UNIVERSIDADE D
COIMBRA

Alessio Rivetti Silva

Neuromorphic Event-based Activity and Anomaly Detection

Dissertação no âmbito do Mestrado Integrado em Engenharia Electrotécnica e de Computadores, na especialização de Automação, orientada pelo Professor Doutor Jorge Manuel Moreira de Campos Pereira Batista e apresentada ao Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Ciências e Tecnologias da Universidade de Coimbra.

Outubro de 2021

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

Alessio Rivetti Silva

Neuromorphic Event-based Activity and Anomaly Detection

Dissertação no âmbito do Mestrado Integrado em Engenharia Electrotécnica e de Computadores, na especialização de Automação, orientada pelo Professor Doutor Jorge Manuel Moreira de Campos Pereira Batista e apresentada ao Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Ciências e Tecnologias da Universidade de Coimbra.

Outubro de 2021

Agradecimentos

O desenvolvimento deste projeto de dissertação não teria sido possível sem o apoio de um conjunto de pessoas que gostaria de mencionar. Em primeiro lugar, queria agradecer ao meu orientador, Professor Jorge Batista, por toda a ajuda e acompanhamento dado ao longo destes meses de trabalho. Também agradecer a sua confiança nas minhas capacidades e todos os desafios propostos, os quais me permitiram crescer bastante enquanto futuro engenheiro.

Um agradecimento muito especial à minha família, os meus pais e o meu irmão, e também à minha namorada, por me terem apoiado em todos os momentos. Foram vocês que me deram a força para seguir em frente e para tentar atingir sempre o meu máximo, mesmo nas alturas mais difíceis. Sem o vosso apoio constante, nada disto teria sido possível.

Quero agradecer também aos meus amigos e colegas de curso, por todas as vivências, pelo companheirismo e pelas boas memórias que levo desta etapa da minha vida. Em particular, queria mencionar duas pessoas que me acompanharam de forma mais próxima ao longo de todo este percurso: Gonçalo Moreira e Afonso Silva. Sem vocês, este percurso não teria sido a mesma coisa, por isso, um grande obrigado por tudo.

Um forte agradecimento aos membros do Laboratório de Visão por Computador, Pedro Martins e Bruno Silva, pela forma como me acolheram, e aos meus colegas de grupo de orientação ainda não mencionados, André Graça e Eurico Almeida, por todas as partilhas de ideias. Quero também deixar um agradecimento a todas as pessoas que colaboraram na gravação de um dos datasets utilizados neste trabalho.

Por último agradecer a todas as pessoas que se foram cruzando comigo ao longo da minha vida e que de certa forma contribuíram para que me tornasse na pessoa que sou hoje.

A todos, um enorme e sentido Obrigado!

Abstract

The development of camera sensors has had a great impact on people's lives, as several types of applications are backed by this technology. These can range from leisure applications to quality control and security applications, among many others. Over the past decade, a different type of camera sensor, designed in the end of the 1980's, started to gather the interest of the Computer Vision's research community. This camera has several denominations, including neuromorphic camera, silicon retina, dynamic vision sensor and the most used, event camera. Event cameras mimic the behaviour of the neuronal structures present in the human eye. Therefore, they capture brightness changes at pixel level, contrasting with the frame capturing performed by conventional cameras. This camera's characteristics have unlocked a new paradigm in the Computer Vision area of research.

Typically, in surveillance applications or scenarios, there is a big concern with respect to people's privacy. Traditional cameras deployed in these applications are able to capture the appearance of an individual, and this aspect has created a barrier to people's approval of local surveillance. There is not a consensus over what is more important: privacy or security? Event cameras can help solve this issue, since no appearance information is gathered by them. Events are triggered by moving edges, thus, it only captures the individual's silhouette, making it very difficult to identify them. Moreover, these cameras are known for performing effectively in low light conditions, allowing usability through entire days without requiring supplementary technologies, such as infrared night vision cameras. Another advantage of this camera technology, over traditional cameras, lies in low power consumption.

This dissertation aims at presenting how this specific camera sensor performs with regards to the detection of anomalous events or activities in a scene. At the beginning of the development of this work, only two articles concerning this subject had been published. This dissertation follows the optical flow-based approach and targets the expansion of the latter with new learning algorithms and by using a different camera model.

Keywords: Event-based Vision, Anomalous Event Detection, Event Cameras, Machine Learning, Computer Vision.

Resumo

O desenvolvimento de sensores na forma de câmara tem vindo a causar um grande impacto na vida das pessoas, dado que diversos tipos de aplicações são suportados por esta tecnologia. Estas variam entre aplicações de lazer até aplicações de controlo de qualidade e segurança, entre muitas outras. Na última década, um tipo de câmara diferente, desenhado no final da década de 1980, começou a captar o interesse da comunidade de investigadores da área da Visão por Computador. Esta câmara tem diversas denominações, tais como câmara neuromórfica, retina de silício, sensor dinâmico de visão e a mais utilizada, câmara de eventos. As câmaras de eventos mimetizam o comportamento das estruturas neuronais presentes no olho humano. Como tal, elas capturam variações de luminosidade ao nível dos pixéis, contrastando com a captura de *frames* efetuada pelas câmaras convencionais. As características destas câmaras abriram caminho para um novo paradigma na área científica da Visão por Computador.

Tipicamente, em aplicações ou cenários de videovigilância, existe uma grande preocupação no que diz respeito à privacidade das pessoas. As câmaras tradicionais que são utilizadas neste tipo de aplicações tem a capacidade de captar a aparência de um indivíduo, e este aspeto tem vindo a colocar uma barreira na aprovação da vigilância local por parte das pessoas. Não existe um consenso sobre qual é mais importante: privacidade ou segurança? As câmaras de eventos podem ajudar a resolver este problema, já que nenhuma informação relativa à aparência é captada por estas. Os eventos são despoletados por extremidades (do inglês, *edges*) em movimento, portanto, apenas é registada a silhueta de um indivíduo, tornando difícil a sua identificação. Para além disso, estas câmaras apresentam um desempenho eficaz em condições de baixa luminosidade, permitindo a sua utilização durante dias inteiros sem a necessidade de tecnologias suplementares, como câmaras de visão noturna através de radiação infravermelha. Outra vantagem desta tecnologia sobre as câmaras tradicionais reside no seu baixo consumo energético.

Esta dissertação procura estudar o desempenho deste tipo de câmaras em situações de deteção de eventos ou atividades anómalas numa cena. No início do desenvolvimento deste trabalho, apenas tinham sido publicados dois artigos referentes a este tema. Esta dissertação segue a abordagem baseada no fluxo ótico e procura expandir esta última com novos algoritmos de aprendizagem e utilizando uma câmara de um modelo diferente.

Palavras-Chave: Visão Baseada em Eventos, Deteção de Eventos Anómalos, Câmaras de Eventos, Aprendizagem de Máquina, Visão por Computador.

Contents

List of Acronyms

ABMOF Adaptive Block-Matching Optical Flow.

AE Autoencoder.

AED Anomalous Event Detection.

AUC Area Under Curve.

CX5 CeleX5.

DEEC Departamento de Engenharia Electrotécnica e de Computadores.

DictL Dictionary Learning.

DL Deep Learning.

DV346 Davis 346.

DVS Dynamic Vision Sensor.

EC Event Camera.

ELM Extreme Learning Machine.

ELM-AE Extreme Learning Machine-Autoencoder.

EMST Event-based Multiscale Spatio-Temporal.

FPN Fixed Pattern Noise.

GAN Generative Adversarial Network.

GT Ground Truth.

GUI Graphical User Interface.

HOF Histogram of Optical Flow.

ISR Institute of Systems and Robotics.

jAER Java tools for Address-Event Representation.

JDK Java Development Kit.

K-SVD K-Singular Value Decomposition.

LASSO Least Absolute Shrinkage and Selection Operator.

LK Lucas-Kanade.

ML Machine Learning.

MSE Mean Square Error.

NN Neural Network.

OCELM One-Class Extreme Learning Machine.

ODL Online Dictionary Learning.

OF Optical Flow.

OR-ELM Online Recurrent Extreme Learning Machine.

OS-ELM Online Sequential Extreme Learning Machine.

RBF Radial Basis Function.

RNN Recurrent Neural Network.

ROC Receiver Operating Characteristic.

SC Sparse Coding.

SDK Software Development Kit.

SLFN Single Hidden Layer Feedforward Network.

SR Sparse Representation.

SVD Singular Value Decomposition.

UC University of Coimbra.

List of Figures

List of Tables

Chapter 1

Introduction

The first chapter presents the context and motivation that gave origin to the developed work. A definition of Anomalous Event Detection (AED) is presented and then illustrated with examples for a better understanding. A depiction of the main objective and the main contributions are also provided, as well as the dissertation's outline.

1.1 Context and Motivation

AED has been an area of great interest in the field of Computer Vision since its inception. For some time now, researchers have dedicated time and effort to improve the reliability of anomaly detection algorithms using conventional frame cameras. As the name suggests, AED focuses on detecting activities that do not comply with the patterns of normality. For instance, if a walkway is under surveillance and subjects are only expected to walk, if there is a subject running or cycling, such activities will be labelled as anomalous. If that same walkway is under surveillance and subjects are only expected to walk in a certain direction, if there is a subject walking in the opposite direction, such activity will be labelled as anomalous. Thus, AED algorithms can be employed in various real-life scenarios and play an important role in local security.

Recently, a different type of camera technology started to gather a lot of interest from the research community. These cameras are called Event Cameras (ECs) and have given the input to a new field in Computer Vision called Event-based Vision. Further concept analysis of this technology will be given in Chapter 2, section 2.1.

These cameras have never been used in our department, therefore, this fact alone provides extra motivation for this dissertation. There was a big sense of responsibility and aspiration to deliver a solid project, so that our department may start to be seen as a contributor to the Event-based Vision field of research. Furthermore, there are very few published articles regarding the use of ECs to perform AED. This fact, along with the scarcity of datasets, posed a big challenge on this project: the implementation of an appropriate framework to perform AED effectively using ECs.

1.2 Main Objective

The main objective of this dissertation was to develop, validate and evaluate different Machine Learning (ML) algorithms in the context of AED using ECs. The ML algorithms that were tested are presented below:

1. Sparse Representation (SR)
2. Extreme Learning Machine (ELM)
    (a) Online Sequential Extreme Learning Machine (OS-ELM)
    (b) Online Recurrent Extreme Learning Machine (OR-ELM)
3. Autoencoder (AE)

A secondary objective was to perform AED in real-time. Unfortunately, despite countless attempts and tests, this objective was not accomplished due to some limitations on the hardware end.

1.3 Main Contributions

In summary, the followed approach relies on the extraction of Optical Flow (OF) to generate descriptors for each region of the frame where a significant amount of events were recorded. Since the authors used the Davis 346 (DV346) camera by *iniVation*, to conduct their study, they relied on Java tools for Address-Event Representation (jAER)[1] to perform the aforementioned extraction. As the work developed in this dissertation was performed with different hardware, more specifically CeleX5 (CX5) by *CelePixel*, jAER could not be used because data was outputted in a different format.

Consequently, an adaptation of the same OF algorithm was adopted, but this time implemented in C++[2] instead of Java. According to the author, Min Liu from ETH Zurich, this code was developed with the objective of sending OF visualization to a PC, via TCP or UDP protocol. Moreover, this source code was intended to receive data from a Davis camera, with far less resolution when compared to CX5. Thus, this code required some adjustments in order to be fully compatible with the owned hardware. As some issues appeared, the previous code[2] was complemented with another source code by the same author[3]. After all the adjustments, the code was able to perform OF extraction for the data obtained by CX5, in a text file format. The

[1] `https://github.com/SensorsINI/jaer`
[2] `https://github.com/wzygzlm/abmof_libcaer`
[3] `https://github.com/SensorsINI/EDFLOW`

modified source code was then uploaded to another branch of the same GitHub repository, for other researchers to use[4].

Another contribution lies in the testing and validation of two ML approaches that were not employed in the original study. These approaches were mentioned in the previous section of this chapter.

1.4 Document Outline

The dissertation adopts the following structure:

- **Chapter 2:** Theoretical knowledge to help understand the concepts supporting the developed framework;
- **Chapter 3:** Presentation of the state of the art for AED using both frame cameras and ECs;
- **Chapter 4:** Materials and methods that aided the developed work;
- **Chapter 5:** Showcase of the developed work, highlighting each step that led to the final results;
- **Chapter 6:** Analysis of the experimental results obtained during the course of the work;
- **Chapter 7:** Sharing of final conclusions and future work recommendations.

[4] `https://github.com/wzygzlm/abmof_libcaer/tree/txt_mod`

Chapter 2

Background Knowledge

This chapter provides all the background knowledge required for a better understanding of the basic notions involved in this dissertation. It contains information regarding ECs and the chosen ML algorithms.

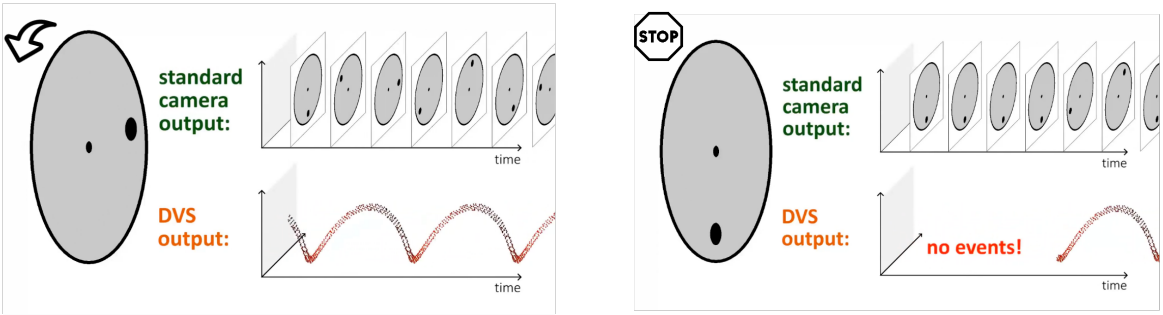
2.1 Event Cameras

ECs are bio-inspired sensors whose output differs from standard cameras. In simple terms, standard cameras output the light intensity captured in a scene, whereas event cameras capture light intensity changes. In other words, instead of capturing images at a fixed rate, ECs asynchronously measure per-pixel brightness changes, and output a stream of events. This stream encodes the time at which each event occurred, its pixel location and sign of the brightness change [1]. A single event i of that stream can be characterized by the equation (2.1):

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

In equation (2.1), t_i is the timestamp, (x_i, y_i) are the pixel coordinates and p_i is the polarity of event i. The polarity refers to the brightness change, therefore if the brightness increases, the polarity will be represented by the number 1 and if the brightness decreases, the polarity will be represented by the number 0. A comparison between the outputs of these cameras is presented in Figure 2.1.

These cameras contain certain properties that can be viewed as advantages when compared to traditional cameras. They have very high temporal resolution and low latency (both in the order of microseconds), very high dynamic range ($140dB$ vs. $60dB$ of standard cameras) and low power consumption [1]. Moreover, it was proven that this technology is able to capture activity (in the form of events) in low light conditions. This characteristic can unlock a new paradigm in the anomaly detection field, since standalone frame cameras cannot capture activity effectively in these conditions. In [1], the authors suggest some of the applications in



(a) Output comparison for a rotating circle with a dot.

(b) Output comparison for a stationary circle with a dot.

Figure 2.1: Output comparison between a standard camera and a Dynamic Vision Sensor (DVS). (a) As the circle rotates, the standard camera captures frames at a specific frame-rate, i.e, in a synchronous fashion; as the DVS only captures the moving parts, namely, the dot present in the circle. (b) As the circle becomes stationary, the DVS stops acquiring information, hence its asynchronous nature. On the other side, the standard camera keeps capturing frames just as before. Images adapted from `https://www.youtube.com/watch?v=cffwH41ReF4`.

which these cameras can be particularly effective when compared to other sensing modalities. These include real-time interaction systems, such as robotics and wearable electronics, where operation in uncontrolled lighting conditions, latency and power consumption are important.

2.2 Event-based Optical Flow

Accurate and fast measurements of OF are necessary requirements for using them in vision tasks. In general terms, OF is a structure obtained from motion information regarding the environment [2]. Several algorithms regarding this topic have been presented, however, the focus of this section is to present some of the most popular event-based OF methods, namely, the Lucas-Kanade (LK) Flow, the Local Plane Fits Flow and the Adaptive Block-Matching Optical Flow (ABMOF).

2.2.1 Lucas-Kanade Flow

This approach starts from a total derivative of brightness formulation presented by Barron et al. [3], whose solution can be found using a technique presented by Lucas and Kanade [4]. For each event, a neighbourhood of dimensions $(n \times n \times \Delta t)$ is constructed and its histogram of previous events is used to estimate a spatial and temporal gradient. This serves as input data to an over-determined system of linear equations that can be solved for the OF vector with Least Squares Estimation [2].

This method is based on the assumption that light intensity $I(x, y, t)$ is invariant during an infinitesimally short time, and thus the gradient constraint equation can be derived:

$$\nabla I^T \begin{bmatrix} v_x \\ v_y \end{bmatrix} = -\frac{\partial I}{\partial t} \tag{2.2}$$

The velocity vector $(v_x, v_y)^T$ is the desired motion flow. Nevertheless, equation (2.2) has two variables, meaning that it is under-determined and needs a second assumption: the local flow, or velocity vector, is constant over the neighbourhood around a specific pixel. Hence, a system of $m = n^2$ can be deducted:

$$\begin{bmatrix} \nabla I(x_1, y_1)^T \\ \vdots \\ \nabla I(x_m, y_m)^T \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} -I_{t_1} \\ \vdots \\ -I_{t_m} \end{bmatrix} \tag{2.3}$$

As stated previously, the solution for this system is obtained via Least Squares Estimation, hence, the solution to this system is represented by:

$$\begin{aligned} Av &= b \\ \Rightarrow v &= (A^T A)^{-1} A^T b \end{aligned} \tag{2.4}$$

If the eigenvalues satisfy $\lambda_1 \geq \lambda_2 > 0$, then the covariance matrix $A^T A$ is invertible. Moreover, the eigenvalues act as confidence measures on the correctness of the computed velocity, and hence, no velocity is computed if both are below a certain confidence threshold τ.

In regards to the event-based LK flow, light intensity I variations are unknown. As such, a conversion from light intensity to events is needed. To achieve this, several approaches were proposed, most of them consisting on an event count over a time interval [5] [6] [7].

2.2.2 Local Plane Fits Flow

This method, presented by Benosman et al. [8], uses the local properties of events' spatio-temporal neighbourhood by fitting a plane to an incoming event's neighbourhood on the surface of recent events. In contrast to the LK method, this does not need an estimation of spatial and temporal gradients [2].

When each event at pixel location (x, y) arrives at a time t, it is drawn into a $3D$ coordinate system and previous events in the same location are discarded. This is known as the surface of active events. Similarly to the previous approach, an assumption of constant local velocity in a small neighbourhood around the events is made. This provides robustness against noise and compensates for missing events when estimating OF. The fitting parameters (a, b, c, d) of this local plane $ax + by + ct + d = 0$ are determined by solving a homogeneous system of equations with least squares regression. Then, an algorithm of iterative improvement over the first fit is

applied. The velocity is given by the inverse gradient:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = -c \begin{bmatrix} \frac{1}{a} \\ \frac{1}{b} \end{bmatrix} \tag{2.5}$$

This formulation assumes time to be a strictly increasing function of space, such that the local derivatives a and b are never zero, however, these are often zero or very small. Some specific recording scenarios can cause the gradient of the local plane to have a vanishing component along a designated orientation. In order to deal with this issue, it is considered that the true direction of motion is encoded by the gradient $g = (-\frac{a}{c}, -\frac{b}{c})$ of the fitted plane. Because its components describe the variation of time with respect to space, it originates a magnitude mismatch. Therefore, a normalization of this gradient vector becomes essential, as well as a multiplication by its correct length, given by the inverse of its magnitude: $|g| = \sqrt{a^2 + b^2}/c$. Finally, the robust velocity vector is given by:

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \frac{1}{|g|^2} g = \frac{-c}{a^2 + b^2} \begin{bmatrix} a \\ b \end{bmatrix} \tag{2.6}$$

2.2.3 Adaptive Block-Matching Optical Flow

The ABMOF algorithm presented an alternative to some of the most formerly used OF extraction methods, such as the ones presented before.

In this algorithm, three time-slices are defined: $t - 2d$, $t - d$ and t. The first two time-slices accumulate previous events and the last one accumulates current events. The parameter d corresponds to the slice duration, typically expressed in ms. The authors in [9] discard polarity data because it requires one bit of pixel memory to record and does not improve accuracy significantly. For each incoming event, a reference block is generated in slice $t - d$, centered in its location. Then, a search is conducted to find the best matching block in slice $t - 2d$, using the sum of absolute difference. Finally, the OF is calculated based on the positional offset of blocks in each time-slice and the time interval. This algorithm also implements a feedback control mechanism to adapt the slice duration in order to obtain better matching results across time-slices. Figure 2.2 provides visual support to the concept behind this algorithm.

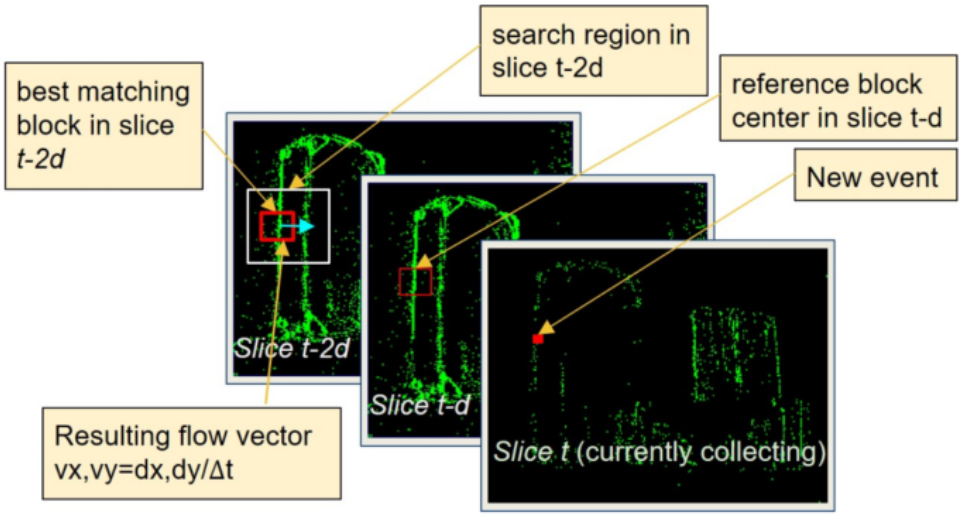


Figure 2.2: ABMOF's working principle. Adapted from [9].

