UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Universidad Autónoma
de Madrid

Master in Deep Learning for
Audio and Video Signal Processing

MASTER THESIS

EVALUATION OF DEEP LEARNING-BASED CLASSIFICATION AND OBJECT DETECTION ALGORITHMS FOR EVENT CAMERAS

Francisco Javier Martín Ameneiro
Advisor: Pablo Carballeira López
Lecturer: José María Martínez Sánchez

June 2021

EVALUATION OF DEEP LEARNING-BASED CLASSIFICATION AND OBJECT DETECTION ALGORITHMS FOR EVENT CAMERAS

Francisco Javier Martín Ameneiro
Advisor: Pablo Carballeira López
Lecturer: José María Martínez Sánchez

Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
June 2021

Video Processing and Understanding Lab
Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Trabajo parcialmente financiado por el Gobierno de España bajo el proyecto

TEC2017-88169-R (MobiNetVideo)

Resumen

El principal objetivo de este Trabajo Fin de Máster es analizar el efecto que tiene la introducción de ruido en señales de eventos que son utilizadas en aplicaciones de visón artificial basadas en Deep learning. En concreto, nosotros nos vamos a centrar en una aplicación basada en Deep learning, la cual es capaz de solventar dos tipos de tareas, clasificación y detección de objetos.

Para ello vamos a utilizar las señales de eventos, las cuales son capturadas a partir de cámaras de eventos. Estas cámaras de eventos, son un nuevo tipo de cámaras que han aparecido hace unos pocos años y tienen la principal característica de estar basadas en el funcionamiento del ojo humano. Las cámaras de eventos tienen una serie de pixeles inteligentes que son capaces de detectar cambios de intensidad, cuando este cambio es mayor a cierto umbral se genera un evento. En comparación con las cámaras tradicionales, las cámaras de eventos se caracterizan por tener una latencia más baja, un mayor rango dinámico y gracias a esto evitar los problemas de *motion blur* y la saturación de los pixeles cuando la diferencia entre el nivel máximo y mínimo de brillo es muy alta.

Debido a la novedad de estas cámaras existe una falta de stock en el mercado, lo que conlleva a una falta de datasets de este tipo de señales y esto tiene una relación directa con el desarrollo de aplicaciones de visión artificial, especialmente aquellas que usan Deep learning. Para solventar esta situación existen una serie de simuladores, los cuales son capaces de, a partir de una señal rgb, generar una señal de eventos, proporcionando de esta manera una herramienta para transformar datasets que han sido capturados con cámaras tradicionales, en datasets de señales de eventos.

Actualmente, no existe ninguna manera de medir la distorsión que estos simuladores generan, por ello, el VPULab está trabajando en diseñar una serie de métricas que sean capaces de medir esta distorsión. Para poder verificar si estas métricas funcionan correctamente, es necesario medir su correlación con los resultados de tareas de visión artificial.

En este trabajo se va a evaluar como la introducción de ruido sobre un total de cuatro datasets de señales de eventos, afectan al rendimiento de las tareas de clasificación y detección de objetos. Vamos a trabajar con un total de cuatro tipos de ruidos y durante los distintos experimentos que hemos realizado vamos a ver como el comportamiento de ambas tareas es similar cuando introducimos el ruido, y como la información espacial es la que tiene una mayor relevancia en ambos casos.

Palabras clave

Cámaras de eventos, Deep Learning, marca de tiempo, polaridad, convolución dispersa, latencia, rango dinámico.

Abstract

The main objective of this Master Thesis is to analyze the effect of the introduction of noise in event signals that are used in artificial vision applications based on Deep learning. Specifically, we are going to focus on an application based on Deep learning, which can solve two types of tasks, classification and object detection.

For this, we are going to use event signals, which are captured from event cameras. These event cameras are a new type of cameras that have appeared a few years ago and have the main characteristic of being based on the functioning of the human eye. Event cameras have a series of intelligent pixels that are able to detect changes in intensity, and when this change is greater than a certain threshold, an event is generated. Compared to traditional cameras, event cameras are characterized by a lower latency, a higher dynamic range and thus avoid the problems of motion blur and saturation of pixels when the difference between the maximum and minimum brightness level is very high.

Due to the novelty of these cameras, there is a lack of stock in the market, which leads to a lack of datasets of these types of signals and this has a direct relationship with the development of artificial vision applications, especially those that use deep learning. To solve this situation there are a series of simulators, which are capable of generating an event signal from an rgb signal, thus providing a tool to transform datasets that have been captured with traditional cameras into event signal datasets.

There is currently no way to measure the distortion that these simulators generate, so VPULab is working on designing a set of metrics that are capable of measuring this distortion. In order to verify whether these metrics work correctly, it is necessary to measure their correlation with the results of computer vision tasks.

In this work, we are going to evaluate how the introduction of noise on a total of four datasets of event signals affects the performance of object classification and detection tasks. We will work with a total of four types of noise and during the different experiments, we will see how the behavior of both tasks is similar when noise is introduced, and how spatial information is the most relevant in both cases.

Keywords

Event cameras, Deep Learning, timestamp, polarity, sparse convolution, latency, dynamic range.

Acknoledgements

First of all, I would like to thank my tutor Pablo Carballeira, for helping me throughout the year to develop this project and to all the time he has devoted to the project's success. I would also like to thank Erik Velasco, who has helped me to understand how event cameras work, something that was completely new to me, and has been able to help me when I sometimes got stuck somewhere in the project.
Next, I would like to thank my parents for supporting me at all times, helping me to make difficult decisions and encouraging me to continue working in the most difficult moments. Without them I would not have reached this point in my life.
I would also like to thank all my colleagues that I have met during this year, because without them nothing would have been the same during this strange year due the pandemic.
Finally, I would like to thank my girlfriend Alicia, who together with my parents is one of the pillars of my life. She is always there to cheer me up when I need it most and she is able to make me smile with just a glance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Event cameras are bio-inspired sensors that have emerged a few years ago. They are characterized by being formed by a series of smart pixels, that, when they detect a change in intensity greater than a certain threshold, generate an event and are activated, as opposed to traditional cameras, where the temporary sampling rate is fixed. In comparison with traditional cameras, event cameras show some advantages: they have a higher dynamic range, latency in the order of microseconds and they can avoid the common problems of the standard cameras, that are caused by the motion blur and when the difference between the maximum and minimum brightness levels is very high.

Due to their recent appearance, there is a shortage on the market and the few available cameras are very expensive for mass marketing. This means that there is a lack of datasets on these types of signals, which poses a problem for research in this area, especially for learning-based approaches to computer vision algorithms for event cameras. Fortunately, some simulators are able to generate an event signal from an rgb signal, thus being able to transform images that have been recorded by traditional cameras into images that would have been recorded by an event camera.

Currently, VPULab is working on designing a set of metrics that are capable of measuring the distortion generated by these simulators. At this point, our work in the TFM is focused on contributing to the validation process of these metrics. To do so, we will use noisy event signals which will be introduced in different computer vision applications based on deep learning. Finally, the results we have obtained will be compared with the results of the VPULab through its metrics to check their correlation levels, so, if the distortion measured with the metrics and the results we have obtained show correlation, it means that the metric is working correctly.

1.2 Objectives

The main objective of this work is to test the influence of noise on the accuracy performance on the task of classification and object detection. The application that we are going to work with is called *Event-based Asynchronous Sparse Convolutional Networks* [1] and it is based on a deep learning architecture, Sparse Neural Network. For this purpose, different noisy versions will be generated from the original datasets used by

the application. Once we have the noisy datasets, different experiments will be carried out to check how this noise influences the accuracy results in the classification task as well as in the object detection task.
All this work is divided into four sections:

1. **State of the art study:** We will define event cameras by explaining their basic concepts and the advantages they show over traditional cameras. In addition, we will explain different computer vision and deep learning-based applications that use this technology.

2. **Design of a proposal:** We will explain the methodology we have followed in this work. In addition, we are going to explain the operation of the noise generator, which is capable of introducing four types of noises of different natures. Finally, we will explain the format converters that we have had to develop in order to be able to work with different datasets, as there are currently several formats for storing events.

3. **Conducting experiments:** We will perform different tests on a total of 4 datasets, these tests are divided into three steps: The first step is to replicate the initial results, without the use of noise. The second step consists of generating the different noisy versions from the original datasets. Finally, the noisy datasets will be tested and the results will be analyzed. The final objective of these experiments is to evaluate the performance of the different datasets and their different configurations (e.g. the type of representation: *histogram* or *event queue*), against the different types of noise proposed.

4. **Conclusions and future work:** Once our experiments have been carried out, we will have to draw a series of conclusions and different tasks will be proposed in order to extend the study of the influence of noise.

1.3 Report structure

This report has the following chapters:

- chapter 1 **Introduction**

- chapter 2 **Related work:** In this chapter, a study of the state of the art will be carried out, in which event cameras will be defined, explaining their fundamentals and the advantages they show over traditional cameras. We will also show examples of computer vision and deep learning applications that use this type of technology.

