronics and Electronics, PhD* (2005) (cit. on p. 17).

[59] P. Lichtsteiner, T. Delbruck, and C.Posh. "A 128×128 120 dB 15 µs Latency Asynchronous Temporal Contrast Vision Sensor." In: *IEEE J. Solid-State Circuits 43* (2008) (cit. on pp. 17–19).

[60] P. Lichtsteiner, T. Delbruck, and C.Posh. "A 128×128 120 dB 30 mW asynchronous vision sensor that responds to relative intensity change." In: *Dig. of Tech. Papers* (2006) (cit. on p. 17).

[61] C. Posch, D. Matolin, and R. Wohlgenannt. "An asynchronous time-based image sensor." In: *ISCAS IEEE International Symposium on Circuits and Systems* (2008) (cit. on pp. 17, 21).

[62] C. Posch, D. Matolin, and R. Wohlgenannt. "A QVGA 143dB Dynamic Range Asynchronous Address-Event PWM Dynamic Image Sensor with Lossless Pixel-Level Video Compression." In: *ISSCC, IEEE Conference on Solid-State Circuits* (2010) (cit. on p. 17).

[63] C. Posch, D. Matolin, and R. Wohlgenannt. "A QVGA 143dB Dynamic Range Frame-free PWM Image Sensor with Lossless Pixel-level Video Compression and Time-Domain CDS." In: *IEEE J. of Solid-State Circuits* (2011) (cit. on pp. 17, 21).

[64] M. Sivilotti. "Wiring considerations in analog VLSI systems with application to field-programmable networks." In: *Ph.D. dissertation, Comput. Neural Syst. CA, USA: California Inst. Technol. (Caltech), Pasadena* (1991) (cit. on p. 18).

[65] M.A. Mahowald. "VLSI analogs of neuronal visual processing: A synthesis of form and function." In: *Ph.D. dissertation, Comput. Neural Syst. CA, USA: California Inst. Technol. (Caltech), Pasadena* (1992) (cit. on p. 18).

[66] M.A. Mahowald. "Christian Brandli and Raphael Berner and Minhao Yang and Shih-Chii Liu and Tobi Delbruck." In: *A 240 × 180 130 dB 3 µs Latency Global Shutter Spatiotemporal Vision Sensor* (2014) (cit. on p. 20).

[67] Jonathan Binas, Daniel Neil, Shih-Chii Liu, and Tobi Delbruck. "DDD17: End-To-End DAVIS Driving Dataset." In: *Arxiv, Presented at the ICML 2017 Workshop on Machine Learning for Autonomous Vehicles* (2017) (cit. on pp. 20, 24, 62).

[68] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman. "HATS: Histograms of averaged time surfaces for robust event-based object classification." In: *Proc. 2018 IEEE/Conf. Computer Vision and Pattern Recognition (CVPR)* (2018), pp. 1731–1740 (cit. on pp. 3, 24, 30, 57, 61, 62).

[69] Pierre de Tournemire, Davide Nitti, Etienne Perot, Davide Migliore, and Amos Sironi. "A Large Scale Event-based Detection Dataset for Automotive." In: *Proc. 2018 IEEE/Conf. Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 1731–1740 (cit. on pp. 24, 68).

[70] Pierre de Tournemire, Davide Nitti, Etienne Perot, Davide Migliore, and Amos Sironi. "The multivehicle stereo event camera dataset: An event camera dataset for 3d perception." In: *IEEE Robot. Autom. Lett., vol. 3, no. 3* (2018), pp. 2032–2039 (cit. on p. 25).

[71] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman. "HOTS: A hierarchy of event-based time-surfaces for pattern recognition." In: *IEEE Trans. Pattern Anal. Mach. Intell.* (2017), pp. 1346–1359 (cit. on p. 30).

[72] J. Achary, V. Padala, and A. Basu. "Spiking neural network based region proposal networks for neuromorphic vision sensors." In: *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)* (2019), pp. 1–5 (cit. on p. 31).

[73] J. H. Lee, T. Delbruck, and M. Pfeiffer. "Training deep spiking neural networks using backpropagation." In: *Front. Neurosci., vol.10* (2016), p. 508 (cit. on p. 31).

[74] Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. "Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization." In: *British Machine Vis. Conf. (BMVC)* (2017) (cit. on p. 25).

[75] Ana I. Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso Garcia, and Davide Scaramuzza. "Event-based vision meets deep learning on steering prediction for self-driving cars." In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)* (2018) (cit. on p. 27).

[76] Ryad Benosman, Charles Clercq, Xavier Lagorce, Sio-Hoi Ieng, and Chiara Bartolozz. "Event-based visual flow." In: *IEEE Trans. Neural Netw. Learn. Syst* (2014) (cit. on p. 27).