2.3 Machine Learning

The detection of abnormal events is performed based on the learned representation of normal events, because the full nature of abnormal events is unknown. Consequently, it is very difficult to label all possible abnormal occurrences. Hence, the objective of the proposed ML architectures is to learn an efficient representation/coding in an unsupervised manner.

This section describes the chosen ML architectures to perform AED. These include two models of Extreme Learning Machines and an Autoencoder. Since SR was used in the original work [10], and to keep it in context, its description will be made in Chapter 4, section 4.3.4.

2.3.1 Extreme Learning Machine

Before delving into the two models of ELMs chosen for this work, it is relevant to provide a brief introduction about ELMs themselves. This architecture's intent was essentially to deliver a faster but still robust learning mechanism when compared to classic feedforward NNs. These are known to have a slow learning speed, due to the use of slow gradient-descent based learning algorithms and also because all the parameters of the networks are tuned iteratively using such learning algorithms [11]. ELMs are Single Hidden Layer Feedforward Networks (SLFNs) that randomly choose hidden nodes and determine its output weights analytically. Figure 2.3 presents the configuration of this network.

From a mathematical standpoint, the output of a SLFN with $\tilde{N}$ hidden nodes (additive or Radial Basis Function (RBF) nodes) can be represented by (2.7):

$$f_{\tilde{N}}(x) = \sum_{i=1}^{\tilde{N}} \beta_i G(a_i, b_i, x), \qquad x \in \mathbb{R}^n, \qquad a_i \in \mathbb{R}^n \tag{2.7}$$

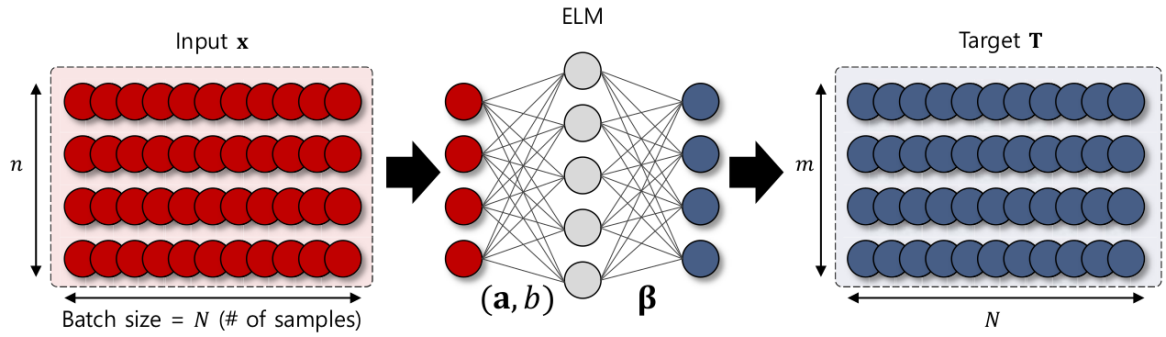


Figure 2.3: Configuration of the ELM network. Adapted from [12].

a_i is the weight vector connecting the input layer to the i^{th} hidden node, b_i is the bias of the i^{th} hidden node and β_i is the weight connecting the i^{th} hidden node to the output node. $G(a_i, b_i, x)$ is the output of the i^{th} hidden node with respect to input x. Depending on the type of hidden node (additive or RBF), $G(a_i, b_i, x)$ will have a different activation function (sigmoid or Gaussian).

Considering N arbitrary distinct samples $(x_i, t_i) \in \mathbb{R}^n \times \mathbb{R}^m$, where x_i is a $n \times 1$ input vector and t_i is a $m \times 1$ target vector, if a SLFN with $\tilde{N}$ nodes can approximate these N samples without error, it means that there exist β_i, a_i and b_i such that:

$$f_{\tilde{N}}(x_j) = \sum_{i=1}^{\tilde{N}} \beta_i G(a_i, b_i, x_j) = t_j, \qquad j = 1, ..., N \tag{2.8}$$

In a more compact fashion, equation (2.8) can be expressed by:

$$H\beta = T \tag{2.9}$$

H is called the hidden layer output matrix and it contains the results obtained from the activation function:

$$H = \begin{bmatrix} G(a_1, b_1, x_1) & \cdots & G(a_{\tilde{N}}, b_{\tilde{N}}, x_1) \\ \vdots & \ddots & \vdots \\ G(a_1, b_1, x_N) & \cdots & G(a_{\tilde{N}}, b_{\tilde{N}}, x_N) \end{bmatrix}_{N \times \tilde{N}} \tag{2.10}$$

β and T are the weight and target value matrices, respectively:

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix} \qquad T = \begin{bmatrix} t_1^T \\ \vdots \\ t_{\tilde{N}}^T \end{bmatrix} \tag{2.11}$$

Huang et al. [11] provided proof that H is invertible with probability one for $\tilde{N}$ arbitrary distinct input vectors and for randomly generated hidden node parameters (a_i, b_i) with any continuous probability distribution. Therefore, equation (2.9) becomes a linear system and the output weights β are estimated as:

$$\beta = H^{\dagger}T \tag{2.12}$$

$H^{\dagger}$ is the Moore-Penrose generalized inverse of the hidden layer output matrix H. This matrix can be calculated with the orthogonal projection method, orthogonalization method, iterative method and Singular Value Decomposition (SVD). Among these, the most used is SVD since it can always be used, without any restrictions nor requirements. One important consideration to be made is that ELM is a batch learning method, hence relying on the availability of complete training data.

Online Sequential Extreme Learning Machine

In some scenarios, especially those in which the amount of training data is very large, it is not possible to supply all the data in one batch to an ELM, memory-wise. Furthermore, in real applications, this data may arrive chunk-by-chunk or one-by-one, and thus, the batch ELM algorithm has to be modified in order to accommodate this behaviour, i.e., to make it online sequential [13].

Let's consider the case where rank$(H) = \tilde{N}$ (number of hidden nodes). Under this assumption, $H^{\dagger}$ is given by:

$$H^{\dagger} = (H^T H)^{-1} H^T \tag{2.13}$$

If $H^T H$ tends to become singular, one can make it nonsingular by constraining the network size, that is, by reducing the number of hidden nodes $\tilde{N}$, or by increasing the amount of training data samples N in the initialization phase. Here, the least-squares solution to equation (2.12) becomes:

$$\beta = (H^T H)^{-1} H^T T \tag{2.14}$$

Given a portion of initial training set $\aleph_0 = \{(x_i, t_i)\}_{i=1}^{N_0}$, with $N_0 \geq \tilde{N}$, the ELM batch algorithm consists only in the minimization of $||H_0\beta - T_0||$. The solution to this minimization is given by $\beta^{(0)} = K_0^{-1} H_0^T T_0$, where $K_0 = H_0^T H_0$. Considering another portion of training data $\aleph_1 = \{(x_i, t_i)\}_{i=N_0+1}^{N_0+N_1}$, with N_1 representing the number of samples in this portion. Now, the problem

becomes a minimization of:

$$\left\| \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} \beta - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix} \right\| \tag{2.15}$$

Considering both portions of training data $\aleph_0$ and $\aleph_1$, the output weight β becomes:

$$\beta^{(1)} = K_1^{-1} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} T_0 \\ T_1 \end{bmatrix} \tag{2.16}$$

As expected, K_1 will be an extension of the formulation presented for K_0:

$$K_1 = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} \tag{2.17}$$

In order to obtain sequential learning, $\beta^{(1)}$ needs to be expressed as a function of $\beta^{(0)}$, K_1, H_1 and T_1. Intuitively, K_1 can assume the form:

$$\begin{aligned} K_1 &= \begin{bmatrix} H_0^T & H_1^T \end{bmatrix} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} \\ &= K_0 + H_1^T H^1 \end{aligned} \tag{2.18}$$

Moreover, the following expression can be deducted:

$$\begin{aligned} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} T_0 \\ T_1 \end{bmatrix} &= H_0^T T_0 + H_1^T T_1 \\ &= K_0 K_0^{-1} H_0^T T_0 + H_1^T T_1 \\ &= K_0 \beta^{(0)} + H_1^T T_1 \\ &= (K_1 - H_1^T H_1)\beta^{(0)} + H_1^T T_1 \\ &= K_1 \beta^{(0)} - H_1^T H_1 \beta^{(0)} + H_1^T T_1 \end{aligned} \tag{2.19}$$

Joining (2.16) and (2.19), $\beta^{(1)}$ is expressed by:

$$\begin{aligned}\beta^{(1)} &= K_1^{-1}\begin{bmatrix}H_0\\H_1\end{bmatrix}^T\begin{bmatrix}T_0\\T_1\end{bmatrix}\\ &= K_1^{-1}(K_1\beta^{(0)} - H_1^T H_1\beta^{(0)} + H_1^T T_1)\\ &= \beta^{(0)} + K_1^{-1}H_1^T(T_1 - H_1\beta^{(0)})\end{aligned} \tag{2.20}$$

Remember that K_1 is given by equation (2.17). Generalizing K and β, as new data arrives, enables a recursive algorithm to update the least-squares solution. For the $(k+1)$ portion of training data, the following can be deducted:

$$\begin{aligned}K_{k+1} &= K_k + H_{k+1}^T H_{k+1}\\ \beta^{(k+1)} &= \beta^{(k)} + K_{k+1}^{-1}H_{k+1}^T(T_{k+1} - H_{k+1}\beta^{(k)})\end{aligned} \tag{2.21}$$

K_{k+1}^{-1} is derived using Woodbury's formula:

$$\begin{aligned}K_{k+1}^{-1} &= (K_k + H_{k+1}^T H_{k+1})^{-1}\\ &= K_k^{-1} - K_k^{-1}H_{k+1}^T(I + H_{k+1}K_k^{-1}H_{k+1}^T)^{-1} \times H_{k+1}K_k^{-1}\end{aligned} \tag{2.22}$$

Letting $P_{k+1} = K_{k+1}^{-1}$, then the updating equations for $\beta^{(k+1)}$ are given by:

$$\begin{aligned}P_{k+1} &= P_k - P_k H_{k+1}^T(I + H_{k+1}P_k H_{k+1}^T)^{-1}H_{k+1}P_k\\ \beta^{(k+1)} &= \beta^{(k)} + P_{k+1}H_{k+1}^T(T_{k+1} - H_{k+1}\beta^{(k)})\end{aligned} \tag{2.23}$$

Online Recurrent Extreme Learning Machine

The OR-ELM was developed with the objective of solving two major drawbacks present in the OS-ELM: the impossibility of adjusting the input weights and the impossibility of applying it to learn Recurrent Neural Network (RNN) [12]. This architecture is able to learn RNN due to the employment of an ELM-Autoencoder and a normalization method called layer normalization.

ELM-AE [14] is able to extract better hidden layer features when compared to classic ELMs. The goal of this architecture is to convert input features to sparse representations, which can then be used to perform unsupervised learning. In order to do so, it differentiates from the ELM by using input data as target data ($t = x$) and by orthogonalizing its randomly assigned input weights a and hidden layer bias values b. According to Kasun et al., this orthogonalization

boosts the generalization performance of ELM-AE. The output weight of ELM-AE β is calculated in the same way as in the ELM, except for a slight difference in the calculation of $H^{\dagger}$:

$$\beta = H^{\dagger}T, \qquad H^{\dagger} = \left(H^{T}H + \frac{I}{C}\right)^{-1} H^{T} \tag{2.24}$$

For the calculation of $H^{\dagger}$, a regularization constant C is added to prevent $H^{T}H$ from being a singular matrix and hence to improve the stability of this architecture. β is the transformation matrix that is able to translate the hidden feature space to input data. Its transpose performs the inverse transformation, from input data to the hidden feature space. Thus, β^{T} is used as input weight ($a = \beta^{T}$). The bias values b of the hidden layer are kept zero to preserve the transformation ability of β^{T}.

In regards to the OR-ELM structure, it consists of three networks: a RNN and two SLFNs, which are auxiliary ELM-AE networks for learning RNN's input and hidden weights. The RNN is the main network, meaning is the one performing the prediction. The auxiliary networks responsible for updating the input and hidden weights are named ELM-AE-IW and ELM-AE-HW respectively. Figure 2.4 illustrates the network structure of the OR-ELM.

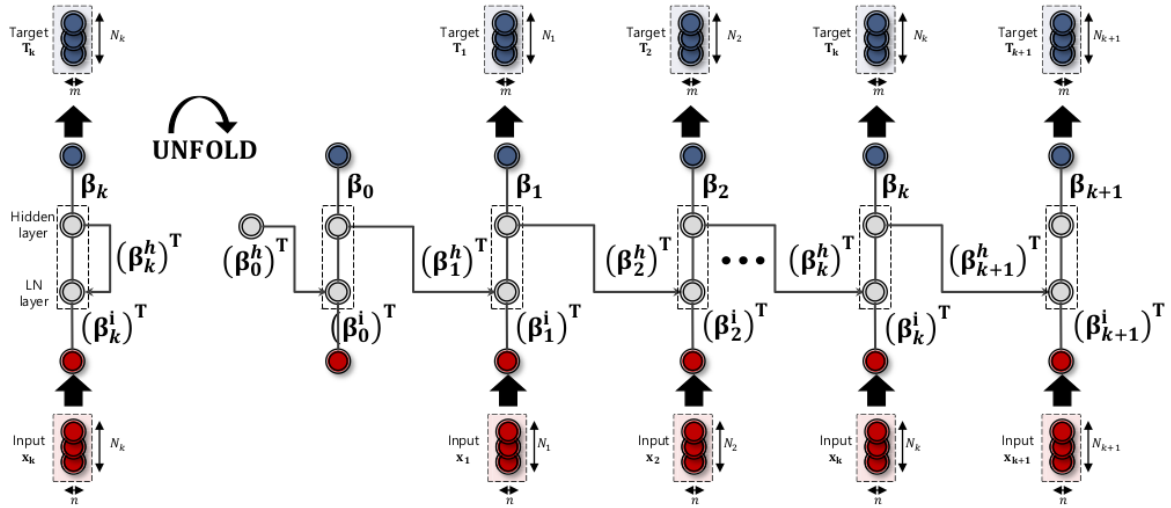


Figure 2.4: OR-ELM network structure. Its structure is identical to a plain RNN except for the normalization. Each red circle represents an input vector. Adapted from [12].

Just like the OS-ELM, this architecture is divided in an initialization phase followed by a sequential learning phase. The initialization phase starts with the definition of the initial output weight β_0 and initial auxiliary matrix P_0, as follows:

$$\beta_0 = 0 \qquad P_0 = \left(\frac{I}{C}\right)^{-1} \tag{2.25}$$

The hidden layer output matrix H_0 is initialized with random values who follow a zero-mean

and a standard deviation of one distribution. Subsequently, the ELM-AEs are initialized, specifically, ELM-AE-IW'a input weight W^i and ELM-AE-HW's input weight W^h are randomly set with a zero-mean and a standard deviation of one, likewise. Their output weights β_0^i, β_0^h and auxiliary matrices P_0^i, P_0^h are initialized with equation (2.25).

Then, the sequential learning phase takes place. Unlike the OS-ELM, this model also tunes the input weights. These weights are updated based on the input weights of the ELM-AE-IW, which propagate the $(k+1)$ input sample $x(k+1) \in \mathbb{R}^{n\times 1}$ to the hidden layer, so that its hidden layer output matrix is given by:

$$H_{k+1}^i = g(norm(W_{k+1}^i x(k+1))) \tag{2.26}$$

The $norm$ function acts as a layer normalization procedure just before the non-linear activation, and is calculated as:

$$\begin{aligned} norm(x) &= \frac{x-\mu^i}{\sqrt{\sigma^{i^2}+\epsilon}} \\ \mu^i = \frac{1}{L}\sum_{j=1}^{L} x_j, \qquad \sigma^i &= \frac{1}{L}\sum_{j=1}^{L}(x_j-\mu^i)^2 \end{aligned} \tag{2.27}$$

Next, the calculation of ELM-AE-IW's output weight β_{k+1}^i is done using recursive least squares method:

$$\begin{aligned} \beta_{k+1}^i &= \beta_k^i + P_{k+1}^i {H_{k+1}^i}^T (x(k+1) - H_{k+1}^i \beta_k^i) \\ P_{k+1}^i &= \frac{1}{\lambda} P_k^i - P_k^i {H_{k+1}^i}^T (\lambda^2 + \lambda H_{k+1}^i P_k^i {H_{k+1}^i}^T)^{-1} H_{k+1}^i P_k^i \end{aligned} \tag{2.28}$$

In contrast to equation (2.23), T_{k+1} is replaced by $x(k+1)$ to perform unsupervised auto-encoding. Furthermore, a constant forgetting factor $\lambda \in [0,1]$ is added in order to tackle the short-term prediction performance of OS-ELMs. The transpose of β_{k+1}^i is used as the input weight of OR-ELM ($W_{k+1} = {\beta_{k+1}^i}^T$).

In like manner, the hidden weight of OR-ELM is updated using ELM-AE-HW, which propagates the OR-ELM's k hidden layer output $H_k \in \mathbb{R}^{L\times 1}$ to its hidden layer, resulting in the following hidden layer output matrix:

$$H_{k+1}^h = g(norm(W_{k+1}^h H_k)) \tag{2.29}$$

β_{k+1}^h and P_{k+1}^h are calculated in the same manner as presented in (2.28), but with the respective data. However, this time T_{k+1} is replaced by H_k, but still to perform unsupervised auto-encoding. The transpose of β_{k+1}^h is used as the hidden weight of OR-ELM ($V_{k+1} = {\beta_{k+1}^h}^T$).

The OR-ELM's hidden layer output matrix H_{k+1} for the $k+1$ input $x(k+1)$ appears as folows:

$$H_{k+1} = g(norm(W_{k+1}x(k+1) + V_{k+1}H_k)) \tag{2.30}$$

To conclude, the output weights β_{k+1} of the OR-ELM are updated using equation (2.23).

2.3.2 Autoencoder

An AE consists of two parts: an encoder and a decoder [15]. Figure 2.5 reveals the structure of this framework. Both encoder and decoder are implemented by NNs and can be interpreted as two functions $z = f(x)$ and $r = g(z)$. Function $f(x)$ maps sample x from data space to feature space, while function $g(z)$ reconstructs sample x by mapping z from feature space to data space. These two functions are usually stochastic functions $p_{encoder}(z|x)$ and $p_{decoder}(r|z)$. Naturally, r is a reconstruction of x.

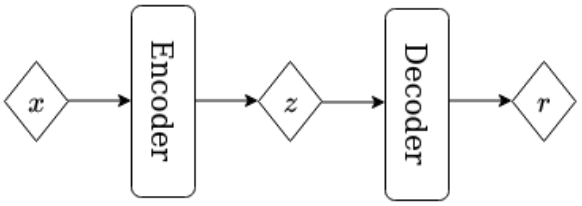


Figure 2.5: Structure of the AE architecture. This structure was firstly introduced by Yann LeCun in his PhD thesis [16]. Inspired by [15].

Given a training set $S = \{x_i | x_i \in \mathbb{R}^d\}$,$1 \leq i \leq n$, the architecture presented in Figure 2.5 can be modeled by the following equation:

$$\begin{cases} z = f(w_e, b_e, x) \\ r = g(w_d, b_d, x) \end{cases} \tag{2.31}$$

As stated previously, f and g are the encoder and decoder functions, respectively. w_e and b_e are the encoder's parameters, while on the other hand, w_d and b_d are the decoder's parameters. Since f and g are NNs themselves, $w_{\{e,d\}}$ and $b_{\{e,d\}}$ are the weight matrices and the bias vectors of these networks. The AE's loss function is defined as:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \|x_i - r_i\|_2^2 \tag{2.32}$$

Training an AE consists in optimizing its loss function, by minimizing the reconstruction error amongst all training data samples. In equation 2.32, $\theta = (w_e, b_e; w_d, b_d)$. In order to solve this optimization problem, one can use the gradient descent or stochastic gradient descent.

Chapter 3

State of the Art

This chapter aims at presenting previous work developed in the AED field, both in the frame camera and EC paradigms. Different modelling strategies will be presented, particularly its Neural Network (NN) architectures and characteristics. As will be demonstrated, event-based AED is largely inspired by methods employed with frame cameras.

3.1 Frame-based AED

Before addressing the work that has been conducted, in the field of detecting anomalies with ECs, it makes sense to first present the different strategies of anomaly detection with traditional cameras.

AED is an unsupervised learning task where the goal is to identify abnormal patterns or motions in data [17]. These patterns are by definition infrequent or rare events. The unsupervised and semi-supervised methodologies discussed in [17] can be decomposed in three categories: reconstruction based, spatio-temporal predictive models and generative models. The results obtained in [17] show that in each strategy (presented in Table 3.1) there are specific architectures that lead to better results.

3.1.1 Generative Adversarial Networks (GANs)

Due to their extended capabilities, GANs have become one of the most popular NN architectures over the past few years. They are composed of two parts: a generator and a discriminator. The generator is fed with a random noise signal and in the training phase, it tries to learn a distribution that will make the discriminator believe the generated image consists of real data.

This type of neural network already showed that it can achieve superior results when compared to some of the other methods used in abnormality detection [18]. In this study, the network was trained using videos exclusively representing normal crowd behaviour. The reason for that lies in the small size of existing datasets with abnormality ground truth. The detection of anomalies in a scene is done by evaluating the amount of overlapping between the generated image

	NN Architectures	Characteristics
Representation learning for reconstruction	Principal Component Analysis (PCA) Autoencoders (AEs) Convolutional Autoencoders (CAEs) Extreme Learning Machines (ELMs)	Build representations that minimize the reconstruction error of training samples
Predictive modeling	Convolutional LSTM 3D Autoencoder Slow Feature Analysis (SFA)	Take into account the spatio-temporal correlation. Trained to minimize the prediction error on spatio-temporal sequences.
Generative models	Variational Autoencoders (VAEs) Generative Adversarial Networks (GANs) Adversarial Autoencoders (AAEs)	Learn to generate samples from the training distribution, while minimizing the reconstruction error as well as distance between generated and training distribution.

Table 3.1: Categorization of the various approaches used in video anomaly detection, the NN architectures employed in each approach and its characteristics.

and the ground truth image. Figure 3.1 shows some of the obtained results in [18].