- chapter 3 **Design and development:** We will show the work methodology that we have followed to carry out the different experiments.

- chapter 4 **Evaluation:** In this chapter, we are going to present the different datasets we are going to use and then proceed to explain the experiments we have carried out.

- chapter 5 **Conclusions and future work:** In the fifth chapter, a series of conclusions will be drawn from the work carried out and different lines of work will be proposed to extend this research in the future.

Chapter 2

Related work

2.1 Introduction

In this second chapter, we define what are the event cameras and the benefits they show over traditional cameras, then we will talk about the main computer vision applications of events technology and after that, the ones that are based on deep learning. In addition, we explain one of the problems that exist with event cameras, related to the lack of enough dataset, and how our project is going to help in the task of solving it by introducing noisy event signals into a deep learning-based computer vision application. Finally, we analyze with detail some applications of computer vision-based on deep learning that use event signals and could be used in our work.

2.2 Basics of event cameras

2.2.1 Definition

Event cameras [2] are bio-inspired sensor which first appeared about 6 years ago. As you will see throughout this project and especially in this second chapter, these event cameras show great qualities when applied to robotics or computer vision applications.

Before going deeper into event cameras, we are going to discuss how we, as humans, capture visual information using our eyes [4]. We use the information our eyes pick up to understand what is happening around us and to be able to react to changes in our environment. This information is acquired through light, this light passes through the cornea to the retina, where it will be captured. Inside the retina, there exist two types of photoreceptor systems, rods and cones. The **rods** are responsible for capturing information in low light conditions and are not able to perceive color. **Cones**, on the other hand, are more sensitive to light and are able to perceive color. In both cases, the light information is acquired asynchronously, that is, when the light hits the retina it is transformed continuously into electrochemical signals, which are sent, via the optic nerve, to the visual cortex, that is the part of the brain dedicated which is dedicated to processing this information, we can visualize this process in Figure 2.1.

Event cameras [6] are made up of a series of pixels that try to imitate the photoreceptors of the retina, which we have mentioned previously, as they work independently and asynchronously, so they will only be activated when there is a light change in the

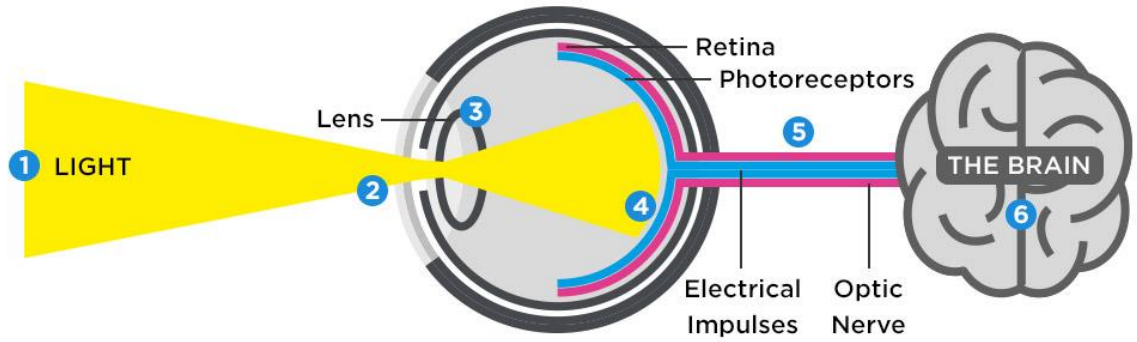


Figure 2.1: In this figure, we can visualize the process of how the light information is acquired and which are the main elements of this process. (Source: [3])

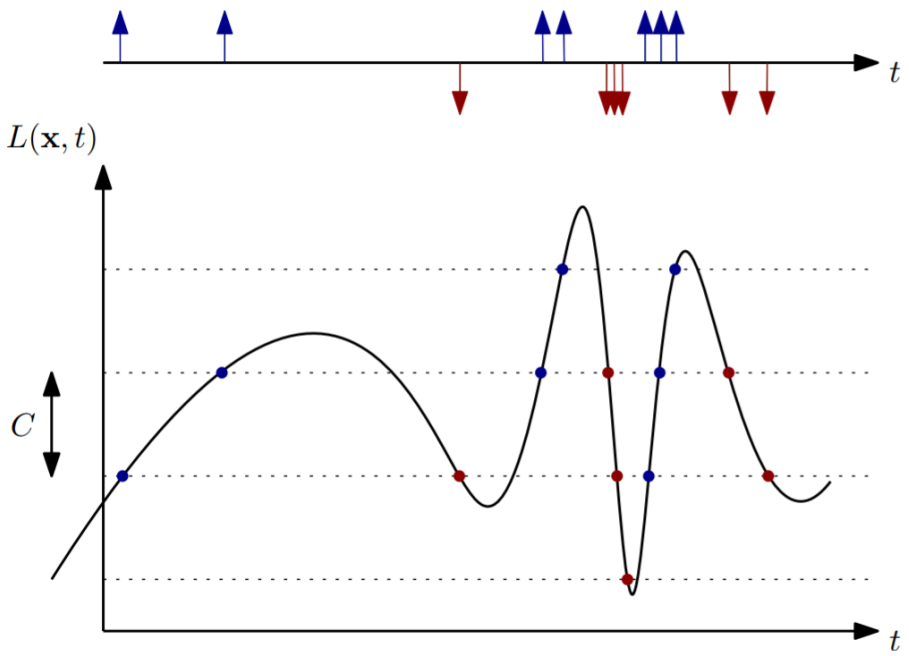


Figure 2.2: In this figure, we can see how events are created when a change of intensity happens. When this intensity change is increasing, the generated events will have a positive polarity value, conversely, when this intensity change is negative, the generated events will have a negative polarity value. (Source: [5])

scene that is being captured. So, every time that a single pixel detects an intensity change, higher than a certain threshold, that pixel will trigger an event, in Figure 2.2 we can visualize an example of how events are generated from changes in intensity. Each event is made up of a total of four variables:

- **X coordinate:** Corresponds to the point on the horizontal axis where the intensity change, that is, the event, has been detected.
- **Y coordinate:** Corresponds to the point on the vertical axis where the event has been detected.
- **Timestamp:** Indicates the instant of time at which the event has occurred.
- **Polarity:** It is used to indicate whether the change in intensity has been upward, in which case the polarity would be positive, or downward, in which case the polarity would be negative.

On the other hand, traditional cameras, use a shutter, which opens and closes to capture the light. This fact means that the speed at which the shutter can open and close will limit the ability of the camera itself to capture high-speed images. Next, we are going to take a closer look at the differences between event cameras and traditional cameras.

2.2.2 Event cameras vs Traditional cameras

The past sixty years of research in the computer vision scenario have been devoted to frame-based cameras but, as we are going to show you, the technology that they use can be improved to obtain better results in the performance of some computer vision algorithms and avoid some concrete problems while capturing scenes that we are going to explain below.

Standard cameras [8], indeed, suffer from different types of problems, the first one is that they work with a **high latency**, which depends on exposure time in ranges between 1 to 100 ms. Due to this first problem, the effect of the **motion blur** emerges. Motion blur [7], in the field of images, it can be defined as the effect produced by moving objects in a photograph or in a sequence of frames that make up a video in Figure 2.3 we can see some examples of images that suffer from motion blur. Finally, standard cameras have a **low dynamic range**, which may result in loss of information due to the lighting conditions of the scene being recorded, most of these lighting problems appear in extreme situations, both when the scene being captured is very bright and pixels may and pixels may get saturated, for example, due to sunlight, and when the scene being captured has low light conditions, for example when recording at night.

In contrast, event cameras do not suffer from any of these problems. As we know, event cameras are novel sensors that are able to capture all the motion that occurs in a scene with the use of smart pixels that conform to them. Although this kind of sensor have been created a few years ago and still has room for improvement, it has some outstanding properties that, in comparison with the ones that standard cameras provide us, can have a huge impact on computer vision and robotic applications. The main properties of event cameras are the following ones [9]:

Figure 2.3: In this figure, we have some visual examples of images that suffer from the problem of motion blur. (Source: [7]).

- **Latency:** They have very low latency, in order of microseconds (us).
- **Motion blur:** Due to the low latency and unlike traditional cameras, event cameras do not suffer from this problem.
- **Dynamic range:** In this case event cameras have a much higher dynamic range than traditional cameras, approximately eight orders of magnitude superior (140 dB vs 60 dB). Thanks to this, event cameras will not suffer from lighting problems when capturing a scene, again setting them apart from traditional cameras.
- **Power consumption:** They have ultra-low power consumption, in fact, if we compare the average power consumption between traditional cameras and event cameras, we can see that the average power consumption of traditional cameras is 1 W, while in event cameras it is a hundred times lower, 1 mW.
- **Bandwidth:** As event cameras are designed to capture motion when there is no movement in the scene that we are recording, no intensity changes will occur, and thus no event will be generated. That is to say, in the parts of the scene being recorded where there is no movement, no information will be obtained, thus avoiding redundant information and making the bandwidth needed to process this type of signal smaller than that required by the signals generated by traditional cameras.