[77] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. "EV-FlowNet: Self-supervised optical flow estimation for event-based cameras." In: *Robotics: Science and Systems (RSS)* (2018) (cit. on pp. 27, 34).

[78] Alex Zihao Zhu, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis. "Unsupervised event-based learning of optical flow, depth, and egomotion." In: *In IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)* (2019) (cit. on pp. 27, 34).

[79] C. Ye, A. Mitrokhin, C. Parameshwara, C. Fermüller, J. A. Yorke, and Y. Aloimonos. " Unsupervised learning of dense optical flow and depth from sparse event data." In: *Arxiv* (2018).

[80] D. Gehrig, A. Loquercio, K. G. Derpanis, and D. Scaramuzza. "End-to-end learning of representations for asynchronous event-based data." In: *IEEE Int. Conf. Computer Vision (ICCV)* (2019) (cit. on pp. 28, 29).

[81] N. F. Y. Chen. "Pseudo-labels for supervised learning on dynamic vision sensor data, applied to object detection under ego-motion." In: *Proc. 2018 IEEE/CVF Conf. Computer Vision and Pattern Recognition Workshops (CVPRW)* (2018) (cit. on p. 35).

[82] J. Li, S. Dong, Z. Yu, Y. Tian, and T. Huang. "Event-based vision enhanced: A joint detection framework in autonomous driving." In: *Proc. 2019 IEEE Int. Conf. Multimedia and Expo (ICME)* (2019) (cit. on p. 36).

[83] G. Chen, H. Cao, C. Ye, Z. Zhang, X. Liu, X. Mo, Z. Qu, and J. Conradt et al. "Multi-cue event information fusion for pedestrian detection with neuromorphic vision sensors." In: *Front. Neurorobot., vol. 13* (2019) (cit. on p. 36).

[84] A. Zanardi, A. Aumiller, J. Zilly, A. Censi, and E. Frazzoli. "Cross-modal learning filters for RGB-neuromorphic wormhole learning." In: *Proc. 15th Robotics: Science and System XV* (2019) (cit. on p. 36).

[85] F. Chollet. "Xception: Deep learning with depthwise separable convolutions." In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017) (cit. on p. 37).

[86] A. I. Maqueda, A. Loquercio, G. Gallego, N. García, and D. Scaramuzza. "Event-based vision meets deep learning on steering prediction for self-driving cars." In: *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (2018) (cit. on p. 37).

[87] Yusuke Sekikawa, Kosuke Hara, and Hideo Saito. "EventNet: Asynchronous Recursive Event Processing." In: *ArXiv, Accessed Online* (2019) (cit. on p. 56).

[88] Qinyi Wang, Yexin Zhang, Junsong Yuan, and Yilong Lu. "Space-Time Event Clouds for Gesture Recognition: From RGB Cameras to Event Cameras." In: *IEEE Winter Conference on Applications of Computer Vision (WACV)* (2019) (cit. on p. 55).

[89] O. Vinyals, S. Bengio, and M. Kudlur. "Order matters: Sequence to sequence for sets." In: *ArXiv, Accessed Online)* (2015) (cit. on p. 43).

[90] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. "3d shapenets: A deep representation for volumetric shapes." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1912–1920 (cit. on p. 52).

[91] D. Maturana and S. Scherer. "Voxnet: A 3d convolutional neural network for real-time object recognition." In: *EEE/RSJ International Conference on Intelligent Robots and Systems* (2015) (cit. on p. 52).

[92] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. Guibas. "Volumetric and multi-view cnns for object classification on 3d data." In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE* (2016) (cit. on p. 52).

[93] H. Su, S. Maji, E. Kalogerakis, and E. G. Learned-Miller. "Multi-view convolutional neural networks for 3D shape recognition." In: *Proc. ICCV* (2015) (cit. on p. 52).

[94] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. "Scannet: Richly-annotated 3d reconstructions of indoor scenes." In: *arXiv preprint arXiv:1702.04405* (2017) (cit. on pp. 52, 53).

[95] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. "Spectral networks and locally connected networks on graphs." In: *arXiv preprint arXiv:1312.6203* (2013) (cit. on p. 53).

[96] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. " The cityscapes dataset for semantic urban scene understanding." In: *Proc. of IEEE conf. on CVPR* (2016) (cit. on p. 62).

[97] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. "Frustum PointNets for 3D Object Detection from RGB-D Data." In: *Proc. of IEEE conf. on CVPR* (2016) (cit. on p. 69).

[98] Etienne Perot, Pierre de Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi. "Learning to Detect Objects with a 1 Megapixel Event Camera." In: *34th Conference on Neural Information Processing Systems (NeurIPS 2020).* (2020) (cit. on p. 71).