A variant of this network has also been proposed by some researchers. It consists of the same architecture (one generator and one discriminator), but with a key difference. The generator now has two inputs: random noise and an image. These are called Conditional Generative Adversarial Networks (cGANs). The main objective of this new added input is to force generation of data that closely resembles the ground truth input image. This expanded version of GAN unlocks new capabilities, and some of them are explored in [19]. The results shown in this paper suggest that cGANs are a promising approach for many image-to-image translations, especially those involving highly structured graphical output. Thus, these networks learn a loss function adapted to the task and data at hand, which can be useful in various applications.

Figure 3.2 adds evidence on the great performance obtained by these networks. As can be seen, it is able to generate output images that closely resemble the input images, but with the desired features.

To improve the performance of GANs, Nguyen et al. proposed a dual discriminator approach [20]. More specifically, this new technique targets the problem of mode collapse encountered

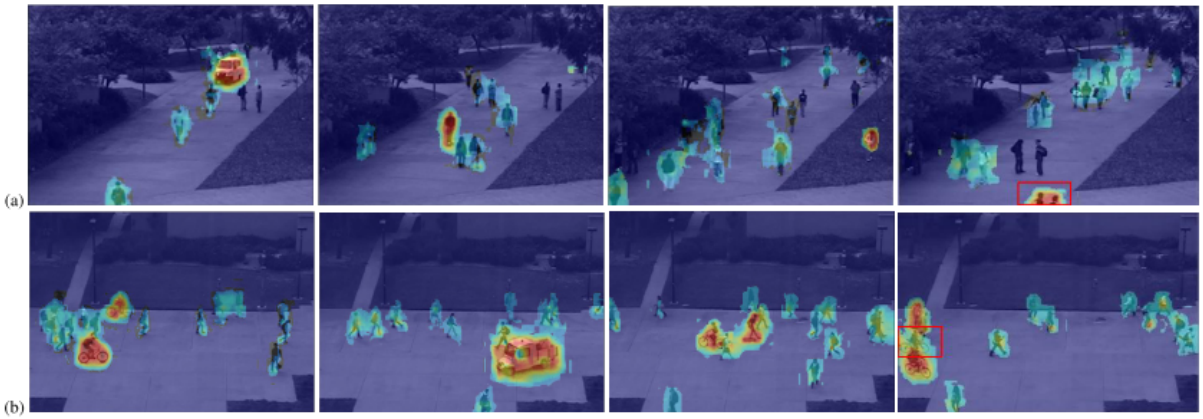


Figure 3.1: AED results obtained by the GAN approach. In the rightmost column, the red rectangles represent detection errors, which may happen occasionally. Adapted from [18].

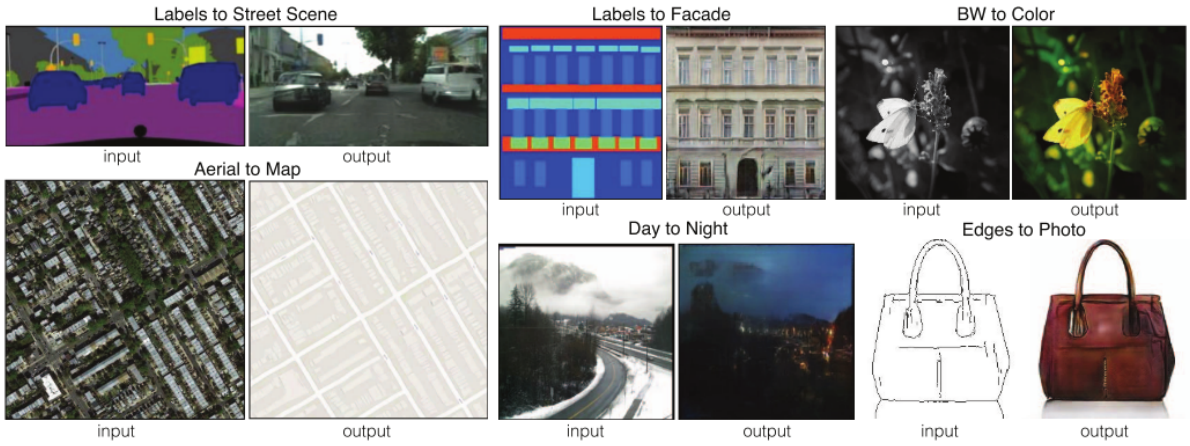


Figure 3.2: Examples of image-to-image translation performed by cGANs. Adapted from [19].

in GANs, which is one key drawback of these types of networks. This dual discriminator GAN is termed D2GAN by the authors in [20]. Mode collapse is an undesired behaviour, in which the outputs tend to be very similar over time, thus neglecting diversity. The objective function of the density estimation problem combines the Kullback-Leibler (KL) and reverse KL divergences in a unified form. Compared to previous state-of-the-art baselines, the proposed approach revealed to be more scalable, in the sense that it can be trained on large datasets (such as ImageNet, for instance).

3.1.2 Extreme Learning Machine (ELM)

ELM gathered the interest of the research community, mainly due to its solution in opposing the time-consuming training methods utilized in NNs. This architecture allowed much faster training times, since its weights are learned in a single step, essentially by learning a linear model. Multiple variations have been presented, and some of them have been used to perform AED tasks.

One-Class Extreme Learning Machine (OCELM)

The authors in [21] introduce the use of a OCELM to perform AED. This is a variation of the classic ELM, explained in chapter 2, section 2.3.1. As the name suggests, the key difference lies in the fact that this network only models one class, instead of multiple classes. In this context, this one class is characterized by normal activities, provided to the model as training inputs. It is expected that training input data originates the same output, always. To achieve this, the output training data is entirely defined by a single value, 1 as suggested in [21]. Also in [21], the authors create descriptors based in OF estimation. One can state that this work motivated the use of ELMs in this dissertation, and also reinforced the validity of OF-based descriptors to perform AED tasks.

Online Sequential Extreme Learning Machine (OS-ELM)

In [22], the authors have used another variation of ELM to perform anomaly detection in the behaviour of an individual driving a wheelchair controlled by a joystick. This variation consists of an Online Sequential Extreme Learning Machine (OS-ELM), and more thorough details about it are provided in chapter 2, section 2.3.1. Essentially, this architecture has an adaptive behaviour that allows periodical updates of the modeled data. This way, the model can adapt its inference behaviour as new data arrives to the model. Although this method was not used to detect anomalies in surveillance environments, it will be tested in such environments in this dissertation.

3.1.3 Autoencoder (AE)

Gong et al. suggested a new AE architecture while confirming that AEs have been extensively used for anomaly detection and have proven effectiveness [23]. This architecture is more thoroughly explained in chapter 2, section 2.3.2. Basically, this NN architecture is able to model high-dimensional data in the unsupervised setting.

3.2 Event-based AED

The first study published on this specific subject was published on 2019 and later updated in February 2020. This study makes use of a GAN acrhitecture to perform AED. A more recent study, published in January 2021, alternatively makes use of the Optical Flow (OF). Both strategies will be described in order to provide a thorough understanding of each one.

3.2.1 GAN-based Approach

Inspired by previous studies performed with frame cameras, the authors in [24] proposed the use of dual discriminator cGANs to perform the AED task at hand with ECs. Section 3.1.1 already introduced some of the aspects that were included in this study. GANs are designed to receive images/frames as input, which are generally not obtained by ECs, due to its sparse data modality. In order to confront this problem, an extra module is introduced, called Deep Learning (DL) Memory Surface Network. Basically, this network makes use of a fully convolutional encoder-decoder architecture to map a discretized volume of event data to a single image called DL memory surface. This memory surface is retrieved from the bottleneck layer of the encoder-decoder network. Figure 3.3 presents the full framework proposed in [24].

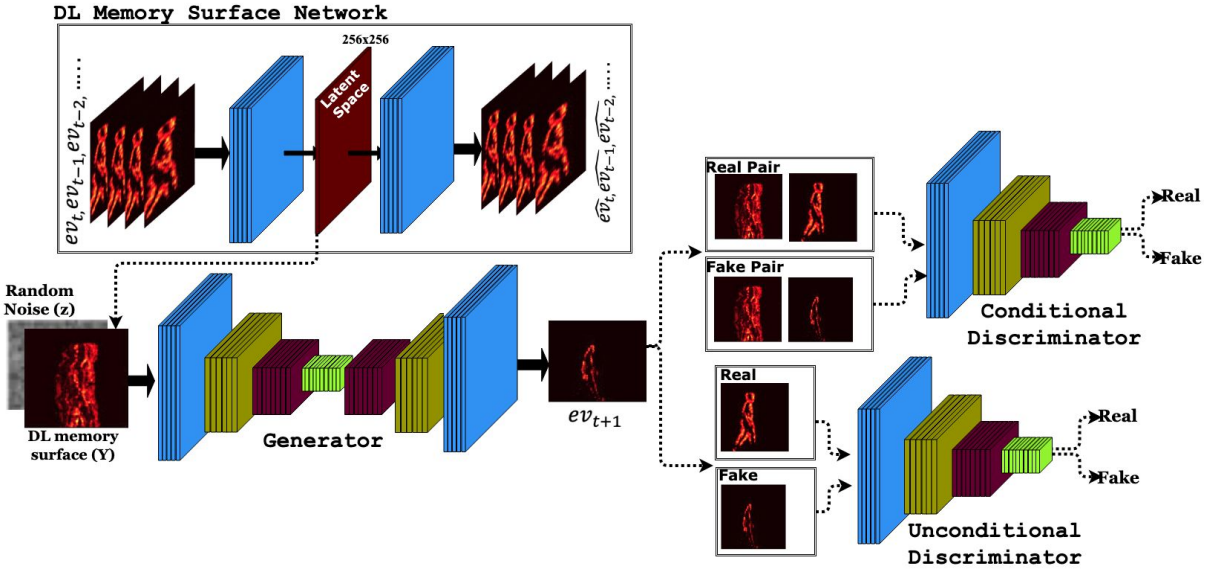


Figure 3.3: Full framework of the proposed approach. It is composed of a DL Memory Surface Network and a dual discriminator cGAN. Adapted from [24].

At inference time, the input image obtained before is considered as being the Ground Truth (GT) image. The reconstructed image obtained is compared to the GT image and the Mean Square Error (MSE) is calculated. If the MSE surpasses the detection threshold, that frame is considered abnormal. Figure 3.4 shows exactly how this inference is made.

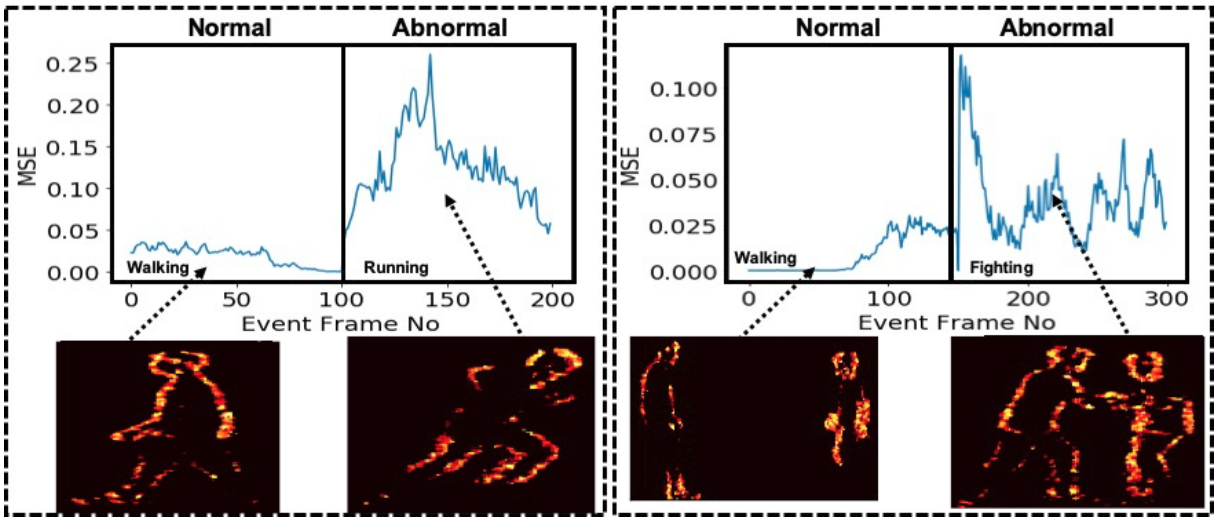


Figure 3.4: Plot of MSE between GT and predicted images. Normal activities, such as walking have a lower MSE, while abnormal activities, such as running and fighting have higher MSEs. Adapted from [24].

3.2.2 Optical Flow Approach

A more recent study has also proved to be effective on detecting abnormal behaviours and/or entities in a scene. In [10], the authors exploited neuromorphic sensors to obtain event-based multiscale spatial-temporal (EMST) descriptors. Essentially, this method relies on optical flow estimation, hence, it is very successful in capturing fast moving entities in a scene. After obtaining the descriptors, these are used to train a Sparse Representation (SR) model, based on Dictionary Learning (DictL) and Sparse Coding (SC). Besides presenting a new algorithm, the authors also created and published the first publicly available neuromorphic dataset for abnormal event detection, named *NeuroAED*.

This approach was chosen for this dissertation and thus, its structure will be further detailed in Chapter 4.

Chapter 4

Materials and Methods

This chapter focuses on presenting the materials and methods chosen to achieve this dissertation's objectives. Essentially, it will present and discuss the EC chosen for this project, a popular tool to process event data and the followed AED methodology, introduced in [10], as well as its dataset and support materials.

4.1 CelePixel CeleX5

The choice of acquiring this camera was mainly due to its specifications. The authors in [1] provided a side-by-side comparison between a wide variety of EC models from different manufacturers, which allowed to conclude that this camera, despite not being the most used, had the best range of specifications from all the others, in respect to the objective of this work. This EC has a resolution of 1280×800 pixels, latency of $8\mu s$, dynamic range of $120dB$ and a power consumption of $400mW$ [1], [25]. Besides capturing events, CX5 also provides grayscale output (with a dynamic range of $120dB$) and OF output. Moreover, it has the capability to output these three modalities concurrently, through a feature named Loop Mode. The acquired model's interface is USB 3.0.

In regards to the camera's Software Development Kit (SDK), the manufacturers provide all the files (libraries, drivers and scripts) needed perform the configuration and installation[1]. The only prerequisite for its correct operation is *OpenCV 3.3.0*, since all visualization features rely on this framework to work properly. The SDK was developed with C++ programming language. A demonstration of all camera features is presented in an application called *DemoGUI*, and Figure 4.1 illustrates its appearance.

Furthermore, the manufacturers provided sample codes that allow users to explore the features embedded in the *DemoGUI* individually, thus offering them the basic modules to develop their own applications. Nevertheless, most of this samples mainly target visualization purposes. As stated in [1], DVSs are noisy because of the inherent shot noise in photons and from tran-

[1] `https://github.com/CelePixel/CeleX5-MIPI`

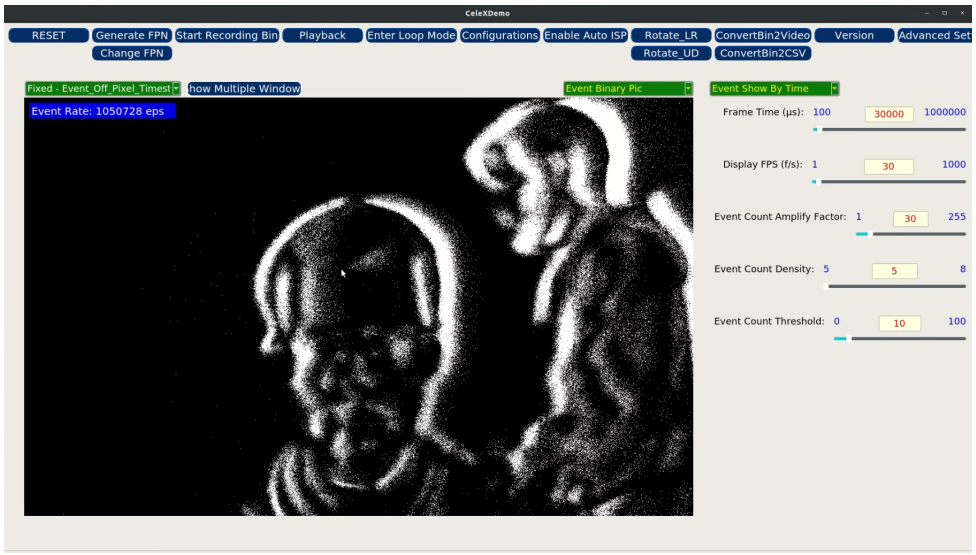


Figure 4.1: Appearance of the *DemoGUI* while in event mode. It features live visualization of events and the recording of these to a *.bin* file. It also features live visualization of different outputs (grayscale and OF) and modes (Fixed Mode and Loop Mode) and adjustment of various parameters and settings.

sistor circuit noise. In order to mitigate this noise effect, CX5 accommodates a software-based mechanism, which makes use of a text file containing the Fixed Pattern Noise (FPN) of the camera. Information regarding the installation process and the SDK features can be consulted in the documentation provided in the respective GitHub repository[1].

4.2 Java tools for Address-Event Representation (jAER)

Certainly one of the most popular tools to process event data acquired by Davis cameras, jAER[2] encompasses an immense set of features. This open-source tool, coded in Java programming language, is being developed with the help of a large group of developers from the event-based vision community. Some of the key features of jAER are the logging and playback of events to/from a file, Matlab compatibility, sending data and controlling hardware over network via UDP, TCP or Multicast protocols and filters. The filters consist of functions developed to obtain specific data or parameters from events, ranging from noise filters to image segmentation, OF extraction and trackers, to name a few.

In [10], the OF extraction process was performed by a filter that implemented the ABMOF algorithm [9]. After recording the dataset, using a DV346 camera, the recordings were opened in jAER and the ABMOF filter results containing OF vector data were logged to text files.

[2] `https://github.com/SensorsINI/jaer`

4.3 NeuroAED

The pipeline presented by the authors in [10] follows the structure presented in Figure 4.2. A breakdown of each module will be provided in the upcoming sections. The recording of events was performed with the Davis 346 (DV346) camera by *iniVation*, which is one of the most utilized models of ECs in the research community, along with other Davis models. This camera has a resolution of 346×260 pixels.

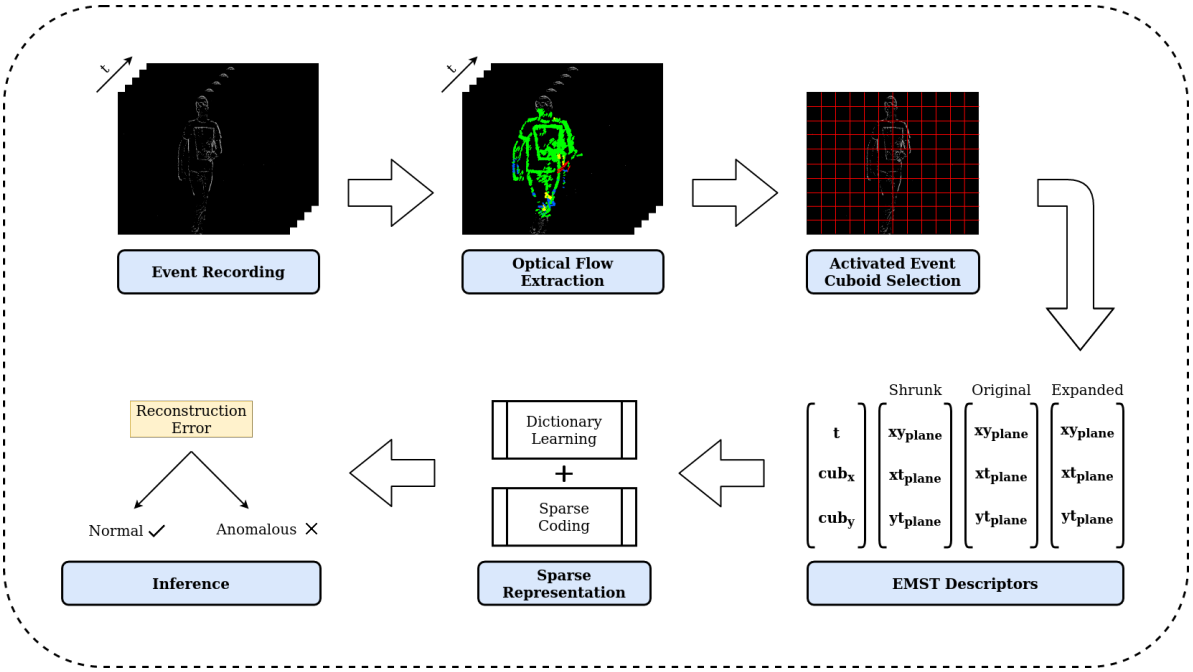


Figure 4.2: NeuroAED framework.

4.3.1 Optical Flow Extraction

Neuromorphic sensors are natural motion detectors. After all, they capture brightness variations (i.e events) in a scene, and such variations can only be provoked by moving objects. Nonetheless, the events themselves do not hold any information with regards to the direction and amplitude of the moving entities. As this approach relies on both direction and amplitude information to distinguish abnormal events from normal events, it became mandatory to adopt an algorithm that could extract such information from raw event data. The adopted algorithm is called Adaptive Block-Matching Optical Flow (ABMOF) [9], detailed in Chapter 2, section 2.2.3. The authors relied on this algorithm's implementation present in jAER.

4.3.2 Activated Event Cuboid Selection

This step aims at selecting only the regions of interest for the EMST descriptor calculation. The reason behind this is to prevent the computation of descriptors for every activated pixel, which

is known to be computationally expensive. These regions of interest are referred to as activated event cuboids. As shown in figure 4.2, the event slice is divided into $M \times N$ non-overlapping event cuboids of dimensions $\Delta x \times \Delta y \times \Delta t$. This event slice division can be viewed as if a grid was placed on top of the event slice, and each rectangle represents an event cuboid. To distinguish activated event cuboids from non-activated event cuboids, an event count threshold is defined. This means that if a specific event cuboid contains a sufficient amount of events to surpass the threshold, it will be considered as activated and thus moves on to the next phase.

4.3.3 Event-based Multiscale Spatio-Temporal Descriptors

With the previous stages completed, the focus shifts towards the construction of a robust descriptor that should contain relevant information in order to distinguish normal from abnormal events. To achieve this, the authors in [10] inspired themselves in the widely used Histogram of Optical Flow (HOF) feature [26]. Hence, the extraction of the event-based HOF (eHOF) is performed for each activated event cuboid. Figure 4.3 brings further insight to this procedure.