In Figure 2.4, we show you some visual examples of the properties that we had described above. Specifically, two examples can be visualized, the first one shows how the low latency of event cameras avoids the problem of motion blur, and the second one shows how event cameras can obtain more information in bad lighting conditions due to a higher dynamic range.

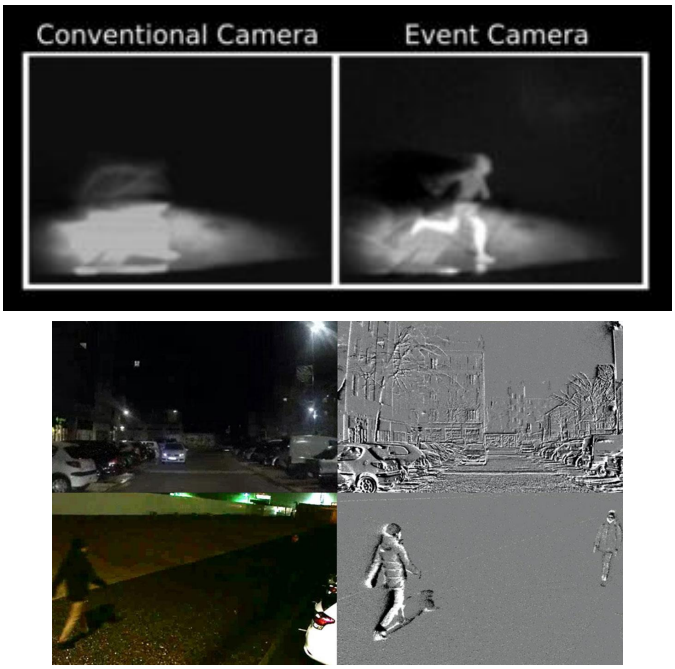


Figure 2.4: In this figure we have some visual examples of images represent the advantages that shows an event camera against a traditional camera. The image that is in the upper part describe the motion blur problem, and the image that is in the bottom part describe the problem of scene brightness. (Sources: [10, 11]).

2.3 Computer vision algorithms on event signals

Once that we know what the event cameras are and the advantages they show over traditional cameras, we are going to look at the main computer vision algorithms that use this technology.

2.3.1 SLAM (Simultaneous Localization And Mapping)

Simultaneous Localization And Mapping [12] most commonly known as SLAM, is a computer vision application whose objective is to give a device (robot, drone, vehicle, etc.) the ability to incrementally create a map of the surrounding environment and to know its location within the map it is creating. These applications have been implemented in a wide range of domains such as indoors [13] and outdoors [14] environments, underwater [15], airborne systems [16], etc.

The incorporation of event cameras [8, 18] in these types of applications results in a great improvement in high-speed scenes, in Figure 2.5 we show you a visual example of how are the frames capture by a standard camera in a high-speed scene, where appears the motion blur problem explained in Section 2.2.2, and the frames that are captured by an event camera in the same situation. This improvement is due to the high temporal resolution of the event cameras. In addition, with the incorporation of event cameras, we will avoid the problems of the problem of abrupt lighting changes as they also have a higher dynamic range than traditional cameras.

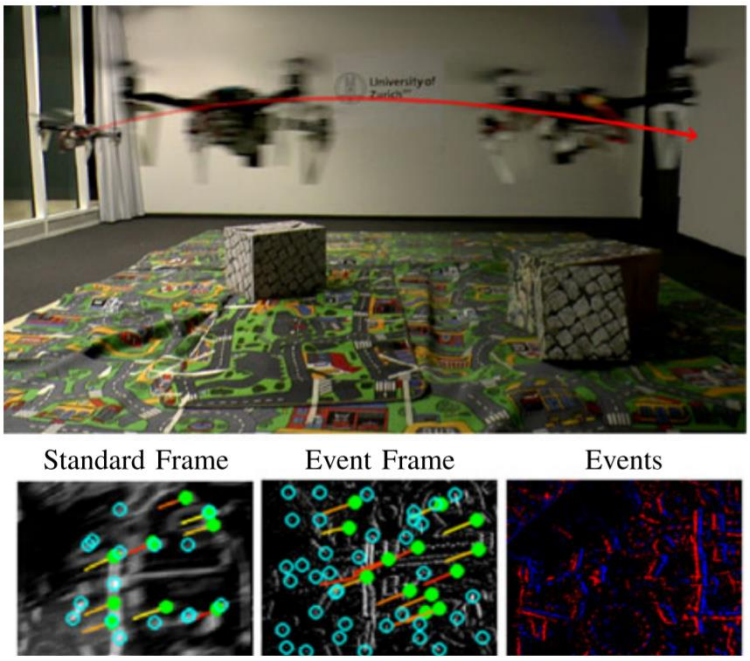


Figure 2.5: In this figure, we can see how does a frame captured by a standard camera in a high-speed scenario is, and how does it looks like when it is captured by an event camera. This illustration shows how event cameras, by having a much lower latency than traditional cameras, avoid the problem of motion blur that appears in the frame that has been captured by a traditional camera. (Source: [17]).

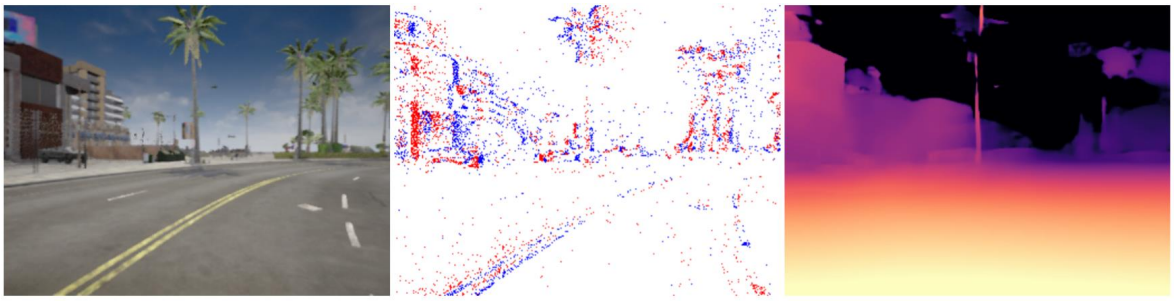
2.3.2 Depth Estimation

Depth estimation [19] applications try to predict the depth of each pixel from an rgb signal. This task is very simple for the human eye, but for computer vision algorithms it is very challenging due to the occlusion of objects, the different textures that appear in the same image, etc. These kinds of applications are of great applicability, they can be used for autonomous driving [20], for 3-dimensional reconstruction applications [21], etc.

When these applications are adapted to be used with event signals [22, 23], we can see how the results are improved compared to those obtained using traditional cameras. As in the previous case, where this remarkable improvement can be observed is in the case of high-speed scenes, where event cameras avoid the appearance of motion blur, and in scenes with large changes in lighting, where event cameras can lose much less information than traditional cameras do. In Figure 2.6, we can see an example of the results that a depth estimation application that uses event signals obtain.

2.3.3 Reconstruction of Visual Information

This type of application is the most common type of computer vision application using event signals [24, 25]. These applications try to reconstruct images from the intensity changes that occur in them, i.e. through events. Employing this, the results obtained in reconstruction applications based on hand-crafted priors can be improved, specifically in cases where motion blur appears and where there is a loss of information due to sudden changes in illumination



(a) Frame (b) Events (c) Ours on events

Figure 2.6: In this figure, we can see the results of a depth estimation application that uses event signals. The figure on the left, represents the original frame, next to it we see the events that are detected in that frame, and finally, we can see of the right of the image the depth estimation. We may observe that in this case, they use colors to identify which objects are further away, with the lighter-colored pixels being closer and the darker-colored pixels being further away.(Source: [22]).

Once the image is reconstructed and has the advantages provided by the event technology, it can be applied to different tasks to improve the performance that was obtained using traditional cameras, some examples of these tasks are object detection [6], face detection [26], etc.

2.3.4 Others

In addition to the three computer vision applications that we have explained in more detail, there are even more types of applications that use event signals, e.g. optical flow [27, 28], tracking [29, 30], semantic segmentation [31, 32], etc. But we will not go into detail in these cases.

2.4 Deep Learning in event cameras

Now that we have seen which are the main computer vision applications that use event signals, we will move on to analyze those applications that also incorporate deep learning technology for this type of signal. There are two main branches, the algorithms that use the so-called *Spiking Neural Networks* [33] and the ones that use the *Sparse Neural Networks* [34].

