1 2 9 0

UNIVERSIDADE Ð
COIMBRA

Simão Almeida Rocha

# Neuromorphic Vision Based Multi-vehicle Detection and Tracking for Intelligent Transportation System

**VOLUME 1**

**Dissertação no âmbito do Mestrado em Engenharia Eletrotécnica e de Computadores, no ramo de Robótica, Controlo e Inteligência Artificial, orientada pelo Professor Doutor Jorge Manuel Moreira de Campos Pereira Batista e apresentada ao Departamento de Engenharia Eletrotécnica e de Computadores.**

Setembro de 2022

· U C ·

FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

# NEUROMORPHIC VISION BASED MULTI-VEHICLE DETECTION AND TRACKING FOR INTELLIGENT TRANSPORTATION SYSTEM

**Simão Almeida Rocha**

Coimbra, September 2022

# NEUROMORPHIC VISION BASED MULTI-VEHICLE DETECTION AND TRACKING FOR INTELLIGENT TRANSPORTATION SYSTEM

**Supervisor:**

Professor Doutor Jorge Manuel Moreira de Campos Pereira Batista

**Jury:**

Professor Doutor Urbano José Carreira Nunes

Professor Doutor Jorge Manuel Moreira de Campos Pereira Batista

Professor Doutor Cristiano Premebida

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, September 2022

# Agradecimentos

Sendo este documento a conclusão do ciclo de estudos, gostaria de começar por agradecer a todos os professores que fizeram parte desta longa jornada. Em especial gostaria de agradecer ao professor Jorge Batista pelo contínuo acompanhamento na partilha de conhecimentos e no incentivo na procura sempre de melhores resultados. Quero também agradecer aos colegas do laboratório, Bruno Silva, Pedro Martins, André Graça, Eurico Almeida, Timur Lomin, Patrícia Boavida e Henrique Cardoso pelo apoio e motivação durante este trabalho e pela partilha de conhecimentos e histórias.

Queria agradecer a todos os meus amigos que fui conhecendo e que fizeram parte desta história que vou guardar para o resto da minha vida, em especial ao José Mota, Ricardo Rodrigues, Carlos Torres, João Henriques, Óscar Martins, Diogo Santos, Pedro Póvoa, Henrique Cardoso, João Silva, Patrícia Boavida, Gabriela Simões, Timur Lomin, Pedro Fernandes, Nuno Marques, André Teixeira, João Sousa, João Jorge.

Queria também agradecer a todas as pessoas que fizeram parte do meu percurso como desportista desde tenra idade, foram estas as pessoas que me ajudaram a moldar como atleta, mas principalmente como pessoa ajudando a ganhar a responsabilidade e o desejo para abraçar novos objetivos.

Agora agradeço à pessoa que mais me apoiou ao longo deste percurso, a minha namorada, Natacha Roseiro. Foi a pessoa que esteve sempre ao meu lado, que me ajudou a crescer pessoalmente e a enfrentar qualquer desafio e obstáculos que foram aparecendo ao longo destes cinco anos.

Por fim, quero agradecer à minha família. Aos meus pais, Maria Eunice e Manuel Rocha, aos meus irmãos, Ariana Rocha e Tomás Rocha, aos meus avós e tios. Foram eles que me deram e proporcionaram a vida que tenho hoje, sem eles nada disto seria possível.

***A todos, um muito obrigado!***

# List of Acronyms

| | |
|---|---|
| **AP** | Average Precision |
| **ANN** | Artificial Neural Networks |
| **AR** | Average Recall |
| **CMC** | Cumulative match curve |
| **CNN** | Convolutional Neural Network |
| **COCO** | Common objects in Context |
| **DBT** | Detection-Based Tracking |
| **DFT** | Detection-Free Tracking |
| **DNN** | Deep Neural Network |
| **FLOPs** | Floating Point Operations |
| **FP** | False Positive |
| **FN** | False Negative |
| **fps** | frame per seconds |
| **GM-PHD** | Gaussian Mixture-Probability Hypothesis Density |
| **GM-CPHD** | Gaussian Mixture-Cardinality Probability Hypothesis Density |
| **HOG** | Histogram of Oriented Gradients |
| **IOU** | Intersection Over Union |
| **ITS** | Intelligent Transportation System |
| **KF** | Kalman Filter |

| | |
|---|---|
| **LSTM** | Long Short-term Memory |
| **mAP** | mean Average Precision |
| **MOT** | Multi-object Tracking |
| **PDAF** | Probabilistic Data Association Filter |
| **RED** | Recurrent Event-based Detector |
| **RNN** | Recurrent Neural Network |
| **SE** | Squeeze-and-Excitation |
| **SOT** | Single Object Tracking |
| **SORT** | Simple Online Real-time Tracker |
| **SSCN** | Submanifold Sparse Convolutional Network |
| **SSD** | Single Shot Multibox Detector |
| **STDP** | Spike-Timing-Dependent Plasticity |
| **TP** | True Positive |
| **YOLO** | You Only Look Once |

# Resumo

O aparecimento da aprendizagem profunda e o crescimento do poder computacional culminou no avanço da resolução de vários problemas em diferentes áreas, como por exemplo o MOT. Esta área sempre foi alvo de grande interesse em visão por computador e com diversas aplicações em diferentes campos, como por exemplo videovigilância, entretenimento e condução autónoma. Durante muito anos os investigadores investiram tempo e esforço para melhorar a tarefa de seguimento de objetos usando câmaras RGB. Contudo, estas câmaras estão sempre limitadas devido à taxa de captura de informação e pelas condições de iluminação. Posto isto, nos últimos anos as câmaras neuromórficas têm sido exploradas pela comunidade científica para ultrapassar as limitações existentes através da capacidade de apenas transmitir mudanças de intensidade, ao nível do pixel, geradas pelo movimento na cena, resultando num conjunto de eventos com baixa latência, baixo consumo e uma dinâmica muito alta. Consequentemente, estes atributos podem gerar um grande contributo no seguimento de veículos. Para alcançar o objetivo deste trabalho, o problema é tratado pela fórmula de seguimento-por-deteção, onde o problema de deteção é avaliado por uma rede neuronal recorrente em duas tipologias: CNN e SSCN. Paralelamente, com o mesmo objetivo, irá ser avaliada uma rede com tipologia SNN com a arquitetura Tiny-YOLO. A fase de seguimento é alcançada usando um algoritmo online de seguimento de múltiplos alvos, SORT, e pela sua evolução, DeepSort, através da adição de um modelo profundo para extrair descritores de aparência. Relativamente aos resultados, a nossa implementação teve o terceiro melhor resultado comparado com o estado da arte no dataset GEN1, que apenas é superado pelos autores do próprio dataset. O melhor resultado obtido é de 40% de mAP com uma rede de 24M de parâmetros enquanto a nossa rede obteve 31% com apenas 2.8M de parâmetros. Além disso, apresentamos resultados como base para trabalhos futuros sobre os datasets UA-DETRAC e BMVC para a deteção de objetos e seguimento de múltiplos veículos.

# Abstract

The arrival of deep learning and the increasing computational power have culminated in the advancement of many problems in different fields, such as MOT. This area has always been a field of great interest in computer vision, with many applications in different fields, like surveillance, entertainment, and autonomous vehicles. For many years, researchers have dedicated time and effort to improving the task of tracking objects in many scenes using RGB cameras. However, RGB cameras are always limited due to the rate of capturing information and light conditions. Therefore, in the past years, neuromorphic cameras have been explored by the scientific community to surpass the limitations through the ability to only transmit local pixel-level intensity changes generated by movement in a scene at the moment of occurrence, resulting in an information-rich stream of events with low latency, low power consumption, and a very high dynamic range. Consequently, these attributes can generate great contributions to the tracking of vehicles for ITS. To achieve the purpose of this work, the problem is treated with a detection-based tracking formulation, where the detection problem is evaluated by a recurrent detection neural network in two typologies, CNN and SSCN. In addition, while working on the same objective, it will be validated an SNN typology with a Tiny-YOLO architecture. The tracking phase is accomplished using an online multi-target tracking algorithm, SORT, and its evolution, DeepSort, through the addition of a deep model for extracting appearance descriptors. The lack of labelled data is a real problem for validating the algorithms for detection and tracking with neuromorphic cameras. To solve this problem, a python software tool v2e is used to synthesize realistic dynamic vision sensor event camera data from UA-DETRAC and BMVC datasets. Regarding the results of this study, our network has advanced to the third best result in the existent results of state-of-art in the GEN1 dataset. The best result obtains 40% in mAP with a network of 24M parameters, while our network has only 2.8M parameters and obtains 31% in mAP. Furthermore, we present a baseline for future work on the UA-DETRAC and BMVC datasets for object detection and multi-vehicle tracking.

*"Everyone has inside of him a piece of good news. The good news is that you don't know how great you can be! How much you can love! What you can accomplish! And what your potential is!"*

— Anne Frank

# Contents

# List of Figures

# List of Tables

# 1 Introduction

The document aims to present the work developed on the topic “Neuromorphic Vision-based Multi-vehicle Detection and Tracking for Intellingent Transportation System”, for the conclusion of the Master’s Degree in Electrical and Computer Engineering at the Faculty of Sciences and Technologies of Coimbra.

## 1.1 Motivation

Multi-object tracking has been a topic of significant interest in computer vision because it is a crucial component of many vision systems. It has numerous applications in traffic control, human-computer interaction, digital forensics, gesture recognition, augmented reality, and visual surveillance. In recent years, researchers have started to explore a different type of camera technology to surpass the limitations of conventional cameras. One of the most promising types are called neuromorphic cameras and they have given the input to a new field in computer vision called event-based vision.

These cameras are inspired by biological vision, which acquire transients in visual scenes using an event-driven frameless approach. Unlike traditional cameras, neuromorphic cameras only transmit local pixel-level intensity changes generated by movement in a scene at the moment of occurrence, resulting in an information-rich stream of events with low latency, low power consumption and a very high dynamic range [1]. This technology does not suffer of motion blur and is able to capture information in low light conditions which is a severe problem of conventional cameras when dealing with high dynamically scenarios and real world conditions.

These traits, as well as the fact that these cameras have only recently begun to be used in our department and the importance of MOT for ITS, provide great motivation to embrace this dissertation topic. Furthermore, since this field still very recent there are not too many works that focus on particular strategies to solve the MOT problem. This fact, along with the paucity of datasets, places a big challenge on this work.

## 1.2 Developed Work

The purpose of this work is to explore the potential of neuromorphic vision sensors through different typologies of Deep Learning models to perform a Multi-Detection framework and Multi-vehicle tracking.

Therefore, using the tracking-by-detection formulation, the work was divided into two tasks: detection and tracking. In the detection phase, three approaches were investigated in order to determine which would be the best approach to apply in the tracking phase. Given the unique properties of these cameras, we cannot always guarantee that each vehicle in the image produces events; for example, vehicles that have stopped moving no longer produce events. To overcome this challenge, a CNN typology with LSTM blocks was implemented to retain past information and help in the detection of vehicles even when they stop generating events. Following the same reasoning, an SSCN typology was implemented, employing the same memory mechanisms, with the distinction being the usage of sparse convolutions to benefit from the sparse properties of the representations built with a set of events generated by the neuromorphic cameras. These representations are given as input to the two mentioned typologies through two algorithms[1]: Time Surface and Event Volume. These algorithms receive a set of events over a time interval and produce frame representations. Finally, a Spiking neural network typology was explored in order to explore the sparse and discrete characteristics of the data coming directly from the neuromorphic cameras.

For the tracking phase, we validate a classical tracking approach, SORT [2], using the bounding boxes of the detection model as input. This approach essentially employs tracking hypothesis with a Kalman Filter, frame-by-frame data association using bounding box overlap as an association metric, and the Hungarian method. A second approach, DeepSort [3], an extension of the aforementioned algorithm, was also implemented. The difference between this approach and the previous one is the use of an additional re-identification model to extract an appearance descriptor for each bounding box and the use of two new metrics for data association, Mahalanobis distance and cosine distance. The re-identification model implementation is based on the work [4] and uses two representations: an event representation and its transformation for polar coordinates. These representations are fed as input to a

[1]Based on information from https://docs.prophesee.ai

ResNet-50 to extract a descriptor appearance by concatenating the feature vectors generated from each input.

The methodologies were evaluated on three separate datasets, each with a distinct set of characteristics, and the results were encouraging, in line with the current state-of-the-art results for the GEN1 Automotive dataset [5] in particular. The other datasets, UA-DETRAC [6] and BMVC, were converted to event data using the v2e [7] python software tool, and the results with the implemented approaches indicate interesting performance. As a result, they can be used as a baseline for future work using neuromorphic cameras for vehicle detection and tracking.

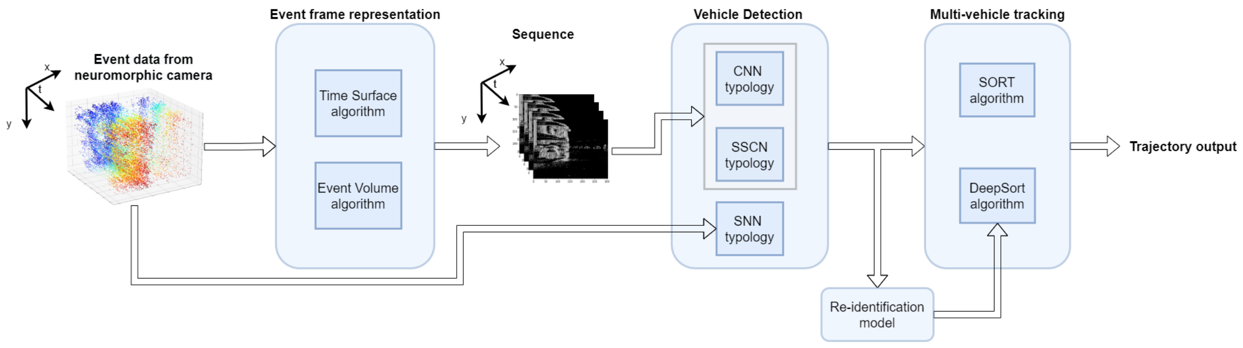


Figure 1.1: A diagram of the implementations carried out for the conclusion of the objective of this dissertation.

## 1.3 Document Structure

This document is split into six chapters. The first chapter introduces the motivation for doing the study, as well as their aims and contributions. The following chapter covers and delves into various significant studies undertaken by the scientific community that contributed to and inspired this work. Second, in chapter 3, some background information is provided to assist in following up on the job implemented. Following that, chapter 4 outlines the work that was implemented in order to achieve the goal of this dissertation. Later in chapter 5, the results of the completed work are carefully reported and discussed. The last chapter highlights all of the essential issues provided throughout the document and offers light on probable future directions.

# 2 State of The Art

In this chapter it will be presented the previous work devoted in Multi-Object Tracking (MOT) and Single Object Tracking (SOT). The first section will revolve around the basic approach for MOT, the second and third section will concentrate on dedicated neural networks to take full potential of Event-Cameras properties and the last section will portray the event-based vision related work applied to MOT.

## 2.1 Procedure of Multi-object tracking

Before presenting the related work for MOT, makes sense to explain the procedure involved in its algorithms. Therefore, most of existing MOT works can be grouped into two sets: Detection-Based Tracking (DBT) and Detection-Free Tracking (DFT).

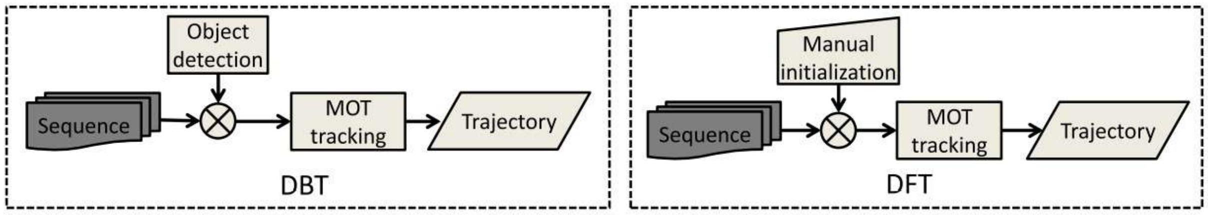


Figure 2.1: A procedure flow of two prominent tracking approaches. **Left**: Detection-Based Tracking (DBT), **Right**: Detection-Free Tracking (DFT).(Taken from [8])

For DBT, the first step is to detect object and then assign into a tracking algorithm. Therefore, given a sequence, a specific object detection is applied in each frame to obtain object detection hypotheses, then the tracking is conducted to link detection hypotheses into an identified object. Intuitively, this strategy brings two problems apart from the issues belonging to MOT. Firstly, since the detector is trained in advance, they specialize in detection of specific kinds of targets. Secondly, the tracking performance depends highly on the detector.

Meanwhile, DFT requires a manual initialization of a fixed number of objects in the first frame then localizes these objects in subsequent frames. The main drawback with this strategy over DBT is that it cannot recover from object disappearance during tracking.

Furthermore, the processing mode of MOT can be categorized into online tracking and offline tracking. Online tracking methods relies solely on data available up to the present frame. The image sequence is handled in a step-by-step manner, and it is also referred to as sequential tracking. The object's location and ID are used to represent the results.

On the other hand, offline tracking methods use observations from both the past and the future. Observations from all the frames are required to be obtained in advance and are analyzed jointly to estimate the final output. This can be heavier computationally, which means it is not always possible to handle all the frames at once.

## 2.2 Spiking Neural Networks

The ANN models attempt to represent the human brain in order to execute tasks such as classification, recognition, and others. To accomplish this, a large number of labeled training examples are required, which corresponds to supervised training using backpropagation, a gradient-based optimization method. ANNs are predominantly built using idealized computing neurons with continuous activation values and a set of weighted inputs.

Contrarily, SNNs are more biologically realistic than ANNs and more closely mimic natural neural networks because of the natural discrete spikes executed by biological neurons. In addition, SNNs are also more hardware-friendly, energy-efficient, and faster than ANNs. These properties, along with the capacity to be intrinsically sensitive to temporal characteristics of information, SNNs have become the focus of a number of recent applications in many areas of pattern recognition, such as visual processing, speech recognition, etc. However, it is important to note that inference time requires specific hardware, which increases the difficulty of implementing this type of modality.

Initially, SNNs were trained in a unsupervised manner due to the fact that activations functions of spiking neurons are asynchronous and non-differentiable in time which prevents using backpropagation, resulting in lower performance compared to other methods. This approach can be made by using spike-timing-dependent plasticity (STDP) [9], Growing Spiking Neural Networks [10], Artola-Bröcher-Singer rule [11], Bienenstock-Cooper-Munro (BCM) rule [12] or Relationship between BCM and STDP rules [13].

Alternatively, more recent studies have proven that despite the lack of a continuous and differentiable activation function, this network topology can be trained using gradient descent algorithms for backpropagation error, i.e., in a supervised manner. Supervised learning for SNNs can be applied with the following methods: SpikeProp [14]; Remote Supervised Method [15]; FreqProp [16];

Furthermore, DNN-to-SNN conversion methods have been widely studied in recent years as another alternative approach [17, 18, 19]. The idea of these methods is based on importing pre-trained parameters from DNN to SNN.

Due to the difficulty of training SNNs, their applications have been limited to relatively simple tasks such as image classification. However, the authors of [20] have studied the performance over a more challenging regression problem, i.e., object detection. Based on an in-depth study by the authors, the issues arise from inefficient conventional normalization methods and the absence of an efficient implementation of leaky-ReLU in an SNN domain. To this end, they have introduced two novel methods: channel-wise normalization and signed neuron with imbalanced threshold, where they claim that both methods provide fast and accurate information transmission over the network. Consequently, the authors have developed a spiked-based object detection model, called Spiking-YOLO, where the results are comparable up to 98% to the standard CNN tiny-YOLO on the COCO and PASCAL datasets. Furthermore, compared to previous SNN conversion methods, the authors claimed that it converges 2 to 4 times faster and consumes approximately 280 times less energy on a neuromorphic chip compared to the DNN model.

Additionally, the authors of [21] have developed a SNN network to train directly on data coming from event cameras in order to investigate the problem of object detection with SNNs. Event data is binary and sparse in space and time, which makes it the ideal input for SNNs. To address the difficulty of having a good performance in real-word applications, the authors follow the latest advancements in the matter of spike backpropagation-surrogate gradient learning, parametric LIF, and SpikingJelly framework [22, 23]. Furthermore, they introduced a novel approach to encoding event data called voxel cube that preserves their binary and temporal information while keeping a low number of timesteps. In addiction, they test four different backbones (SqueezeNet [24], VGG [25], MobileNet [26], DenseNet [27]) with a SSD bounding box regression head. They claim to present the first SNN capable of doing object detection on real-world event data.

## 2.3 Sparse Convolutional Networks

Convolutional networks constitute the state-of-the art method for a wide range of tasks that involve the analysis of data with spatial and/or temporal structure, such as photos, videos, or 3D surface models. However, applying these architectures to point clouds obtained using a LiDAR scanner, RGB-D camera, or neuromorphic cameras is inefficient because of the sparse nature of data and the curse of dimensionality. In such scenarios, it becomes increasingly important to exploit data sparsity whenever possible in order to reduce the computational resources needed for data processing.

Recently, a number of convolutional network implementations that have been presented are tailored to work efficiently on sparse data [28, 29, 30]. The implementation of [28, 29] are mathematically identical to regular convolutions, but they require fewer computational resources in terms of FLOPs and/or memory. [28] uses a space version of *im2col* operation to restrict computation and storage to the active sites, while [29] uses the voting algorithm from [31] to suppress unnecessary multiplications by zero. The last work [30] modifies the convolution operator to produce averaged hidden states in parts of the grid that are outside the region of interest.

One disadvantage of previous sparse convolutional network solutions is that they expand the sparse data in each layer by using full convolutions. Hereupon, in [32] the authors have reached a solution that keeps the same level of sparsity throughout the network. They have developed new implementations for performing sparse convolutions, similar to regular convolution but saving time and computation, and a novel convolution operator termed submanifold sparse convolution (SSC). In addiction, the authors have proved the efficiency of the new type of network by developing an SSCN network for segmentation in 3D point clouds and outperforming all prior state-of-the-art on the test set for the semantic segmentation of the ShapeNet competition.

Subsequently, the authors from [33] have developed an implementation for object detection and object recognition using the same sparse convolutions as the prior work using event data. In addiction, the authors claim that their framework exploits spatio-temporal sparsity of events, reduces the computational complexity up to 20 times with respect to high-latency neural networks, and outperforms state-of-art asynchronous approaches up to 24% in prediction accuracy.

## 2.4 Event-based Vision applied to Multi-Object Tracking

The majority of works address this problem using a DBT method, as explained in section 2.1. Hence, this part will show the relevant work for object detection and tracking with neuromorphic cameras.

### 2.4.1 Object detection

In this subsection, we will discuss the related work of object detection developed with Event-Cameras. Starting with the simple detectors based on statistical methods and then moving on to the more advanced detectors based on deep learning methodologies.

Table 2.1: Detection methods applied to multi-object tracking problems with Event-Cameras.

| **Detector** | **Method** |
|---|---|
| Region Proposal with 1D histograms [34] | Statistical |
| Global sliding window based detector [35] | Statistical |
| Clustering with MeanShift, DBSCAN and WaveCluster [36] | Machine learning |
| YOLE and fcYOLE based on YOLO [37] | Deep learning |
| CNN model with LSTM convolutions [38] | Deep learning |
| Grafted Networks [39] | Deep learning |

The region proposal with 1D histograms [34] is a simple strategy that can be applied since with these cameras, a static scene does not contribute with events, which makes possible the use of two histograms, $H_x$ and $H_y$ that count the number of events in each cartesian position X and Y. The actual 2D region is obtained by finding intersections of the X and Y regions.

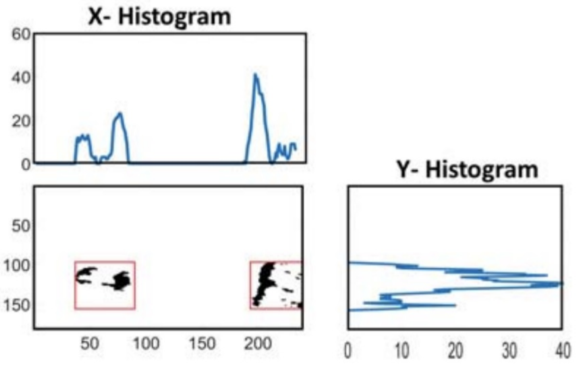


Figure 2.2: Detection performed with 1D histograms. (Taken from [34])

In [35], the authors use a global detector to obtain a region of interest that was initialized by the user, and the detection phase is performed with a detection matrix, M, which keeps track of events that may belong to the object. A quantization function is used for each event to identify which clusters belong to it, and the event's associated location is used to increase M. Then, when enough events have been accumulated in M, determined by a threshold value of at least 10% of events occurred, a sliding window is performed to establish the region with maximal activation corresponding to the object. One important aspect of this methodology is that it has a training phase where the detector learns to deduce which clusters are important to object sample descriptors $X^{w1}$ while rejecting quantization results that are common between two samples, $X^{w1}$ and $X^{w2}$ where $w1$ represents the object and $w2$ the background. To obtain the object clusters for the detection process, they create a new vector $h^{diff}_{w1,w2}$ by making use of a Bayesian bootstrapped representation, $\{h_{1w_1}, h_{2w_1}, ..., h_{N1w_1}\}$ and $\{h_{1w_2}, h_{2w_2}, ..., h_{N1w_2}\}$.

$$h^{diff}_{w1,w2} = \sum_{l=1}^{N_1} h_{lw_1} - \sum_{m=1}^{N_2} h_{mw_2} \tag{2.1}$$

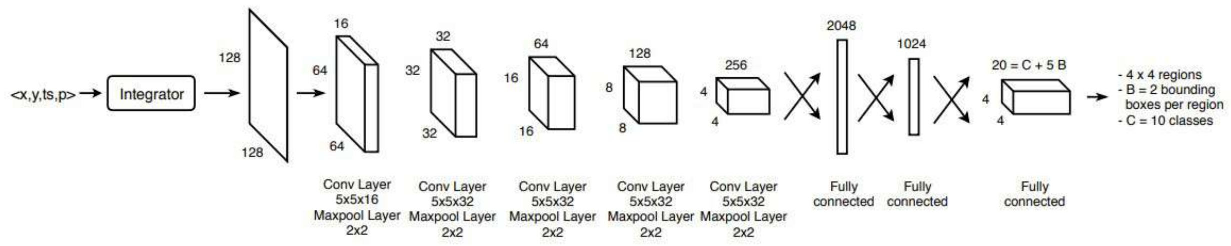
The $h^{diff}_{w1,w2}$ represents the codewords assigned. If the value is positive, it means that more codewords have been assigned to the object more times than it has been assigned to background. In essence, this vector represents the clusters that are important to describe the object samples.

As the neuromorphic camera has the characteristic of giving a set of events in the form of a point cloud (x,y,t), clustering methods are very suitable for this situation of detecting vehicles. For that reason, [36] approaches the problem by generating an object hypothesis directly from the measurements with a classic clustering method. More concretely, the procedure of a tracking system follows the philosophy of "tracking-by-clustering" and the authors have discussed the advantages and disadvantages of using each method: MeanShift, DBSCAN, and WaveCluster.

According to the authors, the MeanShift method is iterative and has the difficulty of filtering out noise. DBSCAN's key benefit is that it can detect clusters of any form, and the WaveCluster has the benefit of filtering noise through the use of a threshold.