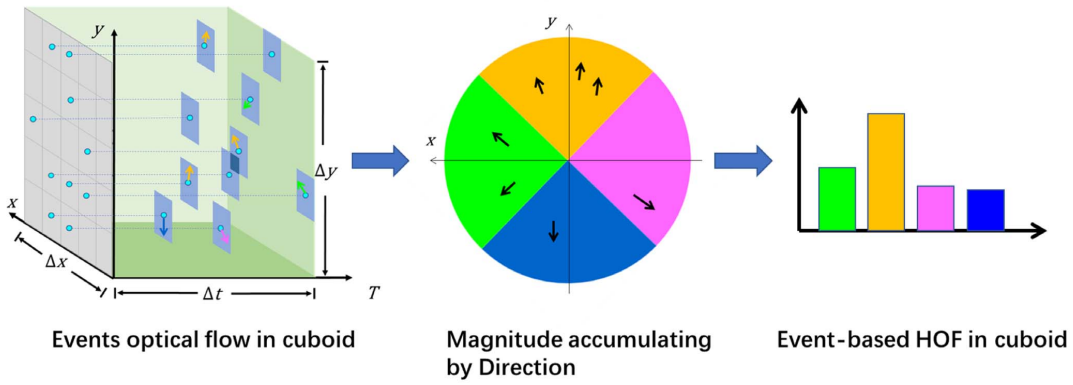


Figure 4.3: Event-based Histogram of Optical Flow (eHOF). Adapted from [10].

Events are processed inside each activated cuboid by accumulating their optical flow magnitude into o bins based on the corresponding direction to form the eHOF feature, which is expressed by:

$$H = (T_1, T_2, ..., T_o) \in R^{1 \times o} \tag{4.1}$$

In equation (4.1), T_i corresponds to the accumulated magnitude in the $i-th$ direction. The central picture in Figure 4.3 shows that the histogram contains 4 bins ($o = 4$), represented by the colours: pink, yellow, green and blue. For example, if one event has an OF vector with a direction of 0° associated, its amplitude will contribute to the magnitude accumulation in the first bin, coloured in green. After this step, a cuboid region of dimension 9×9 centered on the current

activated event cuboid is defined. This means that eight adjacent cuboids around the current activated event cuboid are selected in each of the $xy\text{-}plane$, $xt\text{-}plane$ and $yt\text{-}plane$. Figure 4.4 demonstrates the final result out of this selection.

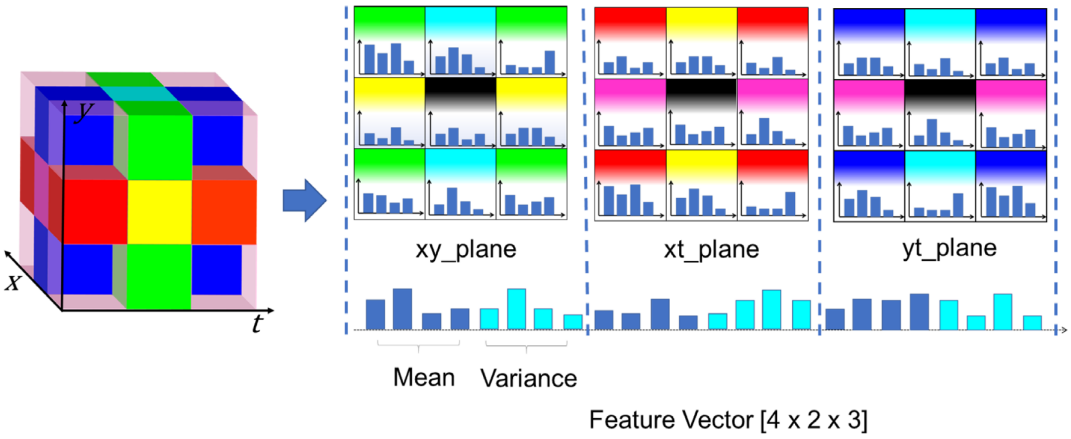


Figure 4.4: Single scale spatio-temporal feature. Adapted from [10].

In Figure 4.4, the activated event cuboid is coloured in black. In each plane, eHOF features are calculated for each neighbouring cuboid, marked with a colour other than black. The neighbouring cuboids, along the vertical and horizontal direction of the plane, indicate intersections between two planes. As an example, for the $xy\text{-}plane$, yellow cuboids indicate an intersection with $xt\text{-}plane$, while aqua green cuboids indicate an intersection with $yt\text{-}plane$. Keeping up with $xy\text{-}plane$ as an example, the accumulated eHOF features of the c ($c = 9$, eight neighbouring cuboids plus the activated event cuboid in the center) cuboids for this plane are obtained:

$$P_{xy} = (H_{xy-1}, H_{xy-2}, ..., H_{xy-c}) \in R^{o \times c} \tag{4.2}$$

Next, the average and variance values are calculated for each bin/direction of the eHOF:

$$\mu_{xy}^{(i)} = \frac{1}{c} \sum_{j=1}^{c} H_{xy-j}^{(i)} \in R^{1 \times o} \tag{4.3}$$

$$\sigma_{xy}^{2(i)} = \frac{1}{c} \sum_{j=1}^{c} (H_{xy-j}^{(i)} - \mu_{xy}^{(i)})^2 \in R^{1 \times o} \tag{4.4}$$

In equations (4.3) and (4.4), $i \in [1, o]$ represents the bin/direction. These values are then combined to form the description vector of $xy\text{-}plane$ $F_{xy} = (\mu_{xy}^{(i)}, \sigma_{xy}^{2(i)}) \in R^{1 \times 2o}$. Intuitively, a fast moving entity on the scene will cause their eHOFs to have larger magnitude on the motion direction, resulting in an higher average in F_{xy}. This is the key principle that leads to a correct identification of abnormal events. F_{xy} focuses on describing motion characteristics with respect to spatial space, whereas F_{xt} and F_{yt} focus on extracting motion features in the time domain.

The variance on the latter contain velocity changes in time dimension, which corresponds to the acceleration value that serves as an important evaluation indicator. Thus, the single scale event-based spatio-temporal descriptor is comprised of the description vector on the three planes $F_s = (F_{xy}, F_{xt}, F_{yt})$. As stated in Figure 4.4, each feature vector has size $[4 \times 2 \times 3] = 24$.

Finally, to deal with the scale variation of moving objects, each activated event cuboid is resized into three scales: shrunk, original and expanded. Figure 4.5 brings additional comprehension to the final form of the EMST descriptor. One can conclude that in the end, each EMST descriptor has size $[4 \times 2 \times 3] \times 3 = 72$.

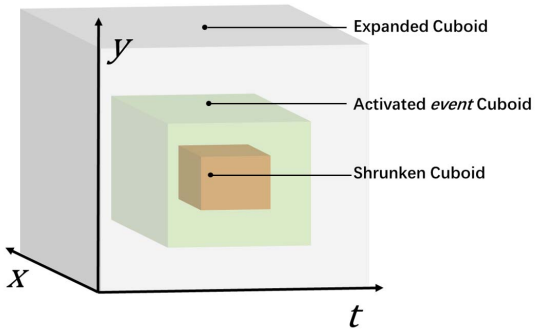


Figure 4.5: Multiscale spatio-temporal cuboids. Adapted from [10].

4.3.4 Sparse Representation

In most cases, anomalous events are hard to characterize due to their unpredictable nature. Therefore, the main idea supporting AED algorithms is to learn the normal event model. Then, at inference time, the inputs will be classified as normal or abnormal depending on their deviation from the trained model, again, consisting only of normal patterns. In [10], the learning strategy employed was Sparse Representation (SR), composed by four parts: Dictionary Learning (DictL), Sparse Coding (SC), Iterative Updating and Abnormal Measurement.

Dictionary Learning

The goal of DictL is to learn the patterns present in a set of features that can lead to acceptable sparse coding results. The dictionary learning can be formulated as:

$$\min_{\beta} \|X - D\beta\|_2^2 \quad s.t. \quad \|\beta\|_p \leq s \tag{4.5}$$

In equation 4.5, $X \in R^{m \times n}$ is the input features, $D \in R^{m \times d}$ is an overcomplete dictionary ($m < d$), β is a set of atoms from dictionary D and $\|\beta\|_p$ is the regularization constraint of parameter s ($\leq n$) to produce a SR. In the training process, D and β are alternatively optimized

and the authors tested both K-Singular Value Decomposition (K-SVD) [27] and Online Dictionary Learning (ODL) [28] algorithms to construct the dictionary.

Sparse Coding

After obtaining the learned dictionary, an algorithm of SC is now employed to obtain a SR of each EMST descriptor. The chosen algorithms were K-LIMAPS [29] and Least Absolute Shrinkage and Selection Operator (LASSO).

Iterative Updating

The dictionary is initialized by randomly sampling from the input features. This dictionary is used to obtain the SRs of the input features. These are then used to update the dictionary, based on the minimization of their reconstruction errors. In the next iteration, the SCs will be calculated with the updated dictionary, and this procedure is repeated until the reconstruction errors lie below a certain threshold. This joint optimization between DictL and SC contributes to more accurate SRs of the input features, which is the main objective of this stage.

Abnormal Measurement

This final step consists of abnormal measurement in input data. The SR allows the capture of the inherent structures and patterns of the input data to perform such measurement. The abnormality of an event is measured according to the rarity similarity between input features. The similarity metric selected was the Mahalanobis distance, which relies on the statistical distribution of the SRs of all training inputs, assembled in the training phase. This distribution is essentially composed of the mean values and the covariance matrix of the SRs. In the testing phase, the Mahalanobis distance between the SR of each testing input and the statistical distribution is calculated. If this distance surpasses a certain detection threshold θ, that testing input will be labeled as abnormal, as shown in the formulation below:

$$Label(X_{test}) = \begin{cases} normal & M(SR(X_{test})) < \theta \\ abnormal & M(SR(X_{test})) \geq \theta \end{cases} \tag{4.6}$$

In equation (4.6), $M(SR(X_{test}))$ is the Mahalanobis distance and $SR(X_{test})$ is the SRs of testing feature X_{test}. As mentioned above, θ is the detection threshold and it corresponds to the maximum distance of all training inputs.

4.4 NeuroAED Dataset

Publicly spread and accepted datasets are at the base of computer vision algorithms' rapid development, once they allow direct comparison between the algorithms. Considering the important role of datasets in the development of abnormal event detection systems and the lack of a neuromorphic vision dataset dedicated to this task, NeuroAED dataset was built. This dataset comprises 152 samples of four distinct indoor and outdoor environments, and thus is split into four sub-datasets, namely Walking, Campus, Square and Stair datasets. Each sub-dataset contains training and testing samples. The training samples consist only of normal events, while testing samples consist of both normal and abnormal events. Moreover, each sub-dataset contains pixel-level and slice-level Ground Truth (GT) files, to evaluate the performance of the detection algorithm.

In this dissertation, only the Walking sub-dataset was used because it lined up with the main objective of this work. It was recorded in a walkway on a sunny day and contains 30 training samples and 28 testing samples, ranging from around 8 to 20 seconds of duration. Most of the samples have a rather sparse crowd density. The event characterization in this sub-dataset is provided in Table 4.1. Illustrations of the normal and abnormal events enumerated in the following table can be found in Appendix I.

Normal Events	Abnormal Events
Pedestrians walking	Pedestrians running
	Cycling
	Motorcycles

Table 4.1: Characterization of normal and abnormal events in Walking sub-dataset of NeuroAED dataset.

4.5 NeuroAED Support Materials

In addition to providing the first publicly available event-based AED dataset, the authors also provided the source codes utilized for data pre-processing and EMST descriptor calculation, via GitHub repository[3]. The pre-processing materials contain source codes for converting *.aedat* files containing raw event data acquired by DV346 to *.txt* files and to fix the negative timestamp bug present in jAER's playback feature.

In July, of the present year, the source codes used for the SR modelling were added to this repository, nevertheless, none were utilized in this dissertation.

[3] `https://github.com/ispc-lab/NeuroAED`

Chapter 5

Developed Work

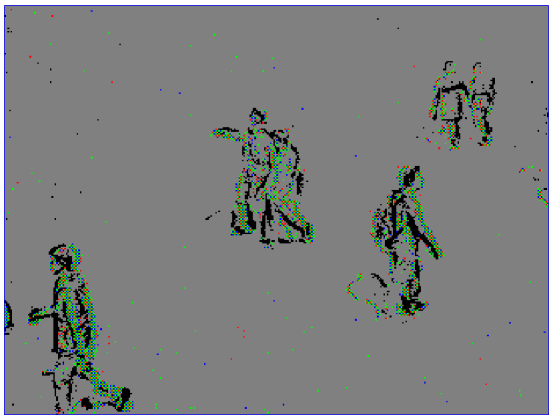
This chapter addresses all the work developed in order to accomplish the main objectives of this dissertation. It contains a preliminary methodology validation, as well as a validation of the alternative ML architectures implemented, all tested with NeuroAED Walking dataset. It also contains information regarding data acquisition using the CX5 and the attempts to perform OF extraction in real time. Finally, it presents the application developed to perform the event data acquisition in a more user-friendly way and also the two datasets recorded in the department's facilities.

5.1 Methodology Validation

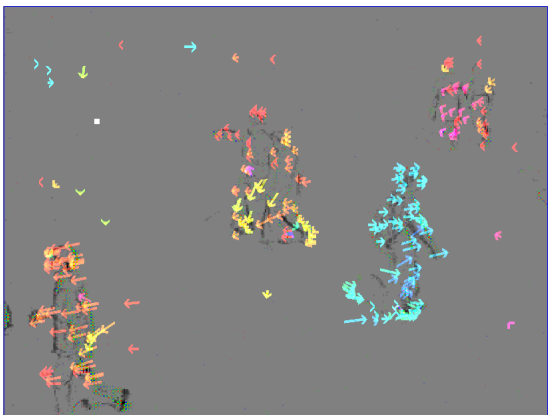
After analysing the state of the art and choosing the methodology to be followed, a preliminary validation of the methodology began. The objective was to analyse its performance and verify if it could become the baseline to develop the intended future work.

The first step consisted in downloading the NeuroAED dataset. Next, jAER was downloaded and installed, since it was necessary to complete the OF extraction process in the event data acquired by DV346. Given the fact that this is a Java application, it required the installation of a Java Development Kit (JDK), in this case, *NetBeans* by *Apache*. Given a sample from NeuroAED dataset, the OF calculated in jAER appears as shown in Figure 5.1.

Once the OF vectors are acquired, the next stage consists in calculating the EMST descriptors for the selected activated event cuboids. As mentioned in Chapter 4, section 4.5, the source codes for performing the pre-processing and EMST descriptor calculation were provided by the authors, and all were used at this stage. The last phase of this preliminary validation consisted in the development of a source code to perform the SR of these cuboids descriptors, in order to distinguish normal activities from abnormal ones. Thus, the *scikit-learn* Python library was used to implement the DictL and SC functions. As referred in preceding chapters, AED is based on the learning of normal patterns; as so, only cuboid descriptors from normal samples were provided to the DictL function. The portion of the source code utilized to perform these tasks



(a) Sample gathered from NeuroAED Walking dataset.



(b) OF extracted from the events present in this sample.

Figure 5.1: Demonstration of the OF extraction process in an event sample from NeuroAED Walking dataset. As seen, the OF is extracted in the form of vectors. In the top-left region of (b), there are wrongfully estimated vectors, primarily due to the noisy nature of ECs.

can be consulted in Appendix II.

The DictL and SC had some parameters that could be changed or tuned. Table 5.1 discloses the parameter values that differ from default values, used in both of them. These parameters were adopted after performing several tests and concluding that these were, in fact, the ones that experienced better results.

Parameter	Value
No. of atoms	20
Sparsity control	1×10^{-8}
Transform algorithm	*Lasso-Lars*
L1 norm penalty	1×10^{-10}

Table 5.1: DictL and SC parameter values that differ from default values.

In order to interpret some results, various plots were made. These will be presented in the next chapter. Another aspect, that should be referenced, is the fact that an outlier rejection method was added, so that noisy data could be disregarded. The reason behind this was to avoid the negative impact of such data in the learned SR model. This method consisted of calculating the Z-Score in the training data and rejecting all the samples whose value was more than 7 standard deviations from the mean.

5.2 Alternative Machine Learning Architectures

The alternative ML architectures that were added were two models of ELM (OS-ELM and OR-ELM) and an AE. Although the original work does not use these, and since they were utilized in previous studies in the field of AED, it seemed interesting to validate such architectures in the

event-based paradigm. A mathematical foundation of these methods was provided in Chapter 2.

5.2.1 Extreme Learning Machine

The base ELM source code for one-class classification was adapted from a GitHub repository[1]. Then, adaptations were made in order to confer online sequential and online recurrent characteristics to this implementation. These were also inspired by previous works posted on GitHub[2,3].

Similarly to the SR model, this architecture is trained with data only representing normal activities and the same outlier rejection method is applied. The ELM only contains one output node, which contains the result of the classification, in this particular case. In the training phase, one needs to provide input data and the corresponding output data. Thus, for all training input data, the defined output value was 1. The network's outputs are compared to the training target outputs (1) and the absolute difference between them is computed. This absolute difference is also known as distance. This model includes a cutoff feature, which is responsible for rejecting the μ% worst training distances. After rejecting these distances, the biggest distance is considered as being the threshold for anomaly detection. In the testing phase, if a sample originates an output with a distance superior to the threshold determined in the training phase, it will be labelled as anomalous. Table 5.2 displays the structure of both OS-ELM and OR-ELM models. The ELMs' source codes can be found in Appendix III.

Parameter	Value
No. of input nodes	72
No. of hidden nodes	500
Mini batch size	1000
Activation function	*Gaussian*
μ	1%

Table 5.2: Structure and parameters of the OS-ELM and OR-ELM architectures.

5.2.2 Autoencoder

On the other hand, the AE architecture was inspired by an article published on the website *Towards Data Science*[4]. It makes use of the *TensorFlow* Python library to construct, train and perform predictions with the aforementioned ML architecture. This article also gathered inspira-

[1] `https://github.com/waitwaitforget/ExtremeLearningMachine`
[2] `https://github.com/claudio-rh/sequentialML_implementations`
[3] `https://github.com/chickenbestlover/Online-Recurrent-Extreme-Learning-Machine`
[4] `https://towardsdatascience.com/anomaly-detection-using-autoencoders-5b032178a1ea`

tion from an example present in the *TensorFlow* documentation. The structure of the employed AE is illustrated in Figure 5.2.

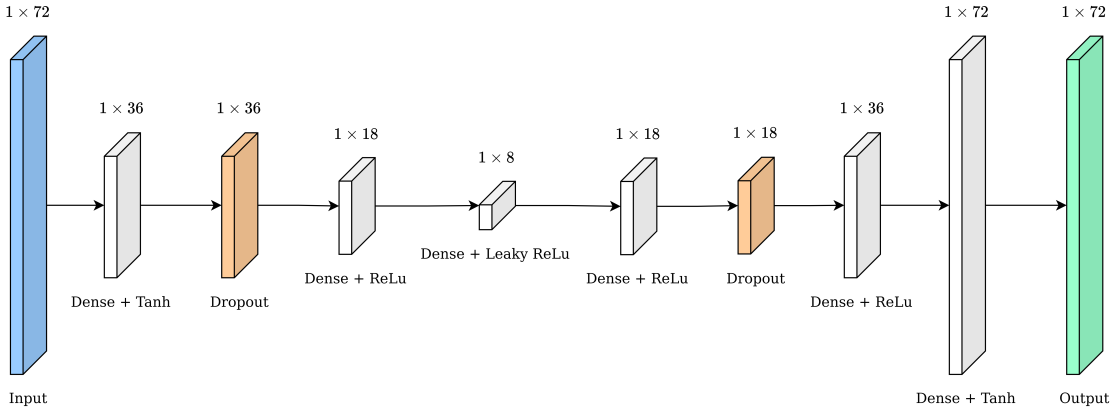


Figure 5.2: Structure of the employed AE. It includes each layer's type and activation function, as well as its dimensions.

The training phase incorporated a mechanism of early stopping, in order to prevent overfitting, as well as unnecessary epochs. The early stopping function tracks the validation loss and the validation data is a fraction of 15% of the training data. The following table represents the parameters chosen for the AE's training:

Parameter	Value
Learning rate	1×10^{-7}
Max. no. of epochs	1000
Mini batch size	5000
Loss	*Cosine similarity*
Optimizer	*Adam*
Dropout rate	0.2

Table 5.3: Training parameters of the AE architecture.

Two anomaly threshold metrics were tested. The first consisted in considering the mean reconstruction error obtained in the training phase. The second consisted in defining this threshold manually, by visualizing data and performing trial and error. The second metric led to better results, at the expense of being a non-automatic process. The source code utilized to define and train the AE can be consulted in Appendix IV.

5.3 Event Data Acquisition with CeleX5

After validating the methodology, the next step lied in acquiring event data using the CX5. The process of acquiring data with the camera was fairly straightforward, since the manufacturers provided sample codes that exemplified its functionalities. One of the samples codes exem-

plified exactly how to acquire and visualize events in a continuous fashion, by making use of a callback method that resembled the working principle of hardware interruptions. Every time a pixel or set of pixels were triggered by events in a specific sampling period, this callback received a buffer containing these events. Since the sample code was intended for visualization purposes, the logging of event data required an adaptation to this code. Hence, this adaptation consisted in writing event data to a *.txt* file.

Due to the high resolution of CX5, the callback received event buffers continuously over time, and their size could increase significantly when greater motion was present on the scene. This meant that to assure a fluid reception of event buffers, even when they escalated to hundreds of thousands of events in a sampling interval, not much computation should be made inside the callback method. Therefore, the writing of events to a file was only performed in the end of the program's execution, meaning that at runtime, event data was stored in a dynamic sized vector.

5.4 Optical Flow Extraction and Real Time Experiments

As acknowledged in the previous section, one may conclude the file format in which event data was stored, differed from the one present in the NeuroAED dataset (*.aedat*). This invalidated the possibility of using jAER's ABMOF algorithm to extract OF. Luckily, the same author of jAER's ABMOF algorithm, Min Liu, also had developed a C++ version of the algorithm. However, this implementation lacked some of the features present in jAER, namely recording features. This implementation mainly targeted visualization purposes, allowing users to send OF information over network protocols to another computer. Hence, such adaptations were made in order to save OF data to a *.txt* file. This task was reasonably demanding, since the author used advanced coding methodologies aiming at runtime optimization, relying substantially in hexadecimal words and bit-wise operations to encode variables. Moreover, minor bugs in the algorithm's implementation were fixed.