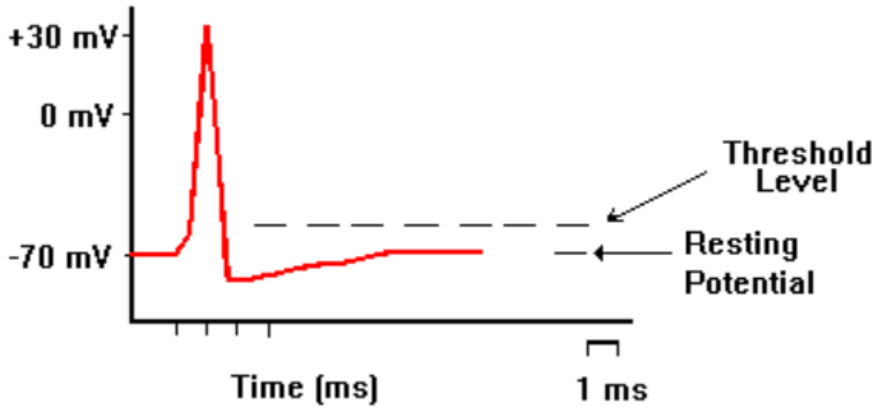


Figure 2.7: In this figure, we can see how a spike looks like. When the voltage signal is above the threshold, the spike is detected. (Source: [35]).

2.4.1 Spiking Neural Networks

Spiking Neural Networks [33] are part of the third generation of neural networks, which are increasingly trying to link neuroscience and machine learning by using models that increasingly resemble the way neurons are wired in our brains. The main quality of spiking neural networks that differentiates them from other neural networks is that they use the spikes voltages. These spikes are created when the value of the voltage is superior to a certain threshold that had been previously fixed, when this happens, the neuron that has detected it generates a spike. In Figure 2.7 we can visualize how a spike looks like.

The main advantage of these spiking neural networks is that they are more energy efficient than traditional neural networks, as each spike is sparse in time, which means that each spike contains a lot of information [36]. But, although these networks were developed a few years ago [37], they did not have much impact on deep learning applications that used traditional signals, mainly because there were not many algorithms to train this type of network, and those that did exist were less efficient than those used in traditional convolutional neural networks. However, this has not slowed down the growth of spiking neural networks and work is being done to continue improving them, in particular, several feature extraction [38] and image classification [39] applications have been developed using this architecture. since, as we mentioned at the beginning, they are one of the networks with the greatest potential for the future due to the good energy consumption conditions they show.

In the field of event cameras, such networks are increasingly being used, as it is a way to exploit even more the main advantage of the spiking neural network, that is the sparsity of each spike in time. By introducing event signals into this networks, they will become even more efficient in terms of energy consumption. There are already some applications that introduce event signals into the described spiking neural network, as we can see in [40, 41].

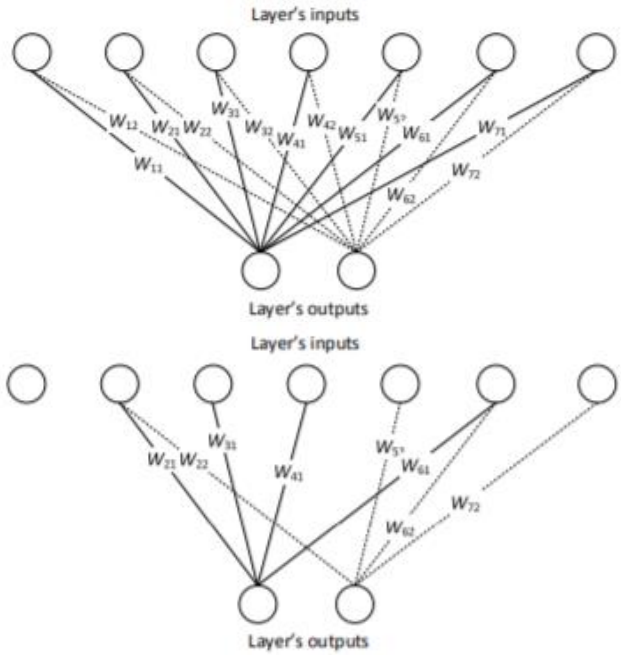


Figure 2.8: In this figure we can see a comparison between the connections in a sully conected layer (upper figure) and a sparsely conected layer (bottom figure). (Source: [42]).

2.4.2 Sparse Neural Networks

The other neural network that is used when we are developing a deep learning algorithm using event data is the *Sparse Neural Network* [34]. These neural networks aim to solve one of the problems that appear in deep neural networks that use fully connected layers. The problem came when, in these neural networks, we try to deal with a huge amount of data, for example, rgb images. In this case, although it has been proven several times that fully connected layers are very powerful, the number of connections increases to a value too high to reach a point where it becomes inefficient. This makes it necessary to set a maximum number of inputs to the fully connected layers. By using **sparsely connected layer** [43], as in the Sparse Neural Networks, we can avoid this problem. Unlike fully connected layers, in these layers, each input node is not connected to all output nodes. In Figure 2.8, we can see a comparison between the connections made in the fully connected layers and the sparsely connected layer. Applying this method, we can see how the number of parameters needed in the sparsely connected layer is much smaller than the number of parameters needed in the fully connected layers, moreover, despite the reduction in the number of parameters, the accuracy of the algorithm is very little affected as demonstrated in [42].

Another characteristic of the sparse neural network is the use of the **sparse convolution** [44] instead of the traditional convolution. This technique provides a much more efficient approach to make convolutions. As the application that we have used to work with during this project is based on the use of sparse convolution, we will give you more details below:

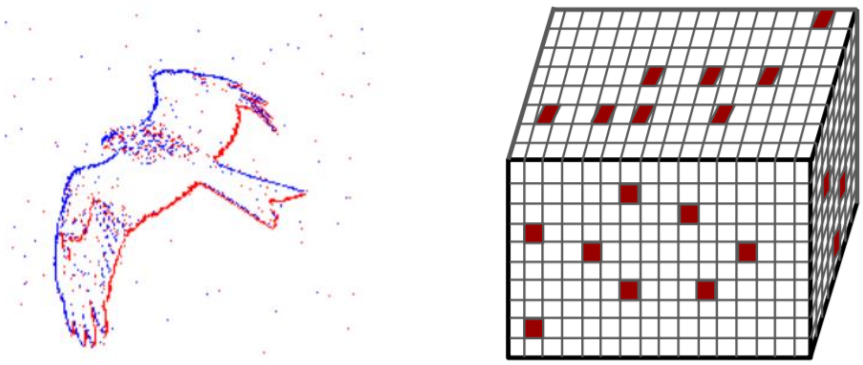


Figure 2.9: The figure shows how the data is sparsed over the representation space in the case of the event representation (image on the left part) and the 3-dimensional representation (image on the right part). (Sources: [1] [45]).

- **Sparse Convolution**

  The operation of convolution is one of the most popular among the neural networks that use deep learning to solve problems that involve the use of images or videos as they are effective when we are dealing with 2 dimensions. The problem here comes when we are not using 2-dimensional images, and we use for example a 3 dimensions signal, where the additional dimension significantly increases the computational complexity of the convolution. In addition, as it happens in 3 dimensions representations and event representations, the information is sparse in the representation space and many of the pixels (or voxels in the case of 3 dimensions) have a zero value. We can better visualize this second problem of the sparsity of the data in Figure 2.9.

  At this point, the sparse convolution [44] emerges as a solution to the problem that we have just described above, as it is a way of calculating the convolution only on those pixels that are active, that is, those pixels whose value is non-zero. In addition to the computational benefits that we have explained, the use of the sparse convolution does not affect the performance of the algorithm in comparison when we use the traditional convolution, in fact, in some cases, it can improve it, as demonstrated over different experiments in [44, 46].

  Now we are going to see, in a brief way, how does this sparse convolution work. The first step will be to define the input data, that it will be pixels whose value is non-zero, the active pixels. It is necessary to know the value of these active pixels and also the location of these active pixels, in order to, in the following operations, being able to locate them.
  Once that the input is defined, a convolutional kernel will be applied to the input data. In this second step, there is no difference compared to the traditional convolution.