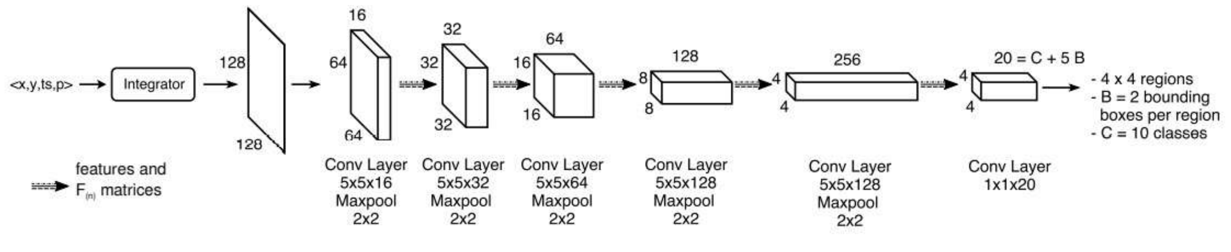
The examples given above are effective on object detection, but they do not use a feature capable of describing the chosen object, they only rely on statistical or mathematical proof. The arise of deep learning brought the capability to detect and generate features to describe each class object, making the system more reliable.

Therefore, in [37] they present two neural network architectures, YOLE (fig. 2.3) and fcYOLE (fig.2.4), for object detection based on the YOLO architecture and leaky surface. The first integrates the events into a surface during a certain time to give input to a standard CNN. The second is an asynchronous fully conventional network that exploits the sparsity of neuromorphic cameras.



<x,y,ts,p>
Integrator
128
128
16
64
64
32
32
32
64
16
16
128
8
8
256
4
4
2048
1024
20 = C + 5 B
4
4
- 4 x 4 regions
- B = 2 bounding boxes per region
- C = 10 classes
Conv Layer 5x5x16 Maxpool Layer 2x2
Conv Layer 5x5x32 Maxpool Layer 2x2
Conv Layer 5x5x32 Maxpool Layer 2x2
Conv Layer 5x5x32 Maxpool Layer 2x2
Conv Layer 5x5x32 Maxpool Layer 2x2
Fully connected
Fully connected
Fully connected

Figure 2.3: YOLE architecture for object detection. (Taken from [37])



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

Figure 2.4: fcYOLE architecture for object detection. (Taken from [37])

To perform this, a novel formalization is presented for convolution (e-conv) and max pooling (e-max-pooling), where these computations are only performed if the input feature map has changed since the last input. One important aspect to take note of is that they use a leaky surface in each layer to represent the changes of the object over time, and the second is that the network has no fully connected layers.

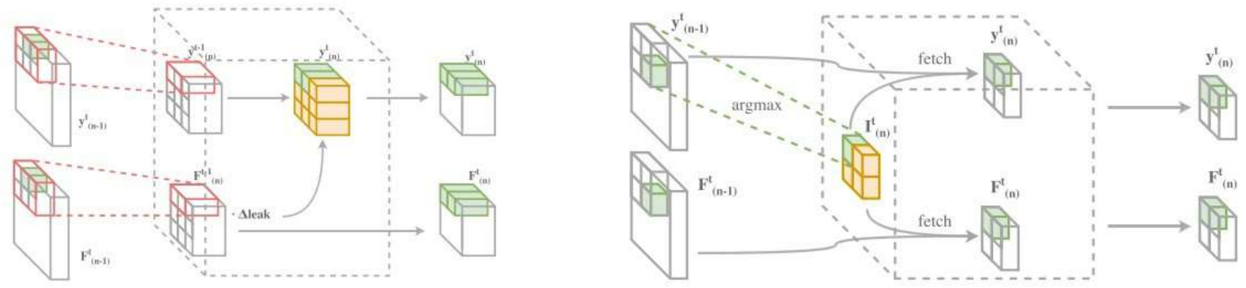


Figure 2.5: The structure of the e-conv (**Left**) and e-max-pooling layers (**Right**). The internal states and the update matrices are recomputed locally only where events are received (green cells) whereas the remaining regions (depicted in yellow) are obtained reusing the previous state. (Taken from [37])

Due to the lack of large labeled datasets for exploiting deep learning methods with information captured by neuromorphic cameras, the authors of [39] have proposed a network grafting algorithm (NGA). This algorithm consists of using two neural networks: one pre-trained network for standard framed datasets and the other is the same architecture except that the first layers are replaced by a grafted network.

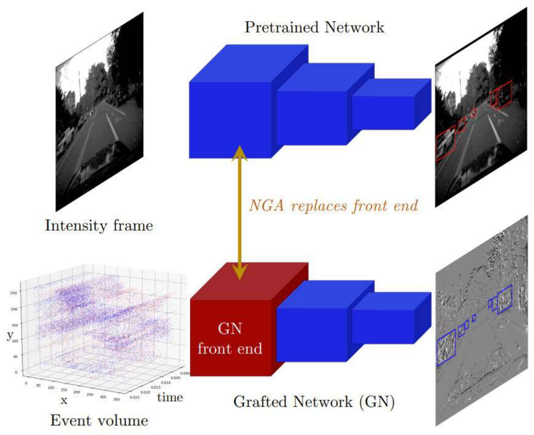


Figure 2.6: (**Top**) Pretrained Network. (**Bottom**) Grafted Network. (Taken from [39])

Within the scope of the student network learning with the knowledge of the teacher network, NGA trains the grafted network (GN) to achieve a similar performance, like the pretrained network (N) by increasing the representation similarity between features. This training is performed with the help of three main loss functions. Mean-Squared-Error for learning similarity representation between the features of the front-end. The second loss is also a Mean-Squared-Error for evaluation features for the middle net layers in the network and draws inspiration from the Perception Loss [40]. Both of these losses terms minimize the magnitude differences between hidden features, which leads the authors to make a third loss to encourage the GN front end to generate intensity textures based on Gram loss [41]. The final loss is the sum of all losses.

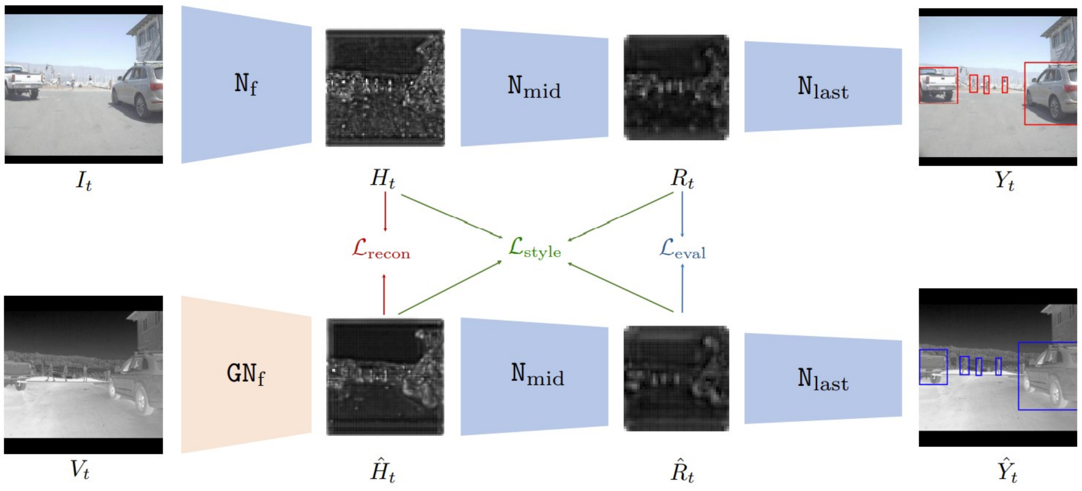


Figure 2.7: (**Top**) Pretrained Network. (**Bottom**) Grafted Network. Learning object detection with different modalities of dataset. (Taken from [39])

In order to explore the temporal characteristics, the authors of [38] introduced a recurrent architecture and temporal consistency loss for better-behaved during the training of object detection model. This model receives a constructed frame by accumulating events during an interval of time and the architecture is formed by three convolutional layers to extract low-level features, in particular Squeeze-and-Excitation layers [42], and five ConvLSTM layers [43] to extract high-level spatio-temporal features. In other words, these layers contain a memory state to accumulate meaningful features over time and to remember the presence of objects even when they stop generating events. The final block is a bounding box regression head, called Single Shot Detector (SSD) [44], that receives multiscale features provided by each ConvLSTM layer to predict the bounding box of each detection.

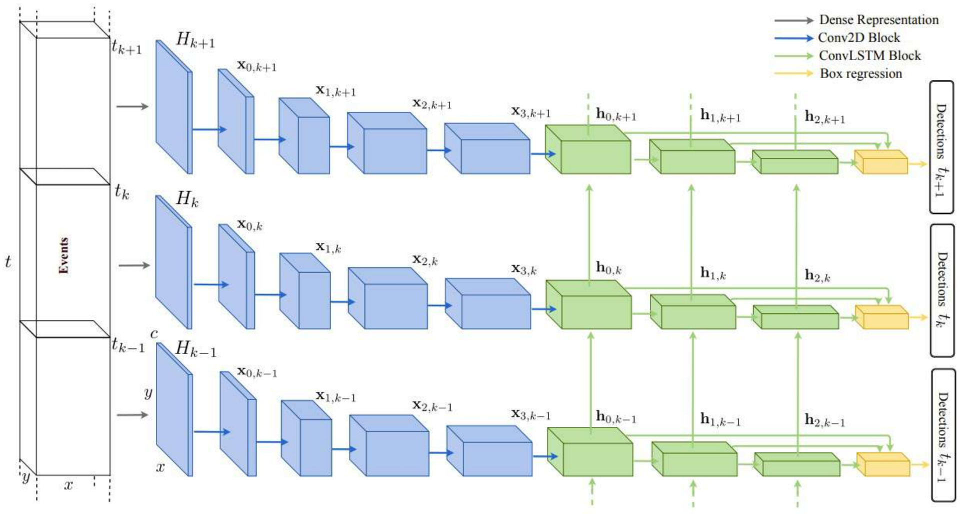


Figure 2.8: Object detection with recurrent neural networks. (Taken from [38])

### 2.4.2 Classic tracking algorithms

The core of MOT is implicitly based in many publications on the use of one of the four standard tracking algorithms: SORT [2], GM-PHD [45], GM-CPHD [46], and PDAF [47]. According to [36], these algorithms have the benefit of making full use of characteristics from event data since they need less computation and are extremely effective.

#### SORT

This algorithm focuses on a single hypothesis tracking methodology with a standard Kalman Filter, then applies a data association with a cost matrix for each detection and for each target based on intersection over union distance (IOU). After that, an Hungarian algorithm is used to optimally solve the assignment problem. It also applies a minimum IOU to reject bad assignments. For track handling, when a new detection happens, it is only considered apt to be tracked if any detection in the current frame has an overlap with the existence of untracked detection in previous frames. This methodology for tracking has been used in [2].

#### GM-PHD

In the context of data association uncertainty, noise, false alarms, and detection uncertainty, the GM-PHD filter is a recursive algorithm that predicts the time-varying number of targets and their states from observation sets. The algorithm treats the target and measurement sets as random finite sets and recursively applies the probability hypothesis density

(PHD), which is essentially the first order-statistic of the random finite set in time, for posterior intensity propagation. The target dynamics and birth process, as well as the posterior intensity at any time step, are considered Gaussian mixtures under linear and Gaussian assumptions. The efficiency of recursions with a large number of Gaussian components is increased.

#### GM-CPHD

The posterior intensity of a random finite set of targets is propagated recursively in the probability hypothesis density (PHD) filter. The posterior intensity and posterior cardinality distribution are propagated together in the cardinalized PHD (CPHD) filter, making it a generalization of PHD recursion. By integrating the cardinality information, the accuracy and stability are improved. This work is essentially a closed-form solution to the CPHD recursion under the premise of linear Gaussian target dynamics and a birth model. When compared to a typical PHD filter, the CPHD filter not only eliminates the requirement for data association in traditional tracking methods, but it also improves the accuracy of individual target state estimates and the variance of the estimated number of targets.

#### PDAF

For each valid measurement, the probabilistic data association filter (PDAF) computes the probability of the target being tracked. This probabilistic or Bayesian information accounts for the measurement origin uncertainty. PDAF operates on confirmed measurements at the present moment, and an association probability is calculated for each measurement to determine the weight of the current measurement in a combined innovation. This integrated innovation helps in updating the state estimation.
Finally, to understand the impact of each tracking method in terms of processing time is presented a table adopted in [36].

Table 2.2: The FPS using different tracking methods.

| **Tracker** | **FPS** |
|---|---|
| SORT [2] | 552 |
| GM-PHD [45] | 3 |
| GM-CPHD [46] | 4 |
| PDAF [47] | 46 |

### 2.4.3 Multi-Object Tracking

According to [48], the idea was to build a real-time and embedded system that makes use of stationary neuromorphic cameras for object tracking applications by only capturing moving objects. The authors adopt a hybrid strategy that differs from event-based or frame-based methods. The asynchronous events are first gathered into a binary picture, and then overlap-based tracking is applied to these frames. The frames are converted back to spikes for efficient processing on the IBM Neuromorphic device for later object categorization. This work follows the rules of the DBT procedure, where the detection is performed by a region proposal with the intersection of two 1D histograms (section 2.4.1) and the tracking purpose is divided into four modules: track assignment, tracker update, occlusion model, and cleanup trackers. The track assignment is based on a 1D region proposal and KF, i.e., the tracker's new position is estimated based on its previous velocity, and the resulting region is evaluated by overlapping against the region proposal. The second module updates the tracker based on an overlap threshold. Next, the occlusion model deals with objects that are occluded during a certain time. The final module eliminates the object trackers that are leaving the scene.

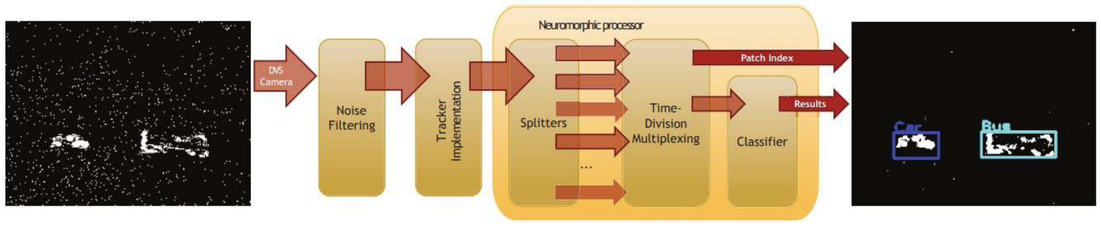


Figure 2.9: Block diagram of the tracking system. (Taken from [48])

In [49] was introduced a simple and efficient object tracking framework, consisting of a local tracker and a global detector, that uses a discriminative representation for the object with online learning which allows to detect and re-track the object when it comes back into the field-of-view. The tracker search is conducted in a local region of the image and the detector is only activated when the tracker is lost and is performed by a sliding window in the entire image (section 2.4.1). Both are trained by a discriminative classifier with samples generated by statistical bootstrapping.

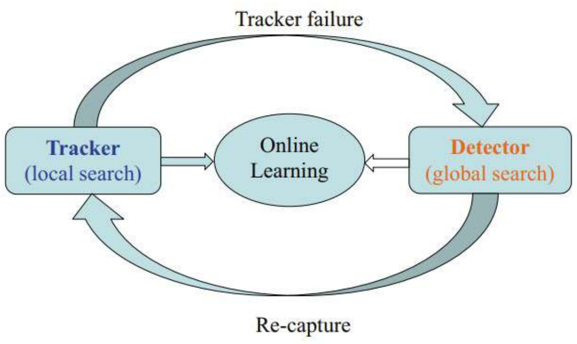


Figure 2.10: Tracker and detector flow. (Taken from [49])

In [50], the idea is an event-based pattern tracking that updates iteratively the model location and orientation to the target 2D image plane based on the arrival of events. In other words, this method performs visual data processing for each incoming event at the time it arrives, asynchronously, and provides a continuous and iterative estimation of the geometric transformation between the model and the events representing the tracked object. According to the authors, this model can handle isometry, similarities, and affine distortion while maintaining real-time performance without the need for expensive hardware. Furthermore, the solution is able to overcome ambiguous scenarios of object occlusion that conventional frame-based algorithms handle poorly by utilizing the dimension of time that is currently underutilized by most artificial vision systems.

Since the neuromorphic cameras output discrete events, the authors in [36] and [51] have explored the tracking by generating an object hypothesis directly from the measurements with a classic clustering method. The first task is to demonstrate the feasibility and potential of the tracking-by-clustering approach. In the detection stage, three classical approaches to clustering were evaluated: MeanShift, DBSCAN, and WaveCluster. In terms of tracking stage, was used online multitarget tracking via four different algorithms: SORT, GM-PHD, GM-CPHD, and PDAF. Alternatively, the second work redefined the MeanShift clustering algorithm to employ asynchronous events to detect vehicles, while the tracking is accomplished by incorporating the Kalman filter to effectively predict and correct the clustering tracking process.

Looking for others strategies for object tracking, UMACE [52], ASEF [53] and, specially, MOSSE filter [54] have shown their great potential. This paradigm can achieve a speed of tracking above 100 frames because of the use of Fourier domain Fast Fourier Transform (FFT) to compute the correlation. In [55], was demonstrated a new way of dealing with translations over the image using the Fourier domain. This leads to more samples by keeping computation demand low. Consequently, it allows the tracker to learn a discriminating SVM classifier between object appearance and the environment. Moreover, Kernelized Correlation Filters (KCFs) were developed to deal with kernel ridge regression issues. The proposed method can be used for multi-channel images, allowing more features to be used in appearance representation, such as HOG.

The tracking flow with these correlation filters (CF) resumes to start with a small window centered on the object in the first frame. Then the CF is applied over a search window in the next frame to track the target that is located at the position corresponding to the maximum value in the correlation output. Next, an online update to the filter is made to capture the dynamic appearance change of the object when tracked, preventing the drifting during the tracking and, consequently, the loss of the object.

Focusing in the tracking with neuromorphic cameras, the authors in [56] present a robust event-stream pattern tracking method based on Correlation Filter mechanism where the feature appearance representation is performed by an hierarchical three convolutional layers of VGG-16 architecture to capture the dynamically appearance change of the object when tracked. To respect the constraints of the input model, three-channel, the events are integrated into a rate coding map during a short interval, then the coding map is assigned to each input layer.

In [57] the authors follow the approach based on DBT for the visual tracking problem for ground vehicles. In the first step, they accumulate events into a representation of events count. Secondly, they apply an object tracking framework that contains a YOLO detector trained offline and a Correlation Filter tracker trained online with hand-crafted HOG [58] features, following the work of [55]. Furthermore, a fusion strategy is used to produce the tracking result, which includes the results from the detector and tracker using the Kalman Filter and data association using the Hungarian Algorithm.

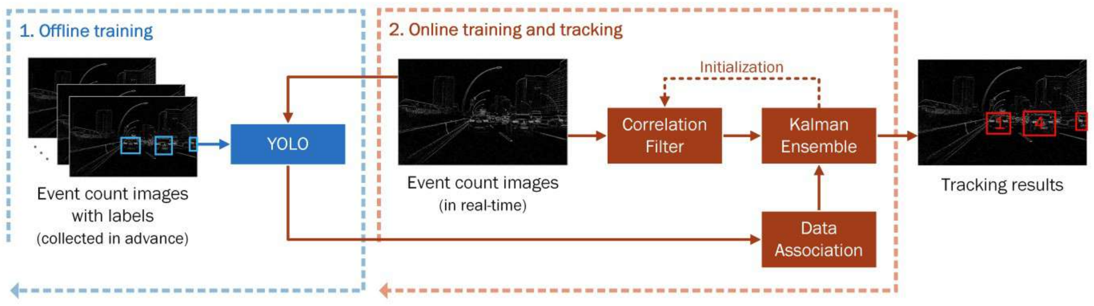

1. Offline training
2. Online training and tracking
YOLO
Initialization
Correlation Filter
Kalman Ensemble
Data Association
Event count images with labels (collected in advance)
Event count images (in real-time)
Tracking results

Figure 2.11: object tracking framework. (Taken from [57])

To summarize and conclude this chapter, it will be proceed the implementation of a recurrent neural architecture technique for the CNN [38] and SSCN [32] typology to achieve the goal of object detection after studying the state of the art centered on the tasks of this study. In addition, while working on the same objective, it will be validated an SNN typology with a Tiny-YOLO architecture [20]. For the final multi-vehicle tracking assignment, it will first examine the classic tracking technique SORT [2], and then a deep model of vehicle re-identification will be constructed based on [4] work to validate and analyze the deepSORT method [3].

# 3 Background Knowledge

In this section is provided information about the camera technology and theoretical concepts that help follow the full work document.

## 3.1 Neuromorphic Vision Sensor

Neuromorphic cameras are bio-inspired vision sensors that attempt to emulate the functioning of biological retinas. As opposed to standard cameras, which generate frames at a constant frame rate, these sensors output data only when a brightness change is detected in the field of view. This results in a sensor able to produce a stream of asynchronous events that sparsely encode changes in the field of view of the camera with microsecond resolution, low power consumption, and low latency. The stream of events encodes the time (t), location (x, y), and polarity (p) of the intensity change (-1 or 1).

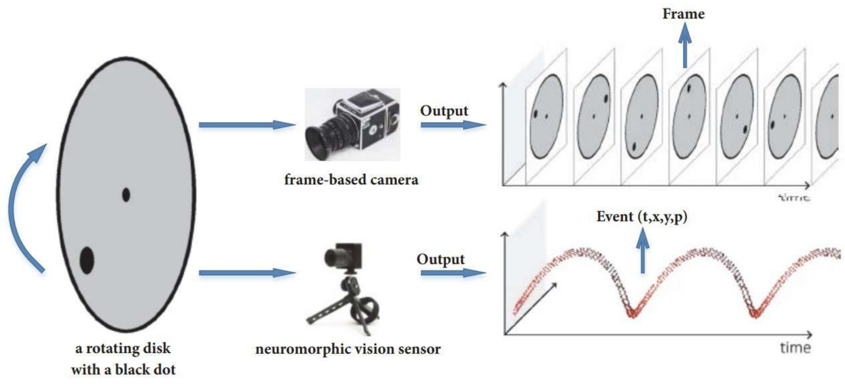


Figure 3.1: Output comparison between frame-based cameras and Neuromorphic vision sensors in terms of the rate of captured information.

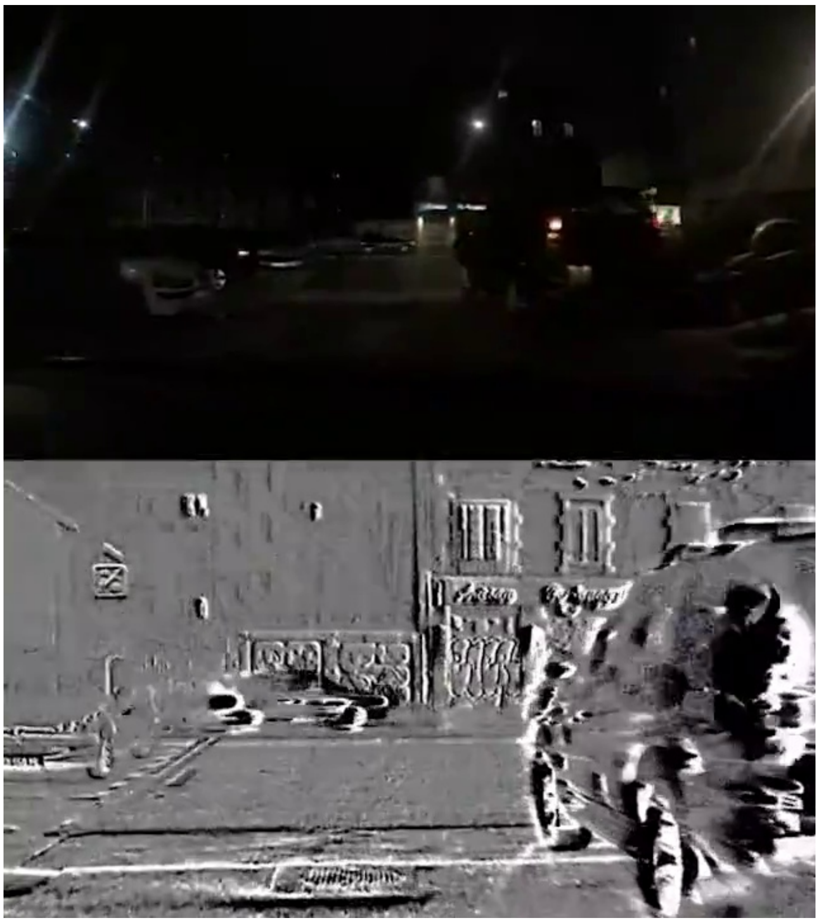


Figure 3.2: Output comparison between frame-based cameras and Neuromorphic vision sensors in terms of low light conditions.

In figures 3.1 and 3.2, it is visible the differences between frame-based cameras and neuromorphic vision sensors at the rate of capture information and low light conditions described in section 1.1.

## 3.2 Representation of Events

Fundamental techniques underlying computer vision are based on the ability to extract meaningful features. To this extent, convolutional neural networks (CNNs) rapidly became the first choice in many computer vision applications, such as image classification, object detection, and semantic scene labeling. However, usually the input of these networks is images. As a result, the majority of works with neuromorphic cameras adhere to the rule of representing the encoded stream of events as a frame, i.e., an accumulation of events on a surface over a time interval. In this work, two representations will be analyzed: Time Surface and Event Volume.

We will assume in the following sections that we are given an input sequence of events $E_i = (x_i, y_i, p_i, t_i)$ in a time interval of size $\triangle t$.

### 3.2.1 Time Surface

The Time Surface[1] (TS) algorithm accumulates events during an interval of time, giving more importance to the newer events and less to the old ones. This is given by an exponential decay according to a parameter $\tau$.

The formula for the exponential decay time surface at time $t_i$ is the following:

$$TS_{t_i}(y, x, p) = exp(-\frac{t_i - t}{\tau}) \ for \ each \ event \ E_i \ when \ t \leq t_i \tag{3.1}$$

Where:

- $t_i$ is the event arriving time.
- t is the timestamp of the last event in the specific position.
- When $\tau$ is small, only recent events may actually contribute to the time surface; when $\tau$ is large, more older events can contribute to the time surface.

Since the goal is to detect and track multiple vehicles, and different vehicles can have distinct velocities, this representation suits well to managing various rates of incoming events thanks to the two different decay values, 100 ms and 10 ms. The Time Surface input shape used in the experiments is $(4, M, N)$, where $M$ and $N$ are the width and height of the input representation frame, and the first dimension refers to the two decays for each polarity.

[1]Based on information from https://docs.prophesee.ai

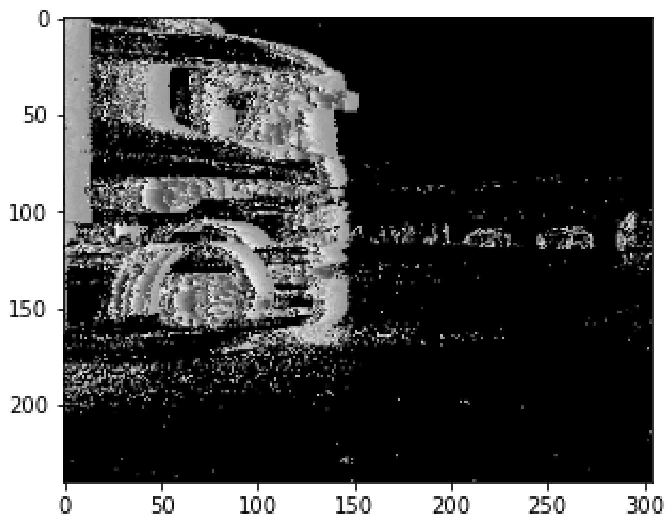


Figure 3.3: Time Surface representation example with event data from GEN1 Automotive dataset [5].