This first adaptation did not lead to good results, and after some discussion with the author, a decision of merging this code with another implementation of the same code was made. This led to far better results. Despite this fact, this code had a long computation time. For a NeuroAED Walking sample of roughly 20 seconds, it could take almost 10 minutes to extract its OF. Thus, it could not be employed in a real time application. Several algorithms were tested in the path to obtain a real time OF extraction method.

5.4.1 Real Time Experiments

The alternative algorithms that were tested were inspired in other commonly used event-based OF extraction algorithms, namely LK Flow and Local Plane Fits Flow. The mathematical background of these algorithms was provided in Chapter 2, section 2.2. In order to obtain real time behaviour, these algorithms were implemented directly in the callback method referenced in the previous section of this chapter. This allowed the algorithm to process event data at arrival time. However, recalling a previous section's disclaimer, the callback method did not perform properly when a slight amount of computation was added to it. The articles presenting these algorithms stated that these had real time performance, requiring less than $10\mu s$ to process each event. Nevertheless, despite all optimization attempts, these revealed to be too computationally expensive to be included in the callback. Every time a significant amount of events arrived in a buffer, the program's execution started lagging. When restricting the computation to an area smaller than the full resolution of the camera, the performance improved. However, to experience real time characteristics, this area needed to be so small that most of the scene dynamics were disregarded, resulting in a massive loss of relevant data.

In Chapter 4, section 4.1, it was said that CX5 featured built-in OF extraction capabilities. When analysing the data resulting from the built-in OF extraction method, it became evident that it was not being correctly calculated. Adjacent activated pixels were originating vectors with the same norm, but opposite directions, thus not complying with the motion pattern present on the scene. Further analysis on the camera's software allowed to conclude that the OF was being calculated based on an incomplete mathematical formulation of the original gradient-based OF formulation.

Lastly, an *OpenCV* method was explored, specifically, its Gunnar Farneback's algorithm implementation. This frame-based method required the construction of frames of events, defeating the purpose of event cameras and its advantages. Although it led to good OF estimation, it still experienced a lagging behaviour when incorporated with the callback containing event buffers. Conclusively, this method, as well as the other presented methods, was not able to perform in a real time fashion. Since the problem lied on the CX5's working principle in acquiring data, the real time behaviour was not accomplished. It is believed that in order to accomplish this feat, significant changes to CX5's base software scripts needed to be made, hence requiring a large time investment.

5.5 Downsampling

Due to its resolution, the amount of events captured by the CX5 is very extensive. This adds a significant workload to the processing of events when trying to implement an algorithm. More specifically, the OF extraction with such amounts of data becomes computationally expensive, hence requiring a mechanism to reduce those amounts of data. The most intuitive approach is downsampling, which consists essentially of a reduction of camera resolution. As resolution decreases, so the number of events decreases. Two downsampling algorithms were tested, namely $everyI$ and $window$, based in [30].

The first method consists of a very simple downscaling method that uses every i^{th} pixel, discarding all other pixels. This approach is very simple and does not require any additional algorithms, which may induce changes to the original data. In this work, only pixels with even coordinates were considered, resulting in a resolution decrease to $1/4$ of the original resolution. Figure 5.3 exemplifies the behaviour of this method.

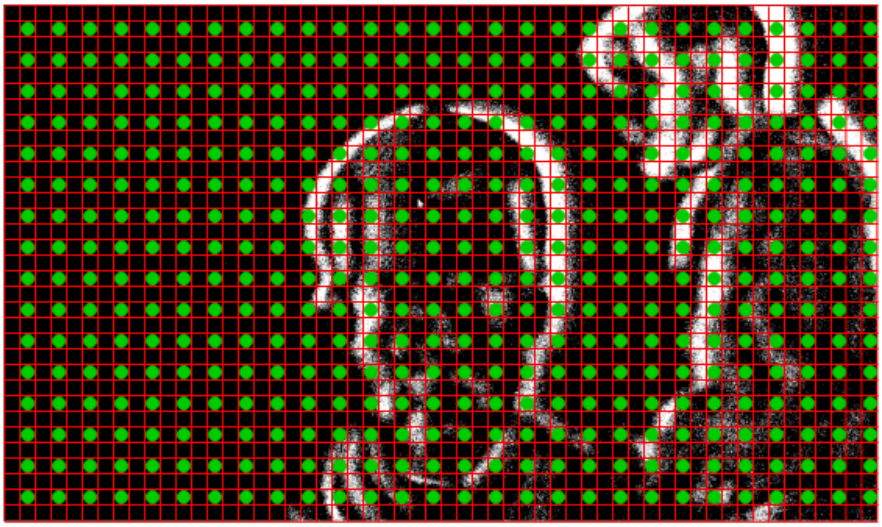


Figure 5.3: $everyI$ pixel selection method. Each square of this image represents a pixel and the green dots mark the utilized pixels, i.e., the ones with even coordinates.

The second method introduces the concepts of superpixel and sliding window. The general idea of $window$ method is to convert a region of pixels in the original data into one superpixel. Each superpixel corresponds to a pixel in the transformed data. To perform this conversion, a sliding window with the same size as the original resolution and length Δt is needed. Events within this window are considered as active. Then, superpixels are triggered if a certain amount of their neighbouring pixels are active. This event produces an event for the downscaled center pixel coordinates. Once Δt passes, the sliding window is reset and a new iteration is started. Typically, the superpixel's neighbourhood encloses nine pixels (center pixel and eight adjacent

pixels) and these neighbourhoods are non-overlapping. Figure 5.4 adds further understanding to this method.

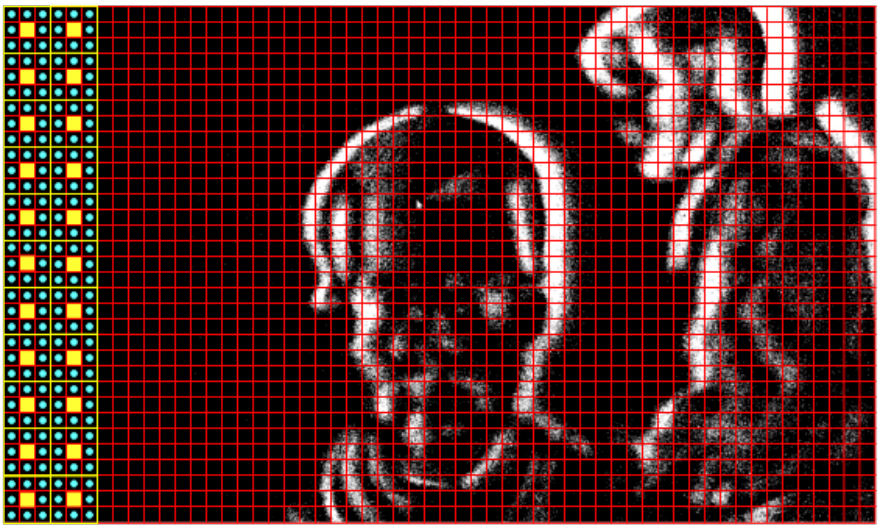


Figure 5.4: $window$ pixel selection method. Each uncoloured square represents a pixel and each region delimited in yellow represents a superpixel neighbourhood. The superpixel is marked with a yellow square and its neighbours are marked with light blue dots.

5.6 Data Acquisition Application

In order to facilitate the process of acquiring and processing event data with CX5, a Graphical User Interface (GUI) application was developed. The tool used to construct it was *Qt Creator*, an IDE that allows users to easily develop C++ GUI applications. Besides having recording capabilities, this GUI also included downsampling and playback features, in order to facilitate the access to these. Figure 5.5 shows the aspect of this application.

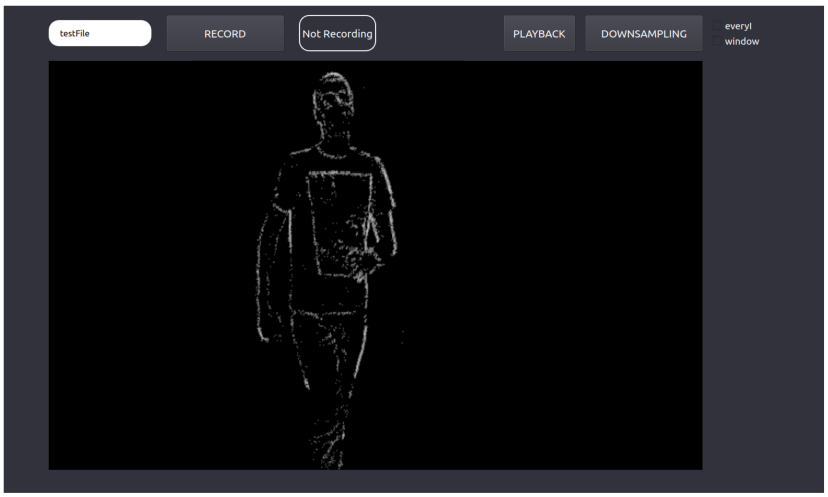


Figure 5.5: GUI application capable of acquiring events and saving them in a text file. It also allows the downsampling of data using two methods and the playback of downsampled files.

5.7 Recorded Datasets

In pursuance of evaluating the performance of the implemented methodology with data acquired by a different camera and in different conditions, two small-sized datasets were recorded: **Lab-AEDataset** and **DEEC-AEDataset**. Results regarding both datasets are revealed in the next chapter. The recording scenes can be consulted in Appendix V.

5.7.1 Lab-AEDataset

The first dataset, **Lab-AEDataset**, was recorded in the Computer Vision laboratory, located in Institute of Systems and Robotics-University of Coimbra (ISR-UC). This dataset only had one subject performing the activities. Normal activities consisted in walking in and out of the laboratory, in any direction. Abnormal activities consisted in running in and out of the laboratory, in any direction. In total, seven normal samples and five abnormal samples were recorded. Similarly to NeuroAED dataset, the subject moved on a geometric plane parallel to that of the camera.

5.7.2 DEEC-AEDataset

The second dataset, **DEEC-AEDataset**, was recorded in one of Departamento de Engenharia Electrotécnica e de Computadores-University of Coimbra's (DEEC-UC's) corridors. This dataset included the presence of seven subjects. Normal activities consisted in walking along the corridor in a specific direction. Abnormal activities consisted in walking along the corridor in the opposite direction and running in any direction. In total, five normal samples and nine abnormal samples were recorded. In contrast with **Lab-AEDataset**, this dataset contained more anomalous activity types to detect, which posed a bigger challenge. Contrary to previously discussed datasets, here the subjects moved on a geometric plane perpendicular to that of the camera.

Chapter 6

Experimental Results

This chapter reveals the experimental results obtained during the development of this dissertation. It starts by showing the results obtained for the different ML methods in the methodology validation with NeuroAED dataset and continues by showing the results obtained for the subsequently recorded datasets.

6.1 Methodology Validation and Added Machine Learning Architectures

After using jAER for the first time, there was a certain level of uncertainty about the tuning of ABMOF's parameters. Therefore, the first results obtained were somewhat underwhelming. The first OF estimation considered a number of scales of 2 units and a block dimension of 21 pixels. The cuboid descriptors extracted in the first test had the following parameters:

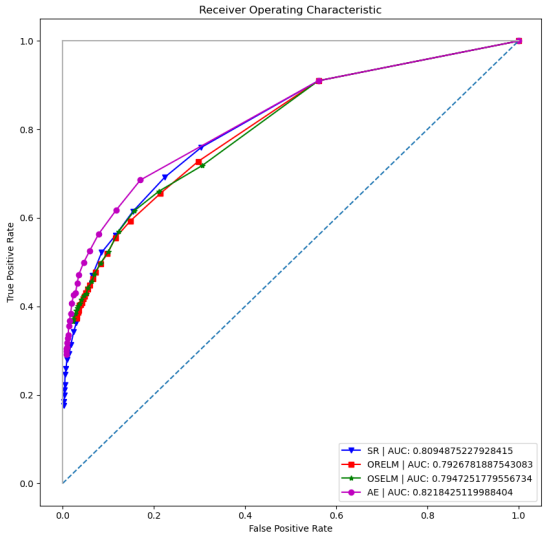
Parameter	Value
Cuboid area ($pixels$)	18×14
No. of events for activation	400
Slice duration (ms)	100

Table 6.1: Parameters used to extract the cuboid descriptors in the first test.

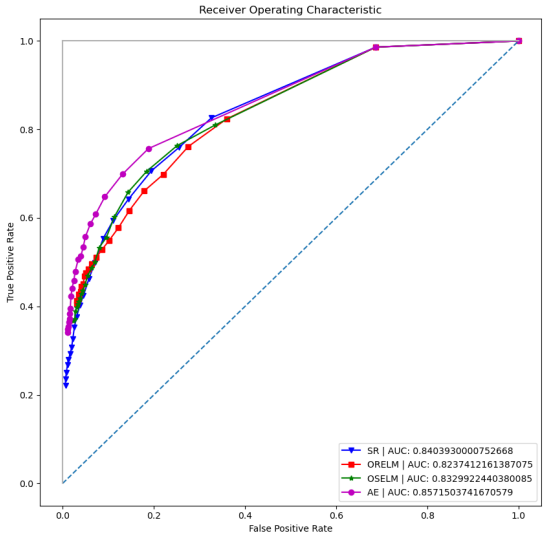
The Receiver Operating Characteristic (ROC) curve and Area Under Curve (AUC), for the abnormal samples, are showcased in Figure 6.1a.

In order to produce superior results, the OF algorithm's parameters were altered. This time, a number of scales of 3 units was considered, and the block dimension was reduced to 9 pixels. Decreasing the block's dimension, the algorithm performs more fine-grained estimations, because it is bounded to smaller regions of pixels. Increasing the number of scales, targets the mitigation of scale variability present in event data. This change of parameters originated slightly better results, which are presented in Figure 6.1b.

At this stage, the C++ version of ABMOF was tested. The tuning of the parameters was



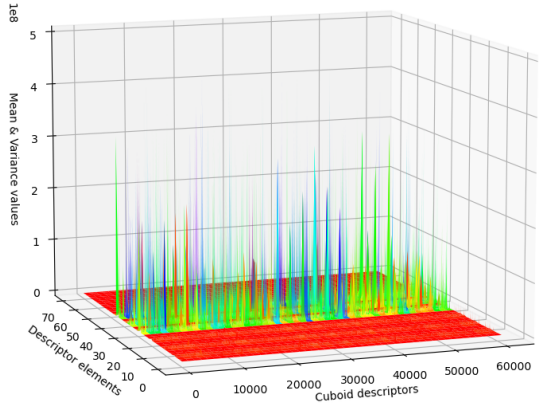
(a) ROC curve and AUC obtained for all the ML models. The AE model performed better than the other models, with an AUC of 82.18% in abnormality detection. With regards to the SR model, these results were far from the ones obtained in the original study, which claimed an AUC of 95.8% for the same dataset. This pointed out that some parameters in OF estimation or cuboid descriptor calculation needed to be tuned more accurately.



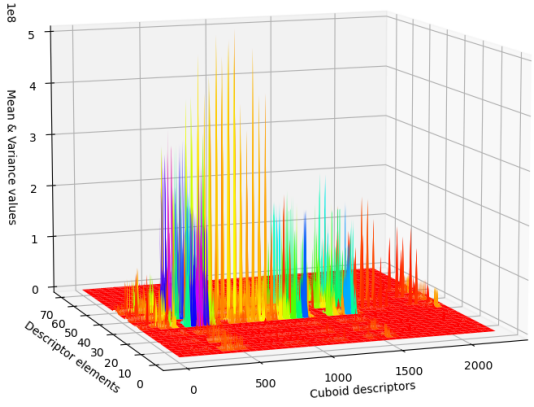
(b) ROC curve and AUC obtained for all the ML models. In this test, the AE model continued to perform better than the other models, with an AUC of 85.72% in abnormality detection. With regards to the SR model, these results were still far from the ones obtained in the original study, which implied that more adjustments needed to be made.

Figure 6.1: First ROC curves and AUCs obtained for the ML models in the described conditions.

supported by the graphical representation of the originated cuboid descriptors, which allowed to interpret their quality before using them in the models. The difference between the descriptors, describing normal and abnormal activities is illustrated in Figure 6.2.



(a) Normal cuboid descriptors used for training the ML models.

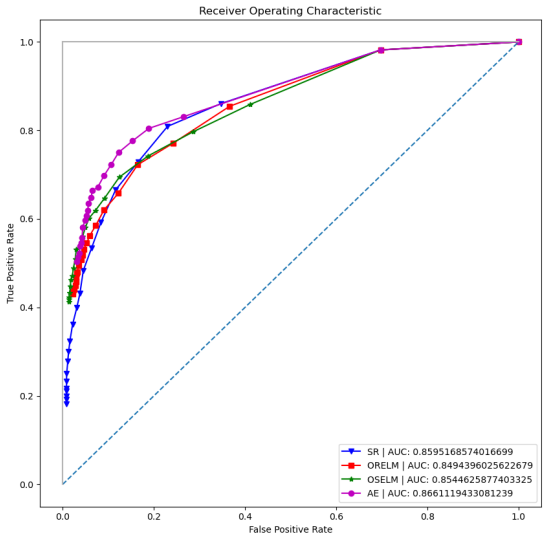


(b) Example of a sample containing abnormal cuboid descriptors to be used in the test phase.

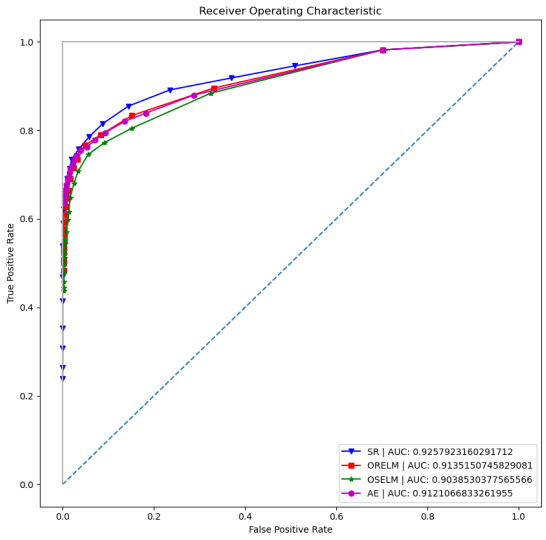
Figure 6.2: Comparison between training data (normal cuboid descriptors) and a test sample containing abnormal cuboid descriptors. These plots evidence the differences between both types of descriptors, as abnormal samples contain higher values in the slices containing anomalous events.

The first ROC curve obtained for the new implementation of ABMOF, considering a block

dimension of 10 pixels and a number of scales of 3 units, while maintaining the same parameters presented in Table 6.1 for the cuboid descriptors, is illustrated in Figure 6.3a. The visualization of the OF estimation from the C++ ABMOF, allowed to conclude that the estimation was not being performed correctly. After fixing the errors, the results improved significantly, as expected. The resulting ROC curve is shown in Figure 6.3b. The comparison between the OF data obtained before and after the correction are shown in Figure 6.4.

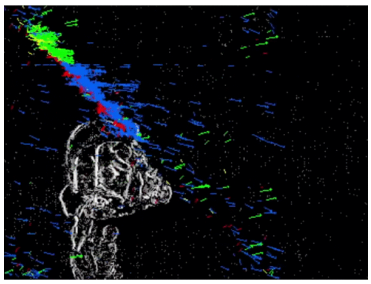


(a) ROC curve and AUC obtained for all the ML models. The AE model continued to perform better than the other models, with an AUC of 86.61% in abnormality detection.

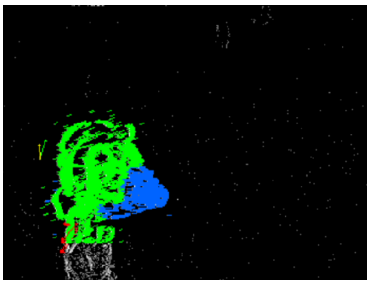


(b) ROC curve and AUC obtained for all the ML models. For the first time, the SR model performed better than the other models, with an AUC of 92.58% in abnormality detection, getting closer to the results obtained in the original study.

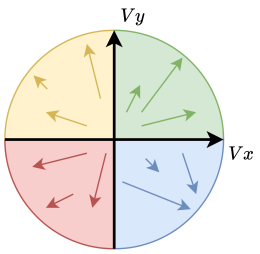
Figure 6.3: ROC curves and AUCs obtained for the ML models using OF estimation provided by the C++ implementation of ABMOF.



(a) OF estimation using first version of C++ ABMOF.



(b) OF estimation after adjusting the first version of C++ ABMOF.



(c) OF vectors' direction colour code.

Figure 6.4: Comparison between OF estimations, and OF vectors' direction colour code.

Aiming towards a more complete analysis of the AED results, more data analysis metrics were introduced: accuracy for each test sample, computation time of processing all test samples and the number of false anomalies detected in normal samples. In all ML models, the

accuracy was calculated for the abnormality threshold determined in the training phase. The plot containing these values and the computation times is presented in Figure 6.5

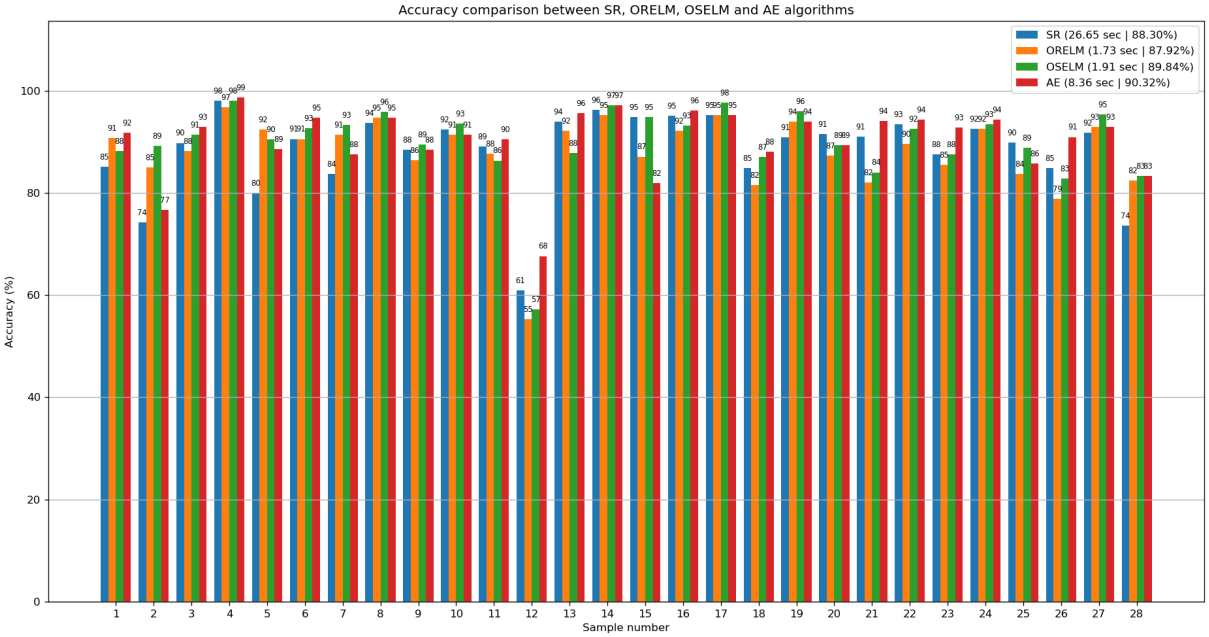


Figure 6.5: Accuracy and computation times obtained for the NeuroAED test samples. As seen, the AE model led to better results, with an average accuracy of 90.32%. The ELM models are the fastest with regards to computation time, with an elapsed time of 1.90 seconds for the OS-ELM and 2.96 seconds for the OR-ELM models.