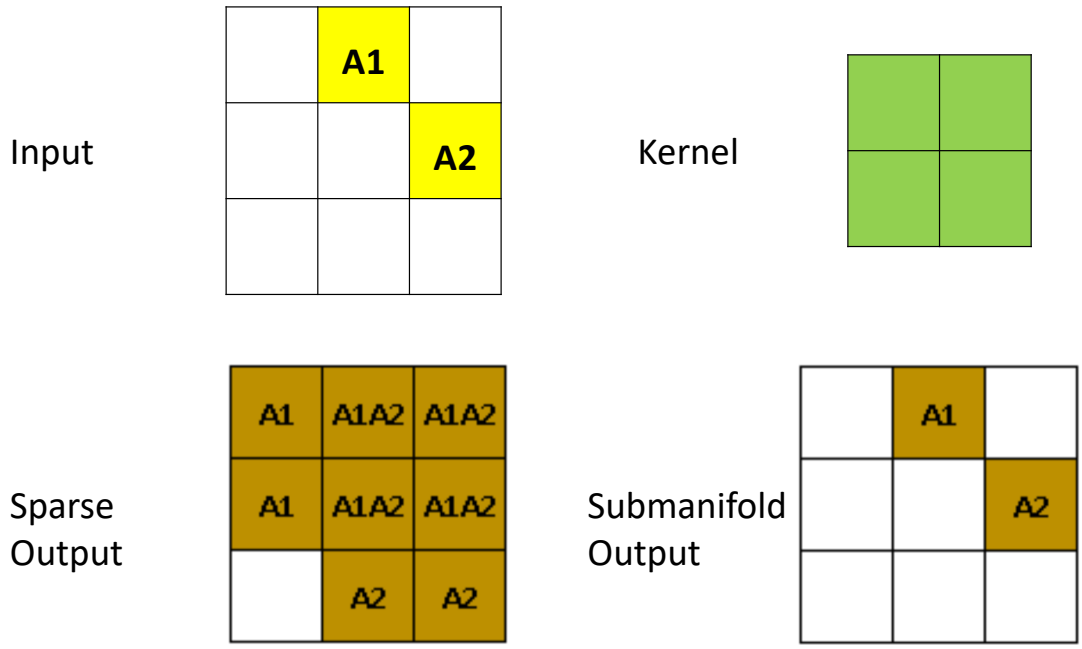


Figure 2.10: The figure shows the two approaches to calculate the output when we are using the sparse convolution. For the first case, in the sparse output, the cells with A1 are the ones that have been obtained when the kernel is on one active pixel, called A1. The same happen with the cells that have A2, but in this case the active pixel is A2. Finally, the cells that have A1A2 are the sum up of the outputs that have been calculated in the two previous cases. For the second case, the submanifold output, we can see how the outputs are in the same positions than the inputs. (Source: [47])

Finally, in the way of how the output is defined, we can use two different approaches as we can see in [44]. The first one is called *Sparse output*, and, from input data, active input data, in this case, a traditional convolution is calculated over all pixels covered by the kernel, it can be understood as a traditional convolution that only applies to active pixels. The second approach, that is the one that the application that we are working with [1] applies to its data, is called *submanifold output*. In this second case, the output will be only calculated when the center of the kernel covers the active input data. In Figure 2.10 we propose you a visual example where you may better understand these two approaches. As it is demonstrated in [44], the submanifold output results in a much efficient approach.

By applying these techniques, especially the submanifold output approach, the sparse convolution is able to give a more efficient approach to calculate convolutions in neural networks by, as we have explained, instead of scanning all the pixels that are contained in an image, only using the active pixels, that is, the non-zero pixels.

As we have seen, the sparse neural networks can be applied to event signals in order to exploit the main benefit that both parts have in common, the sparsity. In Section 2.6.1 we are going to study in detail an example of an application that uses these sparse neural networks with event signals to solve two different tasks, image classification, and object recognition.

2.5 Lack of event datasets

Nowadays, due to the novelty of event cameras, there are not many companies involved in the development and marketing of them. However, the main companies involved are the following [8]:

- **Prophesee:** Which have recently announced a partnership with SONY, they can develop 1M pixels cameras that have an approximate market price of 4.000 USD.
- **Inivation:** As in the previous case, they have announced a partnership with Samsung, they make event cameras with VGA (640 x 480 pixels) resolution that are able to also output standard frames rather than just events and their approximate price is 5.000 USD.
- **Insightness:** They can develop event cameras with very similar properties to Inivation´s cameras have, the same resolution, and the same approximate price.
- **CelexPixel Technology:** This final company develops event cameras of 1M pixel of resolution and that have the particularity that it does not output the polarity, their approximate cost is 1.000 USD.

As you have seen, there are not many companies that support event cameras development and, as a rule, the event cameras on the market are very expensive. This means that when it comes to researching this new technology, the necessary datasets do not exist. This fact, when we are working with computer vision applications and, in particular, with applications based on deep learning, it becomes a very differentiating

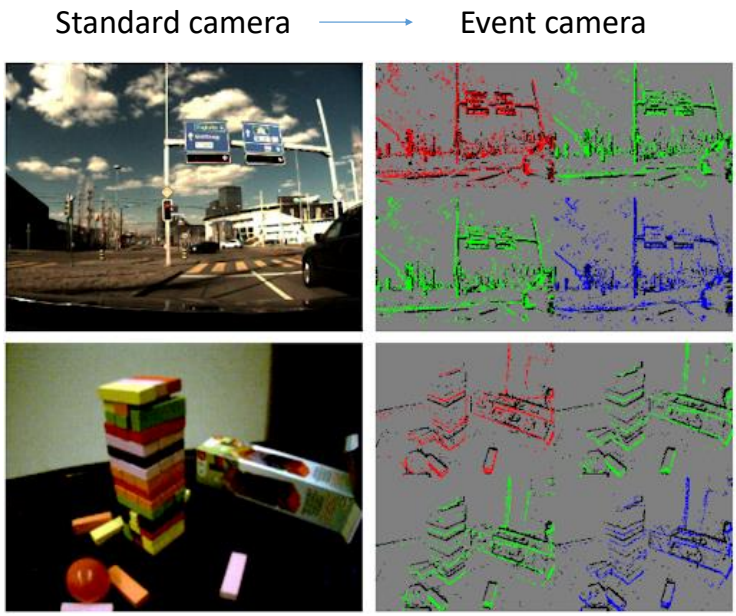


Figure 2.11: In this figure we can see how event cameras simulators works, from a rgb image (situated on the left), the are able to generate an image made up of events in different colors (situated on the right), as if it had been recorded by an event camera. (Source: [48]).

factor. This causes advances in this field to take place at a slower pace.

In order to solve this problem, some simulators have been developed [49, 50, 48]. These simulators can transform images that have been captured with traditional cameras into images that would have been captured with an event camera, in other words, from an rgb signal, they are able to generate an event signal. In Figure 2.11, we can see an example of how these simulators work.
The number of these simulators is growing considerably, as is the number of computer vision applications that use event signals. In the following, we will explain how some of the main event simulators work, although for more information we can find in [51] a list with several of these simulators.

- **ESIM** [5]: This simulator features an adaptive rendering scheme, which gives the simulator the ability to sample frames only when it is necessary. This simulator also gives us the ability to generate large amounts of event data as well as a ground truth.

- **PIX2NVS** [52]: The main feature of this simulator, is that it can transform large annotated datasets of videos that have been captured by traditional cameras such as YouTube-8M [53] or YFCC100M [54] into event signals, thus providing the ability to use these large datasets in event-based applications.

- **V2e** [55]: This last simulator, is characterized by the synthesized generation of realistic event signals. It is very useful for training networks for uncontrolled lighting conditions as it is demonstrated in [55].

In addition to this first problem of missing event data, there is a second problem, that

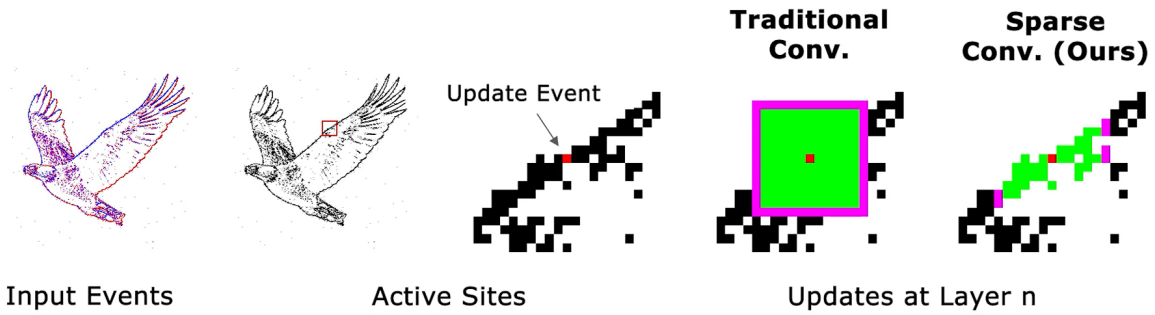


Figure 2.12: In this figure we can see a comparison between the traditional convolution and the sparse convolution when we are facing an event signal (Source: [1])

is also related to the novelty that surrounds all this technology. As we have said the number of available simulators is growing considerably, but there is no way to measure them in order to identify which of the available simulators is the best one or is the one that generates more distortion. To solve this second problem, the VPULab is working on designing some metrics that determine the degree of distortion that an event simulator generates.

At this point, our work comes as a tool for validating these metrics and checking if the results that they are providing are correct. To do this we will use computer vision applications based on deep learning that uses event data which are explained in detail in Section 2.6. The work that we have done is to introduce noisy event signals into these applications, these noisy event signals are developed by ourselves by using our noise generator. Once that we have obtained the results using the noisy event signals we will use them to check their correlation with the results that the VPULab has obtained using their developed metrics.