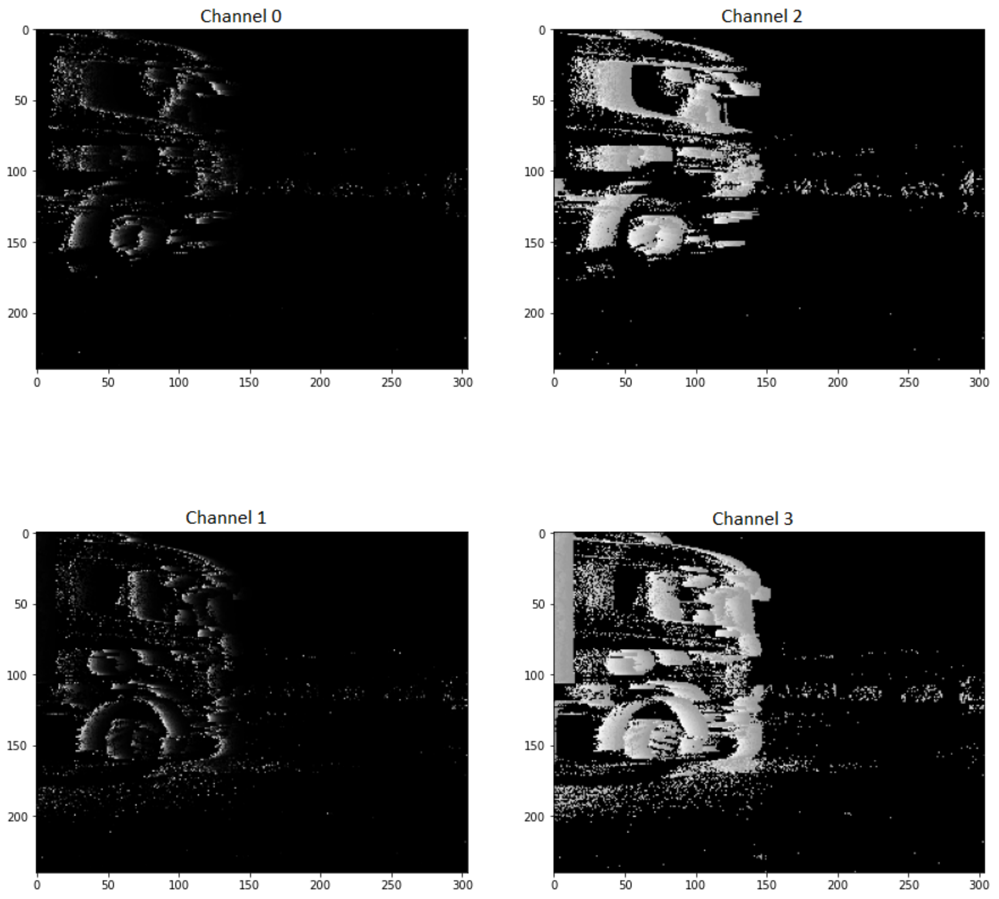


Figure 3.4: Time Surface channel contribution with event data from GEN1 Automotive dataset [5].

### 3.2.2 Event Volume

In Event Volume[2] (EV) each time bin is divided into micro time bins. Each event, similar to histogram creation, is assigned to a cell based on its location (x,y) and to a time bin based on its timestamp (t). Instead of counting events like in Histogram, each event is weighted by its temporal distance from the center of the neighboring micro time bins.

For each event $E_i$, a linearly-weighted histogram update is performed, for each polarity:

$$EV(t_i, c_i, y, x) = EV(t_i, c_i, y, x) + max(0, 1 - |\lfloor t^* \rfloor - t^*|) \tag{3.2}$$

Where:

- $t_0$ is the starting timestamp.
- $\lfloor t^* \rfloor$ is the closest micro bin of $t^*$.
- $t^* = c \times \frac{t_i - t_0}{\triangle} - 0.5$ is the relative temporal distance to the centre of the corresponding micro time bin.
- $\triangle$ is the time interval in each time bin.
- $t_i$ is the event arriving time.
- c is the number of micro time bins.

This representation combines a histogram with time information from a linear time surface to generate an input tensor of shape $(10, M, N)$, with bins and polarity combined in the first dimension, and $M$ and $N$ are the width and height of the input representation frame.

To further understand this representation, here is a figure [3] of two positive events in different temporal positions and their different contributions to each micro bin:

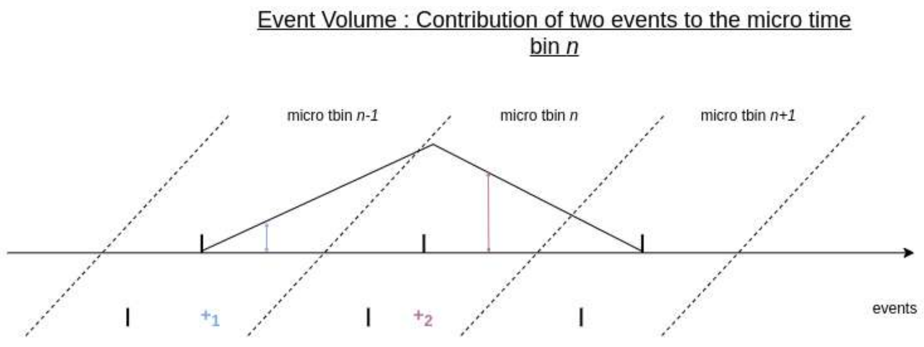

Event Volume : Contribution of two events to the micro time bin *n*
micro tbin *n-1*
micro tbin *n*
micro tbin *n+1*
+1
+2
events

Figure 3.5: Event contribution.

The first event, colored blue in the above image, occurs in the micro time bin $n - 1$ and is temporally far from the center of the micro time bin $n$, so its contribution to the micro

[2]Based on information from https://docs.prophesee.ai

[3]source https://docs.prophesee.ai

time bin $n$ is small. The second event, in purple, is closer to the centre of the micro time bin $n$ in regards to time and hence contributes more to the micro time bin $n$.

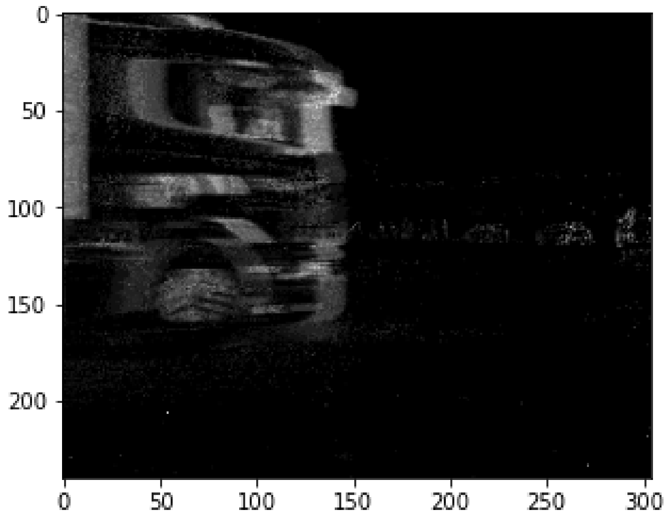


Figure 3.6: Event Volume representation example with event data from the GEN1 Automotive dataset [5].

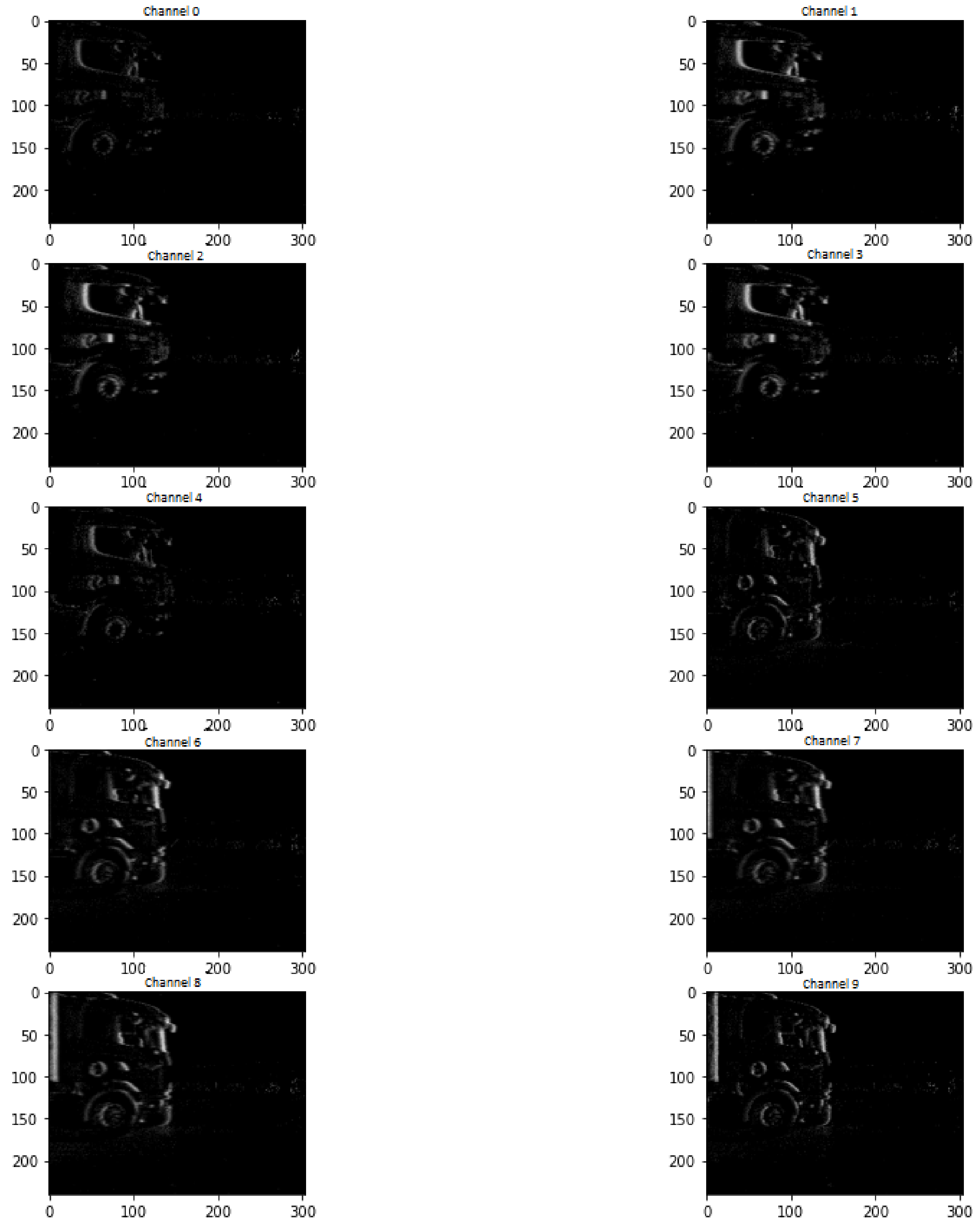


Figure 3.7: Event Volume channel contribution with event data from the GEN1 Automotive dataset[5].

## 3.3 Squeeze Excitation Layer

The authors of [42] introduced a new architectural unit with the goal of improving the quality of representations through a mechanism that allows the network to perform feature recalibration by explicitly modelling the interdependencies between the channels of its convolutional features. In other words, this unit can learn to use global information to selectively emphasise informative features and suppress less useful ones.

The authors prove that the SE block structure has the advantage of being simple and that it can be directly employed in existing state-of-the-art architectures by replacing components with their SE equivalents, where performance may be effectively boosted. SE blocks are likewise computationally light, imposing just a minor increase in model complexity and computational cost.

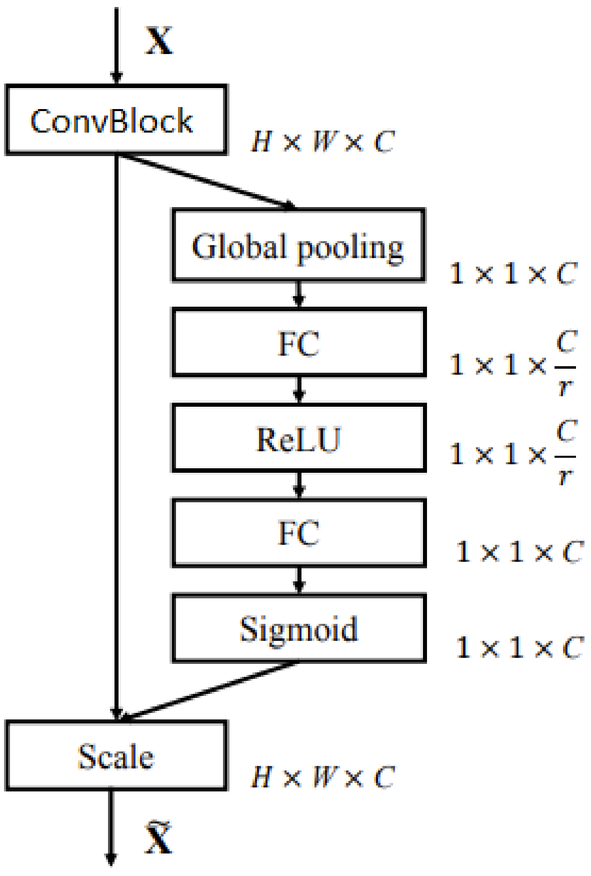


Figure 3.8: A Squeeze-and-Excitation block. (Taken from [42])

The structure of the SE building block is illustrated in the figure 3.8. This process is made up of two steps: squeeze and excitation. The first step is achieved by using global average pooling to generate channel-wise statistics in order to exploit channel dependencies. The second step, excitation, is where the adaptive recalibration of features happens. This is done by using the information aggregated in the Squeeze step, followed by a gating mechanism and sigmoid function. The gating mechanism is constituted by first applying one FC layer and a ReLu function to add some non-linearity and also reduce the output channel by a $r$ ratio. Then, another FC layer with a sigmoid function to smooth and return the output channel dimensionality. To close this unit the output of the block is rescaled to $H \times W \times C$.

## 3.4 LSTM

To enable modelling time-dependent and sequential data tasks, such as stock market prediction, machine translation, text generation, and image captioning, the RNN [59] were introduced. They are basically dynamic systems that have an internal state at each time step of the classification. This is due to circular connections between higher- and lower-layer neurons and optional self-feedback connections. These feedback connections enable RNNs to propagate data from earlier events to current processing steps, fig. 3.9[4]. However, RNNs have vanishing gradients triggered by long-term dependency problems, making learning long data sequences difficult.

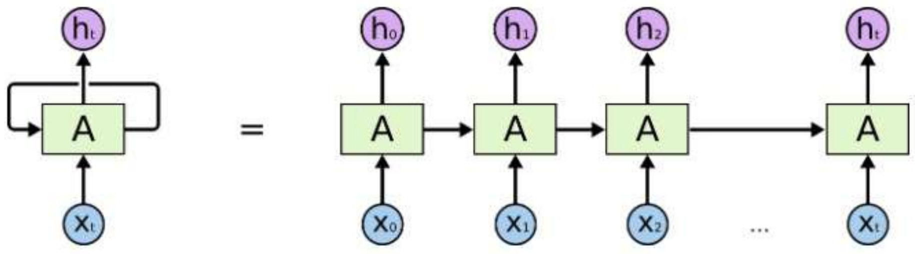


Figure 3.9: RNN Diagram.

To prevent the vanishing gradient, the LSTM [60] architecture was presented, which is a special kind of RNN capable of learning long-term dependencies. As the LSTM is derived from RNN, they also have a chain like structure, but instead of having a simple layer structure, there are four layers with operations between them.

[4]Source: https://colah.github.io

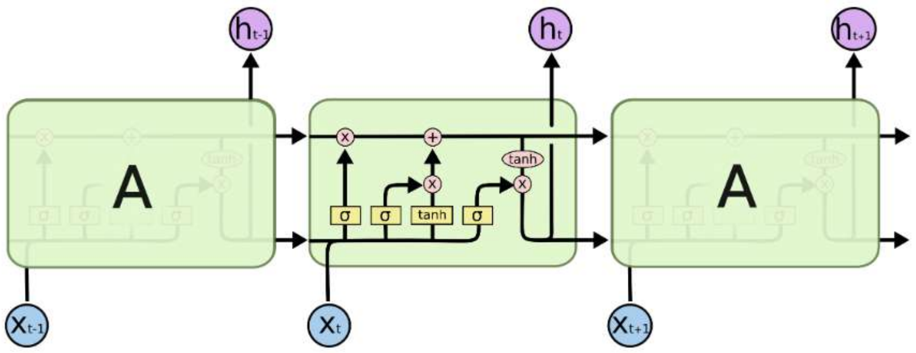


Figure 3.10: LSTM Diagram.

In the above diagram[5], the yellow squares are small learned neural networks, and the pink circles represent pointwise operations. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations. For each green box, there are two inputs, a cell-state (C) and a hidden-state (h), from the output of the prior node to the inputs of the posterior nodes.

The initial stage of LSTM determines whether the information from the cell state is significant or not. This choice is determined by a sigmoid layer known as the "forget gate layer", 3.11a. It looks at the previous hidden state $h_{t-1}$ and the input data $x_t$ and returns a number between 0 and 1 for each number in the cell state $C_{t-1}$, where 1 indicates "totally keep this", whereas 0 indicates "absolutely get rid of this".

The following step is to decide what new information we will maintain in the cell state and is divided into two parts. First, a sigmoid layer known as the "input gate layer" determines which values will be updated. Secondly a tanh layer then generates a vector of new candidate values, $\tilde{C}_t$, which might be added to the state, figure 3.11b.

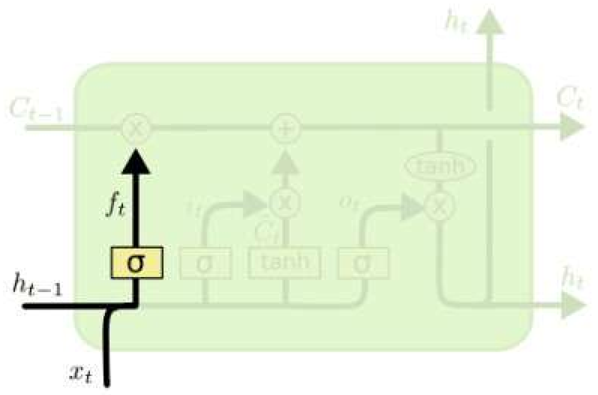
In the third step, it only needs to update the cell state, $C_{t-1}$, to the new cell state, $C_t$, determined in the previous step. The old state is multiplied by $f_t$, erasing the things that were previously determined to be forgotten in the first step. Then $i_t \cdot \tilde{C}_t$ is added to give the final cell state value, figure 3.11c.

Finally, it is necessary to determine what will be sent to the output. This output will be filtered and will be based on the cell state. First, we run a sigmoid layer to determine which parts of the cell state will be output. The cell state is then passed through tanh and multiplied by the output of the sigmoid gate, resulting in only the parts that were determined to be output, figure 3.11d.

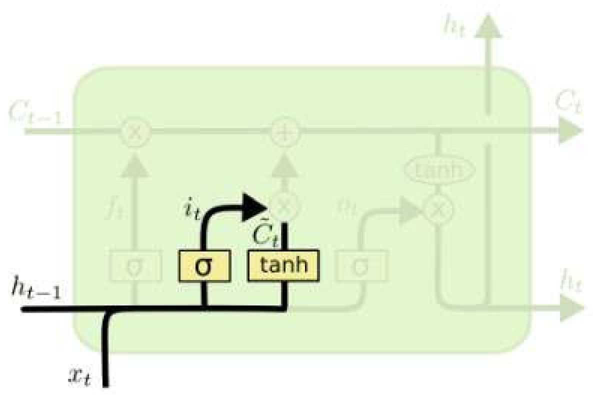
In essence, the presence of gate activations allows the LSTM to better balance gradient

[5]Source: https://colah.github.io

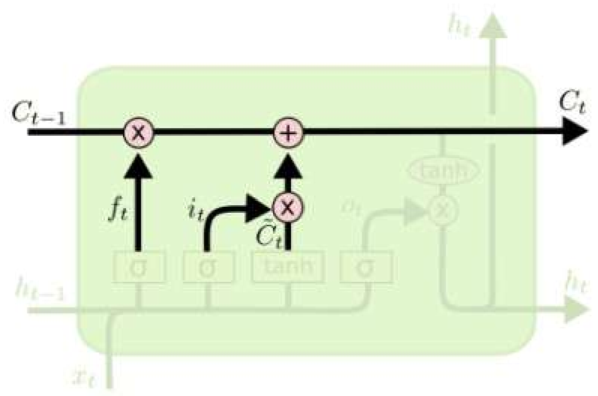
values during backpropagation and, as a result, update the model's parameters appropriately, preventing vanishing gradients.



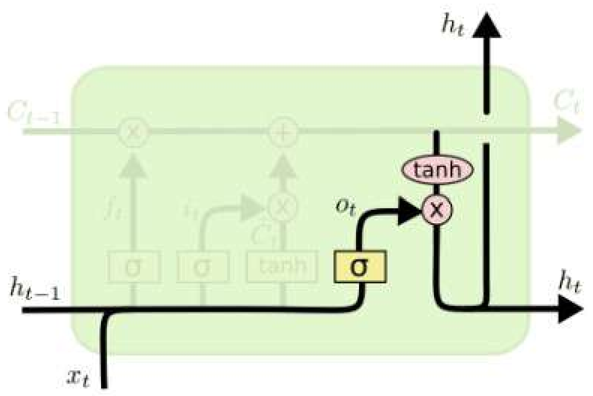
(a) Step 1 → Forget gate layer



(b) Step 2 → Candidate values to the Cell-State



(c) Step 3 → Cell-State Update



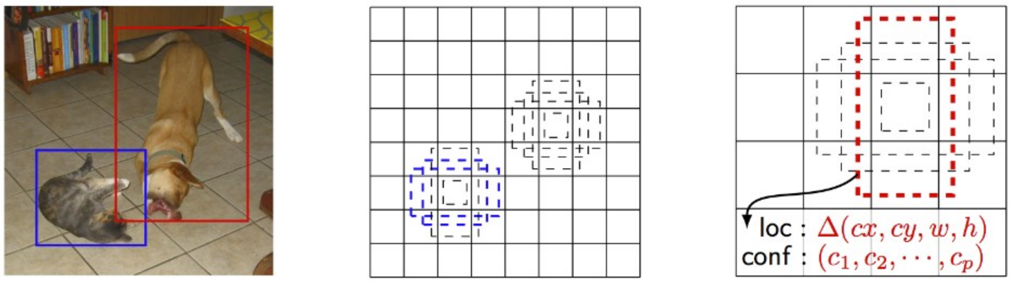
(d) Step 4 → Hidden-State Update

Figure 3.11: The four layer of LSTM model.

## 3.5 Single Shot Multibox Detector

The name of this architecture comes from: **Single Shot Detector** → tasks of object localization and classification are done in a single forward pass of the network; **MultiBox** → technique for bounding box regression developed by Szegedy; **Detector** → the network is an object detector that also classifies those detected objects.

The SSD network computes both location and confidence scores using small convolution filters. In other words, for each feature map, SSD applies a set of convolution filters to each cell to make predictions. This network receives multi-scale feature maps in order to detect objects independently, i.e., higher-resolution feature maps are responsible for detecting small objects and the lower-resolution feature maps are responsible for larger objects.


loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$

(a) Image with GT boxes (b) $8 \times 8$ feature map (c) $4 \times 4$ feature map

Figure 3.12: Example of detection small and larger shape with different feature maps. (Taken from [44])

### Default Bounding Box: Priors

To understand how and why the priors of SSD are created, it is required to know the MultiBox technique. In MultiBox, the researchers created what we call priors, which are precomputed, fixed-size bounding boxes that closely match the distribution of the original ground truth boxes. Those priors are selected in such a way that their IoU is greater than 0,5. Despite the fact that an IoU of 0,5 is still not good enough, it is a better strategy than starting the predictions with random coordinates for the bounding box regression algorithm. Therefore, MultiBox starts with the priors as predictions and attempts to regress closer to the ground truth bounding boxes.

In contrast to MultiBox, SSD has fixed priors and every feature map cell is associated with a set of default bounding boxes with different dimensions and aspect ratios. These priors are manually chosen, whereas in MultiBox, they were chosen because their IoU with respect to the ground truth was over 0,5. In theory the advantage of SSD with fixed prior is a generalization for any type of input. Moreover, SSD also keeps the default boxes to a minimum (4 or 6) with one prediction per default box and instead of using a global coordination for the box location, the boundary box predictions are relative to the default boundary boxes at each cell ($\triangle c_x$, $\triangle c_y$, $\triangle w$, $\triangle h$), i.e the offsets to the default box at each cell for its center ($c_x$, $c_y$) and the width $w$ and the height $h$.

### Matching strategy

For every time that SSD makes predictions, they are classified as positive or negative matches. The function of localization cost is calculated using only the positive matches.

The match is performed by computing the IoU between the default bounding box(not the predicted boundary box) and ground truth. If it is greater than 0,5 is positive, otherwise it is negative. After this, only the corresponding prediction boxes for the positive matches are used to calculate the cost, encouraging each prediction to be closer to the default bounding box and leading to more stable training.

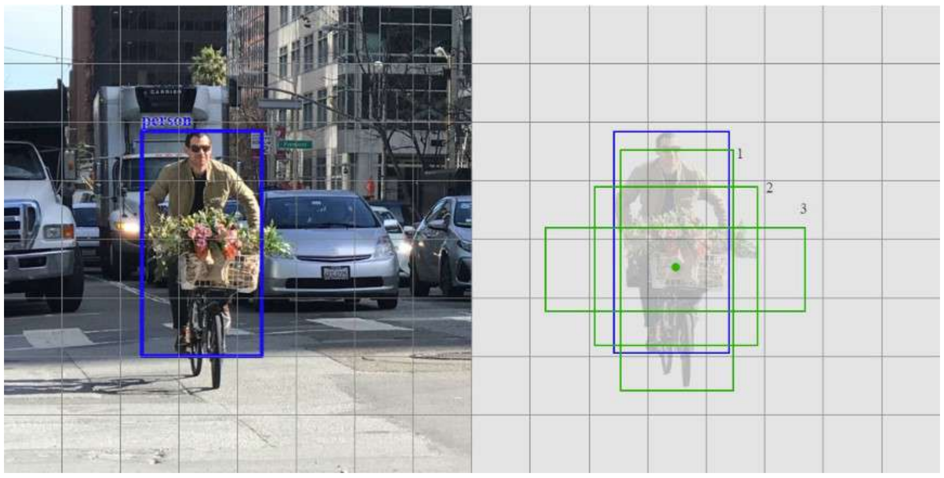


Figure 3.13: Ground truth box (**Blue**), Priors (**Green**). In this example only prior 1 and 2 will be counted as positive match.

### Hard Negative Mining

However, during the training, there are always many more negative than positive examples, leading to an imbalanced dataset that causes difficult training for detection objects. Therefore, in practice, it is advised to keep the ratio between negative and positive at about 3:1. The way of doing this is just by sorting the negative examples by the confidence loss and picking examples with more loss value, leading, once more, to faster and more stable training. It is needed to keep the negative samples so that the network continues to learn what is background and is explicitly told what constitutes an incorrect detection.

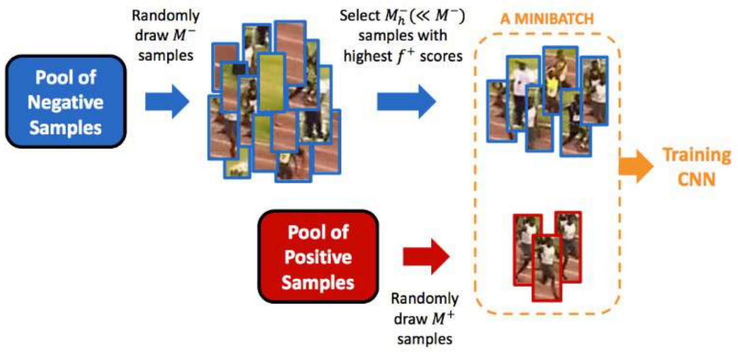

Randomly draw $M^-$ samples
Select $M_h^- (\ll M^-)$ samples with highest $f^+$ scores
A MINIBATCH
Pool of Negative Samples
Training CNN
Pool of Positive Samples
Randomly draw $M^+$ samples

Figure 3.14: Example of hard negative mining.