Figure 6.5 might suggest that the OS-ELM and OR-ELM architectures are the ones which had better overall performance. However, by analysing the frequency of wrongfully detected anomalies in normal samples, it can be concluded that these are the architectures that lead to a bigger number of false positives. This analysis is illustrated in Figure 6.6.

The last analysis leads to the deduction that the ELM-based models struggle in distinguishing normality from abnormality, while the other two models are able to correctly distinguish both. The visualization of anomalous events is exemplified in Figure 6.7.

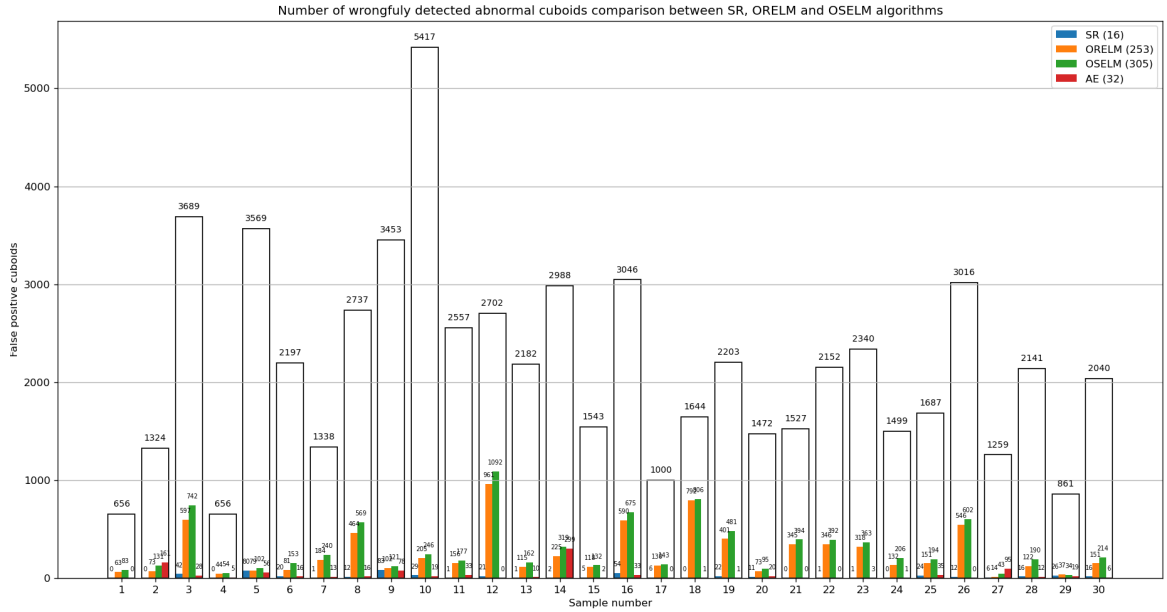
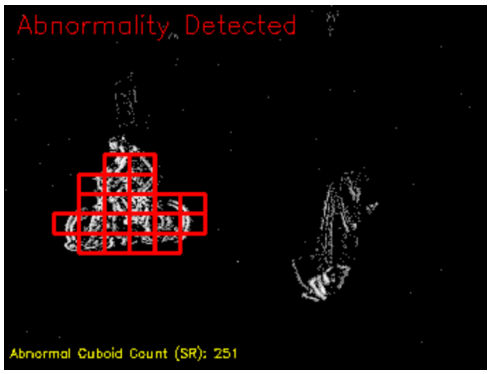
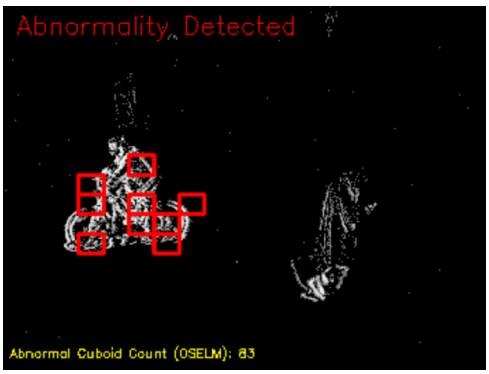


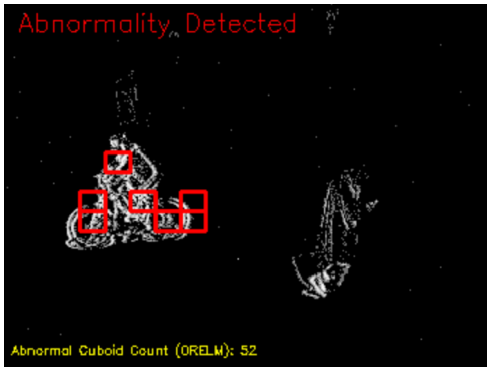
Figure 6.6: Number of false positives detected by each model for all normal samples. In this graph, the best performance is verified by the SR model, with an average of 16 false positives. The worst performing models are the ELM-based models, with averages surpassing 200 false positives. The colourless bars indicate the total number of cuboids in each sample.



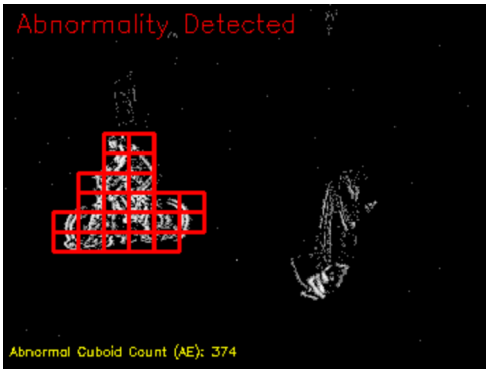
(a) Anomalous cuboids detected by SR model.



(b) Anomalous cuboids detected by OS-ELM model.



(c) Anomalous cuboids detected by OR-ELM model.



(d) Anomalous cuboids detected by AE model.

Figure 6.7: Example of anomalous event visualization in a test sample with a person cycling in the scene. People walking in the scene do not trigger any abnormal cuboids, whereas a person cycling, does. In some cases, people walking may trigger abnormal cuboids, hence the number false positives referenced in Figure 6.6.

6.2 Real Time Optical Flow Experiments

As stated in the previous chapter, the task of estimating OF in real time, using event data recorded by CX5, was not accomplished, despite countless efforts. Firstly, the OF estimation mode of CX5 provided an erroneous estimation, which is illustrated in Figure 6.8.

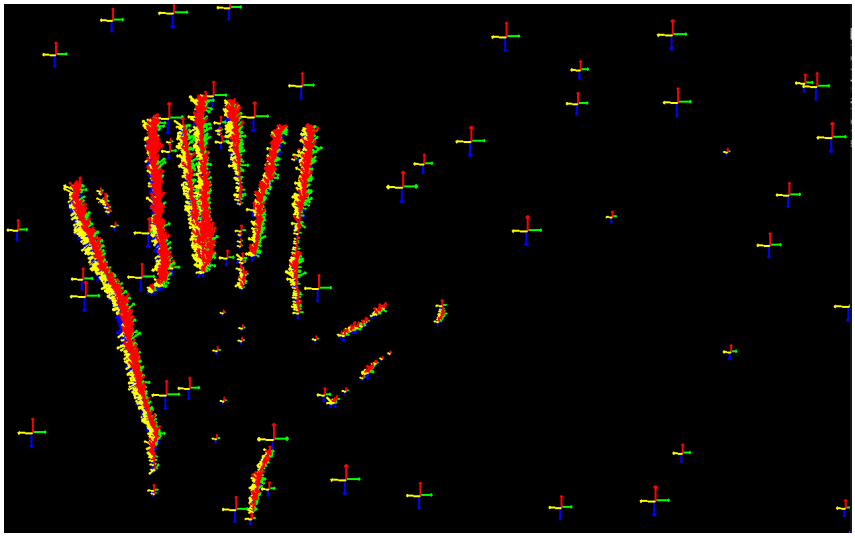
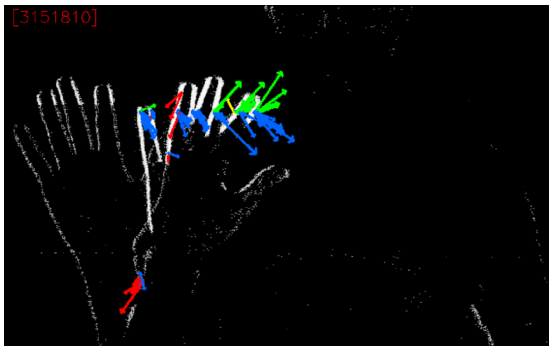
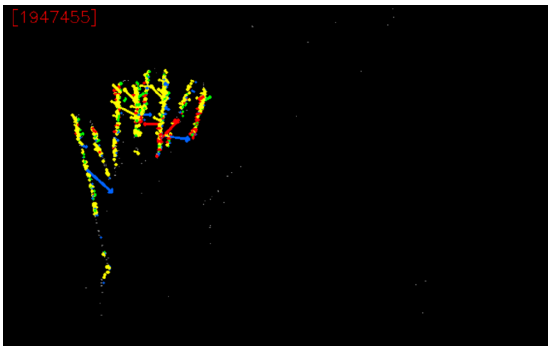


Figure 6.8: OF estimation obtained from CX5 OF mode. By looking at the OF generated by noisy events, it is noticeable that the camera's algorithm calculates the vertical and horizontal derivatives over the adjacent pixels of the triggered events. This causes the generation of OF vectors with opposite directions for a neighbourhood of events. In this example, the hand was moving from the right to the left of the frame, meaning it should only generate yellow or red vectors, which is not the case.

The following experiments were to implement the LK Flow method and Local Plane Fits Flow. None of them performed in real time, as asserted in section 5.4.1 of the previous chapter. The results obtained in both algorithms are displayed in Figure 6.9.



(a) LK Flow estimation



(b) Local Planes Fits Flow estimation

Figure 6.9: OF estimations performed by both algorithms. In **(a)**, the hand was moving from the left to the right of the frame. Although there were correct estimations, these were strongly affected by the overhead of gathering events and estimating OF directly in the acquisition function. In **(b)**, the hand was moving from the right to the left of the frame and similarly, as before, the estimations were deeply affected by the algorithm's overhead. Nevertheless, this revealed to be less computationally expensive when compared to the latter.

Finally, the Gunnar Farneback's algorithm was tested. Despite not having real time characteristics, this algorithm produced decent OF estimations, which are illustrated in Figure 6.10.

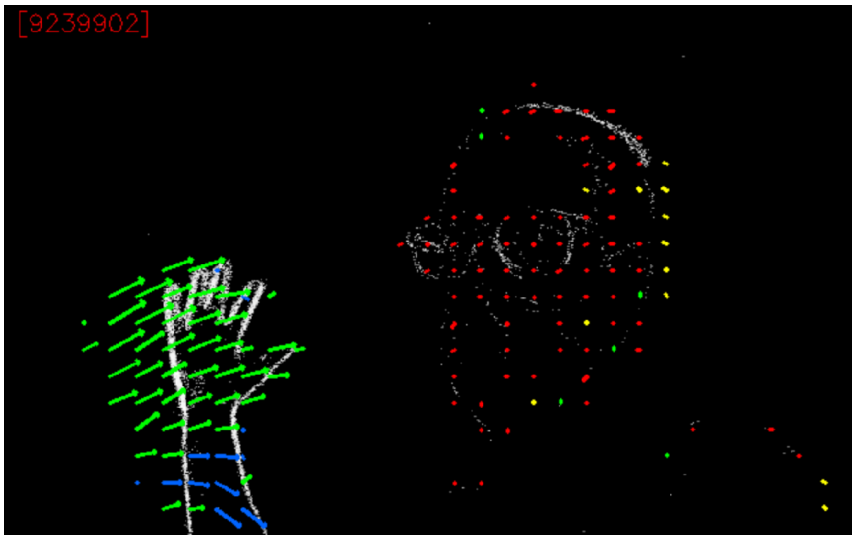


Figure 6.10: OF estimation obtained from Gunnar Farneback's *OpenCV* function. The vectors are estimated correctly, however, these are computed in a grid format, reducing the density of the estimated OF.

Unfortunately, the OF estimation in real time was not accomplished due to a key factor: the CX5's acquisition method API. The acquisition method used by the camera returns a buffer of events at a specific rate. The default rate for buffer arrival is $30\ ms$. Increasing reductions of this rate induce increased lagging in the reception of event buffers. Intuitively, the amount of events triggered in the scene amplifies this issue, since it increases the buffer's size. When the access to the buffer's data surpasses the rate of buffer arrival, the API displays a message warning that the buffer is full. When this occurs, the information being accessed belongs to prior time events, rather than actual time events.

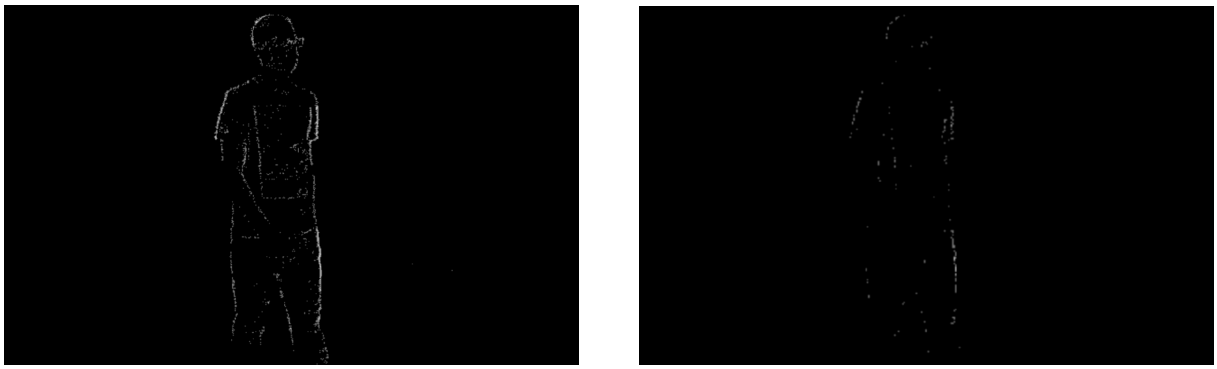
Likewise, when trying to perform computations of arriving event buffers, a lagging effect is also experienced. In state of the art algorithms, like the tested ones, the calculations needed to estimate OF are enough to trigger this behaviour in the API. The articles presenting the OF algorithms state that the estimations can be performed at a rate of units of microseconds per event. Nonetheless, this triggered the lagging behaviour in the acquisition API, confirming that the issue lies in the latter, instead of the algorithms themselves. To further analyse this behaviour, a small squared region was defined in the middle of the frame and OF estimations were bounded to events inside that region. The lagging effect lowered in function of the area of that region, meaning the callback was able to handle calculations in a small set of events.

Despite not achieving the expected results in this phase, it is worth mentioning the issues encountered in the various experiments. This information is valuable and may help the devel-

oping of future implementations of such applications. In conclusion, performing real time OF estimations using CX5 requires significant modifications of the functions supporting the API. The acquisition method needs to be adapted in order to support data processing alongside data acquisition. Most probably, such adaptations may demand a huge time investment, since they require an extensive analysis of the underlying functions supporting the API, having little backing documentation.

6.3 Lab-AEDataset

As claimed in the previous chapter, the data acquired by CX5 was immense, due to its large resolution. Therefore, two downsampling strategies were implemented with the objective of reducing the amount of data supplied to the entire framework, hence reducing computation times substantially. The output of both strategies is illustrated in Figure 6.11



(a) Downsampling using $everyI$ method.

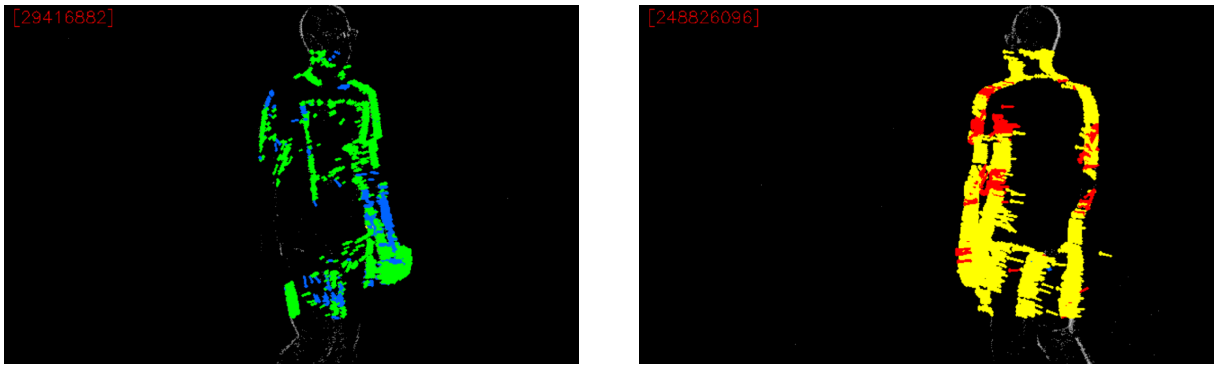
(b) Downsampling using $window$ method.

Figure 6.11: Comparison between downsampling methods. The *window* method reduces the resolution by 3 times in each axis, while $everyI$ method reduces the resolution by 2 times in each axis. The downsampling performed by $window$ results in an exaggerated loss of information, rendering event information almost imperceptible.

It is important to notice that the density of events deeply impacts the quality of OF estimation, because fewer events make it difficult to accomplish block matching in the ABMOF algorithm. From here on, the downsampling was always executed with $window$ method. The OF estimation obtained is exemplified in Figure 6.12.

The data recorded by CX5 is less dense than the one recorded by DV346, with the downsampling amplifying this issue. This fact, along with the increase of resolution when compared to DV346, even after downsampling, required an adjustment of the parameters tuning the cuboid descriptors' calculation algorithm. The cuboid area was increased, and the minimum number of events to activate a cuboid was decreased. The new parameters, which originated better results, are presented in Table 6.2.

The ROC curve for abnormality detection, generated with these descriptors, is shown in



(a) Sample of a subject walking from left to right.

(b) Sample of a subject walking from right to left.

Figure 6.12: Two examples of OF estimation performed in **Lab-AEDataset** samples. The vectors' direction colour code is the same as presented in Figure 6.4c.

Parameter	Value
Cuboid area ($pixels$)	24×20
No. of events for activation	100
Slice duration (ms)	100

Table 6.2: Parameters used to extract the cuboid descriptors from **Lab-AEDataset** samples.

Figure 6.13.

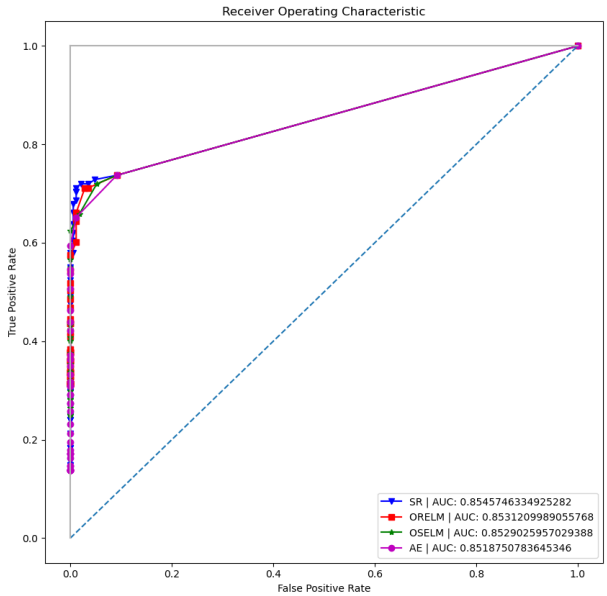


Figure 6.13: ROC curve and AUC obtained for all the ML models. The SR model performed better than the other models, with an AUC of 85.46% in abnormality detection.

In terms of accuracy with the training abnormality thresholds, and computation times, the results are disclaimed in Figure 6.14. The number of false positives detected in normal samples is displayed in Figure 6.15. For a sample containing anomalous events, the abnormal event visualization is showcased in Figure 6.16.