2.6 Deep Learning-based CV applications using event signals

2.6.1 Event-based Asynchronous Sparse Convolutional Networks

This first application called *'Event-based Asynchronous Sparse Convolutional Networks'* [1], is the one we have used throughout the project to carry out all our tests and experiments, so we have explained it in detail.

		N-Caltech101		N-Cars	
	Representation	Accuracy ↑	MFLOP/ev ↓	Accuracy ↑	MFLOP/ev ↓
Standard Conv.	Event Histogram	**0.761**	1621	**0.945**	321
Ours		0.745	**202**	0.944	**21.5**
Standard Conv.	Event Queue	0.755	2014	0.936	419
Ours		0.741	**202**	0.936	**21.5**

Table 2.1: Results obtained by the authors of *Event-based Asynchronous Sparse Convolutional Networks* to demonstrate the efficiency of their method against the traditional one in the task of image classification. (Source: [1]).

The application, as they said in their paper, was developed to create a framework that can convert a model that has been trained using synchronous images of events into an asynchronous model that has the same output as the previous one. By doing this, they affirm that they are able to take advantage of the very characteristics that events have by nature, which are asynchronous and sparsed in the representation space. This sparsity is translated in an improvement in terms of computational cost, being efficient in this aspect, while at the same time they maintain the same level of accuracy as they obtained with the synchronous model. One of the main tools that they use to take advantage of this sparsity that the events have is the **sparse convolution** that we have explained in Section 2.4.2. The fact of using this sparse convolution, instead of a traditional convolution, will make the application work much more efficiently. In Figure 2.12 we can see an example of how traditional convolution and sparse convolution are applied when working with event signals.

Returning to the actual functioning of the *Event-based Asynchronous Sparse Convolutional Networks*, a key aspect that it has is that it is able to solve two different tasks, the first one is **image classification** and the second one is **object detection**. With this, the authors of the paper wanted to prove the statement that we have been repeating along this section, that the sparse convolution allows us to work more efficiently without worsening the accuracy results, over the two mentioned tasks. They made several experiments over the two proposed tasks, in which they measure the two key aspects that we have said, the accuracy and the efficiency.

In Table 2.1 we can see the results on image classification task. As we have previously said, we can see that they use two different datasets and they also use two different kinds of representations, *Event Histogram* and *Event Queue*.

- **Event Histogram:** This kind of representation uses a sliding window that contains a constant number of events. In this representation the events are stored in two different channels, in one of them will be the positive polarity events and in the other the negative polarity events.

- **Event Queue:** This second representation also creates a sliding window, but in this case, there is a queue that stores the values of the polarities and the timestamps of each event. This queue has a fixed length of 15 events, so when a new event comes, the oldest one is discarded.

By analyzing the results that they obtain we can see that they use two metrics in order of comparing both models, the one that uses the standard convolution and the one

		N-Caltech101		Gen1 Automotive	
	Representation	mAP ↑	MFLOP ↓	mAP ↑	MFLOP ↓
YOLE [56]	Leaky Surface	0.398	3682	-	-
Standard Conv.	Event Queue	0.619	1977	**0.149**	2614
Ours		0.615	**200**	0.119	**205**
Standard Conv.	Event Histogram	0.623	1584	0.145	2098
Ours		**0.643**	**200**	0.129	**205**

Table 2.2: Results obtained by the authors of *Event-based Asynchronous Sparse Convolutional Networks* to demonstrate the efficiency of their method against the traditional in the task of object detection. (Source: [1]).

that uses the sparse convolution. The first metric is to measure is the accuracy, and it is used for measuring the quality of the predictions that the evaluated model is doing, for this first metric the results are practically the same for both models, so we can assume that the sparse convolution does not decrease the accuracy levels. The second metric measures the computational complexity of processing an input event, so, the lower this value is, the lower the number of operations required and thus the lower the computational cost. In this second metric, we can see that it exists a huge difference between both models, we can see that the model using sparse convolutions is much more efficient than the one using standard convolutions.

In Table 2.2 we will find the results that they obtained in the task of object classification. These results are very similar to the previous ones in the sense that the levels of accuracy are still very similar between the two models, the model that uses sparse convolutions requires far fewer operations than the one using standard convolutions. In addition, they have added a third model of the state of the art in the comparison, YOLE [56], and we can see that in both metrics it is outperformed by the model developed by the authors of the paper.
With these two experiments, the author of the paper demonstrated the advantages that the model that they had developed in comparison with traditional models that used standard convolution, for further information about their work we recommend you to read their original paper [1].

2.6.2 Convolutional spiking neural networks for spatio-temporal feature extraction

This application called *Convolutional spiking neural networks for spatio-temporal feature extraction* [40] is based on the use of Spiking Neural Networks, explained in Section 2.4.1, with event signals. As we said then, Spiking Neural Networks can work in a much more efficient way, but in this paper, the authors wanted to test if this neural network could improve the results, in the task of feature extraction, obtained by some of the state of the art algorithms based on C3D [57], ConvLSTM [58], etc.

In addition, they have created their own neural network, called *STS-ResNet*. This network can be described as a fusion between the traditional Spiking Neural Network and the ResNet-18 [59] and, as we will see in the experiments that the authors carry out below, it obtains good results. The main advantage that this neural network show

Method	Config	MNIST	Seq1	Seq2	Seq3	Seq4	Seq5
CNN	256-256	99%	98.89%	98.2%	98.8%	98.27%	20.1%
CNN+LSTM	256-256(CNN) + 256(LSTM)	96.84%	67.74%	**98.93%**	98.96%	94.88%	**91.2%**
ConvLSTM	256-256	99%	**99.11%**	98.9%	30.0%	97.43%	40.7%
C3D	256-256	99.03%	98.49%	98.32%	99.17%	97.73%	64.0%
ConvSNN	48-48	**99.4%**	98.6%	98.4%	**99.36%**	**98.8%**	89.5%
STS-ResNet	18 layers	**99.7%**	**99.26%**	**99.2%**	**99.43%**	**99.1%**	**92.7%**

Table 2.3: Results obtained by the authors of *Convolutional spiking neural networks for spatio-temporal feature extraction* to demonstrate the precision in terms of accuracy that Spiking Neural Network and the STS-RestNet obtain against other approaches of the state of the art in feature extraction. (Source: [40]).

is that it solves the problems of training a Spiking Neural Network.

For making the experiments that the authors were interested in, the have created their own synthetic dataset that is based in the MNIST dataset [60]. This synthetic dataset is made up of a total of 5 cases, each of them being a different challenge for the feature extraction task.

- **Sequence 1:** This first sequence adds zoom in and zoom out to the MNIST images.
- **Sequence 2:** The second sequence add rotations from 0 to 360 degrees and the other way around
- **Sequence 3:** This sequence is also based on zoom in and zoom out, but in this case instead of being a progressive change, the changes are 50%, so they are more abrupt than in the first case.
- **Sequence 4:** The fourth case add masks to the images in order to generate occlusions
- **Sequence 5:** Finally, the fifth sequence, which is the most difficult one for the human eye, is to add rotations, like in sequence 2, but in this case they will be done randomly.

In Table 2.4 we can see the results that they have obtained in their first experiment, where they wanted to demonstrate the precision of the Spiking Neural Network and of their neural network, STS-ResNet. Additionally, they made another experiment to compare STS-ResNet with other neural networks in the task of image classification, but in this case using event signals. For this second experiment, where event signals appears, they have use different event dataset to evaluate themshelves like NMNIST [61], DVS-CIFAR10 [62] or DVS-Gesture [63]. In Table 2.4 the results of this second experiment are shown, and we can see how STS-ResNet outperforms the other networks in all proposed datasets. For further knowledge you can read [40].

2.6.3 Inceptive Event Time-Surfaces for Object Classification using Neuromorphic Cameras

The last application that we think could be interesting to use in our project is called *Inceptive Event Time-Surfaces for Object Classification using Neuromorphic Cameras*

Model	Dataset	Accuracy
SPA	NMIST	96.3%
SpiNNaker	NMIST	98.5%
STBP	NMIST	97.92%
Natural ConvSNN	NMNIST	99.42%
NeuNorm	NMNIST	99.53%
STS-ResNet	NMNIST	**99.6**%
CNN	DVS-Gesture	94.59%
pointnet++	DVS-Gesture	95.32%
STS-ResNet	DVS-Gesture	**96.7**%
HAT	DVS-CIFAR10	52.4%
Natural ConvSNN	DVS-CIFAR10	60.3%
NeuNorm	DVS-CIFAR10	60.5%
SPA	DVS-CIFAR10	67.8%
STS-ResNet	DVS-CIFAR10	**69.2**%
ResNet18 (ANN+scratch)	UCF-101	**42.4**%
STS-ResNet	UCF-101	42.1%
ResNet18 (ANN+scratch)	HMDB-51	17.1%
STS-ResNet	HMDB-51	**21.5**%

Table 2.4: Results obtained by the authors of *Convolutional spiking neural networks for spatio-temporal feature extraction* to demonstrate the precision in terms of accuracy that the STS-RestNet obtain over other approaches of the state of the art in image classification using event signals. (Source: [40]).