### Non-Maximum Suppression (NMS)

Due to the nature of SSD a large number of boxes are generated during the inference time, which means that it is imperative to eliminate those that have lower confidence (e.g. less than 0.01) and IoU (e.g. less than 0.45), and keep only the top N predictions by applying a non-maximum suppression. This guarantees that the network retains just the most likely predictions while removing the duplicated pointing to the same object.

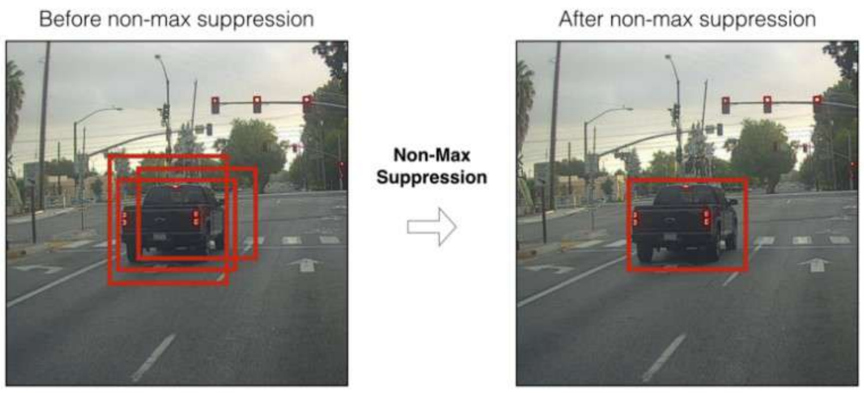


Figure 3.15: Example of non-maximum suppression.

To achieve this, we use the hard non-maximum suppression method, where the bounding boxes are initially sorted by confidence score in descending order to retain just the top N candidates. Then, iteratively, we retain the boxes with higher confidence and calculate the overlap with the rest of the candidates. The boxes with an IoU lower than the chosen threshold are discarded for the next iteration.

## 3.6 Sparse Convolutional Layers

Instead of FFT or *im2col* functions to build the computational pipeline of standard convolution, Sparse Convolution collects all atomic operations and saves them in a rulebook to instruct the computation pipeline.

As our network receives as input the coordinates and values of a 2D image, the concept of sparse convolution is explained with a 2D dimensional example. However, this can be extended to other dimensions with no essential differences because the input is represented in the same way, i.e., it only needs the coordinates and values of each non-zero value of the point cloud.

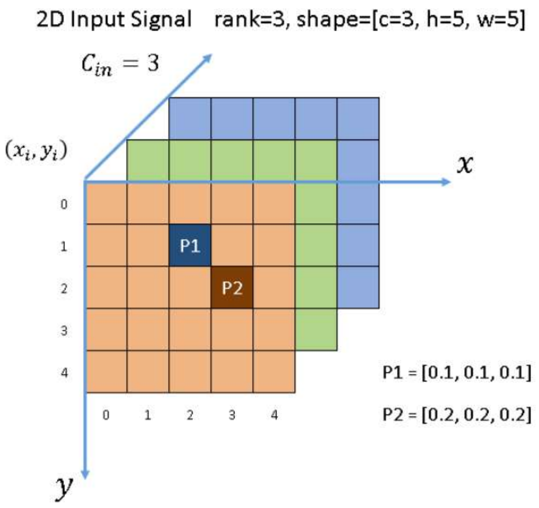


Figure 3.16: 2D input for sparse convolution.

In fig. 3.16[6] we have a input 5x5 with 3 channels where all the pixels have zero value with exception of points P1 and P2. These points are called active input sites according to [32]. The shape of the dense input would be 1x3x5x5, i.e 1 number of samples, 3 channels and a size of 5x5 pixels. In sparse form, the data list is [ [0.1, 0.1, 0.1], [0.2, 0.2, 0.2] ] and the index list is [ [2,1], [3,2] ] with XY order.

The output of the sparse convolution has two definitions according to [32]. One is a regular output that calculates the output sites as long as the kernel covers an input site, similar to the convolution output, however saves a lot of computation by discarding any computation on the regions of non-active sites. The other one is called the submanifold output. The convolution output will only be counted when the kernel center covers an input site.

[6]Source https://towardsdatascience.com

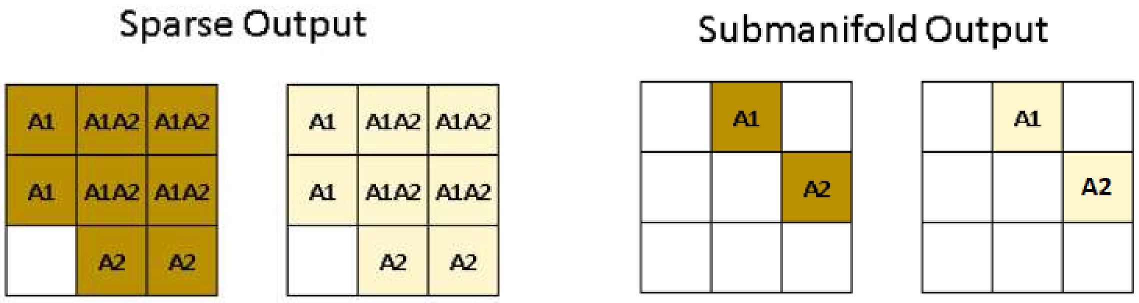


Figure 3.17: Example of the two output definitions by Zhiliang Zhou .

In the figure 3.17[7] we present a result from applying a $3 \times 3$ kernel with 2 output channels and a stride of 1. The convolution result from P1 and P2 is denoted by A1 and A2, respectively. While the A1A2 are the sum of outputs from P1 and P2.

Before continuing with the example, it is important to understand the impact of regular convolutions over a sparse image. In this context, the authors of [61] analyzed this and have called it the submanifold expansion.

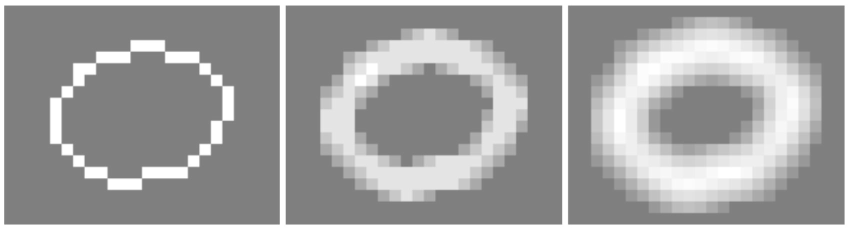


Figure 3.18: Example of "submanifold" dilation. **Left:** Original curve. **Middle:** Result of applying a regular $3 \times 3$ convolution. **Right:** Result of applying the same convolution again (Taken from [61]).

In fig. 3.18 we can see the sparsity on the grid rapidly disappears after applying regular convolutions. However, if we restrict the output of the convolution only to the set of active input points, hidden layers in the network cannot capture a lot of information that may be relevant to the final output.

The implementation of sparse convolution is accomplished by generating an index and a rulebook to schedule and execute all atomic operations. This is done in two steps: hash table generation and rule book generation.

The hash-map is created containing information regarding active input sites and the corresponding active output sites, considering either the sparse or submanifold convolution definition as per the implementation.

[7]Adapted from https://towardsdatascience.com

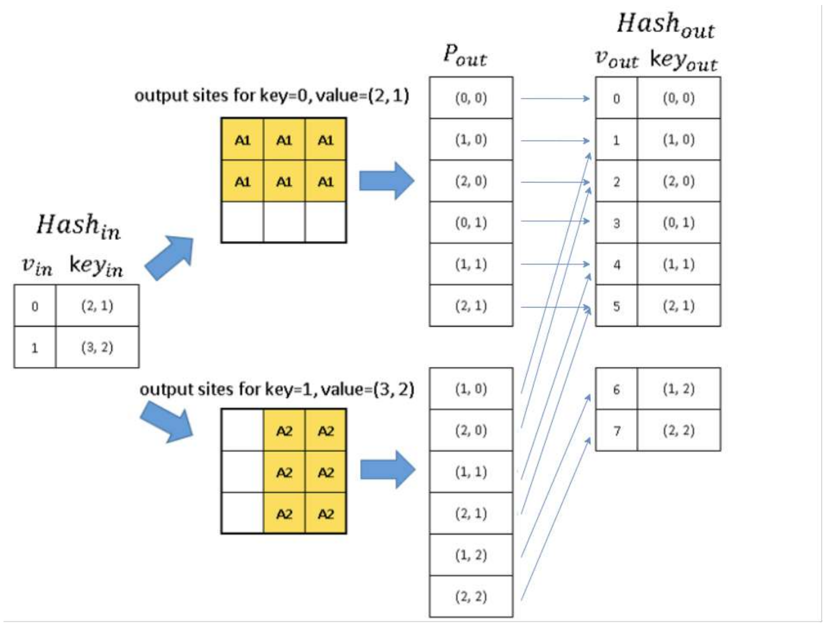
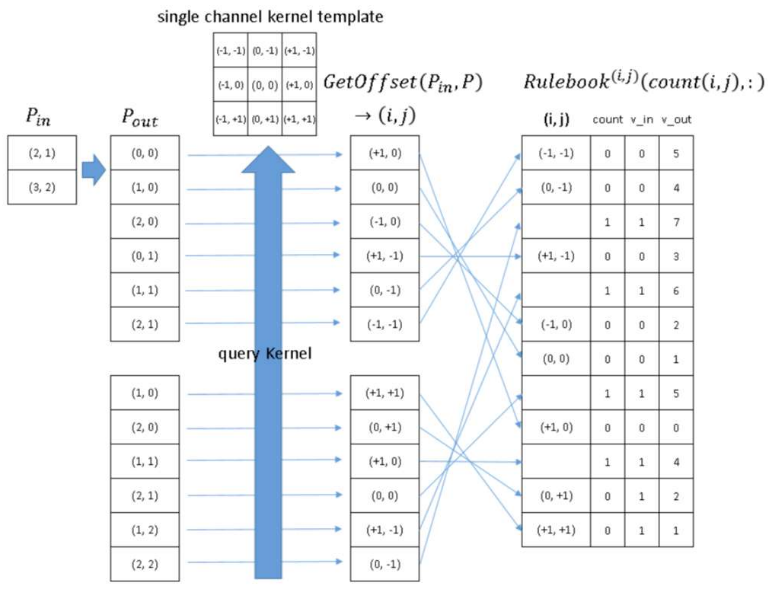


Figure 3.19: Hash table from sparse convolutions.

$Hash_{in}$ is the input hash table in the image above[8]. It maintains all of the active input sites, where $v_{in}$ is the hash table index and $key_{in}$ is the spatial location of the active input sites. The output sites describe the output generated when convolving a convolution kernel over the sparse input data as per the rules of sparse convolution. $P_{out}$ holds the spatial location of these output active sites. At last, an output hash table $Hash_{out}$ is generated which holds all output spatial locations as well as keys.

The second step is to build the rulebook, where the purpose is similar to *im2col*, used in traditional convolutions to convert convolution operations from mathematical form into an efficient programmable form.

[8]Adapted from https://towardsdatascience.com



| (i, j) | count | v_in | v_out |
|---|---|---|---|
| (-1, -1) | 0 | 0 | 5 |
| (0, -1) | 0 | 0 | 4 |
| | 1 | 1 | 7 |
| (+1, -1) | 0 | 0 | 3 |
| | 1 | 1 | 6 |
| (-1, 0) | 0 | 0 | 2 |
| (0, 0) | 0 | 0 | 1 |
| | 1 | 1 | 5 |
| (+1, 0) | 0 | 0 | 0 |
| | 1 | 1 | 4 |
| (0, +1) | 0 | 1 | 2 |
| (+1, +1) | 0 | 1 | 1 |

Figure 3.20: Rulebook atomic operation.

In figure 3.20 [9], $P_{in}$ is the input list that describes the spatial location of the active input sites. $P_{out}$ holds the spatial location of the active output sites generated by convolving the active input sites with the convolution kernel. The right-most table is the rulebook. The first column holds the convolution kernel element index. The second column is a counter and index of how many atomic operations this involves. The third and fourth columns are the input and output hash table keys involved in this atomic operation, respectively.

As seen in fig 3.21[10], when the convolution operation is carried out, the convolution kernel 3x3 is centered at the spatial location (1, 1). In this case, the active input site P1 would generate an active output site at spatial location (0, 0), because the index (+1, 0) of the convolution kernel interacts with the active input site P1. The calculation of the output when the described convolution kernel index interacts with the active input site is an atomic operation and is recoded into the rulebook.

[9] Adapted from https://towardsdatascience.com

[10] Source: https://medium.com

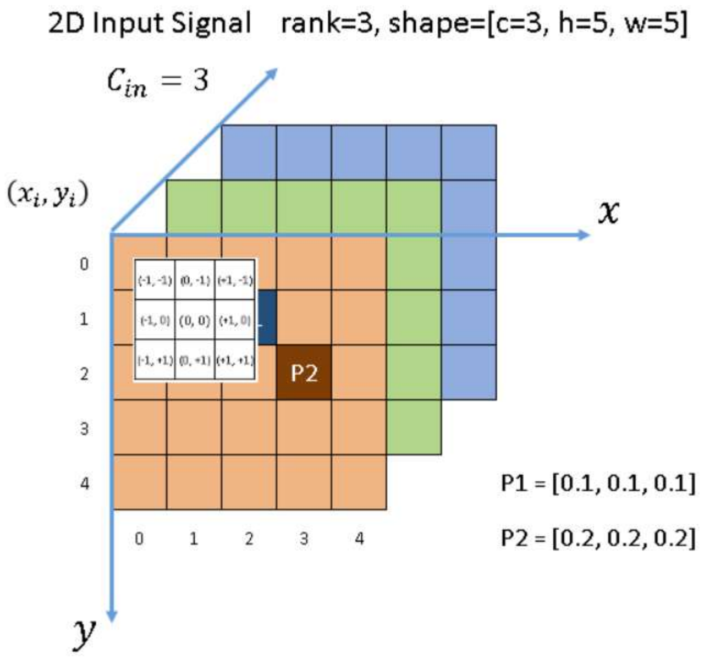
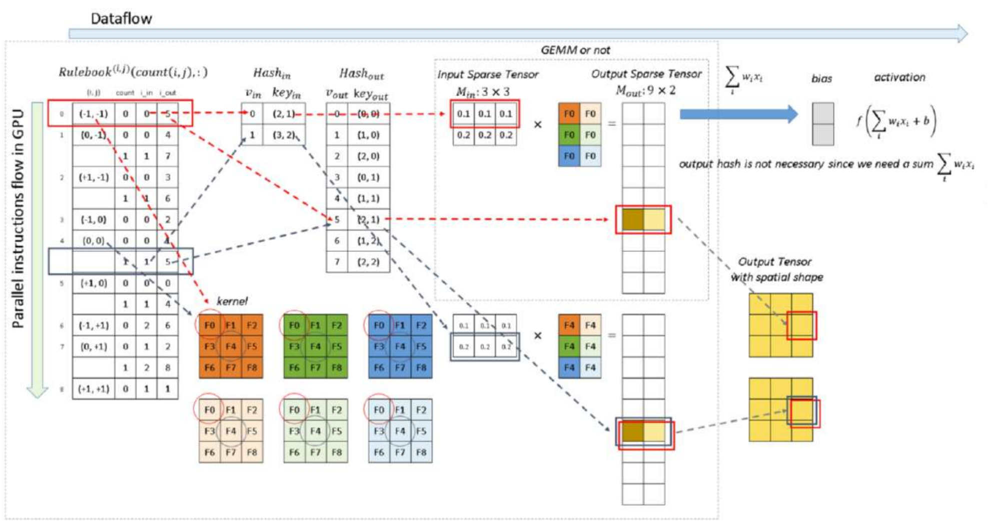


Figure 3.21: Rulebook atomic operation.

Finally the follow image represents all the processing steps involved in sparse convolutions layers for an input 3x5x5 and 2x3x3 kernel filters.



Dataflow
GEMM or not
Parallel instructions flow in GPU
Rulebook
Hash_in
Hash_out
Input Sparse Tensor
Output Sparse Tensor
bias
activation
kernel
output hash is not necessary since we need a sum
Output Tensor with spatial shape

Figure 3.22: Sparse Convolution Computation pipeline.

In the figure 3.22[11], the red and blue arrows represent two different calculation examples. The initial atomic action for the kernel element is represented by the red arrow (-1, -1). According to the rulebook, this atomic operation has input from P1 with position (2, 1) and output with position (2, 1). Similarly, the blue arrow denotes another atomic operation with

[11]Source: https://towardsdatascience.com

the same output location. The results of the red arrow and blue arrow instances may be combined together.

## 3.7 Spiking Neural Networks

SNN attempts to resemble a biological neural network as precisely as possible by updating the membrane potential based on the memorized state of each neuron and current inputs. The model fires a binary spike whenever the membrane potential surpasses a defined threshold. Contrarily, the ANN neurons communicate through continuous activation. In addition, the SNN networks have the advantage of carrying both spatial and temporal information compared to almost all ANNs, with the exception of RNN and LSTM networks.

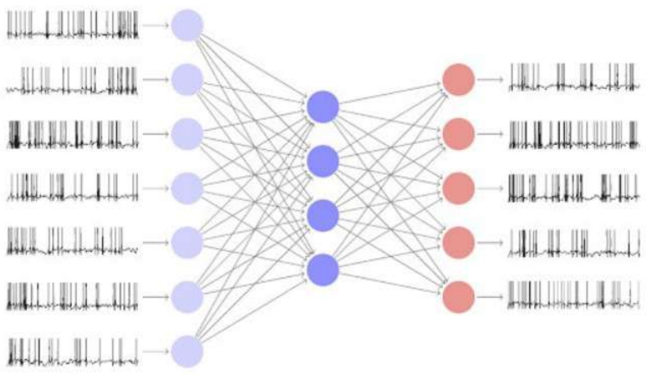


Figure 3.23: An example of input and output data from SNN.

Before diving into the details of the SNN model that we use to solve the object detection task, it will be necessary to highlight some differences between SNNs and ANNs, in particular the LSTM, because in one of the methods that is also followed.

According to the authors of [62] SNNs and LSTM have similarities and differences. The similarities are present in the way both models pass the spatio-temporal dynamics through the hidden state. The internal membrane potential of each spiking neuron affects the neuronal state of the next timestep, which acts like the forget gate of LSTM. The differences are in terms of forward propagation where the SNNs have less intermediate states because of the gates of LSTM, figure 3.25; there are several algorithms to use SNN networks (unsupervised learning, ANN-to-SNN conversion and supervised learning) whereas the LSTM are usually trained by supervised-manner; SNNs only have self-recurrence within each neuron, while LSTMs have cross-recurrence among neurons, figure 3.24; The recurrent weights of SNNs are determined by the leakage factor of the membrane potential, while in LSTM they are trainable parameters.

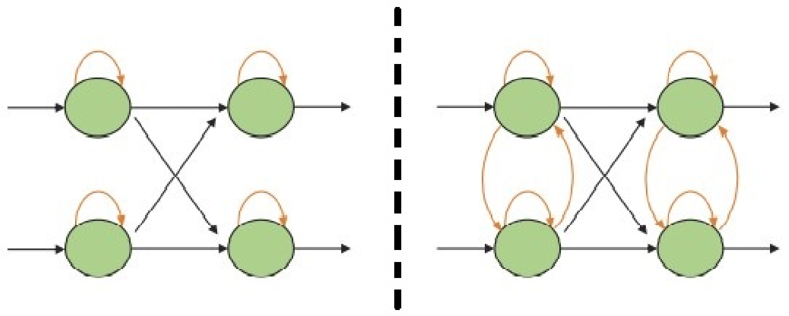


Figure 3.24: Connection pattern of (Left) SNNs and (Right) LSTM. (Adapted from [62]).

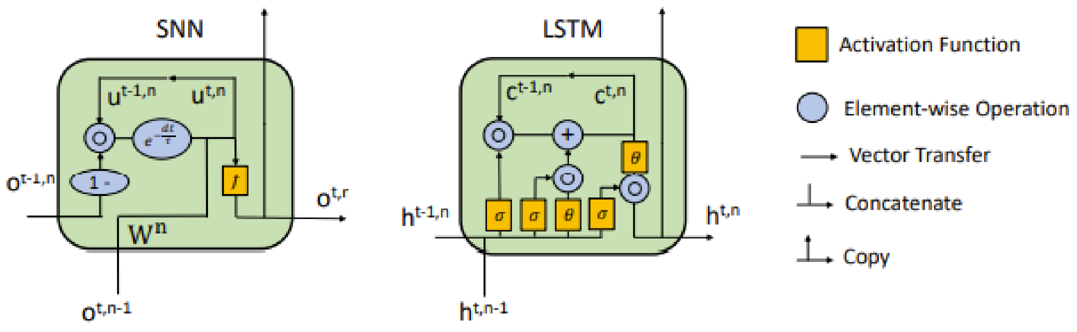


Figure 3.25: Neuron unit of (Left) SNNs and (Right) LSTM. (Adapted from [62]).

The application of the studied SNN to the task of object detection presents two new methods for converting DNN-to-SNN architecture without losing performance. These methods are channel-wise normalization and imbalanced threshold. The first method allows for a better firing rate over all neurons, while the second takes into account both positive and negative activations for firing spikes. The next subsections explain the differences between the new and old methods.

### 3.7.1 Channel-wise vs layer-norm data-based normalization

For good performance in an SNN, it is important to guarantee the transmission of a spike convoy without any information loss from input magnitude. However, given a fixed number of timesteps, information loss can occur from under or over-activation in the neurons. For example, if a threshold voltage $V_{th}$ is too small or the input is too large, the membrane potential $V_{mem}$ will most likely exceed $V_{th}$ and the neuron will generate spikes regardless of the input value, leading to over-activation. Conversely, if the $V_{th}$ is extremely large or the input is small, the $V_{mem}$ will require a long time to reach $V_{th}$ resulting in a low firing rate and under-activation.

For sufficient and balanced activation of the neuron, both weights and threshold voltage have to be chosen carefully to prevent under or over-activation of the neurons. For that purpose, various data-based normalizations have been proposed. The algorithm of layer-wise normalization (or layer-norm) is one of the most famous methods. This method normalizes weights in a specific layer using the maximum activation from the corresponding layer. This is achieved by running a DNN over the training data, assuming that both the training and test datasets have similar distributions. To increase the robustness to outliers and to ensure a sufficient firing rate of neurons, an extended version of Layer-norm can be used by normalizing the activations using the 99.9th percentile of maximum activation. Layer-norm can be calculated as:

$$\tilde{w}^l = w^l \cdot \frac{\lambda^{l-1}}{\lambda^l} \qquad (3.3) \qquad\qquad \tilde{b}^l = \frac{b^l}{\lambda^l} \qquad (3.4)$$

where $w$, $\lambda$, and $b$ are the weights, the maximum activations calculated from the training dataset, and bias in layer $l$, respectively.

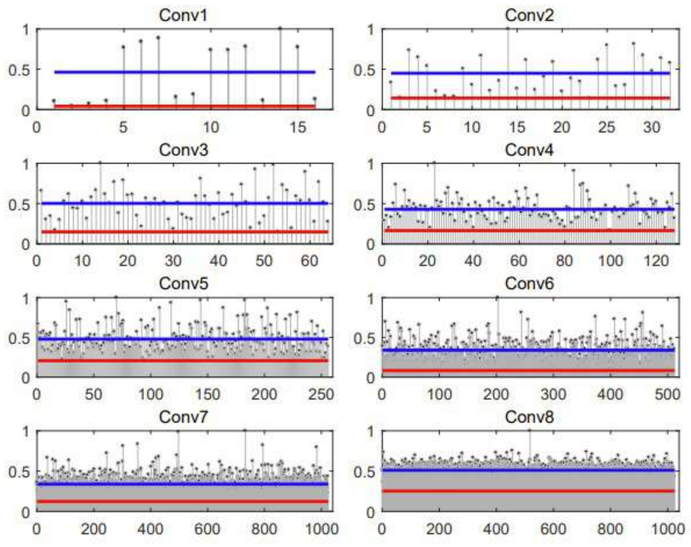


Figure 3.26: Normalized maximum activation via layer-wise normalization in each channel for eight convolutional layers in Tiny YOLO. Blue and red lines indicate the average and minimum of the normalized activations, respectively. (Taken from [20]).

In the image above, for a specific convolutional layer from Tiny-Yolo, the deviation of the normalized activations on each channel is relatively large. We can see that the normalized maximum activation is close to 1 for some channels and close to 0 for others. This clearly shows that Layer-norm yields under-activations in numerous channels that had relatively small activation values prior to the normalization. This situation is extremely problematic in solving regression problems in deep SNNs.

To solve this problem, the authors of [20] have proposed a new method for normalizing the weights from the network. The method is called "channel-wise normalization" (or channel-norm) and he normalizes the weights by the maximum possible activation, using the 99.9th percentile in a channel-wise manner instead of a layer-wise manner. It can be expressed as:

$$\tilde{w}^l_{i,j} = w^l_{i,j} \cdot \frac{\lambda^{l-1}_i}{\lambda^l_j} \tag{3.5}$$

$$\tilde{b}^l_j = \frac{b^l_j}{\lambda^l_j} \tag{3.6}$$

where $i$ and $j$ are indices of channels. Weights $w$ in a layer $l$ are normalized by maximum activation $\lambda^l_j$ in each channel. In the following layer, the normalized activations must be multiplied by $\lambda^{l-1}_i$ to obtain the original activation prior to the normalization.

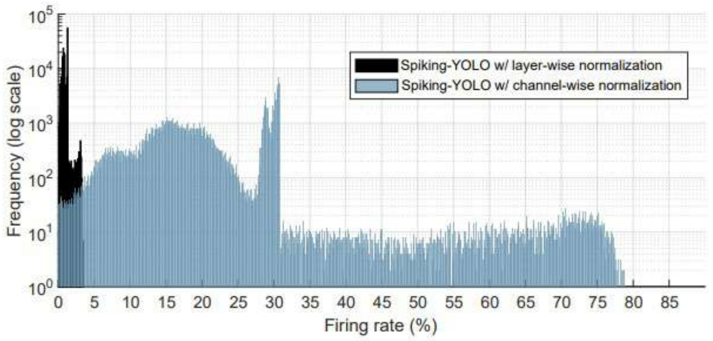


Figure 3.27: Firing rate distribution for layer-norm and channel-norm on channel 2 of Conv1 layer from Tiny YOLO (Taken from [20]).

In figure 3.27 we can see that channel-norm produces numerous neurons to generate a firing rate of up to 80% compared to layer-norm, where most of the neurons generate a firing rate in the range between 0% and 3.5%. This is a clear indication that channel-norm eliminates extremely small activations and that more neurons are producing a higher yet proper firing rate, which leads to accurate information transmission in a short period of time.

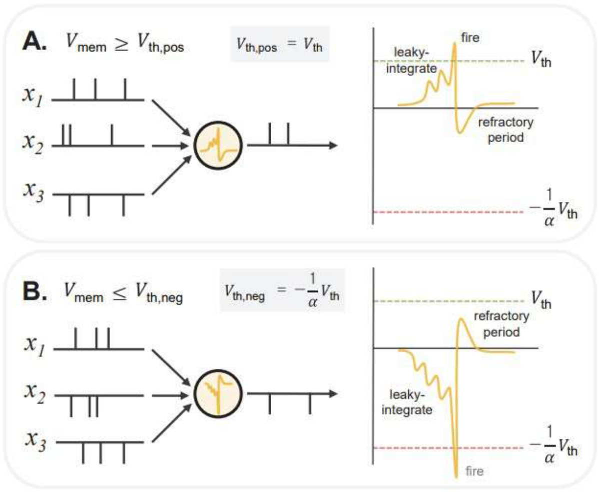
### 3.7.2 Signed neuron featuring imbalanced threshold

This comes from the limitations of implementing Leaky-ReLu in SNNs. Most of the previous DNN-to-SNN conversion methods have focused on converting integrate-and-fire neurons to ReLU while completely neglecting the leakage term in the negative region of the activation function. However, the negative activations in Tiny YOLO are around 51%. Currently, various DNNs use leaky-ReLU as an activation function, yet an accurate and efficient method of implementing leaky-ReLU in an SNN domain has not been proposed. The authors say that this activate function can be implemented in SNNs by simply multiplying negative activations by the slope $\alpha$ in addition to a second $V_{th}$.