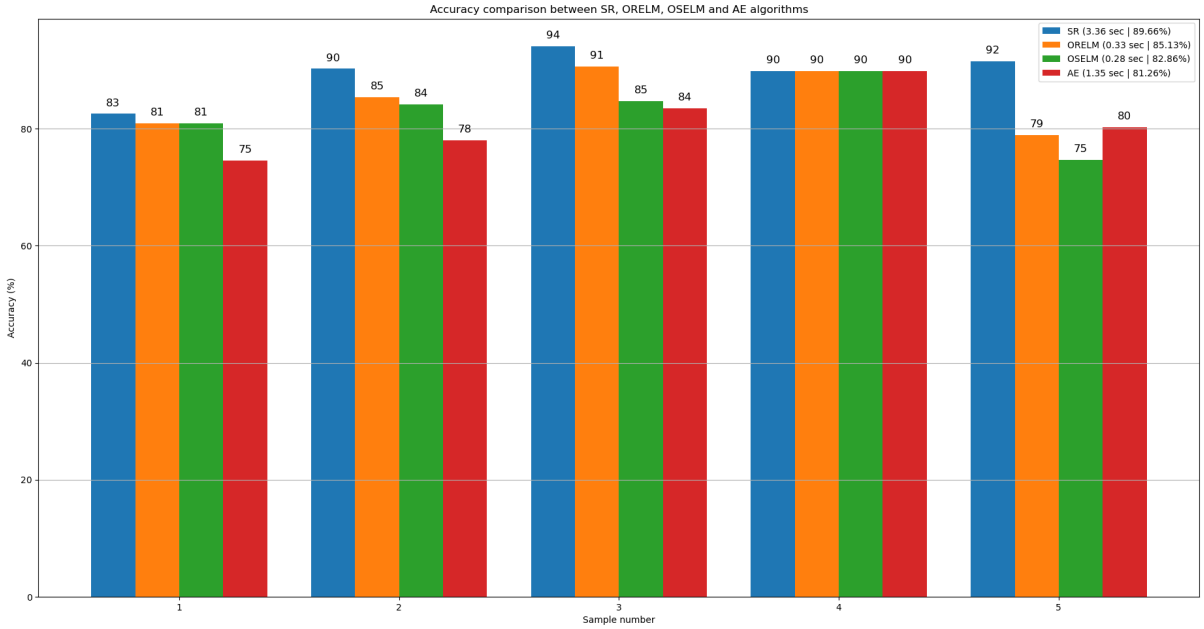


Figure 6.14: Accuracy and computations times obtained for the **Lab-AEDataset** test samples. As seen, the SR model produced better results, with an average accuracy of 89.66%. The ELM models continue to be the fastest with regards to computation time, with an elapsed time of 0.33 seconds for the OR-ELM and 0.28 seconds for the OS-ELM models.

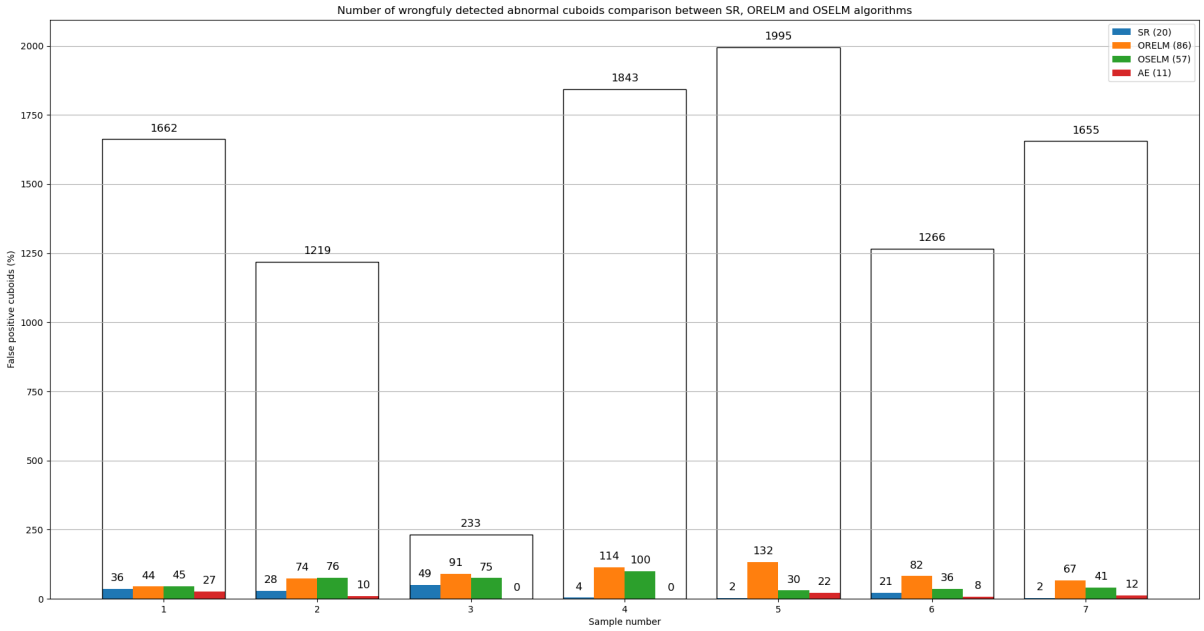
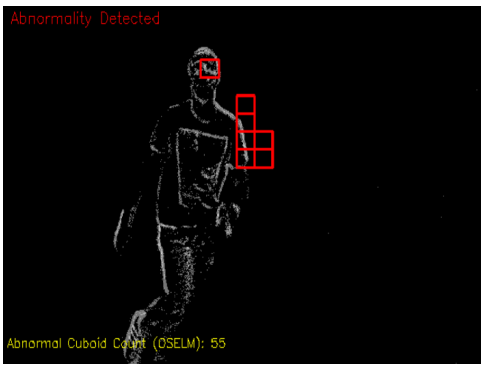


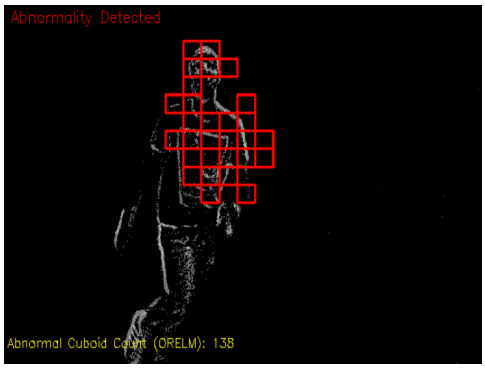
Figure 6.15: Number of false positives detected by each model for all normal samples. In this statistic, the best performance is verified by the AE model, with an average of 11 false positives. The worst performing models are the ELM-based models, with averages surpassing the 50 false positives. The colourless bars indicate the total number of cuboids in each sample.

In this dataset, the results obtained were not as satisfying as the ones obtained for NeuroAED dataset. Possible motives could lie in the small size of the dataset and/or in the quality of event data recorded by CX5. There is evidence that points towards the lens mounted in the camera being one of the main factors jeopardizing event data quality. The focal point of this lens

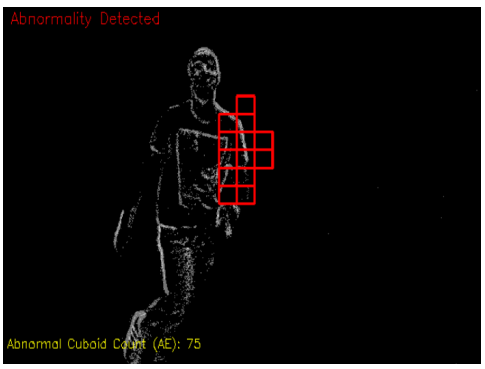
(a) Anomalous cuboids detected by SR model.



(b) Anomalous cuboids detected by OS-ELM model.



(c) Anomalous cuboids detected by OR-ELM model.



(d) Anomalous cuboids detected by AE model.

Figure 6.16: Example of anomalous event visualization in a test sample with a person running in the scene.

is very precise, meaning whenever the subject deviates slightly from this point, the detection of event data becomes inferior. In this dataset, the subject moves in an approximately parallel plane to that of the camera, thus sometimes getting further or closer to the camera, causing variations in the capturing of events.

6.4 DEEC-AEDataset

This dataset, on the contrary to previous datasets used in this work, contains scenes were the subjects are moving in a plane perpendicular to that of the camera. The objective here was to infer if these recording conditions affected the accuracy of AED. The camera's standpoint was higher in height in comparison to the floor plane where subjects were moving. The latter, alone, impacted the quantity of recorded events, a problem that would be amplified by downsampling. Figure 6.17 shows a sample of this dataset.

The low density of event data had a negative effect on OF estimation. Although the algorithm computed estimations, these were not as good as the ones experienced in the previous datasets. An example of the estimated OF is given by Figure 6.18.

Given the reduced density of events, the parameters of the cuboid descriptor generating

Figure 6.17: DEEC-AEDataset sample. The quantity of events has highly decreased in comparison to the previous datasets. This has to due with the height at which the camera was positioned with respect to the plane of movement, and also to the lens' focal point, an issue discussed in the previous section. Notice, however, that the camera was positioned a floor above the plane of movement, thus not being in a very high position.

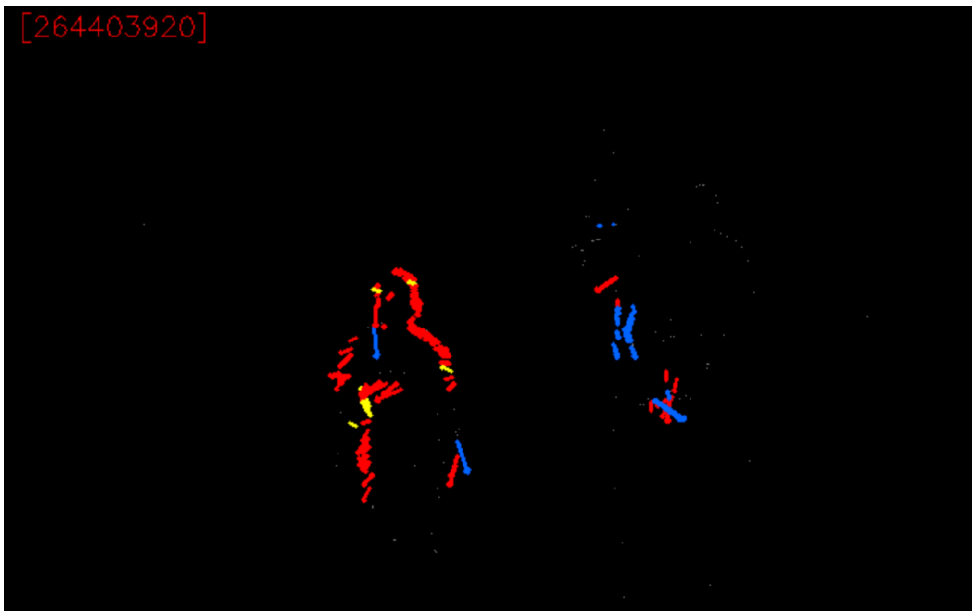


Figure 6.18: Examples of an OF estimation in **DEEC-AEDataset**. The quantity of estimations is far less when compared to the previous datasets. In comparison with **Lab-AEDataset**, this had the camera positioned further away from the subjects, therefore recording less events, which led to worse OF estimations.

algorithm needed to be tuned again, more specifically, the minimum number of events to activate a cuboid. The parameters used are referenced in Table 6.3.

The abnormality detection ROC curve obtained in this dataset is presented in Figure 6.19.

In terms of accuracy, with the training abnormality thresholds, and computation times, the

Parameter	Value
Cuboid area ($pixels$)	24×20
No. of events for activation	5
Slice duration (ms)	100

Table 6.3: Parameters used to extract the cuboid descriptors from **Lab-AEDataset** samples.

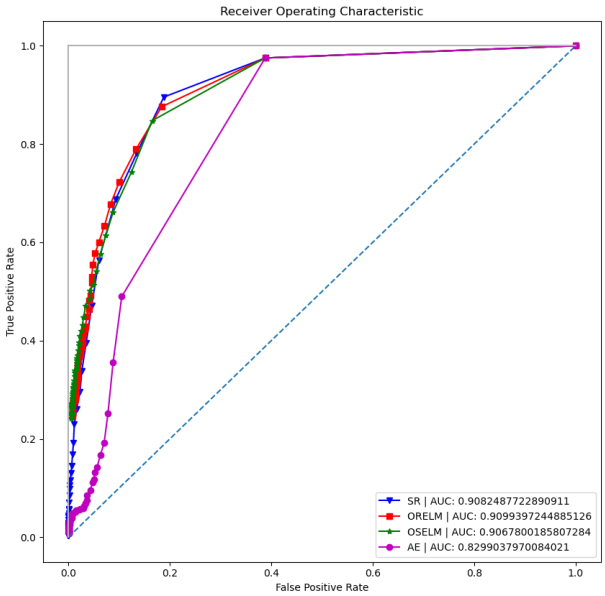


Figure 6.19: ROC curve and AUC obtained for all the ML models. The OR-ELM model performed better than the other models, with an AUC of 90.99% in abnormality detection. In this dataset, the AE model experienced a huge loss in performance, compared to previous datasets.

results are presented in Figure 6.20.

The number of false positives detected in normal samples is displayed in Figure 6.21.

Figure 6.22 exemplifies the anomalous event visualization for a sample containing anomalies.

The results obtained in this dataset reveal that the density of recorded events has paramount importance concerning the quality of AED. Fewer events generate worse OF estimations, thus leading to worse cuboid descriptors, and ultimately to an inadequate learning of normal activities by the models.The results obtained in this dataset reveal that the density of recorded events has paramount importance concerning the quality of AED. Fewer events generate worse OF estimations, thus leading to worse cuboid descriptors, and ultimately to an inadequate learning of normal activities by the models.

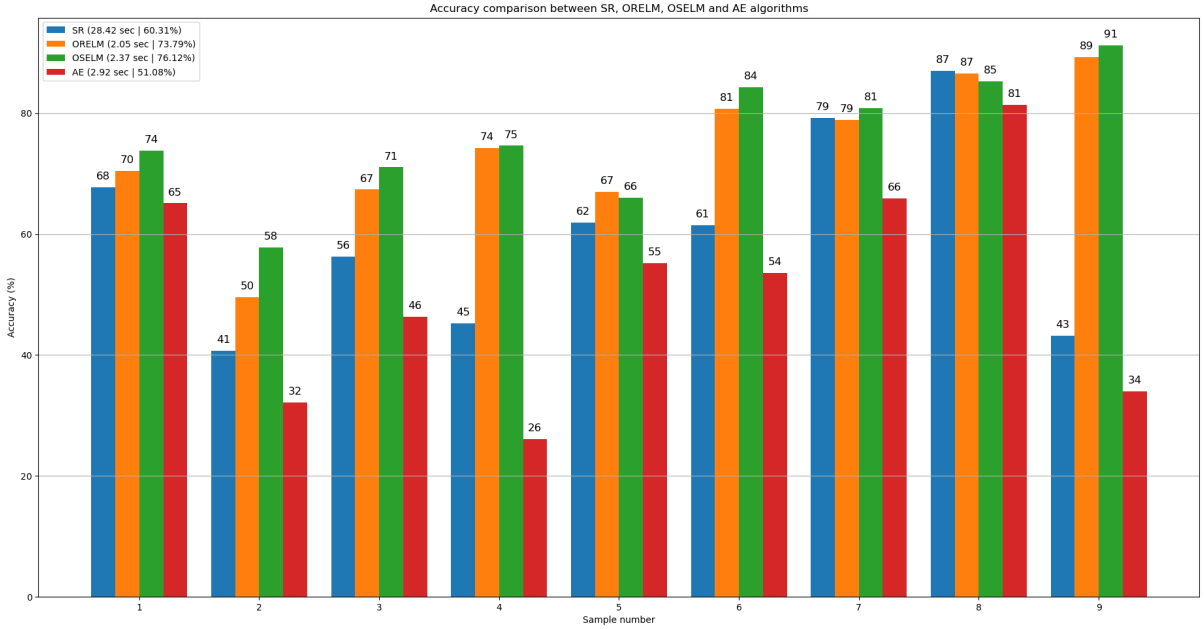


Figure 6.20: Accuracy and computations times obtained for the **DEEC-AEDataset** test samples. As seen, the OS-ELM model produced better results, with an average accuracy of 76.12%. The ELM models were still the fastest with regards to computation time, with an elapsed time of 2.05 seconds for the OR-ELM and 2.37 seconds for the OS-ELM models.

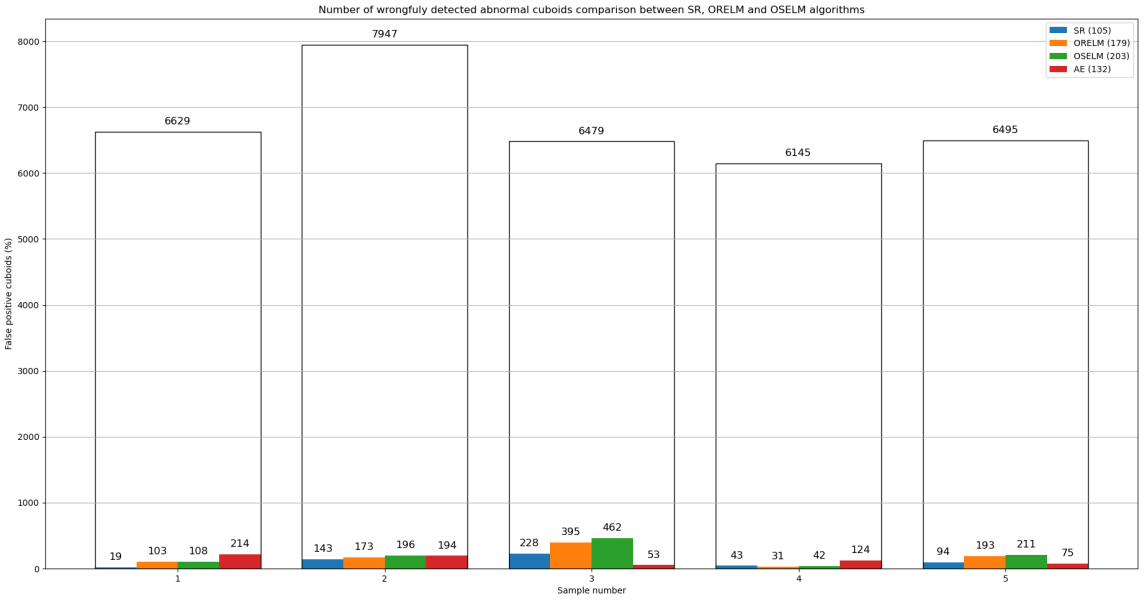
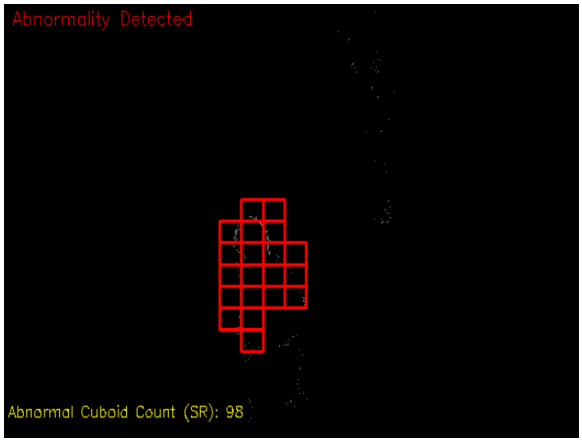
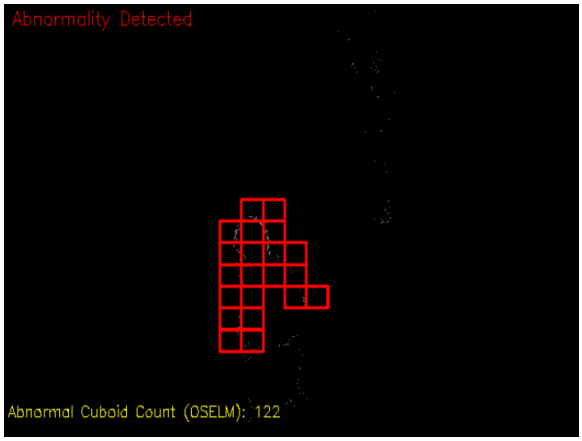


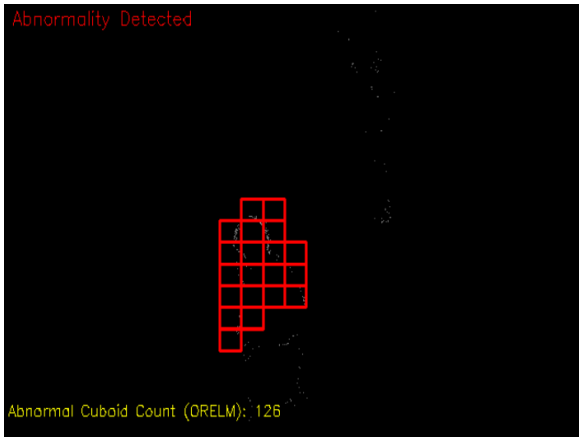
Figure 6.21: Number of false positives detected by each model in all normal samples. In this statistic, the best performance is verified by the SR model, with an average of 105 false positives. The worst performing models are the ELM-based models, with averages rounding 200 false positives. The colourless bars indicate the total number of cuboids in each sample.



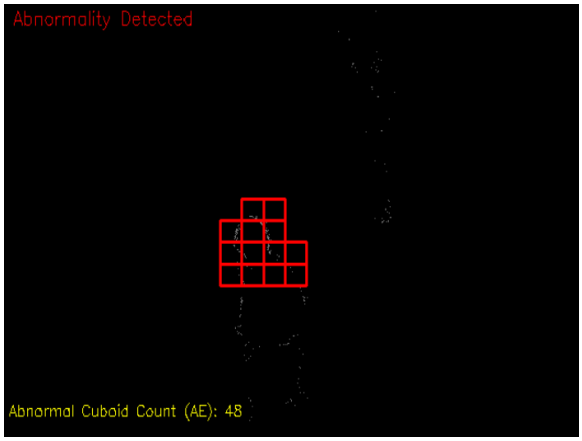
(a) Anomalous cuboids detected by SR model.



(b) Anomalous cuboids detected by OS-ELM model.



(c) Anomalous cuboids detected by OR-ELM model.



(d) Anomalous cuboids detected by AE model.

Figure 6.22: Example of anomalous event visualization in a test sample with subjects walking in opposite direction to the correct one.

6.5 Computation Times

The computations times for processing and preparing the files, from **Lab-AEDataset** and **DEEC-AEDataset**, and those for training the models are depicted in the following tables.

		Samples	Normal							Abnormal				
			N1	N2	N3	N4	N5	N6	N7	ABN1	ABN2	ABN3	ABN4	ABN5
	Clip duration (s)		10.63	9.32	12.94	15.71	20.13	13.99	17.02	6.17	8.06	8.31	6.79	6.98
Computation time (s)	Optical Flow		120.86	137.68	19.55	71.80	87.26	72.87	134.45	180.13	204.38	190.31	31.07	66.57
	Cuboid Descriptors		116.13							131.16				
	Training	SR	18.12							NA				
		OSELM	1.52							NA				
		ORELM	2.04							NA				
		AE	20.36							NA				

Table 6.4: Computation times for estimating OF, calculating cuboid descriptors and training the ML models with data from **Lab-AEDataset**. The clip duration of each sample is also included.