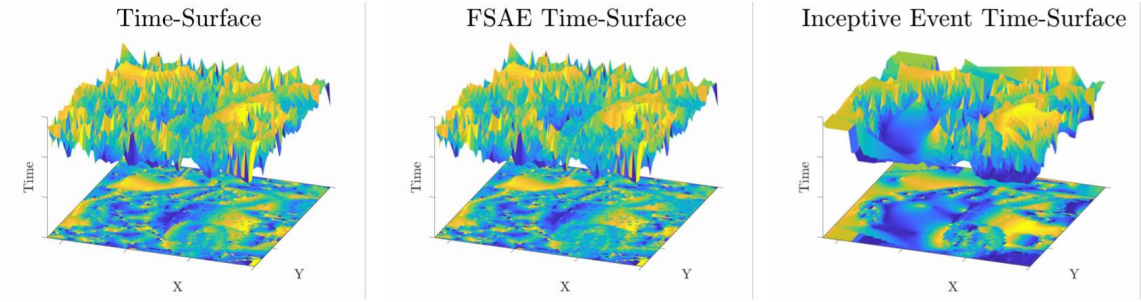


Figure 2.13: The image (a) represent the time surface that is used, made up of a total of approximately 17k events. Image (b), is the time surface after FSAE, where the number of events has been reduced to approximately 8k. Finally, figure (c) represents the time surface after IETS, where the number of events has been drastically reduced to approximately 3k. (Source: [64]).

[64] and once again it is focused on solving the task of image classification using event signals.

The main characteristic of this application is the introduction of the Inceptive Event Time Surface (IETS). IETS is an enhancement of Filtered Surface of Active Events (FSAE) [65]. FSAE is a technique that improves the time surface of events by only using the first events of a sequence. This technique was used for the task of detecting corners and tracking, and, as you can see in the original paper [65], FSAE is a very effective technique for this task. However, in the image classification task, the objects that appear are more complex than in the corner detection task, so the FSAE performance is worse in this case. At this point, the authors of *Inceptive Event Time-Surfaces for Object Classification using Neuromorphic Cameras* developed IETS, which is able to remove more noise in complex objects compared to FSAE. The main factor that sets IETS apart from other algorithms is that instead of using hand-crafted features to remove noise, you use neural networks that learn to extract features from event images with less noise. In Figure 2.13 we can see a comparison between the time surface after applying FSAE and the time surface after applying IETS.

In Table 2.5, we can see the results that they obtain in the task of image classification. In this experiment, they have compared other algorithms that work on the task of image classification and we can observe how IETS is the one that obtains the best results.

Algorithm	**H-First**	**HOTS**	**Gabor**	**HATS**	**HATS**	**FSAE**	**IEST**
Classifier	SNN	SVM	SNN	SVM	CNN	CNN	**CNN**
Accuracy	0.561	0.624	0.789	0.902	0.929	0.961	**0.976**
AUC	0.408	0.568	0.735	0.945	0.984	0.993	**0.997**

Table 2.5: Results obtained by the authors of *Inceptive Event Time-Surfaces for Object Classification using Neuromorphic Cameras* in the task of image classification, where their algorithm, IETS, is the one which obtain better results for both proposed metrics . (Source: [64]).

Chapter 3

Design and development

3.1 Introduction

Once that we have seen the state of the art of event cameras, in this third chapter, we explain the methodology we have followed throughout this project, which is divided into four main parts: the datasets, the format converter, the noise generator and the classifier or the object detector. Finally, we explain to you the most relevant parameters that we have needed to adjust to make the experiments that are shown in Chapter 4.

3.2 Methodology

The main objective of this project is to analyze the influence that, adding noise into a dataset, has on the quality of the predictions made by a model that has already been trained. To do this, although in Section 2.6 we have analyzed three possible applications that we could use to carry out our work, we have focused on the first one, *Event-based Asynchronous Sparse Convolutional Networks* [1]. However, in the future, we will expand our work on the other two remaining applications.

As we commented in Section 2.6.1, *Event-based Asynchronous Sparse Convolutional Networks* application is able to solve two different tasks **image classification** and **object detection** and for that it uses two different datasets for each task. To reach our objective and analyze how the noise affects the performance of the algorithm in both tasks, first, we have trained a total of four models, one for each available dataset. Then, we have introduced the noise to the four datasets, in order to create a noisy version of them. Finally, we have used the noisy dataset to test the models that had been trained in the first step with the original datasets, without noise.

In Figure 3.1, we show you a block diagram that helps to visualize the main parts that make up our evaluation framework: the datasets, the format converter, the noise generator, and the classifier or object detector. The noise generator and the format converter are explained below, while the datasets that we use and the results that we have obtained in the classifier or object detector are explained in Section 4.

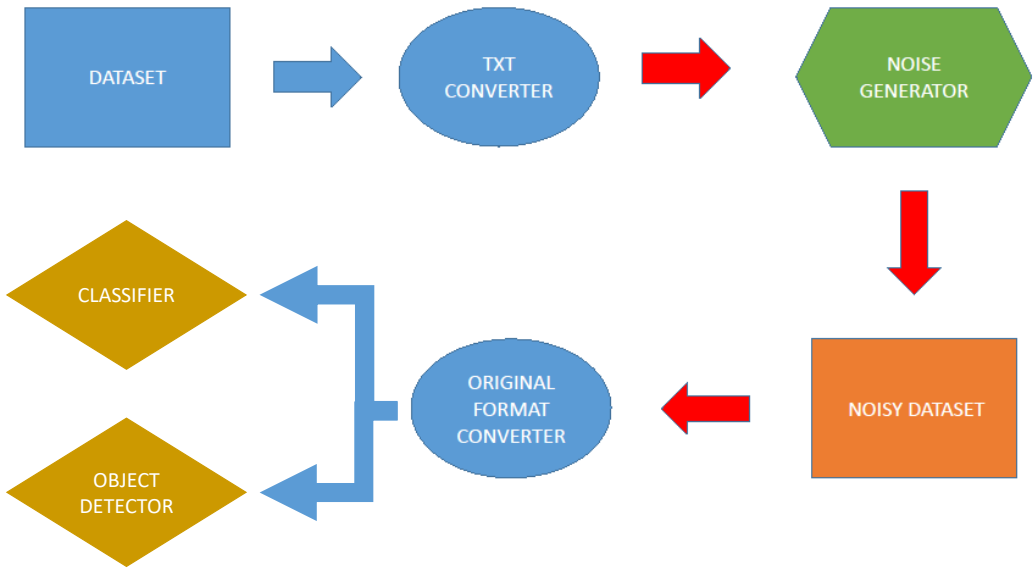


Figure 3.1: In this block diagram, we show the methodology that we have follow during this project.

3.3 Noise Generator

The noise generator has the crucial function of add noise into the datasets that we have used during this project. As the original application does, we will work with the same four datasets:

- **NCaltech101** [66]**:** It is used for image classification task.
- **NCars** [67]**:** It is also used for image classification task.
- **Object Detection NCaltech101** [66]**:** It is used for object detection task.
- **Prophesee Gen1 Automotive** [68]**:** It is use for object detection task.

The four of these datasets will be explained in more detail lately in Section 4.2.

When one of the datasets is introduced into the noise generator, four different types of noises are added. Each of the four noises is from a different nature and they are applied to the different dimensions of an event signal. These four noises that are added are the following ones:

- **Spatial noise (add noise xy):** This type of noise is in charge of modifying the x and y coordinates of each input event, the rest of the variables (polarity and timestamp) remain unchanged.

  The distribution that this noise can adopt can be of two types, *Gaussian* or *Beta*, and they are selected by using the parameter **noise_type**. In our case, we have only use the *Gaussian* distribution.
  While the value of the mean of the distribution is set to 0, the variance is determined by the parameter **p_dg**, which modulates the value of the variance in

such a way that the higher the p_dg, the higher the variance of the spatial noise introduced.
Finally, the variance of the noise is limited by the **width** and **height** values. These values define the diagonal of the sensor that has been used to obtain the events, so they will be different for each dataset. Therefore, we can conclude that the spatial noise variance corresponds to a percentage of the sensor diagonal, which is defined by the value of p_dg.

- **Temporal noise (add noise ts):** In this second type of noise, the variable that is modified is the timestamp.

  Analogous to spatial noise, the distribution of temporal noise can be of two types, *Gaussian* or *Beta*. Again, we have chosen to use the *Gaussian* distribution to create this type of noise.
  The mean value of the distribution is set to 0, and in this case, the variance in this type of noise is limited between 0 and the maximum timestamp value of each sequence of input events.
  The variance is modulated by two different parameters. The first one is **dt**, which is a unit of time that is fixed at the value of 10 ms that, according to a statistical study done by the VPULab, it is the meantime between events. The other parameter that modulates the value of the variance is **p_dt** which defines the percentage of the variance to be applied.