For this purpose, they have introduced a signed neuron featuring an imbalanced threshold that can interpret both positive and negative activations and, in addiction, compensates the leakage term in the negative regions of leaky-ReLU. Furthermore, by introducing different threshold voltage for the negative region $V_{th,neg}$ the proposed method can also retain the discrete characteristic of spikes. The equation 3.7 represents how the firing rate is computed for both positive and negative activations.

$$fire(V_{mem}) = \begin{cases} 1 & \text{if } V_{mem} \geq V_{th,pos}(V_{th}) \\ -1 & \text{if } V_{mem} \leq V_{th,neg}(-\frac{1}{\alpha} \cdot V_{th}) \\ 0 & \text{otherwise, no firing.} \end{cases} \tag{3.7}$$

As shown in figure 3.28 , if the slope is $\alpha = 0.1$ then the threshold voltage responsible for a positive activation $V_{th,pos}$ is 1V , and that for a negative activation, $V_{th,neg}$, is -10V. Therefore $V_{mem}$ must be integrated ten times more to generate a spike for the negative activations in leaky-ReLU.


A. $V_{mem} \geq V_{th,pos}$
$V_{th,pos} = V_{th}$
$x_1$
$x_2$
$x_3$
fire
leaky-integrate
$V_{th}$
refractory period
$-\frac{1}{\alpha}V_{th}$
B. $V_{mem} \leq V_{th,neg}$
$V_{th,neg} = -\frac{1}{\alpha}V_{th}$
$x_1$
$x_2$
$x_3$
$V_{th}$
refractory period
leaky-integrate
$-\frac{1}{\alpha}V_{th}$
fire
Figure 3.28: Overview of proposed signed neuron featuring imbalanced threshold; two possible cases for a spiking neuron (Taken from [20]).

## 3.8 SORT

### Kalman Filter - Estimation Model

The KF is used as an estimation model that predicts the displacement based on state space given by $x = \{u, v, s, r, \dot{u}, \dot{v}, \dot{s}\}$ where $(u, v)$ are the bounding box center of each detection, $(r)$ aspect ratios, $(s)$ area scale, and $(\dot{u}, \dot{v}, \dot{s})$ the respective velocities in image coordinates.

This method uses an independent linear constant velocity model to estimate the frame-level displacements of each identified vehicle. The first four values are direct observations of the detected object state, with the aspect ratio of the target's bounding box assumed to be constant. Furthermore, the detected bounding box is used to update the target state, with the velocity components ideally solved using a Kalman filter framework [63]. If no detection is associated with the target, its state is simply projected using the linear velocity model without adjustment.

### Data Association

Predicting the new location for each target bounding box geometry in the current frame is the first step in solving the assignment problem. Then, using the IoU distance between each detection and all predicted bounding boxes from the existing targets, a cost matrix is generated. The Hungarian algorithm provides the most effective solution to this. Additionally, if the detection to target overlap is less than $IOU_{min}$, assignments will be rejected.

### Track Handling

The track handling resumes to create and destroy unique identities when the vehicles enter and leave the image. Any detection with an overlap less than $IoU_{min}$ in relation to the untracked vehicles forms the basis for the creation of trackers. Then, the tracking is initialised using the values of the detection bounding box, with the velocity set to zero. Since, at this point, the velocity is unobserved, the covariance matrix is initialized with large uncertainty. Additionally, the new tracker must successfully complete a probationary period during which the target must be associated with detections in order to gather sufficient proof to avoid false positives.

On the other hand, the tracks are terminated if they are not detected for a certain period. Moreover, to add efficiency to the method, an earlier deletion of tracker is performed. This prevents an unbounded growth in the number of trackers and localisation errors caused by predictions over long durations without corrections from the detector.

## 3.9 DeepSORT

### Kalman Filter - Estimation Model

The KF is used as estimation model that predicts the displacement based on state space given by $x = \{u, v, \lambda, h, \dot{x}, \dot{y}, \dot{\lambda}, \dot{h}\}$, where (u,v) are the center of bounding box, $\lambda$ is the aspect ration, h is height, and their respective velocities in image coordinates. The algorithm is used as a standard KF with constant velocity motion and a linear observation model, where the $(u, v, \lambda, h)$ measures are taken as observations.

### Assignment Problem

The formalization of this problem is done by integrating information about the motion and appearance of each tracker and detection. For the motion information is used the mahalanobis distance between the prediction state of the KF $(y_i)$ and the detection $(d_j)$, i.e, measure how many standard deviations the detection is away from the mean track location.

$$d^1_{(i,j)} = (d_j - y_i)^T S_i^{-1}(d_j - y_i) \tag{3.8}$$

Furthermore, a gate is computed to see if the association is admissible, gives 1, or not, gives 0.

$$b_{i,j}^{(1)} = \mathbb{1} \cdot \left[ d_{(i,j)}^{(1)} \leq t^{(1)} \right] \tag{3.9}$$

Since the observation model has four measurement the corresponding mahalanobis threshold is $t^{(1)} = 9.4877$.

The information of appearance leads to integrate another metric into the assignment problem. For each bounding box detection $d_j$ an appearance descriptor $r_j$ is computed and kept the last $L_k$ descriptors in a gallery $R_k = \{r_k^{(i)}\}_{k=1}^{L_k}$ for each track k. Then, the second metric computes the smallest cosine distance between the track i and j detection:

$$d_{(i,j)}^{(2)} = \min \left\{ 1 - r_j^T r_k^{(i)} | r_k^{(i)} \in R_i \right\} \tag{3.10}$$

To see if the association is admissible, another gate is introduced:

$$b_{i,j}^{(2)} = \mathbb{1} \cdot \left[ d_{(i,j)}^{(2)} \leq t^{(2)} \right] \tag{3.11}$$

where $t^{(2)} = 0.2$ represents the maximum distance between descriptors.

These combined metrics complement each other by serving different aspects of the assignment problem. The cosine distance considers appearance information that is useful for recovering track identities after long-term occlusions. On the other hand, for short-term predictions, the mahalanobis distance is based on motion to give information about the possible location. Therefore, the combination of both metrics is used to solve the association problem with a cost function, a sum of 3.8 and 3.10:

$$C_{i,j} = d_{(i,j)}^{(1)} + d_{(i,j)}^{(2)} \tag{3.12}$$

if the association is admissible, i.e., if is within the gate region of both metrics:

$$b_{i,j} = \prod_{m=1}^{2} b_{i,j}^{(m)} \tag{3.13}$$

### Matching Cascade

The assignment problem is solved by a cascade method leading to a series of sub-problems instead of solving measurement-to-track associations as a global assignment. This approach was introduced due to a weakness in the mahalanobis distance for objects that are occluded for a longer period. For these situations, the KF predictions have an increased uncertainty associated leading to a minor distance given by the mahalanobis distance, equation 3.8. Consequently, when two tracks compete for the same detection, the track with greater uncertainty is favored. This is an undesirable behavior as it can lead to increased track fragmentation and unstable tracks. Therefore, a matching cascade is introduced, giving priority to more frequently seen objects.

**Algorithm 1** Matching Cascade Algorithm

```
Require: Track indices T = { 1, ..., N }, Detection indices D = { 1, ...,M }, Maximum age $A_{max}$
 1: Compute cost matrix C = $[c_{i,j}]$ using 3.12
 2: Compute gate matrix B = $[b_{i,j}]$ using 3.13
 3: Initialize set of matches M $\leftarrow \emptyset$
 4: Initialize set of unmatched detections U $\leftarrow$ D
 5: for n $\in \{1, ..., A_{max}\}$ do
 6:     Select tracks by age $T_n \leftarrow \{i \in T | a_i = n\}$
 7:     $[x_{i,j}] \leftarrow$ min_cost_matching(C,$T_n$,U)
 8:     M $\leftarrow$ M $\cup \{(i,j) \mid b_{i,j} \cdot x_{i,j} > 0\}$
 9:     U $\leftarrow$ U $\setminus \{j \mid \sum_i b_{i,j} \cdot x_{i,j} > 0\}$
10: end for
11: Return: M,U
```

Algorithm 1 outlines the cascade matching algorithm. As input is provided a set of tracks T and detections D indices as well as the maximum age $A_{max}$. In lines 1 and 2, the association cost matrix and the matrix of admissible associations are computed. Then a cycle over the track age $n$ is made to solve the linear assignment problem for tracks of increasing age. In line 6 is selected the subset of tracks $T_n$ that have not been associated with a detection in the last $n$ frames. In line 7 is solve the linear assignment between tracks in $T_n$ and unmatched detections U. In the next two lines, 8 and 9, the set of matches and unmatched detections are updated.

Furthermore, after the matching stage and following the logic of [2], the metric of IoU is run on the set of unconfirmed and unmatched tracks of age n=1, helping to account for the sudden appearance changes, e.g., due to partial occlusion with static scene geometry, and to increase robustness against erroneous initialization.

### Track Handling

The number of frames since the previous successful measurement association is counted for each track. In other words, after each KF prediction, the counter is increased, and it is reinitialized whenever there is an association between the track and detection. On the other hand, if the trackers exceed the maximum frame's age, $A_{max}$, without any association, they are classified as having left the scene and, consequently, eliminated from the track records. Furthermore, for a new tracker hypothesis to be created, there has to be a detection without any association to an existent tracker. If during the next $X$ frames an association occurs, a new tracker is assembled. In opposition, the tracker hypothesis is deleted.

# 4 Developed Work

The core of this chapter is to illustrate the methods used to accomplish the final purpose of tracking multiple vehicles using neuromorphic cameras. The first section has a detailed description of the detection models implemented, while the second one covers the tracking approaches.

## 4.1 Object Detection Methods

The choice of following the method [38] briefly presented in the chapter 2 with two typologies, CNN and SSCN, resides in the fact of being able to extract spatial and temporal features. The temporal features are the key to this method because of the nature of event data. Because it contains only relative change information, an event-based object detector must keep a memory of the past because of the fact that when the apparent motion of an object is zero, it does not generate events anymore.

Following the same reason, i.e trying to develop typologies that take advantages of event-data in terms of being able to extract both spatial and temporal features, a spiking-Yolo is also analyzed.

### 4.1.1 RED Neural Network

The input network is formed by one dense representation constructed by the algorithms described in the sections 3.2.1 and 3.2.2. A batch of events is aggregated during an interval of time $\triangle t$, i.e for each network input a set of events are accumulated during a time of $\triangle t$ to build a frame representation.

The first block of the architecture uses SE layers to extract relevant spatial features, taking into account that the authors of [38] and [42] claim they perform better results. It is critical to note that this block can be replaced with any other backbone; in fact, in this implementation, a ResNet will be tested as the backbone.

Furthermore, ConvLSTM layers [43] are utilised to add an internal state memory, i.e., the ability to remember the presence of objects even when they stop generating events. After the data passes towards the SE and ConvLSTM layers, the output of each ConvLSTM is fed into a bounding box regression head of a SSD network with different feature resolution to extract objects within a large range of scales.

Table 4.1: Architecture of Neural Network for extracting features

| | **Layer Type** | **Kernel Size** | **Channels Out** | **Stride** |
|---|---|---|---|---|
| Layer1 | BNConvReLU | 7 | 32 | 2 |
| Layer2 | Squeeze-Excitation | 3 | 64 | 2 |
| Layer3 | Squeeze-Excitation | 3 | 64 | 2 |
| Layer4 | ConvLSTM | 3 | 256 | 2 |
| Layer5 | ConvLSTM | 3 | 256 | 2 |
| Layer6 | ConvLSTM | 3 | 256 | 2 |
| Layer7 | ConvLSTM | 3 | 256 | 2 |
| Layer8 | ConvLSTM | 3 | 256 | 2 |

Table 4.2: Architecture of the Squeeze-Excitation Layers [42].

| | **Layer Type** | **Kernel Size** | **Channels Out** | **Stride** |
|---|---|---|---|---|
| Layer1 | BNConvReLU | 3 | C | 2 |
| Layer2 | BNConvReLU | 3 | 64 | 2 |
| Layer3 | BNConv | 3 | 64 | 2 |
| Layer4 | GlobalAvgeragePooling | MxN | 64 | MxN |
| Layer5 | DenseReLU | 1 | 16 | 1 |
| Layer6 | DenseSigmoid | 1 | 64 | 1 |
| Layer7 | Elementwise-Multiplication | 1 | 64 | 1 |
| Layer8 | Skip-Sum | 1 | 64 | 1 |

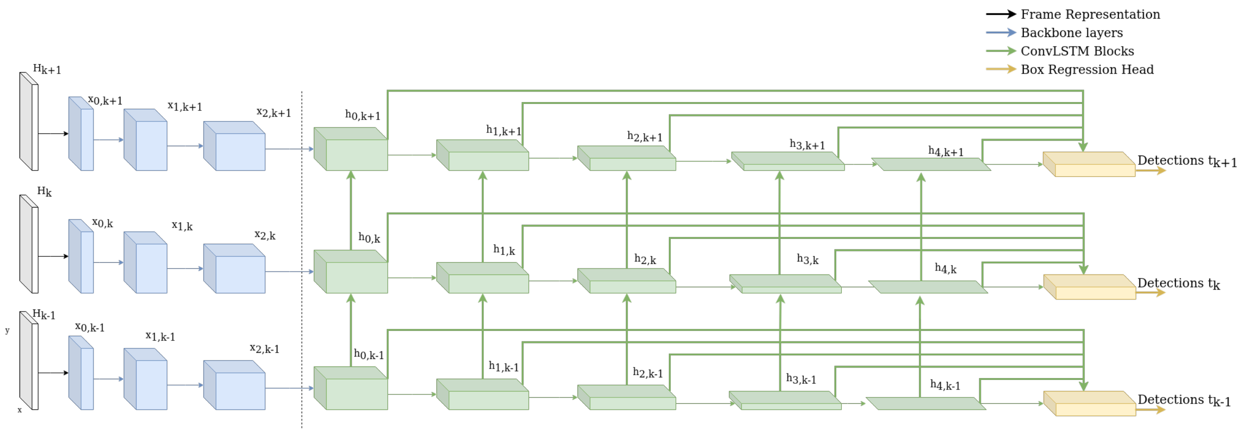


Figure 4.1: Our RED architecture.

### Loss Funtions

Once again, following the work [38] the parameteres of the network are trained using a loss function composed by a regression term $L_r$ for the coordinate boxes and a classification term $L_c$ for the class, as typically done for object detection. For regression, a smooth L1 loss, $L_s$, with $\beta = 1$ is employed, and a Softmax focal loss for classification. For a set of $J$ ground-truth bounding boxes at each example, a tensor $B^*$ with size $(J \cdot R, D)$ represents the encoded coordinates, where $R$ is the number of default bounding boxes of SSD network and $D$ the number of prediction boxes per cell. Let $(B, p)$ be the output of the regression head where $p$ represents the class probability of all predictions. Subsequently, the regression and classification terms are calculated by:

$$L_s = mean(L) = L = \begin{cases} 0.5 \cdot \frac{(B-B^*)^2}{\beta} & \text{if } |B - B^*| < \beta \\ |B - B^*| - 0.5 \cdot \beta & \text{otherwise} \end{cases} \tag{4.1}$$

$$L_r = L_s(B, B^*) \tag{4.2}$$

$$L_c = -(1 - p_l)^{\lambda} \cdot \log p_l. \tag{4.3}$$

where $p_l$ is the probability of the correct class and $\lambda$ is equal to 2.

### Dual Regression Head and Temporal Consistency Loss

With the objective of improving the consistency of detection it was introduced an additional dual regression head and auxiliary loss, $L_t$, to predict the bounding box one time-step into the future. In practice, given an input tensor computed in the time interval between

$t_{k-1}$ and $t_k$ the two regression heads will output $B_k$ and $B'_{k+1}$ while trying to match the ground truth of $B^*_k$ and $B^*_{k+1}$ during the train of the network, respectively.

The auxiliary loss between $B'_{k+1}$ and $B^*_{k+1}$ constrains the output $B'_{k+1}$ of the second head to be close to the predictions of $B_k$ of the first head at the next time step. However, since the two heads are independent, they could converge to different solutions. Therefore, to regularize training is added another loss term explicitly imposing $B'_k$ to be closer to $B_k$. The logic of this method can be seen in 5.1.

$$L_t = L_s(B'_{k+1}, B^*_{k+1}) + L_s(B'_k, B_k) \tag{4.4}$$

The final loss for training is $L = L_t + L_r + L_c$ and is minimized by an Adam optimizer and backpropagation.

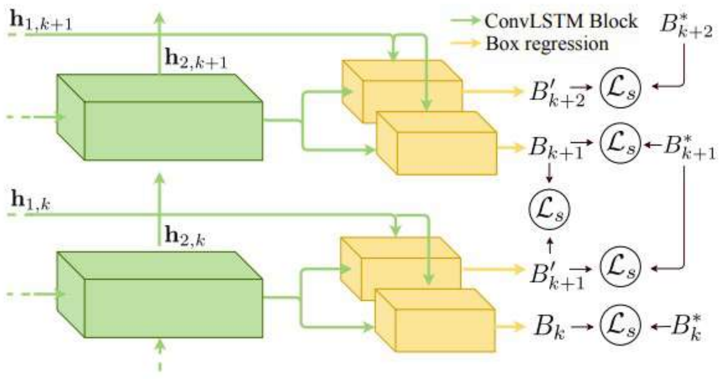


Figure 4.2: Illustrative example of Dual regression head. (Taken from [38])

### 4.1.2 Sparse Convolution Model

Sparse Convolution Networks have shown great potential when applied to LIDAR data processing. On the other hand, the CNN pipeline is very robust and efficient for 2D image processing. However, applying this method to sparse data reveals a certain inefficiency because many multiplications are applied into an area of zeros. With these considerations in mind, we have reconstructed our RED network with sparse convolutions based on the previous work done in [33] and [32].

The input network has two phases. First, a frame representation is constructed by the Time Surface, 3.2.1, and then we create two lists, where one has the coordinates of none-zero pixels and the other keeps the feature value of each coordinate. All the representations are done by accumulating a batch of events over a temporal instant $\triangle t$ of 50ms. The architecture of this network compared to the previous one is similar. The changes are related to the extraction of spatial features. Instead of using SE layers, we changed to simple sparse convolutions, and for extracting temporal features, we just decreased the number of ConvLSTM from 5 to 3. For obtaining the bounding boxes, we continue using the SSD network that is fed by the output of each ConvLSTM.

Table 4.3: Architecture of Sparse Neural Network for extracting features

| | **Layer Type** | **Kernel Size** | **Channels Out** | **Stride** |
|---|---|---|---|---|
| Layer1 | BNConvReLU | 7 | 32 | 2 |
| Layer2 | BNConvReLU | 3 | 64 | 2 |
| Layer3 | BNConvReLU | 3 | 64 | 2 |
| Layer4 | ConvLSTM | 3 | 256 | 2 |
| Layer5 | ConvLSTM | 3 | 256 | 2 |
| Layer6 | ConvLSTM | 3 | 256 | 2 |

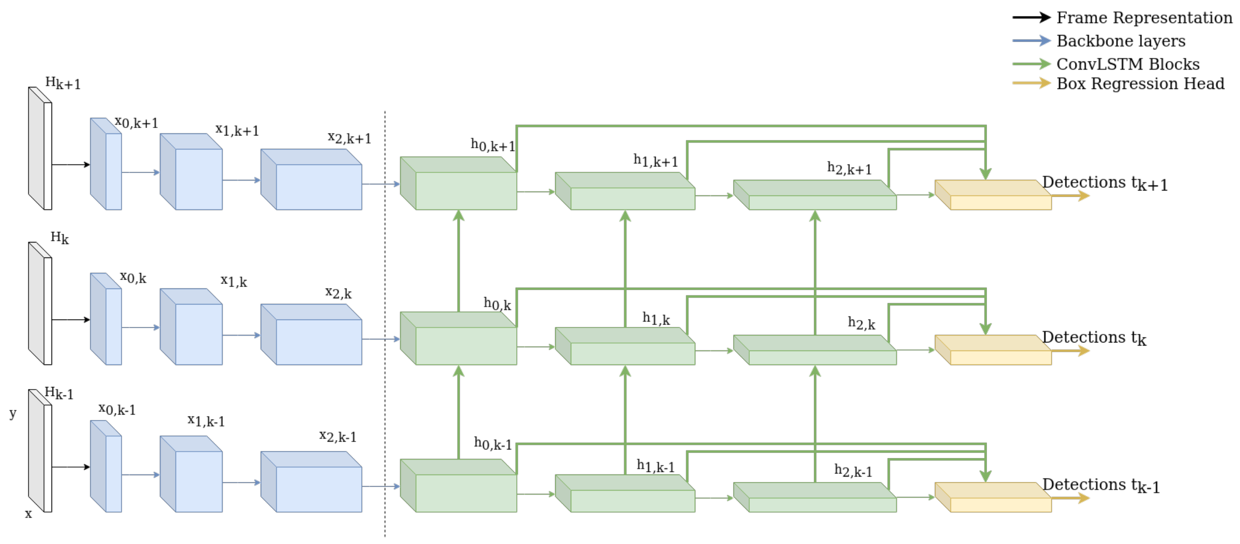


Figure 4.3: Sparse RED architecture.

### 4.1.3 Yolov3 as Spiking Neural Network

In order to explore the sparse nature of the data, we will evaluate a new typology of neural networks called spiking neural networks. The architecture will be based on the YOLOv3 [64] and tiny-YOLOv3 [65] networks.

Following the work of [20], we started by training the tiny-yolov3 CNN network and only during inference the network is converted to an SNN typology. The conversion method was explained in the 3.7 section.

Furthermore, for comparison with other architectures and implementations of SNNs, we validate the work of [21], where they explore four different CNN backbones (SqueezeNet, VGG, MobileNet, and DenseNet) feeding a SSD bounding box regression head.

## 4.2 Tracking Methods

This section describes the main methodologies for achieving the task of multi-vehicle tracking with neuromorphic cameras. In this study, we evaluated, as a baseline, a classic multi-vehicle tracking method called SORT. In order to validate a more robust method (proven in RGB multi-pedestrian tracking), we evaluated the extension of the previous method, DeepSort. It is important to note that the last method implicitly led to the creation of the Re-identification model on event data, which is also explained in this section.

### 4.2.1 SORT

The SORT [2] method was chosen to be the initial approach since it has proven to be very successful, straightforward, and a thoroughly investigated standard way for multi-object tracking. Initially, the algorithm was developed and tested for tracking pedestrians, but, as the authors said, it can be generalized for other classes since it only depends on the detection model. This technique essentially employs a single hypothesis tracking methodology with a typical Kalman Filter, frame-by-frame data association utilizing bounding box overlap as an association metric, and the Hungarian method.

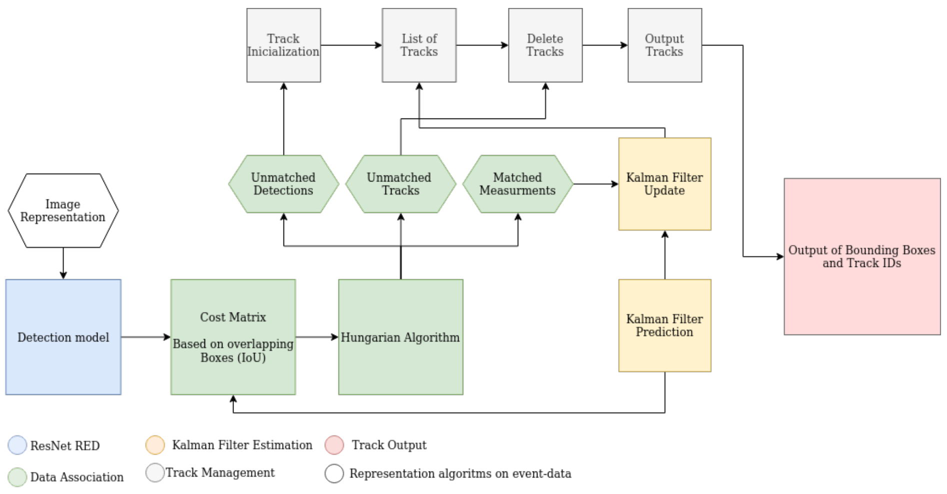


Figure 4.4: SORT diagram.

### 4.2.2 DeepSORT

The method is an extension of the previous method described. As a result, the single hypothesis tracking is likewise based on a conventional Kalman Filter and the Hungarian method for data association. The difference between this approach and the previous one is the use of an additional deep learning model to extract an appearance descriptor and the use of two new metrics for data association, mahalanobis distance and cosine distance. Furthermore, the assignment model was changed to prioritize more commonly seen vehicles via the matching cascade.

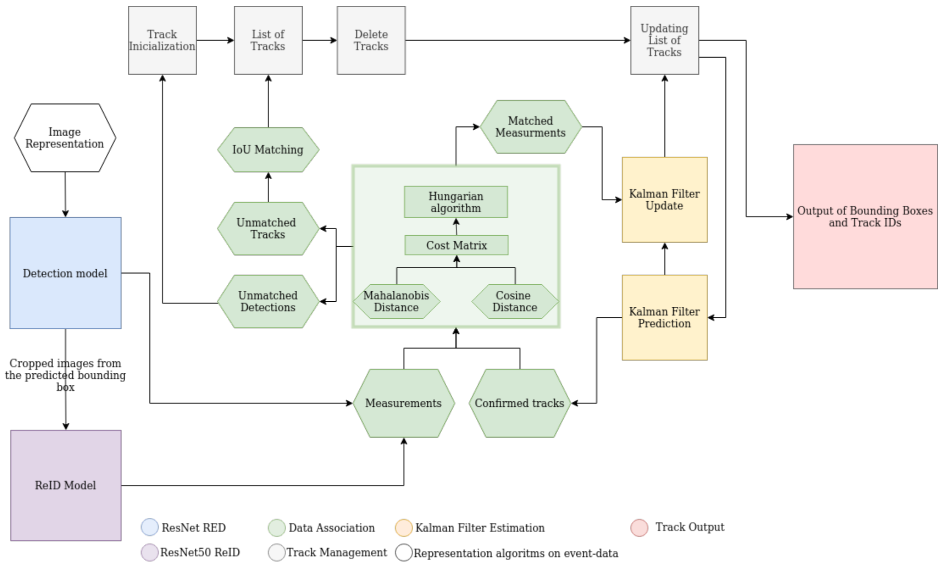


Figure 4.5: DeepSort diagram.

### 4.2.3 Re-identification model

To extract appearance information from event-based patterns, the work of [4] was followed. This work is the first to perform a re-identification with event-camera information and a deep neural network. Taking this into account, the neural network is fed with two different representations: event representation (3.2.2 or 3.2.1), containing meaningful spatial and time details, and their transformation to polar coordinates, which carries more distinct edge patterns. These two representations are relative to the bounding box labels extracted by the detection model. Moreover, for the spatial polar representation [66] is assumed that the event representation can be related to a vehicle contour sketch.

The model for extracting an appearence descriptor is ResNet-50 [67] and, as already been said, the network receives two input representations, event-frame $X_e$ and its polar transformation $X_p$ of size $128 \times 256$. The feature map of the last residual block feeds into the global average pooling layer and a linear layer (FC+BN+ReLU) to compute a 256-D feature embedding. To predict the identity of input vehicle during training, the two extracted feature vectors ($F_e$ and $F_p$) are submit into a classifier, which consists of a fully-connected layer. Further, two classifications loss are applied (event loss and polar loss) to facilitate the learning feature $F_e$ from event-frame and feature $F_p$ from edge patterns in polar image respectively. Therefore, the overall loss of our model is:

$$L_T = \alpha L_e + \beta L_p \tag{4.5}$$

where $L_e$ and $L_p$ represent the cross-entropy loss of ID classification of feature $F_e$ and $F_p$ respectively, both $\alpha$ and $\beta$ are coefficients to control the contribution of the each loss.