		Samples	Normal					Abnormal								
			N1	N2	N3	N4	N5	ABN1	ABN2	ABN3	ABN4	ABN5	ABN6	ABN7	ABN8	ABN9
	Clip duration (s)		31.98	33.23	26.28	22.05	24.54	14.72	32.57	18.85	27.03	31.31	16.41	30.68	22.97	20.43
Computation time (s)	Optical Flow		80.86	96.45	81.96	77.14	87.47	21.34	83.25	65.99	93.13	89.69	38.16	103.04	70.68	60.78
	Cuboid Descriptors		71.83					90.56								
	Training	SR	56.86					NA								
		OSELM	7.58					NA								
		ORELM	7.33					NA								
		AE	35.57					NA								

Table 6.5: Computation times for estimating OF, calculating cuboid descriptors and training the ML models with data from **DEEC-AEDataset**. The clip duration of each sample is also included.

Once more, the results show that the architectures based in the ELM are the fastest to train, while the SR model is the slowest overall. The time difference existing between the OF estimation and cuboid descriptors calculation, in the two datasets, reveals that **Lab-AEDataset** has a higher density of events in its samples. As verified, this is true due to the fact that these samples were recorded from a scene closer to the camera, when compared to **DEEC-AEDataset**. All experiments were done using a PC equipped with an *Intel Core i7 6700HQ 2.60GHz* CPU and 24GB of RAM.

Chapter 7

Conclusions and Future Work

Lastly, one can conclude that an approach using OF to perform AED has proved effective. Furthermore, the added ML models have proven to perform almost as effectively as the original SR model employed. The only drawback verified was that, despite being faster, they would trigger more false positives when compared to the original model. This issue was more accentuated in the ELM models.

Another key aspect, that should be mentioned, is that the quality of the OF estimation deeply affects the results obtained when using this framework. Moreover, one can affirm that it induces a snowball effect in the framework's pipeline, because poor OF data leads to poor cuboid descriptors, hence affecting the modelling capacity of the ML architectures. It should be noted that poor OF estimation can result from deficient event data obtained from the neuromorphic sensor or from an inaccurate tuning of the algorithm's parameters. In the two created datasets, the density of event data was greatly inferior when compared to the NeuroAED dataset. The **Lab-AEDataset**, although having decreased event density, still had enough density to perform reasonable OF estimations. On the other hand, the **DEEC-AEDataset** did not have OF estimations as satisfactory as the latter, due to the lack of event density in the recordings, caused by a distant camera standpoint with relation to the plane of movement and to the precise lens' focal point.

Regarding real time behaviour, the CX5 camera's software needs significant adjustments in order to perform the acquisition of events and advanced processing of these in a real time manner. Currently used methods are capable of acquiring events, but unfit when it comes to performing moderate computations of the acquired data. The function the camera is using to read events receives a buffer at a specific rate and this buffer is assembled pixel-wise, instead of time-wise. This means that the events arrive sorted by pixel coordinates, instead of timestamp. The OF algorithms rely heavily on time relationship between pixels, thus requiring buffer event data to be ordered by timestamp, a process that is moderately costly in itself. Adding the estimation algorithms further increases the overhead present in the event acquisition function,

causing severe performance issues, namely lagging and acquisition break stops.

In terms of future work, further improvements of the framework's accuracy across the different ML models, could be studied, while also studying strategies that could bring real time characteristics to the entire framework described in this dissertation. The latter, would most likely require a thorough analysis of CX5's software and firmware, in order to adjust the procedure the camera uses to process and transmit event data.

References

[1] G. Gallego, T. Delbruck, G. M. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020. DOI: 10.1109/TPAMI.2020.3008413.

[2] B. Rueckauer and T. Delbruck, "Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor," *Frontiers in neuroscience*, vol. 10, p. 176, 2016.

[3] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International journal of computer vision*, vol. 12, no. 1, pp. 43–77, 1994.

[4] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision," Vancouver, British Columbia, 1981.

[5] R. Benosman, S.-H. Ieng, C. Clercq, C. Bartolozzi, and M. Srinivasan, "Asynchronous frameless event-based optical flow," *Neural Networks*, vol. 27, pp. 32–37, 2012.

[6] T. Brosch, S. Tschechne, and H. Neumann, "On event-based optical flow detection," *Frontiers in neuroscience*, vol. 9, p. 137, 2015.

[7] D. J. Thornley, "Anisotropic multidimensional savitzky golay kernels for smoothing, differentiation and reconstruction," *Department of Computing Technical Report*, vol. 8, 2006.

[8] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 2, pp. 407–417, 2013.

[9] M. Liu and T. Delbrück, "ABMOF: A novel optical flow algorithm for dynamic vision sensors," *CoRR*, vol. abs/1805.03988, 2018. arXiv: 1805.03988. [Online]. Available: http://arxiv.org/abs/1805.03988.

[10] G. Chen, P. Liu, Z. Liu, H. Tang, L. Hong, J. Dong, J. Conradt, and A. Knoll, "Neuroaed: Towards efficient abnormal event detection in visual surveillance with neuromorphic vision sensor," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 923–936, 2021.

[11] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[12] J.-M. Park and J.-H. Kim, "Online recurrent extreme learning machine and its application to time-series prediction," in *2017 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2017, pp. 1983–1990.

[13] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on neural networks*, vol. 17, no. 6, pp. 1411–1423, 2006.

[14] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong, "Representational learning with elms for big data," 2013.

[15] J. Zhai, S. Zhang, J. Chen, and Q. He, "Autoencoder and its various variants," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, 2018, pp. 415–419.

[16] Y. Lecun, “Phd thesis: Modeles connexionnistes de l’apprentissage (connectionist learning models),” 1987.

[17] B. R. Kiran, D. M. Thomas, and R. Parakkal, “An overview of deep learning based methods for unsupervised and semi-supervised anomaly detection in videos,” *Journal of Imaging*, vol. 4, no. 2, p. 36, 2018.

[18] M. Ravanbakhsh, E. Sangineto, M. Nabi, and N. Sebe, “Training adversarial discriminators for cross-channel abnormal event detection in crowds,” in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2019, pp. 1896–1904.

[19] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[20] T. Nguyen, T. Le, H. Vu, and D. Phung, “Dual discriminator generative adversarial nets,” in *Advances in neural information processing systems*, 2017, pp. 2670–2680.

[21] S. Wang, E. Zhu, J. Yin, and F. Porikli, “Video anomaly detection and localization by local motion based joint video representation and ocelm,” *Neurocomputing*, vol. 277, pp. 161–175, 2018.

[22] H. Oikawa, T. Nishida, R. Sakamoto, H. Matsutani, and M. Kondo, “Fast semi-supervised anomaly detection of drivers’ behavior using online sequential extreme learning machine,” in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2020, pp. 1–8.

[23] D. Gong, L. Liu, V. Le, B. Saha, M. R. Mansour, S. Venkatesh, and A. v. d. Hengel, “Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1705–1714.

[24] L. Annamalai, A. Chakraborty, and C. S. Thakur, “Evan: Neuromorphic event-based anomaly detection,” *arXiv preprint arXiv:1911.09722v2*, 2020.

[25] S. Chen and M. Guo, “Live demonstration: Celex-v: A 1m pixel multi-mode event-based sensor,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, IEEE, 2019, pp. 1682–1683.

[26] T. Wang and H. Snoussi, “Detection of abnormal visual events via global optical flow orientation histogram,” *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 6, pp. 988–998, 2014.

[27] M. Aharon, M. Elad, and A. Bruckstein, “K-svd: An algorithm for designing overcomplete dictionaries for sparse representation,” *IEEE Transactions on signal processing*, vol. 54, no. 11, pp. 4311–4322, 2006.

[28] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, “Online learning for matrix factorization and sparse coding.,” *Journal of Machine Learning Research*, vol. 11, no. 1, 2010.

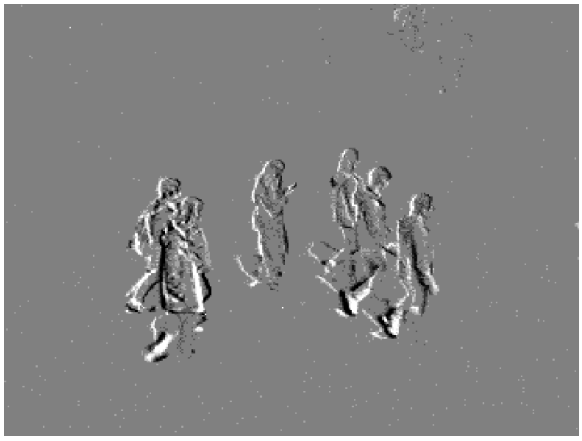
[29] A. Adamo, G. Grossi, R. Lanzarotti, and J. Lin, “Sparse decomposition by iterating lipschitzian-type mappings,” *Theoretical Computer Science*, vol. 664, pp. 12–28, 2017.

[30] S. Hellberg and D. Hollidt, *Evaluation of camera resolution in optical flow estimation using event-based cameras*, 2020.

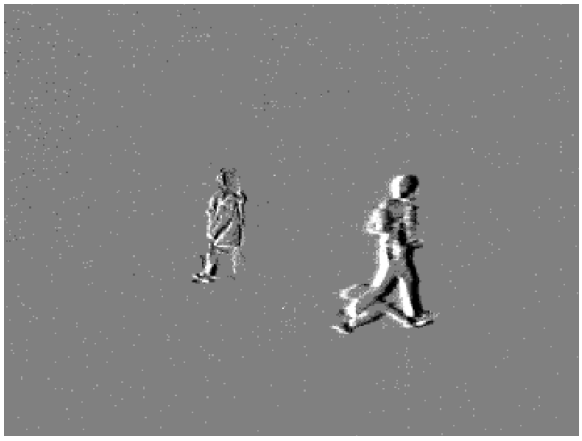
Appendices

Appendix I

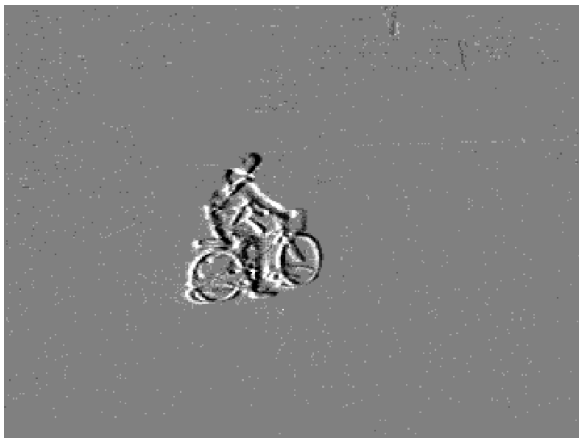
NeuroAED's Walking Dataset Samples



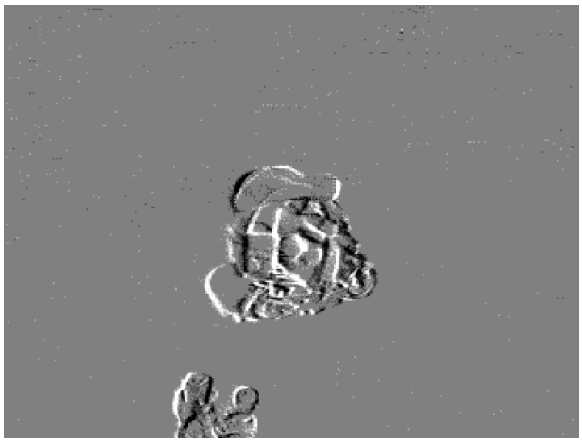
(a) Pedestrians walking.



(b) Pedestrian running.



(c) Cycling.



(d) Motorcycle.

Figure I.1: Normal ((a)) and abnormal ((b), (c), (d)) events contained in NeuroAED's Walking dataset.

Appendix II

Sparse Representation using *scikit-learn* Python library

```
# DICTIONARY LEARNING - TRAINING PHASE
while True:
    print('Learn new dictionary [0] or import existing dictionary [1]: ', end='')
    opt = input()

    if opt == '0':
        dict_learner = DictionaryLearning(n_components=nComp, alpha=1e-8, transform_algorithm='lasso_lars', transform_alpha=1e-10, n_jobs=-1, verbose=True, max_iter=1000)
        D = dict_learner.fit_transform(X_train)

        # Save dictionary and sparse coding for future use
        joblib.dump(dict_learner, 'dictionary_components.sav')
        joblib.dump(D, 'sparse_code.sav')
        break

    elif opt == '1':
        dict_learner = joblib.load('dictionary_components.sav')
        D = joblib.load('sparse_code.sav')
        break
    else:
        print('Enter a valid option ([0] or [1])')

# Size of dictionary and sparse coding
print('\nDictionary Size: ', dict_learner.components_.shape)
print('\nSparse Code Size: ', D.shape)

# Statistical distribution of the sparse representations of all input
```

```
    data
trainMeanMat = np.mean(D.T, axis=1)
trainCovMat = np.cov(D.T)
# trainVarMat = np.var(dict_learner.components_, axis=1)
# print('\nVariance: ', trainVarMat)
print('\nTrain Mean Matrix Size: ', trainMeanMat.shape)
print('\nTrain Covariance Matrix Size: ', trainCovMat.shape)

# ================================================================
# SPARSE REPRESENTATION OF TEST DATA - TEST PHASE
coder = SparseCoder(dictionary=dict_learner.components_,
   transform_algorithm='lasso_lars', transform_alpha=1e-10, n_jobs
   =-1)

# Sparse representation of normal and abnormal test data
X_trainSampleSR = coder.transform(X_trainSample)
X_abnTestSR = coder.transform(X_abnTest)
X_normalTestSR = coder.transform(X_normalTest)
```

Appendix III

One-Class Extreme Learning Machine with OS-ELM and OR-ELM Extensions

```
import numpy as np
from scipy.linalg import pinv2, pinv
from sklearn.preprocessing import LabelBinarizer

class ELM(object):
  def __init__(self, input_dim, hidden_dim, C):

    self.weight = np.zeros((input_dim, hidden_dim))
    self.bias = np.zeros((hidden_dim, 1))
    self.beta = np.zeros((hidden_dim, 1))

    self.binarizer=LabelBinarizer(neg_label=-1, pos_label=1)

    self.C = C
    self._init_weights()

  def _init_weights(self):
    self.weight = np.random.randn(self.weight.shape[0], self.weight.
   shape[1])
    self.bias = np.random.randn(self.bias.shape[0],)

  def sigmoid(self, x):
    return 1.0 /(1.0 + np.exp(-x))

  def gaussian(self, x):
    return np.exp(-pow(x, 2.0))

  def _compute_input_activations(self, x):
    acts = np.add(np.dot(x, self.weight), self.bias)
```

```
    return acts

  def fit(self, input, target):
    H = self.sigmoid(self._compute_input_activations(input))
    self.classes_ = np.unique(target)

    y_bin = self.binarizer.fit_transform(target)

    self.beta = np.dot(H.T, pinv2(1.0/self.C + np.dot(H, H.T)))
    self.beta = np.dot(self.beta, y_bin)

  def predict(self, input):
    pred = self.sigmoid(self._compute_input_activations(input))

    pred = np.dot(pred, self.beta)
    dist = np.zeros((input.shape[0], len(self.classes_)))

    for i in range(len(self.classes_)):
      dist[:,i] = np.abs(pred - self.classes_[i])[:,0]
    pred = np.argmax(dist,1)
    pred = np.array(self.classes_)[pred]
    return pred


class OCELM(ELM):
  def __init__(self, input_dim, hidden_dim, C, mu):
    super(OCELM, self).__init__(input_dim, hidden_dim, C)
    self.mu = mu
    self.M = pinv(0.0001 * np.eye(hidden_dim))
    self.forgettingFactor = 0.999
    self.hidden_dim = hidden_dim

  def fit(self, input, target):
    H = self.gaussian(self._compute_input_activations(input))
    self.classes_ = np.unique(target)
    assert len(self.classes_)==1, 'target should only have one class'

    y_bin = self.binarizer.fit_transform(target)

    self.beta = np.dot(H.T, pinv2(1.0/self.C + np.dot(H, H.T)))
    self.beta = np.dot(self.beta, y_bin)
```

```
70
71      distance  = np.abs(np.add(np.dot(H, self.beta), -target))
72      distance  = np.sort(distance, axis=None)
73
74      # set threshold
75      N = input.shape[0]
76      cutoff = int(np.floor(N * self.mu))
77      self.threshold = distance[-cutoff]
78
79    def fitOnlineRecurrent(self, input, target):
80      H = self.gaussian(self._compute_input_activations(input))
81      self.classes_ = np.unique(target)
82      assert len(self.classes_)==1, 'target should only have one class'
83
84      y_bin = self.binarizer.fit_transform(target)
85
86      self.RLS_k = np.dot(np.dot(self.M, H.T), pinv2(self.
     forgettingFactor*np.eye(input.shape[0])/self.C + np.dot(H, np.dot(
     self.M, H.T))))
87      self.RLS_e = target - np.dot(H,self.beta)
88
89      self.beta = self.beta + np.dot(self.RLS_k,self.RLS_e)
90  #     self.beta = np.dot(self.RLS_k, y_bin)
91      self.M = 1/self.forgettingFactor * (self.M - np.dot(self.RLS_k,
     np.dot(H, self.M)))
92
93      distance  = np.abs(np.add(np.dot(H, self.beta), -target))
94      distance  = np.sort(distance, axis=None)
95
96      # set threshold
97      N = input.shape[0]
98      cutoff = int(np.floor(N * self.mu))
99      self.threshold = distance[-cutoff]
100
101
102   def fitOnlineSequential(self, input, target, iteration):
103     if iteration == 0:
104       self.H = self.gaussian(self._compute_input_activations(input))
105       self.classes_ = np.unique(target)
106       assert len(self.classes_)==1, 'target should only have one
     class'
```

```
107
108       self.P = pinv2(np.dot(self.H.T, self.H))
109       self.beta = np.dot(np.dot(self.P, self.H.T), target)
110
111     else:
112       self.H = self.gaussian(self._compute_input_activations(input))
113       self.classes_ = np.unique(target)
114       assert len(self.classes_)==1, 'target should only have one
    class'
115
116       self.P = self.P - np.dot(np.dot(np.dot(self.P, self.H.T), pinv2
    (np.eye(input.shape[0]) + np.dot(np.dot(self.H, self.P), self.H.T)
    )), np.dot(self.H, self.P))
117       self.beta = self.beta + np.dot(np.dot(self.P, self.H.T), target
    - np.dot(self.H, self.beta))
118
119     distance  = np.abs(np.add(np.dot(self.H, self.beta), -target))
120     distance  = np.sort(distance, axis=None)
121
122     # set threshold
123     N = input.shape[0]
124     cutoff = int(np.floor(N * self.mu))
125     self.threshold = distance[-cutoff]
126
127   def predict(self, input, errorThresh):
128     H = self.gaussian(self._compute_input_activations(input))
129     D = np.abs(np.dot(H, self.beta) - self.classes_)
130     #print(D)
131     pos_idx = np.where(D < self.threshold*errorThresh)
132
133     pred = np.zeros((input.shape[0],))
134     pred[pos_idx[0]] = 1
135     return pred
```

Appendix IV

Autoencoder structure and training using *Tensorflow* Python library

```
# Training parameter values
nEpochs = 1000
batchSize = 5000
inputDim = X_train.shape[1]
encodingDim = 36
hiddenDim_1 = int(encodingDim / 2)
hiddenDim_2 = 8
learningRate = 1e-7

# Input layer
inputLayer = tf.keras.layers.Input(shape=(inputDim, ))

# Encoder
encoder = tf.keras.layers.Dense(encodingDim, activation="tanh",
    activity_regularizer=tf.keras.regularizers.l2(learningRate))(
    inputLayer)
encoder = tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hiddenDim_1, activation='relu')(
    encoder)
encoder = tf.keras.layers.Dense(hiddenDim_2, activation=tf.nn.
    leaky_relu)(encoder)

# Decoder
decoder = tf.keras.layers.Dense(hiddenDim_1, activation='relu')(
    encoder)
decoder=tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encodingDim, activation='relu')(
    decoder)
decoder = tf.keras.layers.Dense(inputDim, activation='tanh')(decoder)
```

```
24
25 # Autoencoder
26 autoencoder = tf.keras.Model(inputs=inputLayer, outputs=decoder)
27 autoencoder.summary()
28
29 # Callbacks for checkpoints and early stopping
30 cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder.h5",
31                         mode='min', monitor='val_loss', verbose=2,
    save_best_only=True)
32
33 # Define our early stopping
34 early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
    min_delta=0, patience=10, verbose=1, mode='min',
    restore_best_weights=True)
35
36 # Compile the autoencoder
37 autoencoder.compile(metrics=['accuracy'], loss='cosine_similarity',
    optimizer='adam')
38
39 # Train the autoencoder using normal data
40 trainValRatio = int(np.floor(0.85*X_train.shape[0]))
41 history = autoencoder.fit(X_train.iloc[:trainValRatio, :], X_train.
    iloc[:trainValRatio, :], epochs=nEpochs,
42             batch_size=batchSize,
43             shuffle=True,
44             validation_data=(X_train.iloc[trainValRatio:, :], X_train
    .iloc[trainValRatio:, :]),
45             verbose=1,
46             callbacks=[cp, early_stop]
47             ).history
48
49 autoencoder.save('autoencoder.h5')
```

Appendix V

Lab-AEDataset and DEEC-AEDataset Scenes

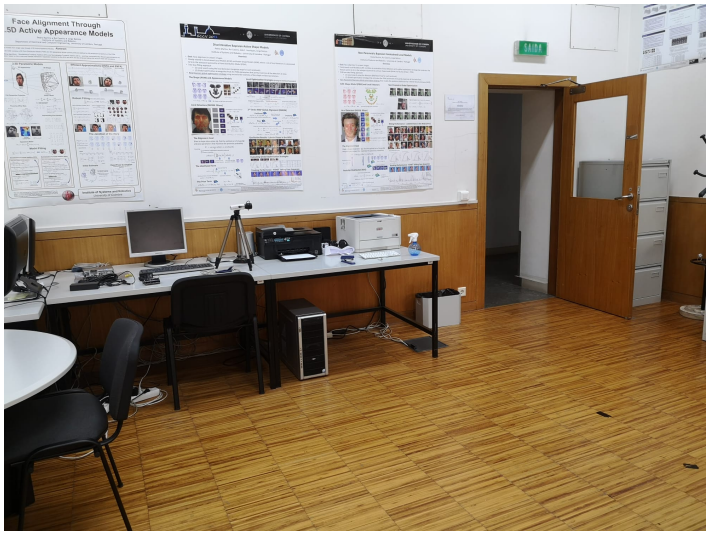


Figure V.1: Lab-AEDataset scene.

Figure V.2: DEEC-AEDataset scene. The green arrow indicates the correct direction of movement, while the red arrow indicates the wrong direction of movement.