During the inference time, the classifier is pulled off and to extract the vehicle re-id descriptor, the network encodes the image by feature $F_{Concat}$ as:

$$F_{Concat} = Concat(F_e, F_p) \tag{4.6}$$

where $F_e$ and $F_p$ are the learned features from each representations.

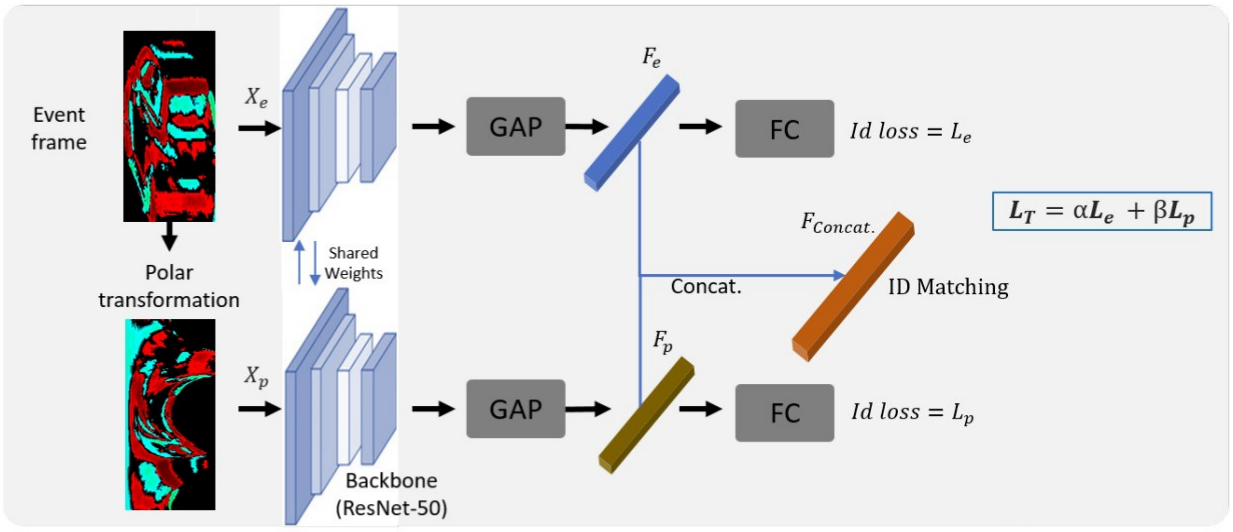


Figure 4.6: Re-identification model architecture. (Adapted from [4])

# 5 Results

This chapter is divided into several sections. The first one presents the used datasets and the information relative to the v2e software [7] to simulate event data. The second section shows the metrics for evaluating our work. The following sections provide the detailed evaluation and analysis of the detection and tracking tasks explained in the previous chapters. The performance of the developed model for object detection in the three datasets (GEN1 Automotive, UA-DETRAC, and BMVC) will be the primary focus of the study. Then, we will discuss the tracking performance over the UA-DETRAC dataset.

All the experiments were conducted and evaluated with the Ryzen Threadripper PRO 3955WX 16-Cores CPU with 3.8 GHz and a RTX3090 GPU.

## 5.1 Datasets for Detection and Tracking

To perform this dissertation work and its different associated tasks, three datasets were used: GEN1 Automotive Detection Dataset [5], UA-DETRAC [68] and BMVC-2014.

The Prophesee GEN1 Automotive Detection Dataset for neuromorphic cameras was created for training and evaluating detection frameworks. Still, the authors believe that it can benefit other tasks like optical flow, structure from motion, and tracking. The dataset was recorded using a PROPHESEE GEN1 sensor 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 labelling manually, by humans, at a lower frequency in order to maximize the variety of objects aspects and scenes. After this process, the dataset contains 39 hours, with 228,123 cars and 27,658 pedestrians bounding boxes of open road and various driving scenarios ranging from urban, highway, suburbs, and countryside scenes, as well as different weather and illumination conditions.

The UA-DETRAC benchmark consists of 100 video sequences, which are selected from over 10 hours of videos at 24 different locations in China, representing various common traffic types and conditions, including urban highways, traffic crossings, and junctions captured with an unstable camera. The videos are recorded at 25 fps with a spatial resolution of $960 \times 540$ pixels. There are over 140 000 manually annotated frames in the UA-DETRAC benchmark, corresponding to 8250 different vehicles, for a total of 1.21 million labeled bounding boxes of objects. The UA-DETRAC benchmark is split into two parts: training and testing, with 60 and 40 sequences, respectively. The training videos are recorded in distinct locations compared to the testing videos, but with similar traffic and attributes. The dataset was used to train and evaluate three stages of this work: detection, re-identification, and tracking.

Since the neurmorphic cameras only capture movement in the scene and the two other datasets contain events related to non-vehicles, the BMVC dataset was chosen because of its simpler attributes for evaluating the developed detection framework. This dataset consists of 51 video sequences, captured with a stable camera in urban highway traffic with different weather and illumination conditions. In total, there are more than 240 thousand frames with at least one vehicle, giving an immense vehicle bounding boxes for training and evaluation captured through an yolov5 network. More specifically, the dataset was divided into to 70% for training and 15% for validation and testing.

It is important to note that these two last datasets contain video sequences captured with a standard RGB camera. So, due to the nature of this work being event-based, it employed the v2e [7] python tool for converting the data to neuromorphic event-data. Furthermore, it was imperative to create a preprocessing to eliminate the bounding boxes related to static and occluded vehicles because of not generating events during the conversion.

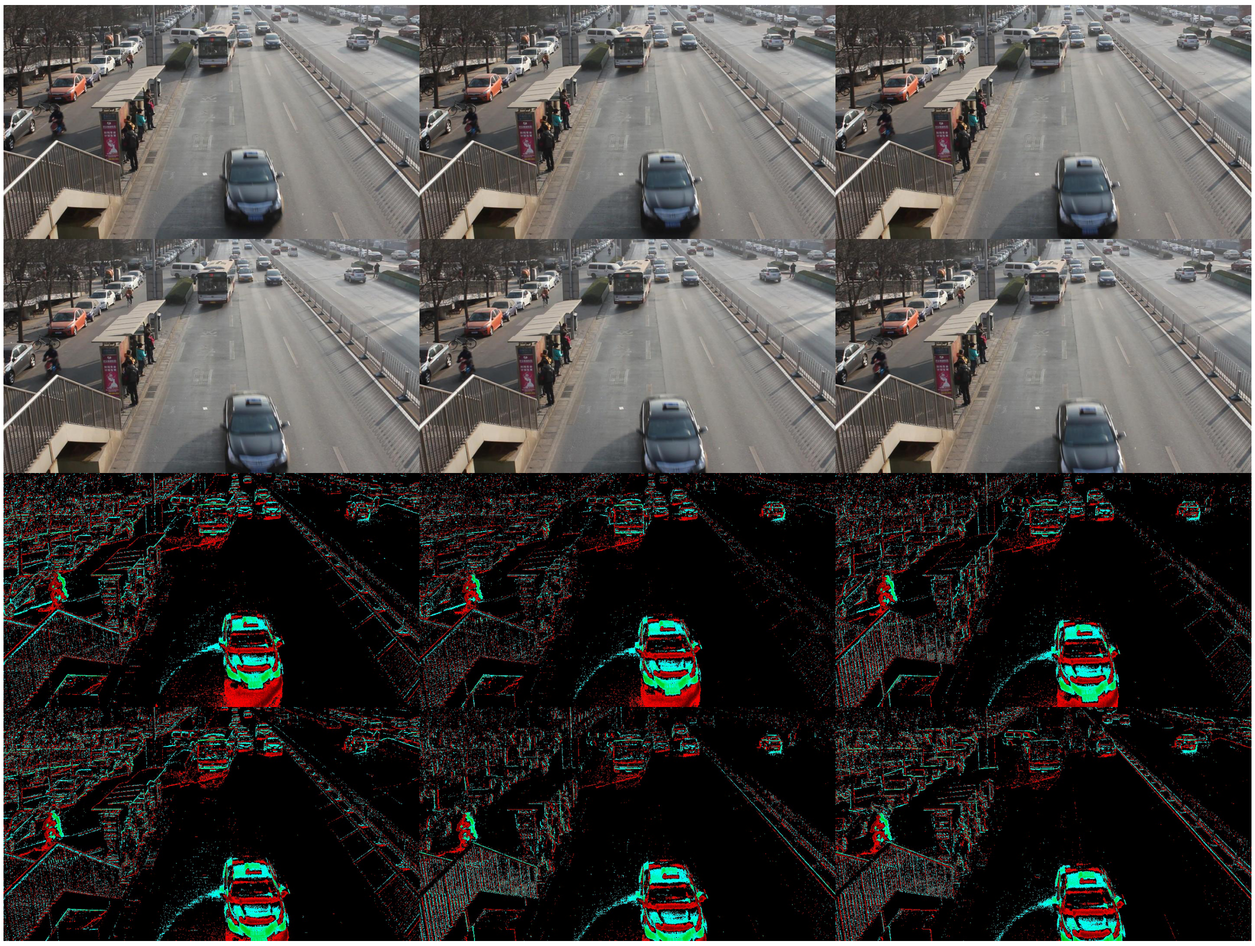


Figure 5.1: Transformation example of RGB data to event-data through v2e software.

The v2e tool has input parameters that we have to select. However the creators provide two sets of default parameters, “Clean” and “Noise”. As the name says, the “Noise” set yields to a transformation of RGB-data to event-data with added noise to the rest of the events generated by scene movement. In contrast, the “Clean” mode generates events only for the objects that move in the scene.

Therefore, to choose the right parameters for our study, we analyzed the behavior of each one through the experiments presented in the table 5.1. The “Dvs exposure” and “frame rate” parameters were not changed to maintain the characteristics of the RGB-data and, at the same time, not affect the ground truth for training the models.

Table 5.1: Different parameters used to study the v2e software tool.

| | | Parameters | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Mode | Experiments | Dvs exposure | Frame rate | Event thres. | Sigma | Cutoff | Leaky rate hz | Noise rate hz |
| Clean | (a) | 40ms | 25 | 0.2 | 0.02 | 0 | 0 | 0 |
| Noisy | (b) | 40ms | 25 | 0.2 | 0.05 | 30 | 0.1 | 5 |
| Custom | (c) | 40ms | 25 | 1 | 0.02 | 0 | 0 | 0 |
| Custom | (d) | 40ms | 25 | 0.05 | 0.02 | 0 | 0 | 0 |
| Custom | (e) | 40ms | 25 | 0.2 | 0.25 | 0 | 0 | 0 |
| Custom | (f) | 40ms | 25 | 0.2 | 0.02 | 0 | 10 | 0 |
| Custom | (g) | 40ms | 25 | 0.2 | 0.02 | 0 | 0 | 10 |

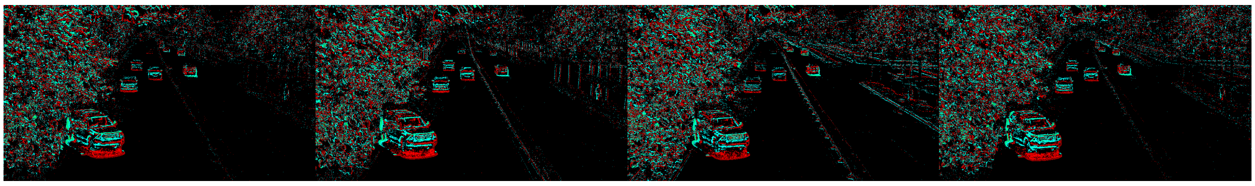


Figure 5.2: Experiment (a).

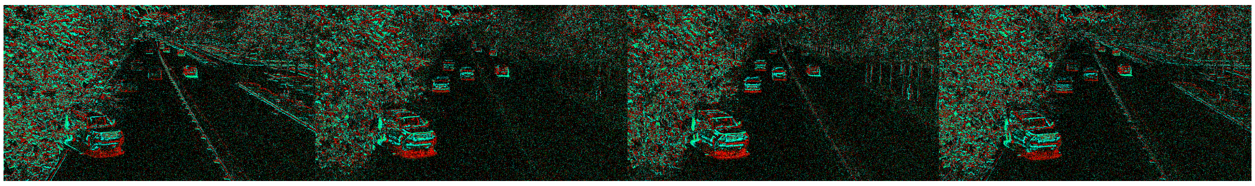


Figure 5.3: Experiment (b).

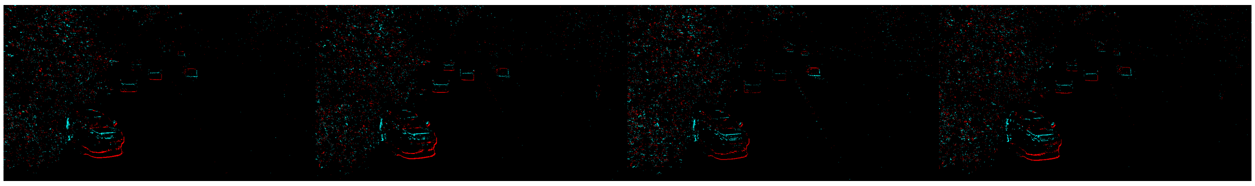


Figure 5.4: Experiment (c).

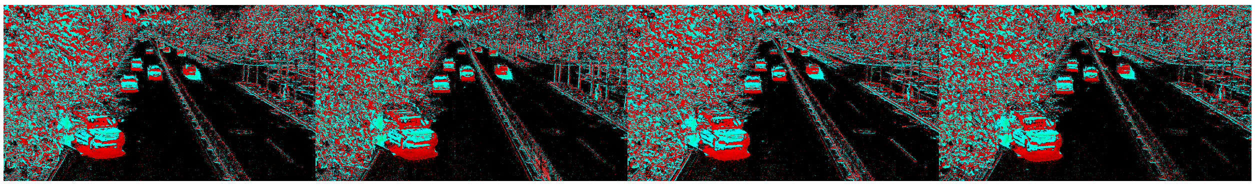


Figure 5.5: Experiment (d).

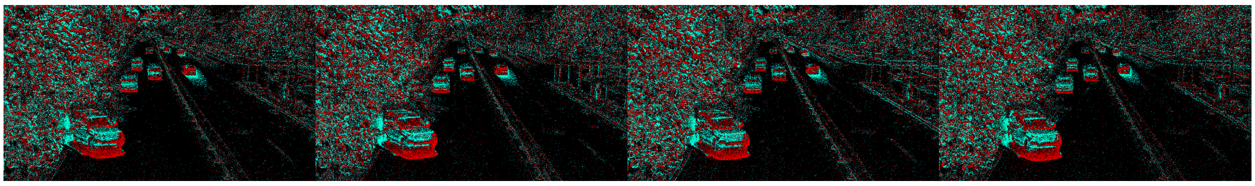


Figure 5.6: Experiment (e).

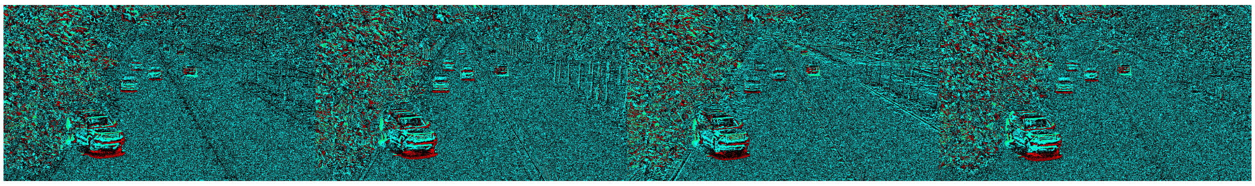


Figure 5.7: Experiment (f).

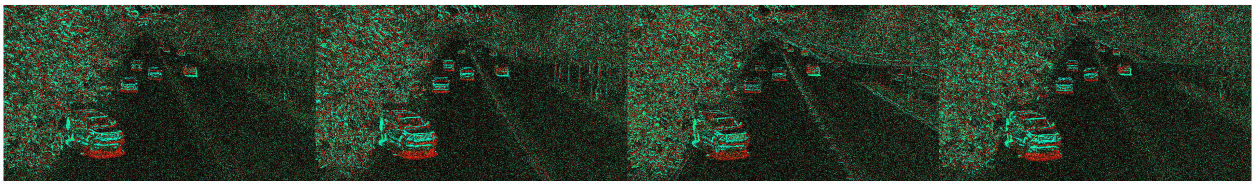


Figure 5.8: Experiment (g).

The “Event thres.” and “Sigma” parameters control the number of events generated from each movement object. If we decrease the “Event thres.” more events will be generated and vice-versa. The “Sigma” parameter works in the opposite way.

Furthermore, the “Leaky rate hz” and “Noise rate hz” are responsible for how much noise is added to the event-data. The “Leaky rate hz” causes noise on each pixel in the image, while the “Noise rate hz” adds noise to the darkest parts of the image. The added noise is proportional to the value of each parameter, i.e., if we increase the values, more noise is added and vice-versa.

Taking into account the experiments, we chose to create two event datasets from the BMCV and DETRAC datasets using the "Clean" mode because it results in a better presentation of object classes.

## 5.2 Evaluated Metrics

This section contains the metrics for the three different modalities: object detection, re-identification, and multi-object tracking. Despite the fact that these procedures are built for frame-based vision sensors, they are still appropriate for quantitative evaluation of our approaches since we aggregate across sliding windows to construct a representation in a 2D image.

## 5.2.1 Object Detection Metrics

To evaluate the performance of object detection, we selected the first six metrics out of the twelve COCO metrics. These metrics are based on the calculus of IoU given by the overlap between the ground-truth and predicted boxes divided by the area of union between them.

- mAP $\rightarrow$ mAP is averaged over 10 IoU thresholds (0.50 to 0.95 with a 0.05 step) and is the primary challenge metric;
- $mAP_{50}$ $\rightarrow$ mAP considering the TP with at least an IoU greater or equal to 0.50;
- $mAP_{75}$ $\rightarrow$ mAP considering the TP with at least an IoU greater or equal to 0.75;
- $mAP_{small}$ $\rightarrow$ mAP for small objects that covers area less than $32^2$;
- $mAP_{medium}$ $\rightarrow$ mAP for medium objects that covers area greater than $32^2$ but less than $96^2$;
- $mAP_{large}$ $\rightarrow$ mAP for large objects that covers area greater than $96^2$;

## 5.2.2 Re-identification Metrics

The performance of the re-identification model is evaluated by two metrics: Mean Average Precision and Cumulative Matching Characteristics.

The mean average precision for evaluating the capacity of re-identification is based on the ability to sort the gallery set according to the evaluated query image. The below image shows that the correct retrieval of the gallery set has more weight for the first images and less for the last to calculate the AP. To sum up, AP penalises the models that are not able to sort the gallery with TP leading the set.

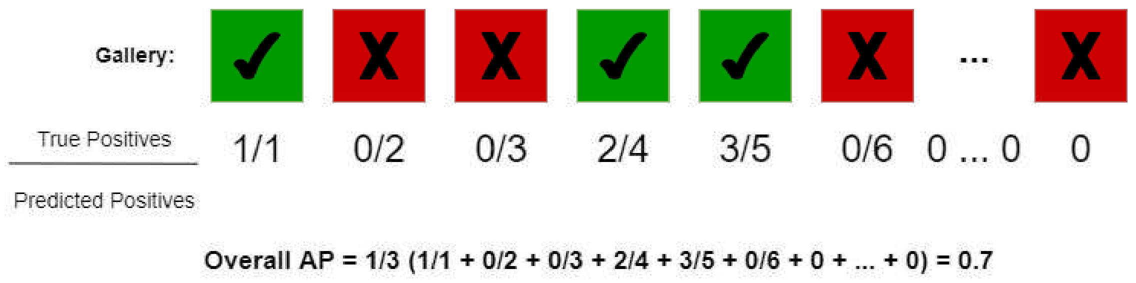


Figure 5.9: Calculation of AP for the sorted gallery set.

Finally, the mAP is simply the mean of all the AP for the existing queries:

$$mAP = \frac{\sum_{i=1}^{N} AP_i}{N} \tag{5.1}$$

The CMC curves are the most popular evaluation metrics for re-identification methods. Consider a simple single-gallery-shot setting, where each gallery identity has only one instance. For each query, an algorithm will rank all the gallery samples according to their distances to the query, from small to large, and the CMC top-k accuracy is:

$$CMC_K = \begin{cases} 1 & \text{if top-k ranked gallery samples contain the query identity} \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

### 5.2.3 MOT Metrics

To evaluate the efficiency and performance of the tracking algorithm, we use several metrics from the *MOT challenge.* Such as MOTP(↑), MOTA(↑), Recall(↑), Precision(↑), Mostly tracked(↑), Partially tracked(↑), Mostly lost(↓), number FP(↓), number of missed objects(↓), number of identification switches(↓), number of fragmentations during the tracking(↓). Evaluation measures with ↑, higher scores denote better performance. As opposite evaluation metrics with ↓, lower scores denote better performance.

Table 5.2: Evaluated metrics for multi-object tracking.

| **Metric** | **Description** |
|---|---|
| MOTP (↑) | Measures the accuracy of localization of detection boxes. |
| MOTA (↑) | Measures the overall accuracy of both the tracker and detection. |
| Recall (Rec ↑) | Number of detections over number of objects. |
| Precision (Prc ↑) | Number of detected objects over sum of detected and false positives. |
| Mostly tracked (MT ↑) | Number of objects tracked for at least 80 percent of lifespan. |
| Partially tracked (PT ↑) | Number of objects tracked between 20 and 80 percent of lifespan. |
| Mostly lost (ML ↓) | Number of objects tracked less than 20 percent of lifespan. |
| Num FP (FP ↓) | Total number of false positives. |
| Num misses (FN ↓) | Total number of misses. |
| Num switches ($ID_SW$ ↓) | Total number of track switches. |
| Num fragmentations (FM ↓) | Total number of switches from tracked to not tracked. |

## 5.3 Detection Results

This section contains the results of the developed detection framework for the three types of datasets as explained in section 5.1. Moreover, as the detection framework was inspired by the work of the same company that created the dataset, the subsection 5.3.1 has more experiments. The next two subsections evaluate different datasets and conditions of captured event data.

### 5.3.1 GEN1 Detection Dataset

For the GEN1 dataset, we will present the results over various experiences so we can demonstrate the effect and behaviour of the present network modules during the training and inference phase. The experiences are: the effect of the number of ConvLSTM, the effect of memory, the module of dual regression head, the effect of using different backbones and representations, and training with different sequence sizes.

All the experiments were trained with ADAM optimizer, initial learning rate $1 \times 10^{-3}$ with a exponential decay rate of 0.98 over 200 epochs and a batch size of 32.

#### Number of ConvLSTM

As described in 4.1.1, each convLSTM of the network was fed a SSD with the feature map. Therefore, the table below has the results of using a different number of ConvLSTM with different feature map sizes in order to detect a high range of object sizes.

Table 5.3: Contribution of different number of LSTM

| | 1 ConvLSTM | 2 ConvLSTM | 3 ConvLSTM | 4 ConvLSTM | 5 ConvLSTM |
|---|---|---|---|---|---|
| mAP | 0.19 | 0.22 | 0.22 | 0.23 | 0.24 |
| mAP_50 | 0.41 | 0.45 | 0.44 | 0.47 | 0.48 |
| mAP_75 | 0.16 | 0.20 | 0.20 | 0.22 | 0.23 |
| mAP_small | 0.16 | 0.17 | 0.17 | 0.18 | 0.19 |
| mAP_medium | 0.23 | 0.26 | 0.25 | 0.27 | 0.28 |
| mAP_large | 0.11 | 0.14 | 0.16 | 0.17 | 0.17 |

Table 5.4: Number of priors during each train.

| | 1 ConvLSTM | 2 ConvLSTM | 3 ConvLSTM | 4 ConvLSTM | 5 ConvLSTM |
|---|---|---|---|---|---|
| number of priors | 1024 | 1408 | 1504 | 1528 | 1532 |

As expected, increasing the number of ConvLSTM leads to better performance in all metrics. As described in 3.5, the purpose of feeding the SSD regression head with different feature map sizes is to let the smaller maps detect the larger objects and vice-versa. Therefore, along the network, we feed the SSD with smaller and smaller feature maps. The introduction of more ConvLSTM leads to an improving detection for small, medium and large objects up to 3%, 5% and 6% respectively. The metric for the larger objects was the one that improved the most, which proves what was said above.

### Impact of memory

Due to the nature of data, the non-existence of objects movement leads to the non-existence of intensity changes, i.e., the camera is incapable of generating event data for the object. This factor was the principal reason for creating a model capable of retaining memory of past occurrences. The table below shows that the use of memory has an impact of between 12% to 30% over all metrics. Looking at the metric of larger objects, we see the importance that the memory mechanism has. These are the objects that undergo the greatest change in terms of the events generated as they stop.

Table 5.5: The impact of memory

| | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| With memory | 0.24 | 0.48 | 0.23 | 0.19 | 0.28 | 0.17 |
| Without memory | 0.06 | 0.18 | 0.02 | 0.07 | 0.07 | 0.00 |

### Dual Regression Head

This module was implemented following the work of [38]. The idea is to improve the consistency of detection by taking into account the bounding box one time-step into the future,$t_{k+1}$. In contrast to the findings shown by the authors, using this module reduces the performance of our network. This can be explained by the low frequency of ground truth over image sequences in our test set. The authors have tested a more robust dataset in terms of the frequency of labels.

Table 5.6: 1 regression head Vs 2 regression head

| | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| 1 Regression Head | 0.24 | 0.48 | 0.23 | 0.19 | 0.28 | 0.17 |
| 2 Regression Head | 0.18 | 0.38 | 0.16 | 0.14 | 0.21 | 0.12 |

### Change of Backbone

The use of a pretrained model in ImageNet is a crucial step for achieving good performance for any task. Since our backbone has not been trained on the ImageNet dataset and we lack the resources to do so, we have chosen to switch to a 10 layer from ResNet-18 model as our backbone without major changes in the final network model compared to the one we started with. This small adjustment has led to an improvement in performance of between 5% and 10% across all metrics.

Table 5.7: Application of a different backbone and the use of pretrained model.

| | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| Squeeze-Excitation Layers | 0.24 | 0.48 | 0.23 | 0.19 | 0.28 | 0.17 |
| Resnet Layers(pretrained) | 0.31 | 0.58 | 0.29 | 0.24 | 0.36 | 0.22 |

### Different Representations

The followed work has tested the framework in three different representations: histogram, time surface, and event volume. In our study, we chose to evaluate the developed framework using the two best representations reported by the authors: time surface and event volume, with the last having a superior outcome by 2%. However, as shown in the table below, we did not see a significant difference between them in our trials.

Table 5.8: Time Surface Vs Event Volume Representations

| | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| Time Surface | 0.24 | 0.48 | 0.23 | 0.19 | 0.28 | 0.17 |
| Event Volume | 0.24 | 0.47 | 0.23 | 0.19 | 0.28 | 0.19 |

### Train with different number of sequences

To train our network, it is necessary to provide a sequence of images so that the network can learn to filter important information to retain in memory. To evaluate this behavior, sequences with 5, 10, and 20 images were provided, with the smallest sequence being the one that brought the best results.

It was expected that larger sequences would bring better results, but the fact that the dataset does not have a ground-truth with great frequency we cannot guarantee that large sequences of images are correct, thus making it difficult to train the network.

Table 5.9: Train with different number of sequence images

| | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| 5 images per sequence | 0.25 | 0.49 | 0.23 | 0.20 | 0.29 | 0.17 |
| 10 images per sequence | 0.24 | 0.48 | 0.23 | 0.19 | 0.28 | 0.17 |
| 20 images per sequence | 0.19 | 0.40 | 0.18 | 0.15 | 0.23 | 0.17 |

### Class results

Here, we list the findings for each of the dataset classes. The dataset has a significant imbalance between the number of labeled cars and the number of pedestrians, as was already indicated in the methodology section 5.1, leading to a greater ease in learning the class of cars than the pedestrian class, as presented in the table below. In terms of the mAP metric, there is a difference in both studied networks of about 33%. Additionally, these results demonstrate that our architecture is capable of detecting cars in the majority of scenarios with at least an IoU larger than 0,5.

Table 5.10: Class results from RED network.

| | AP | AP_.50 | AP_.75 | AP_small | AP_medium | AP_large |
|---|---|---|---|---|---|---|
| Car | 0.41 | 0.70 | 0.44 | 0.32 | 0.46 | 0.30 |
| Pedestrian | 0.09 | 0.28 | 0.02 | 0.08 | 0.11 | 0.05 |

Table 5.11: Class results from RED-ResNet network.

| | AP | AP_.50 | AP_.75 | AP_small | AP_medium | AP_large |
|---|---|---|---|---|---|---|
| Car | 0.47 | 0.75 | 0.53 | 0.37 | 0.53 | 0.38 |
| Pedestrian | 0.14 | 0.42 | 0.05 | 0.12 | 0.19 | 0.06 |

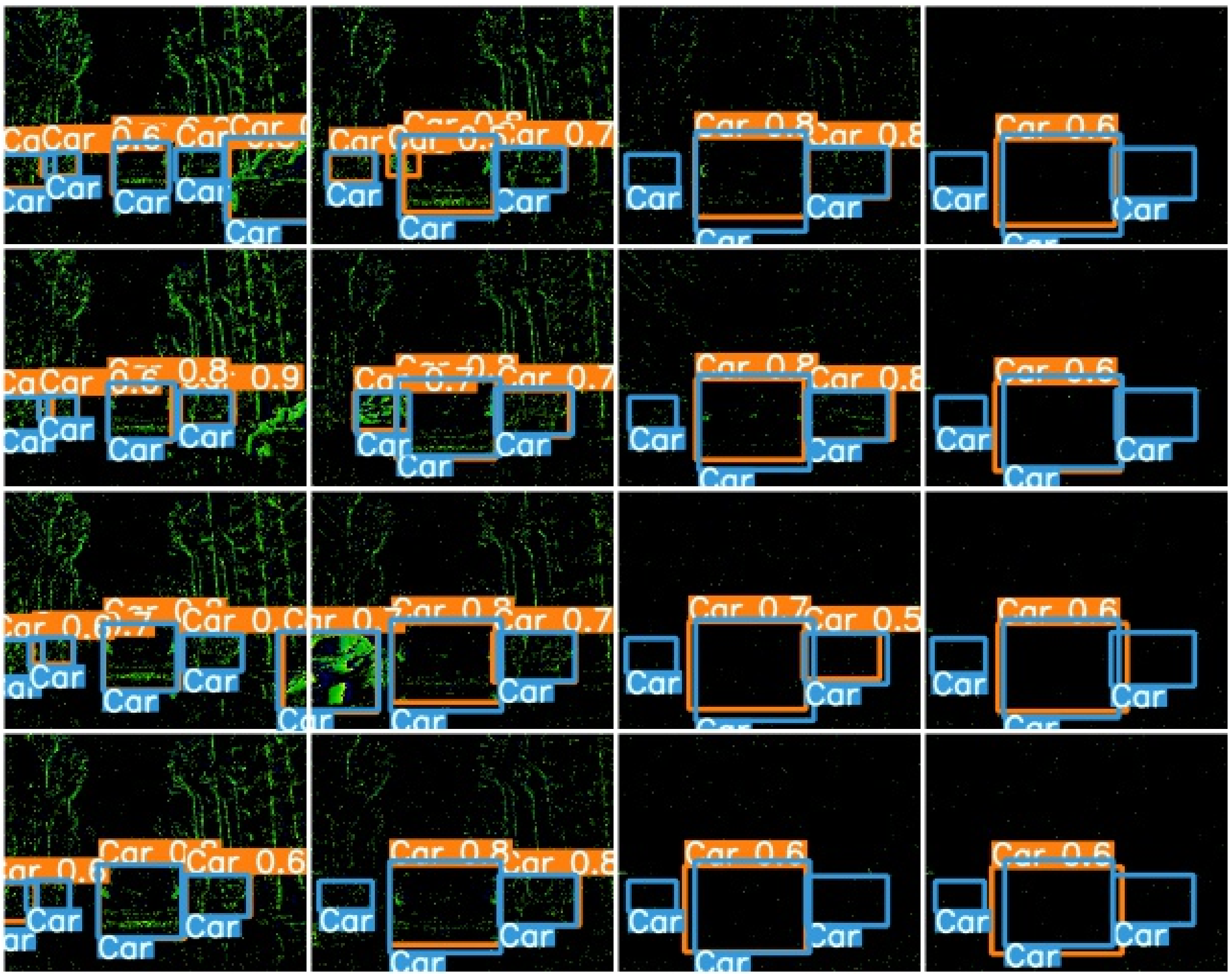


Figure 5.10: Example of ResNet-RED network output for a sequence of 16 images from the GEN1 Automotive Dataset. The blue boxes represent the ground truth, whereas the orange boxes indicate the predicted box. In these images we can see the car detection even without generating events, which proves the functioning of the memory mechanism.

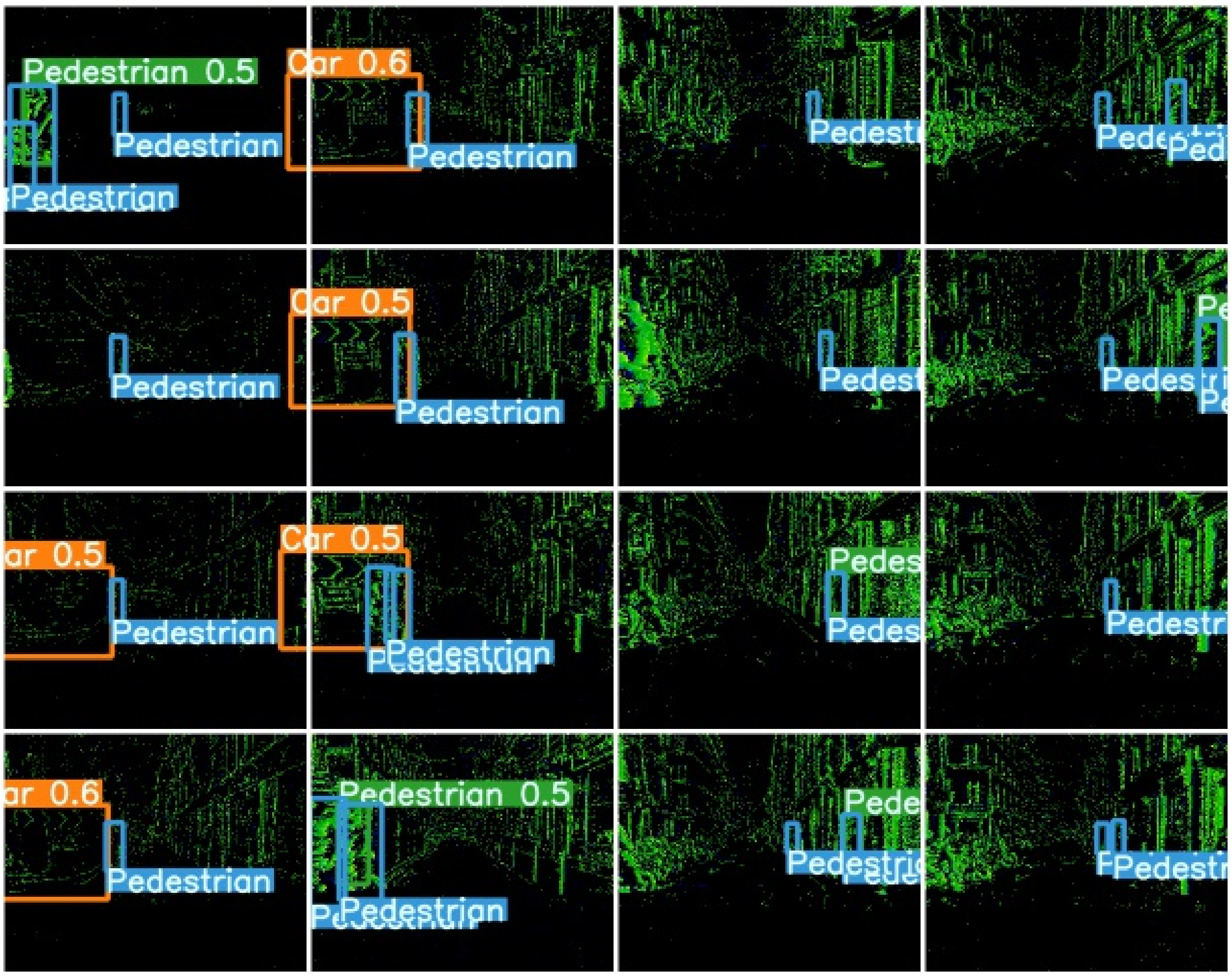


Figure 5.11: Example of ResNet-RED network output for a sequence of 16 images from the GEN1 Automotive Dataset. The blue boxes represent the ground truth, whereas the orange boxes indicate the predicted box. In these images we can see the challenge of pedestrian detection with event-data, since this class is easily camouflaged with background events, as seen in the image.

## 5.3.2 UA-DETRAC Dataset

The results for this dataset will be evaluated over different difficulties; easy, medium, and hard, as reported by the authors [6]. The video sequences contain larger variations in scale, pose, illumination, occlusion, and background clutter, which lead to the three levels of difficulty based on the recall rate of the EdgeBox method [69]. The average recall rates of these three levels are 97.0%, 85.0%, and 64.0%, respectively, with 5000 proposals per frame.

Moreover, giving the results of the last section (sec: 5.3.1) the evaluated performance will be based on RED-ResNet CNN. In addition, given the size of the images (960x540) in this dataset, the model will be evaluated over images sizes of 640x640 and 320x320 to evaluate the impact of losing some velocity of processing over performance for the future task of multi-vehicle tracking.

The experiments were trained with ADAM optimizer, initial learning rate $1 \times 10^{-3}$ with a exponential decay rate of 0.98 over 60 epochs and a batch size of 16.

Table 5.12: ResNet-RED 320x320 results over three different levels of difficulty.

| | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| Easy | 0.33 | 0.52 | 0.37 | 0.07 | 0.36 | 0.30 |
| Medium | 0.34 | 0.55 | 0.37 | 0.06 | 0.33 | 0.40 |
| Hard | 0.19 | 0.35 | 0.18 | 0.06 | 0.17 | 0.29 |
| Overall | 0.27 | 0.46 | 0.29 | 0.04 | 0.27 | 0.35 |

Table 5.13: ResNet-RED 640x640 results over three different levels of difficulty.

| | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| Easy | 0.39 | 0.60 | 0.45 | 0.16 | 0.44 | 0.33 |
| Medium | 0.40 | 0.63 | 0.46 | 0.15 | 0.41 | 0.42 |
| Hard | 0.25 | 0.44 | 0.26 | 0.13 | 0.25 | 0.33 |
| Overall | 0.34 | 0.54 | 0.38 | 0.12 | 0.35 | 0.37 |

Due to the difficulties of detecting cars using simulated event data, we feel our findings are adequate and may be used as a baseline for future event camera feature development. Furthermore, we observed that when the event-intensity for an object is low, our system cannot identify it most of the time, even despite employing memory techniques. On the other hand, it is fully capable of detecting when the event-intensity is high.

Looking at the network's computation velocity at each input size, we get 50ms per frame for 640x640 and 16.7ms for 320x320. In exchange for 7% on mAP, we get around 33% of computation velocity.

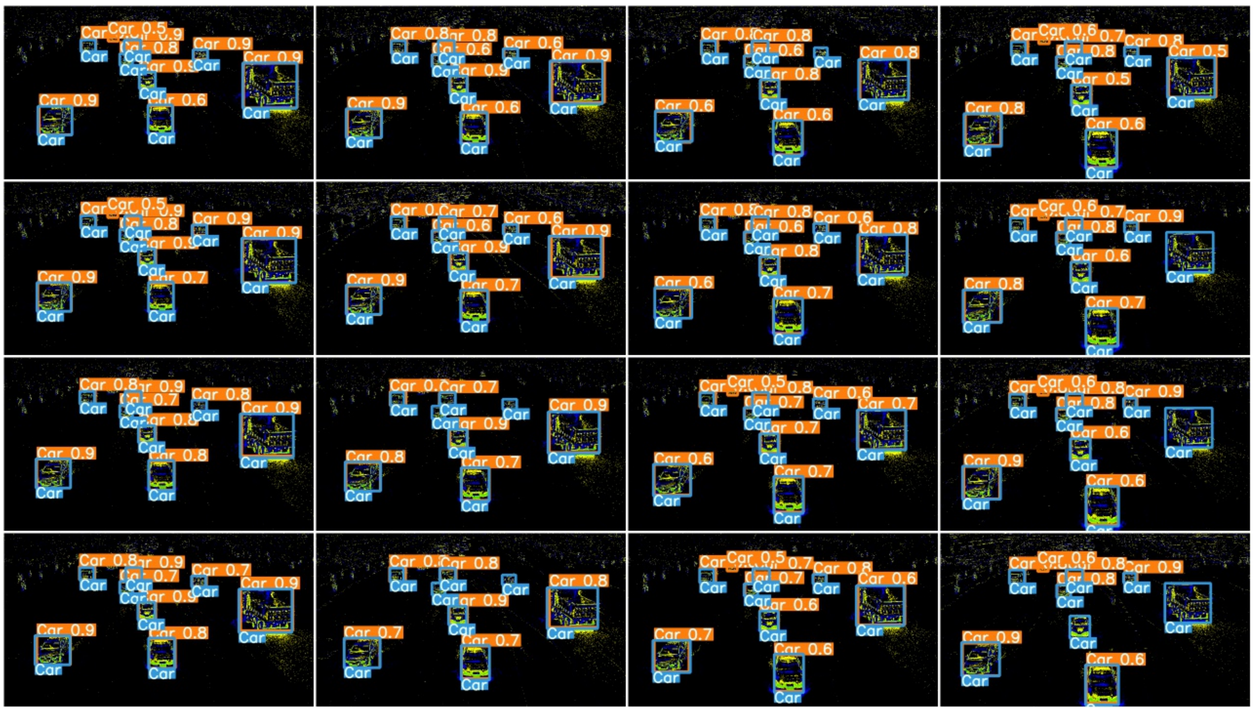


Figure 5.12: Example of ResNet-RED network output for a sequence of 16 images from the UA-DETRAC Dataset. The blue boxes represent the ground truth, whereas the orange boxes indicate the predicted box.

### 5.3.3 BMVC Dataset

The purpose for selecting a third dataset was to assess the detection network under more "easier" conditions. The dataset was captured using a static camera recording on a highway road where the number of cars is reduced and they are plainly visible in most frames. Moreover, for the same reason as in the last section, the results will be evaluated over the ResNet RED CNN with different input sizes, 640x640 and 320x320, since the original shape is 960x540.

The training follows the same parameters as in section 5.3.2.

Table 5.14: Detection results for BMVC datatet with different input sizes.

| Model | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| ResNet 320x320 | 0.44 | 0.73 | 0.46 | 0.25 | 0.66 | 0.72 |
| ResNet 640x640 | 0.55 | 0.81 | 0.62 | 0.44 | 0.72 | 0.45 |

When compared to the findings obtained in the other two datasets, we notice an improvement of roughly 17 percent for input 320x320 and 21 percent for input 640x640. Again, because this is a dataset created using software that replicates RGB visuals for data in events, the results serve as a baseline for future investigations.

Moreover, in terms of computing speed, we get 55ms per frame for an input of 640x640 and 17ms for 320x320. Once more, in exchange of 11% of mAP, we get around 31% of processing speed.

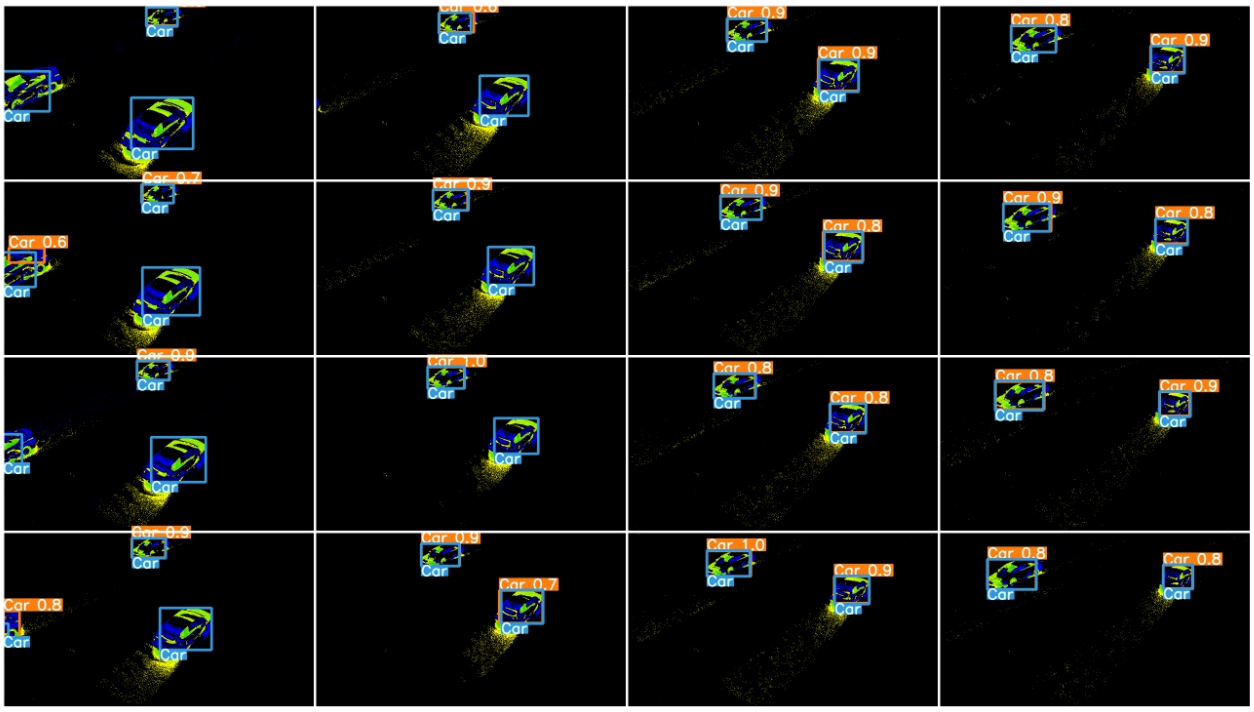


Figure 5.13: Example of ResNet-RED network output for a sequence of 16 images from the BMVC Dataset. The blue boxes represent the ground truth, whereas the orange boxes indicate the predicted box.

### 5.3.4 Sparse Neural Network

In an attempt to find a typology more adequate to the sparse nature of neuromorphic cameras, we modified the structure of the RED CNN to sparse convolutions, giving rise to the sparse-RED network. In the table below, we present the results obtained.

Table 5.15: Sparse-RED object detection results.

| | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| Sparse-RED | 0.06 | 0.15 | 0.05 | 0.06 | 0.08 | 0.04 |
| Asynet | 0.05 | 0.15 | 0.05 | 0.05 | 0.07 | 0.03 |

When we compare this network's performance against that of previous CNN-type networks, we can observe that the results are incredibly poor. However, when we examine the state of the art for this SSCN typology, we discover a work that achieves precisely the same result as our network, 15% on mAP_50, but our network uses only 2.3M of parameters versus 133M of the comparative network.

Additionally, the purpose of building this network was to increase processing speed by directing network kernels to areas with only events, non-zero pixels. But what actually occurred was a speed of 200ms on each frame, which is obviously much slower than what we had previously presented. We also recognized how challenging it was to train our network. This may be due to the fact that the authors of [32] stated, namely that it is crucial to construct a network that balances the utilization of sparse convolutions and submanifold convolutions. Only using sparse convolutions has the same impact as using regular convolutions, which reduces the sparsity of the data. On the other hand, using only submanifold convolutions, we will not pass enough information to the subsequent layers, and consequently, the network will not be able to learn the desired task.

### 5.3.5 Spiking Neural Network

Here, we show the feasibility of using SNN for object detection over three datasets and make comparisons to the best results of the proposed CNN framework. In addition, we could measure the energy efficiency and computation performance, but unfortunately, we do not have the necessary hardware.

Table 5.16: Object detection results for the three dataset.

| Dataset | mAP | mAP_50 | mAP_75 | mAP_small | mAP_medium | mAP_large |
|---|---|---|---|---|---|---|
| GEN1 Automotive | 0.22 | 0.48 | 0.18 | 0.24 | 0.26 | 0.00 |
| UA-DETRAC | 0.13 | 0.23 | 0.14 | 0.06 | 0.20 | 0.13 |
| BMVC | 0.34 | 0.51 | 0.36 | 0.23 | 0.72 | 0.42 |

Comparing the results presented above with those presented with the CNN typology, we can see that there is a drop in performance.

### 5.3.6 Final results

The table below compares our networks against state-of-the-art methods over the GEN1 dataset. These methods were carried out over different types of networks: conventional CNN, SNN, and SSCN.

Table 5.17: Comparison between our detection framework against the State-of-Art methods

| Methods | Network-Type | #Params | mAP | mAp_50 | runtime(ms) |
|---|---|---|---|---|---|
| Asynet | SSCN | 133M | 0.05 | 0.15 | 50 |
| MatrixLSTM-YOLOv3 | CNN | 65M | 0.31 | – | – |
| Events-RetinaNet | CNN | 32.8M | 0.34 | – | 18.29* |
| RED(original) | CNN | 24M | **0.40** | – | 16.70* |
| MobileNet-64+SSD | SNN | 24.26M | 0.15 | 0.35 | – |
| VGG-11+SSD | SNN | 12.64M | 0.17 | 0.38 | 200 |
| Spiking-YOLO | SNN | 9.59M | **0.22** | 0.48 | 222 |
| Dense-Net121-64+SSD | SNN | 8.2M | 0.19 | 0.37 | – |
| ResNet-RED(ours) | CNN | **2.77M** | **0.31** | 0.58 | **14.28** |
| RED(ours) | CNN | 2.23M | 0.25 | 0.49 | 13.7 |
| SparseRED(ours) | SSCN | 2.23M | 0.06 | 0.15 | 200 |

* Measure with on a i7 CPU at 2.70GHz and a GTX980 GPU.

Our CNN network, ResNet-RED, has advanced to the third best network, but it is still 9 percent behind the top result. At the same time, our network has about 10 times fewer parameters, resulting in quicker object detection performance, which is appropriate for our dissertation's end goal of multi-object tracking.

Despite the SNN and Sparse-CNN having more similarity with the nature of events generated by the neuromorphic cameras, we can see through the results that they have not achieved great results and performance, principally the Sparse-CNN. Moreover, Asynet and SparseRED are the unique networks, within our knowledge, that have evaluated this type of network for the task of object detection.

## 5.4 Tracking Results

Finally, we have reached the final objective of this dissertation, which is to evaluate multi-vehicle tracking using Neuromorphic cameras. Taking this into account, the dataset for evaluating this task is UA-DETRAC and, similar to the detection task, it is also divided into three levels of difficulty: easy, medium, and hard. Based on combinations of six object tracking methods (GOG, CEM, DCT, IHTLS, H2T, and CMOT) and four object detection methods (DPM, ACF , R-CNN, and CompACT), these levels of difficulty are determined by the average PR-MOTA score.

As described in chapter 4 for the tracking purpose, two methods will be evaluated, initially, the classical method SORT and followed by the DeepSort method. These two methods require a detection phase that is accomplished by our network, ResNet-RED, with an input size of 640x640. However, the DeepSort methodology adds an extra deep learning network for extracting a feature appearance for each detected vehicle. As a consequence of that, it will be presenting the results for the re-identification network and for the tracking methods.

## 5.4.1 Re-Identification Model

Three models were trained to evaluate the performance of the re-identification network. One model was trained using the two representations, events and polar, as input, while the other two were trained with either events or polar representation. Moreover, to evaluate the performance of this task, a small dataset was created containing 2330 query images, one for each class, and 23221 gallery images.

All networks were trained for 25 epochs using ADAM, with a learning rate of 0.01 with a step decay of 0.1 every 10 epochs and a batch size of 64.

Table 5.18: Ablation study of the loss function with different $\beta$ coefficients.

| Method | mAP | CMC 1 | CMC 5 | CMC 10 |
|---|---|---|---|---|
| L$_t$ $=\alpha L_e + \beta L_p$, $w/\alpha = 1$ | | | | |
| $\beta = 0$ | 0.235 | 0.385 | 0.665 | 0.752 |
| $\beta = 0.1$ | 0.230 | 0.438 | 0.679 | 0.763 |
| $\beta = 0.2$ | 0.231 | 0.436 | 0.659 | 0.732 |
| $\beta = \mathbf{0.3}$ | 0.248 | 0.474 | 0.695 | 0.765 |
| $\beta = 0.4$ | 0.204 | 0.392 | 0.626 | 0.702 |
| $\beta = 0.5$ | 0.234 | 0.448 | 0.680 | 0.759 |
| $\beta = 0.6$ | 0.206 | 0.412 | 0.646 | 0.732 |
| $\beta = 0.7$ | 0.218 | 0.414 | 0.647 | 0.752 |
| $\beta = \mathbf{0.8}$ | 0.246 | 0.487 | 0.697 | 0.774 |
| $\beta = 0.9$ | 0.230 | 0.438 | 0.658 | 0.739 |
| $\beta = 1$ | 0.237 | 0.479 | 0.679 | 0.748 |

Table 5.19: ReID performance for different modalities.

| Modality | mAP | CMC 1 | CMC 5 | CMC 10 |
|---|---|---|---|---|
| Events | 0.24 | 0.39 | 0.67 | 0.75 |
| Polar | 0.2 | 0.35 | 0.61 | 0.70 |
| Events + Polar | **0.25** | **0.49** | **0.70** | **0.77** |

The table above shows that our implementations follow the insights of the work [4], i.e., the network produces higher outcomes when event and polar information are combined, despite dealing with different types of objects. The only difference between the two implementations is the value of $\beta$ at the loss function. Our experiments show that the network can learn efficiently with $\alpha = 1$ and $\beta = 0.8$. However our network has the similar results with beta $\beta = 0.3$ which also is the result obtained in [4].

To have some visual results to highlight the re-identification model's robustness, an algorithm was created to extract the appearance descriptor for each ground truth bounding box in the current frame using the re-identification model. The track ID is then assigned using the hungarian method, taking into consideration the extracted descriptors between the previous and current frames.

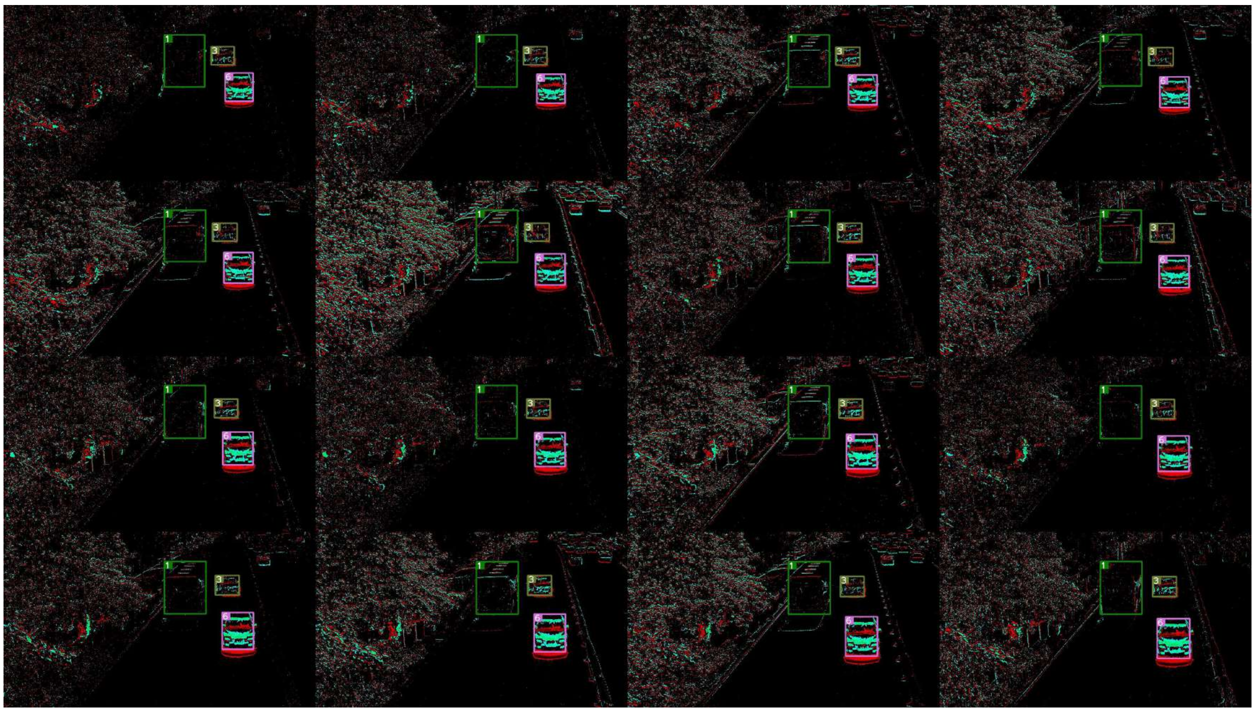


Figure 5.14: Example 1 of re-identification association for a clearly events from vehicles.

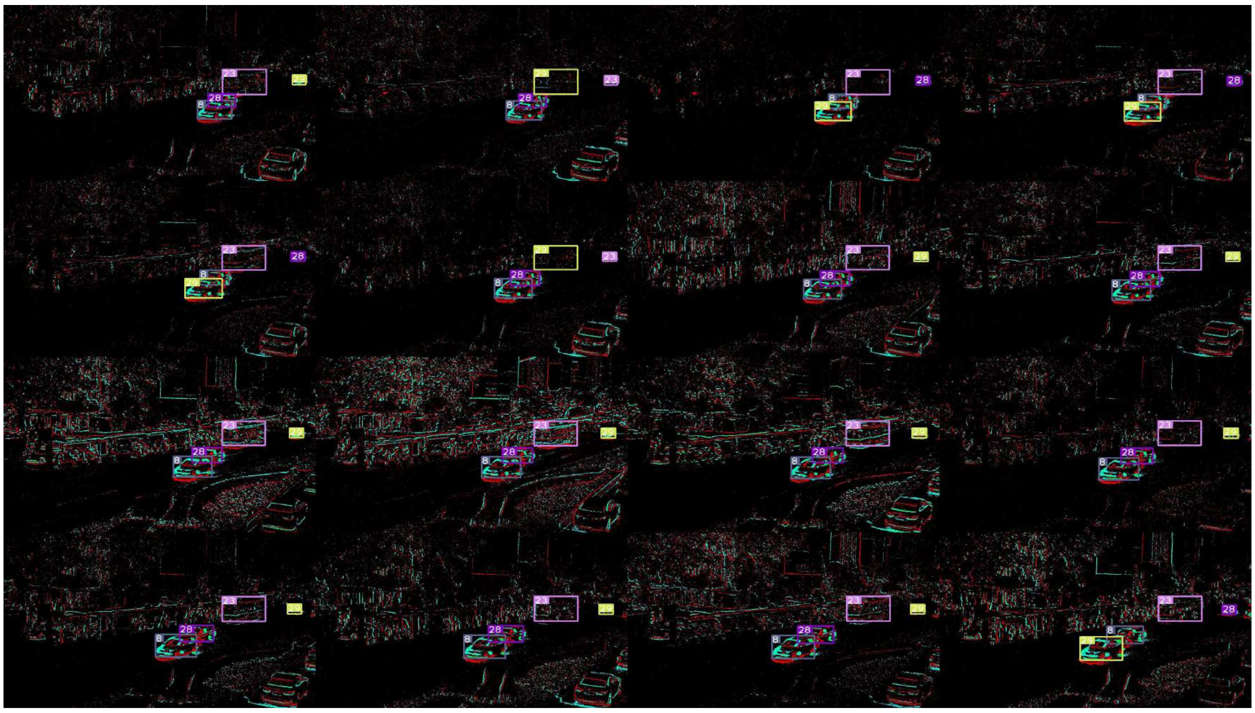


Figure 5.15: Example 2 of re-identification association with occlusions.

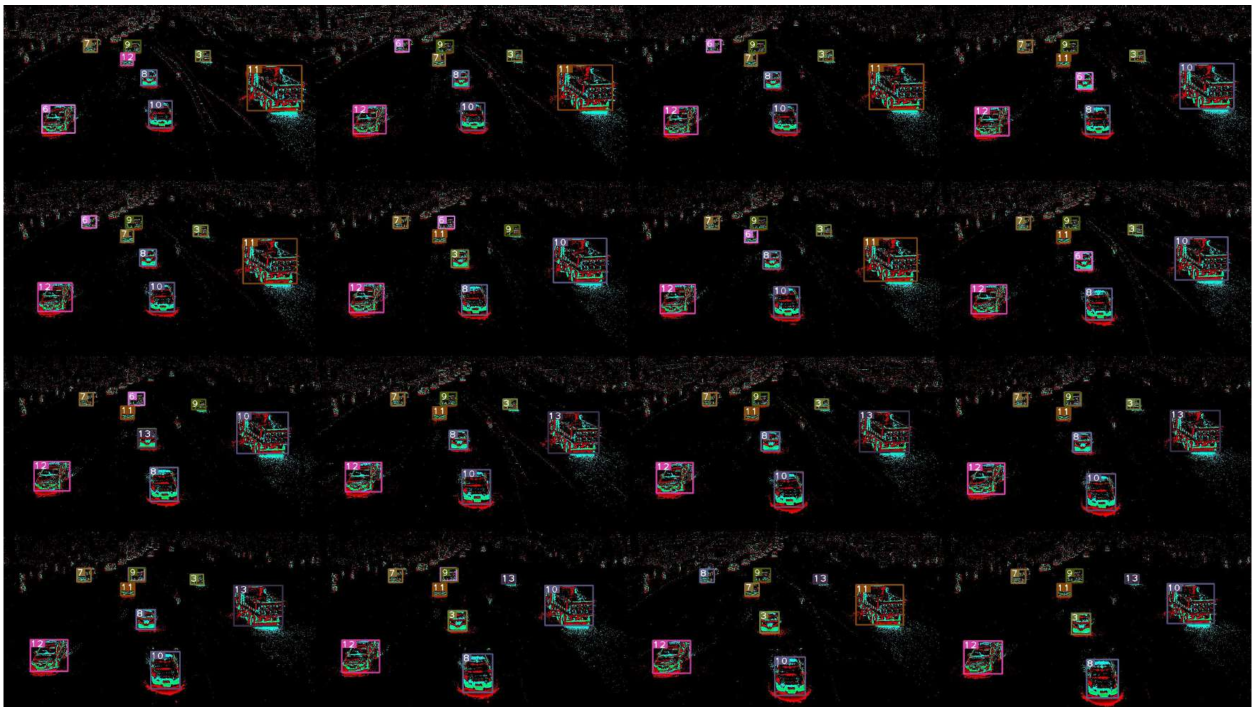


Figure 5.16: Example 3 of re-identification association with a large traffic.

The figures 5.14, 5.15, and 5.16 exhibit the behavior of the re-identification model in three distinct scenarios: one with visible cars throughout the sequence, one with occlusions during the sequence, and the other with a high volume of traffic. The model in the first image displays excellent behavior, resulting in the proper identification of all cars throughout the images. The second has some identity changes caused by the occlusion with two cars, which have stabilized once the occlusion ceases. The final figure shows that the model struggles to extract a descriptor capable of distinguishing each car when there are numerous to compare with.

### 5.4.2 SORT and DeepSORT

Table 5.20: Tracking results for the two methods: SORT and DeepSort

| | Method | MOTA ↑ | MOTP ↑ | Rec ↑ | Prc ↑ | MT ↑ | PT ↑ | ML ↓ | FP ↓ | FN ↓ | ID_SW ↓ | FM ↓ | runtime(Hz) ↑ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Easy | SORT | 0.56 | 0.77 | 0.72 | 0.82 | 0.66 | 0.30 | 0.04 | 21806 | 38563 | 206 | **881** | 37 |
| Medium | SORT | 0.35 | 0.76 | 0.46 | 0.81 | 0.43 | 0.42 | 0.15 | 40829 | 208713 | 976 | **3133** | 35 |
| Hard | SORT | **0.18** | 0.74 | 0.39 | 0.65 | 0.35 | 0.49 | 0.16 | 31749 | 93489 | 381 | **992** | 33 |
| Overall | SORT | 0.35 | 0.76 | 0.50 | 0.78 | 0.48 | 0.40 | 0.12 | **94384** | 340765 | 1563 | **5006** | **35** |
| Easy | DeepSORT | 0.51 | 0.78 | 0.78 | 0.75 | **0.74** | 0.23 | 0.03 | 36015 | 30145 | **149** | 1491 | 24 |
| Medium | DeepSORT | 0.36 | 0.77 | 0.55 | 0.75 | **0.52** | 0.37 | 0.11 | 71675 | 174394 | **794** | 6193 | 19 |
| Hard | DeepSORT | 0.08 | 0.76 | 0.43 | 0.55 | **0.45** | 0.40 | 0.15 | 53335 | 86832 | **372** | 1671 | 16 |
| Overall | DeepSORT | 0.33 | 0.77 | 0.57 | 0.70 | **0.57** | 0.34 | 0.09 | 161025 | **291371** | **1321** | 9355 | 20 |

Both approaches track objects similarly in terms of detection and track ID (MOTA metric), as well as the average overlap between all correctly matched predictions and their ground truth (MOTP metric). However, the SORT method is more reliable when dealing with more challenging tracking conditions, outperforming the MOTA metric by more than 10% in the hard difficulty and is two times faster than DeepSORT.

Furthermore, the SORT algorithm results in fewer vehicle fragmentations and nearly half the false positives, implying that there are fewer predictions to no ground truth tracker. Despite the fact that the SORT approach appears to be more trustworthy, we can observe that DeepSort has fewer track ID changes, fewer false negatives, and can track more vehicles for more than 80% of its trajectory.

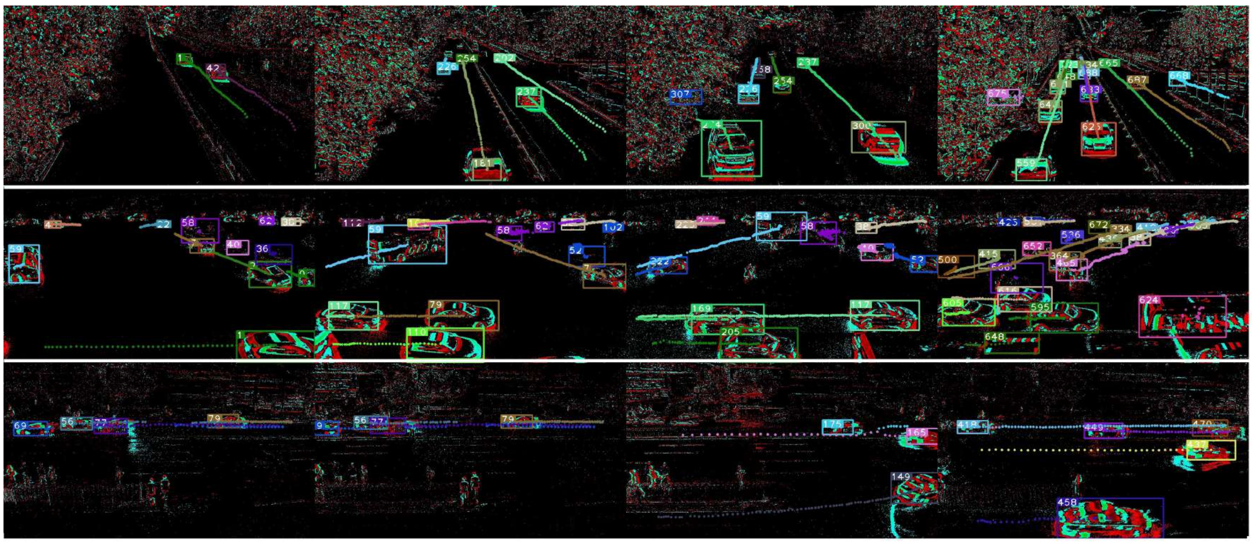


Figure 5.17: Some visual examples for the tracking task with three different sequences (One sequence for each row).

Table 5.21: Comparison of our tracking results with a public benchmark results from the UA-DETRAC dataset.

| Methods | MOTA ↑ | MOTP ↑ | Rec ↑ | Prc ↑ | MT ↑ | PT ↑ | ML ↓ | FP ↓ | FN ↓ | ID_SW ↓ | FM ↓ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (1)CompACT [70] + GOG [71] | 0.38 | 0.74 | 0.42 | 0.92 | 0.28 | 0.33 | 0.40 | 20238 | 373261 | 6457 | 5853 |
| (2)CompACT [70] + RMOT | 0.43 | 0.75 | 0.48 | 0.92 | 0.35 | 0.31 | 0.35 | 32366 | 333866 | 478 | 1146 |
| (3)CompACT [70] + CEM [72] | 0.14 | 0.73 | 0.19 | 0.83 | 0.07 | 0,23 | 0.71 | 25727 | 529408 | 621 | 860 |
| (4)RED-ResNet 640x640 +DeepSort | 0.33 | 0.77 | 0.57 | 0.70 | 0.57 | 0.34 | 0.09 | 161025 | 291371 | 1321 | 9355 |
| (5)RED-ResNet 640x640 +SORT | 0.35 | 0.76 | 0.50 | 0.78 | 0.48 | 0.40 | 0.12 | 94384 | 340765 | 1563 | 5006 |

The table 5.21 compares the performance of multi-vehicle tracking between our work in event-data with some of the methods in RGB data shared by the authors through a matlab toolkit. Looking to the results, in terms of MOTA and MOTP, our methodologies (4 and 5) surpass almost double the MOTA metric compared with the (3) method. However, it has a bad performance of between 5% and 10% compared with the other two methods (1 and 2). Still, in terms of tracking the object during at least 80% of its lifespan, MT metric, our methods have between 22% and 50% better performance and, as a consequence, an improvement in PT and ML metric. Our results show an improved performance in MOTA, MOTP, MT, PT, and ML. How come the FM, ID_SW have bad performance? The reason is the nature of data. Since our ground truth is from RGB data and the event-data is only generated when the object is moving, the fragmentation of the trackers happens with more frequency. On the other hand, trying to extract a feature descriptor from event-data compared with RGB data also penalizes our method (5).

# 6 Conclusions and Future Work

In conclusion, with this work we have achieved our final objective of detecting and tracking multiple vehicles. In fact, in this work we start to understand the importance of having a memory mechanism in our models. Without this, our detection performance decreases because the model is unable to detect static vehicles on the road since they have stopped generating events on the camera. Although we did not achieve state-of-the-art results on the GEN1 dataset, we achieved an excellent result using only 10% of the number of parameters used in the best method using a CNN typology. This leads to a faster performing on detecting vehicles, which is an important aspect in real-time systems. Furthermore, our result is highly influenced by the fact that the dataset is unbalanced in the number of existing examples for each class.

Regarding the other two datasets, we are unable to compare to other state-of-art methods since we have transformed the RGB data to event data. As a result, with this work we have created a baseline for detection and tracking with UA-DETRAC dataset and a detection baseline for the BMVC dataset.

With respect to the typology of Sparse Neural Networks (SSCN), our results have reached the same as the other state-of-the-art method presented on the same dataset, GEN1 Automotive. However, once again, we have accomplished this work by only using 2.3M of parameters against 133M of the comparative network. Furthermore, when creating these types of networks, it is crucial to balance the utilization of sparse convolutions and submanifold convolutions. Only using sparse convolutions has the same impact as using regular convolutions, which reduces the sparsity of the data. On the other hand, using only submanifold convolutions, we will not pass enough information to the subsequent layers, and consequently, the network will not be able to learn the desired task.

At the same time, the Spiking Neural Network (SNN) also explores the sparsity of event data. Our results showed that this network did not reach its full potential in terms of object detection performance, but it outperformed SSCN networks and the other existing SNN state-of-the-art methods on the GEN1 Automotive dataset. On the other hand, we could not measure the efficiency and computation performance because of the absence of hardware. This made it impossible to prove the theoretical advantages.

The last objective was tackled initially using the traditional method of multi-object tracking, SORT, and then, in order to improve outcomes, we went on to the evolution of the algorithm itself, DeepSort, with the creation of a new network to extract an appearance descriptor. That said, the classic algorithm proved to be much more robust, mainly for more complicated scenarios for the tracking task. However, the developed re-identification model brought an improvement in reducing the number of identity changes in tracking, which is the main function for the addition of the new network. However, we have the drawback that by adding the extra model, we add more computational weight, which reduces the processing speed by almost half. It is worth mentioning that these algorithms are heavily influenced by the previous detection model used, and it was confirmed during the discussion of results for the UA-DETRAC dataset that, despite the presence of a memory mechanism, the model was not always able to detect vehicles that had left to generate events on the camera.

Since this is a complex and extensive problem, we can design many objectives to continue to develop and improve this task. Starting with the detection problem, we can then add some complexity to the network with the CNN typology that has already shown excellent results with so few parameters. Then, revisit sparse networks in order to overcome the challenges of network design and training through a better balance between sparse convolutions and submanifold sparse convolutions. Continuing with the sparse typologies but with SNN networks, it would be interesting to demonstrate the network's efficiency and performance on specific hardware to determine whether it is worthwhile to continue with the investigation. Finally, given that several scientists have investigated various tasks with event-data through the development of graph neural networks due to their ability to analyze spatial and temporal information, it would be interesting to develop a model and compare it to the models that have already been developed.

Concerning the second problem imposed by the task, it is first necessary to improve the detection level of the model that feeds the tracking algorithms through the solutions presented above. In a second phase, since the re-identification model revealed that there is still a lot of room for improvement, we could create a more robust model using Transformers networks, which have already demonstrated excellent results for RGB images. However, the implementation of this type of architecture would add a large computational cost. Another, maybe less computationally expensive, option would be to extract its descriptor straight from the detection network, which already extracts the bounding box for each vehicle and identifies it as such. This solution, to our knowledge, has not yet been explored, which brings uncertainty about its success. Finally, we can also analyze more recent tracking algorithms such as: FairMOT [73], TransMOT [74], ByteTrack [75].

# 7 Bibliography

[1] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128 $\times$ 128 120 db 15 s latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43:566–576, 2 2008.

[2] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking, 2017.

[3] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649, 2017.

[4] Shafiq Ahmad, Gianluca Scarpellini, Pietro Morerio, and Alessio Del Bue. Event-driven re-id: A new benchmark and method towards privacy-preserving person re-identification. In *2022 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, pages 459–468, Jan 2022.

[5] Pierre de Tournemire, Davide Nitti, Etienne Perot, Davide Migliore, and Amos Sironi. A large scale event-based detection dataset for automotive, 2020.

[6] Siwei Lyu, Ming-Ching Chang, Dawei Du, Longyin Wen, Honggang Qi, Yuezun Li, Yi Wei, Lipeng Ke, Tao Hu, Marco Del Coco, et al. Ua-detrac 2017: Report of avss2017 & iwt4s challenge on advanced traffic monitoring. In *Advanced Video and Signal Based Surveillance (AVSS), 2017 14th IEEE International Conference on*, pages 1–7. IEEE, 2017.

[7] Y Hu, S C Liu, and T Delbruck. v2e: From video frames to realistic DVS events. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2021.

[8] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, and Tae-Kyun Kim. Multiple object tracking: A literature review. *Artificial Intelligence*, 293:103448, apr 2021.

[9] J. Sjöström and W. Gerstner. Spike-timing dependent plasticity. *Scholarpedia*, 5(2):1362, 2010. revision #184913.

[10] Hananel Hazan, Daniel Saunders, Darpan T. Sanghavi, Hava Siegelmann, and Robert Kozma. Unsupervised learning with self-organizing spiking neural networks. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, jul 2018.

[11] Alain Artola and Wolf Singer. Long-term depression of excitatory synaptic transmission and its relationship to long-term potentiation. *Trends in Neurosciences*, 16(11):480–487, 1993.

[12] B. S. Blais and L. Cooper. BCM theory. *Scholarpedia*, 3(3):1570, 2008. revision #91041.

[13] Eugene M. Izhikevich and Niraj S. Desai. Relating stdp to bcm. *Neural Computation*, 15(7):1511–1523, 2003.

[14] Sander Bohte, Joost Kok, and Johannes Poutré. Spikeprop: backpropagation for networks of spiking neurons. volume 48, pages 419–424, 01 2000.

[15] Filip Ponulak and Andrzej Kasiński. Resume learning method for spiking neural networks dedicated to neuroprostheses control. 01 2006.

[16] R. Bogacz, M.W. Brown, and C. Giraud-Carrier. Frequency-based error backpropagation in a cortical network. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 2, pages 211–216 vol.2, 2000.

[17] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113:54–66, 2014.

[18] Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.

[19] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures, 2018.

[20] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-yolo: Spiking neural network for energy-efficient object detection, 2019.

[21] Loïc Cordone, Benoît Miramond, and Philippe Thierion. Object detection with spiking neural networks on automotive event data, 2022.

[22] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[23] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothee Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks, 2020.

[24] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and lt;0.5mb model size, 2016.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[26] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[27] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2016.

[28] Benjamin Graham. Sparse 3d convolutional neural networks. In *BMVC*, 2015.

[29] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks, 2016.

[30] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions, 2016.

[31] Dominic Zeng Wang and Ingmar Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, 2015.

[32] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks, 2017.

[33] Nico Messikommer, Daniel Gehrig, Antonio Loquercio, and Davide Scaramuzza. Event-based asynchronous sparse convolutional networks. 2020.

[34] Jyotibdha Acharya, Andres Ussa Caycedo, Vandana Reddy Padala, Rishi Raj Singh Sidhu, Garrick Orchard, Bharath Ramesh, and Arindam Basu. Ebbiot: A low-complexity tracking algorithm for surveillance in iovt using stationary neuromorphic vision sensors. volume 2019-September, pages 318–323. IEEE Computer Society, 9 2019.

[35] Bharath Ramesh, Shihao Zhang, Zhi Wei Lee, Zhi Gao, Garrick Orchard, and Cheng Xiang. Long-term object tracking with a moving event camera.

[36] Guang Chen, Hu Cao, Muhammad Aafaque, Jieneng Chen, Canbo Ye, Florian Röhrbein, Jörg Conradt, Kai Chen, Zhenshan Bing, Xingbo Liu, Gereon Hinz, Walter Stechele, and Alois Knoll. Neuromorphic vision based multivehicle detection and tracking for intelligent transportation system. *Journal of Advanced Transportation*, 2018, 2018.

[37] Marco Cannici, Marco Ciccone, Andrea Romanoni, and Matteo Matteucci. Asynchronous convolutional networks for object detection in neuromorphic cameras. 5 2018.

[38] Etienne Perot, Pierre de Tournemire, Davide Nitti, Jonathan Masci, and Amos Sironi. Learning to detect objects with a 1 megapixel event camera. 9 2020.

[39] Yuhuang Hu, Tobi Delbruck, and Shih Chii Liu. Learning to exploit multiple vision modalities by using grafted networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12361 LNCS:85–101, 2020.

[40] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.

[41] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, 2016.

[42] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2017.

[43] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2015.

[44] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[45] B.-N. Vo and W.-K. Ma. The gaussian mixture probability hypothesis density filter. *IEEE Transactions on Signal Processing*, 54(11):4091–4104, 2006.

[46] Ba-Tuong Vo, Ba-Ngu Vo, and Antonio Cantoni. Analytic implementations of the cardinalized probability hypothesis density filter. *IEEE Transactions on Signal Processing*, 55(7):3553–3567, 2007.

[47] Yaakov Bar-Shalom, Fred Daum, and Jim Huang. The probabilistic data association filter. *IEEE Control Systems Magazine*, 29(6):82–100, 2009.

[48] Andres Ussa, Chockalingam Senthil Rajen, Deepak Singla, Jyotibdha Acharya, Gideon Fu Chuanrong, Arindam Basu, and Bharath Ramesh. A hybrid neuromorphic object tracking and classification framework for real-time systems. 7 2020.

[49] Bharath Ramesh, Shihao Zhang, Hong Yang, Andres Ussa, Matthew Ong, Garrick Orchard, and Cheng Xiang. e-tld: Event-based framework for dynamic object tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3996–4006, Oct 2021.

[50] Zhenjiang Ni, Sio Hoi Ieng, Christoph Posch, Stéphane Régnier, and Ryad Benosman. Visual tracking using neuromorphic asynchronous event-based cameras, 4 2015.

[51] Francisco Barranco, Cornelia Fermuller, and Eduardo Ros. Real-time clustering and multi-target tracking using event-based sensors. 7 2018.

[52] M. Savvides and B. V.K. Vijaya Kumar. Efficient design of advanced correlation filters for robust distortion-tolerant face recognition. pages 45–52. Institute of Electrical and Electronics Engineers Inc., 2003.

[53] David S. Bolme, Bruce A. Draper, and J. Ross Beveridge. Average of synthetic exact filters. pages 2105–2112. Institute of Electrical and Electronics Engineers (IEEE), 3 2010.

[54] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters.

[55] João F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. 4 2014.

[56] Hongmin Li and Luping Shi. Robust event-based object tracking combining correlation filter and cnn representation. *Frontiers in Neurorobotics*, 13, 2019.

[57] Rui Jiang, Xiaozheng Mou, Shunshun Shi, Yueyin Zhou, Qinyi Wang, Meng Dong, and Shoushun Chen. Object tracking on event cameras with offline-online learning. *CAAI Transactions on Intelligence Technology*, 5, 04 2020.

[58] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[59] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, June 1989.

[60] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[61] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks, 2017.

[62] Weihua He, Yujie Wu, Lei Deng, Guoqi Li, Haoyu Wang, Yang Tian, Wei Ding, Wenhui Wang, and Yuan Xie. Comparing snns and rnns on neuromorphic vision datasets: Similarities and differences. *Neural Networks*, 132, 08 2020.

[63] Rudolf E. Kálmán. A new approach to linear filtering and prediction problems" transaction of the asme journal of basic. 1960.

[64] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[65] Pranav Adarsh, Pratibha Rathi, and Manoj Kumar. Yolo v3-tiny: Object detection and recognition using one stage improved model. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 687–694, 2020.

[66] Qize Yang, Ancong Wu, and Wei-Shi Zheng. Person re-identification by contour sketch under moderate clothing change. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(6):2029–2046, June 2021.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[68] Longyin Wen, Dawei Du, Zhaowei Cai, Zhen Lei, Ming-Ching Chang, Honggang Qi, Jongwoo Lim, Ming-Hsuan Yang, and Siwei Lyu. UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking. *Computer Vision and Image Understanding*, 2020.

[69] C. Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*, 2014.

[70] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos. Learning complexity-aware cascades for deep pedestrian detection, 2015.

[71] Hamed Pirsiavash, Deva Ramanan, and Charless Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. pages 1201 – 1208, 07 2011.

[72] Anton Andriyenko and Konrad Schindler. Multi-target tracking by continuous energy minimization. In *CVPR 2011*, pages 1265–1272, 2011.

[73] Yifu Zhang, Chunyu Wang, Xinggang Wang, Wenjun Zeng, and Wenyu Liu. FairMOT: On the fairness of detection and re-identification in multiple object tracking. *International Journal of Computer Vision*, 129(11):3069–3087, sep 2021.

[74] Peng Chu, Jiang Wang, Quanzeng You, Haibin Ling, and Zicheng Liu. Transmot: Spatial-temporal graph transformer for multiple object tracking, 2021.

[75] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. Bytetrack: Multi-object tracking by associating every detection box, 2021.