- **Add events:** This third noise, as the name suggests, has the function of adding more events. These events are created randomly, so it has the effect of introducing useless events into the original dataset.

  We are going to explain how one event is created since the rest will do the same. First, the timestamp is created by selecting a random number between zero and the maximum value of the timestamp for the original event. Then, to create the x and y coordinates it is selected a random value between zero and the values of width, in the case of the x coordinate, and height, in the case of y coordinate. These two values vary depending on the sensor that has been used to capture the events. Finally, the polarity is created by a random choice between two possible values, 0 or 1.
  There is a **perct** parameter that defines the percentage of events that will be created out of the total number of events.

- **Remove events:** Finally, the fourth noise that uses in our work, has the opposite function to the previous one, instead of adding events, this type of noise removes them. In this way, the noisy dataset will contain less information than the original dataset.

  This last noise is the simplest one as it just has one parameter that is **perct**, which has the same function as it has in the case of *add events*, that is to define the percentage over the total number of events that are going to be removed.

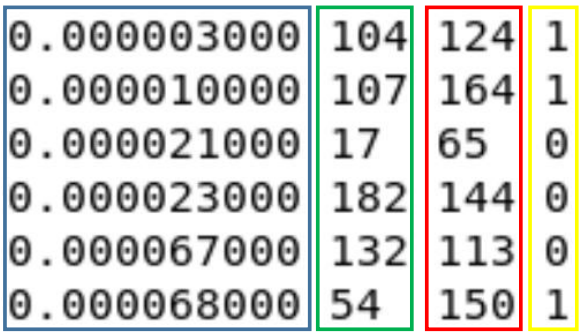


Figure 3.2: In this figure, we can see some examples of how the events are organized in the *.txt* format required by the noise generator. The blue box shows the timestamp values, the green and red boxes show the values of the x and y coordinates respectively, and finally, the yellow box shows the polarity data.

3.4 Format converter

As we have seen during Chapter 2, all the technology related to event cameras has emerged only a few years ago. Because of this, there is no specific format in which events must be stored. This applies both to the variables themselves that form an event (x-coordinate, y-coordinate, timestamp, and polarity), and to the format in which these events are stored, they can be stored in a text format (*.txt*), in binary format (*.bin*), etc. So depending on the dataset that we are working with the format of the events may vary.

This aspect has been a limitation for us because for the noise generator to work properly, it needs a specific format, unfortunately, this format does not match the one used by three of the four datasets we work with. Therefore, we have had to develop two format converters for each case, one to be able to introduce the dataset in the noise generator, and the other to return the noisy dataset to its original format so that we can carry out the tests we consider necessary.

Our noise generator needs the events to be stored in a text format (*.txt*), following the following structure:

- The events must be formed in four columns, one for each of the four variables that make up an event.
- The first column will be filled by the timestamps variable, which must be in second with 9 decimal numbers.
- The second and the third column will be filled by the x and y coordinates respectively, each of both values must be given in pixels.
- Finally, the fourth column corresponds to the polarity that must have the values 0 (for negative) or 1 (for positive).

In Figure 3.2 we show you some examples of how events are organized in *.txt* format before going through the noise generator, following the guidelines described above.

Of the four datasets we work with, only the NCars dataset meets the requirement

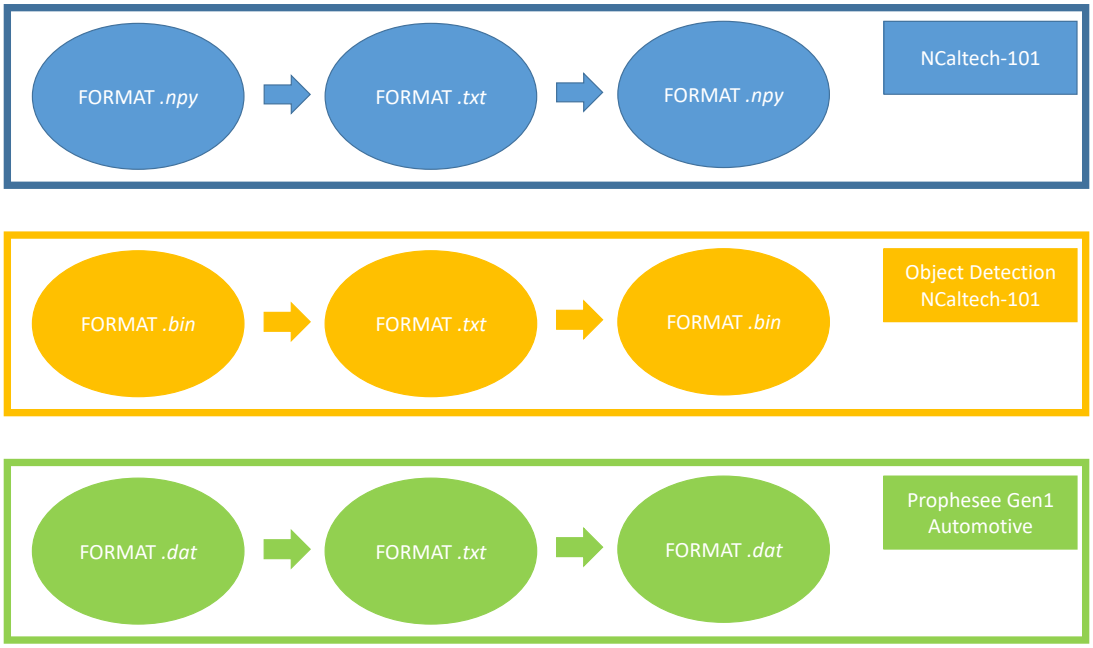


Figure 3.3: In this figure, we show the format converters we have developed for the three datasets that required it, each of them represented with a color.

of storing the events in a text format. Therefore, we needed to develop different converters to each of the datasets that do not store the events in a text file. In these cases, when the noisy datasets were created we needed to develop another converter that returns the original format of the dataset so the application could run correctly. We have needed to follow this process for the cases of NCaltech101 dataset (uses a *.npy* format), NCaltech101 Object Detection dataset (uses a *.bin* format) and Prophesee Gen1 Automotive (uses a *.dat* format). It is important to note that in the datasets dedicated to the object detection task, only the files containing information on the event sequences have been modified and not those containing information on the bounding boxes. In Figure 3.3 we illustrate how these format converters have been created for each of the datasets that need it.

3.5 Implementation details

In this last section of the third chapter, we have introduced certain details of the implementation of the *Event-based Asynchronous Sparse Convolutional Networks* application [1] that we have considered necessary to be able to correctly understand its operation.

The application can perform two types of tasks, image classification and object detection using four different datasets. In addition, its original functionality, as it is explained in Section 2.6.1, was to compare the efficiency of networks using sparse convolutions and those using traditional convolutions. In order to modulate all these factors, we use a file called *settings.yalm* whose most relevant parameters for our interests are:

- **name:** Indicates the name of the dataset that you are going to use . It can be

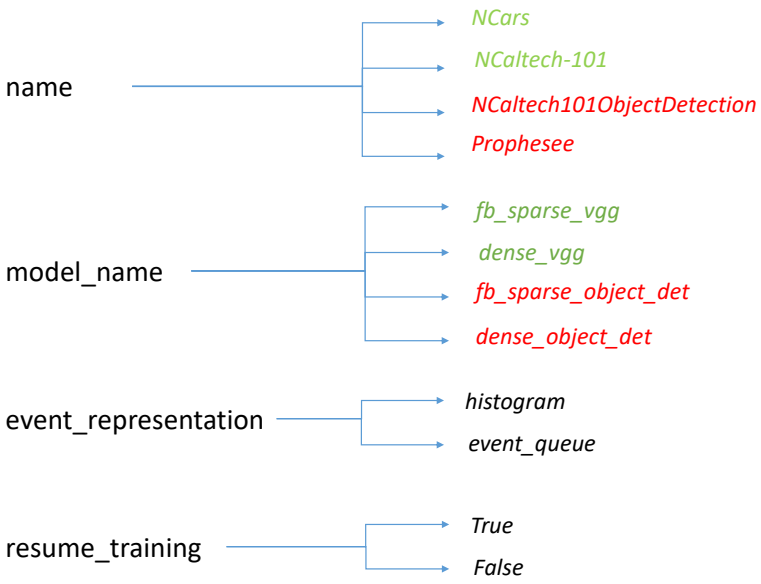


Figure 3.4: In this figure, we can see an outline of relevant *settings.yalm* file parameters. The parameters in green are those related to the image classification task and those in red are those related to the object detection task.

NCaltech101, NCaltech101_ObjectDetection, Prophesee or *NCars*.

- **model_name:** Indicates the kind of model that you want to use depending on the task that you want to solve during your training. It can be *fb_sparse_vgg, dense_vgg, fb_sparse_object_det* or *dense_object_det*. In our case we have just used two of them, the *fb_sparse_vgg* when we where interested in solving the task of **image classification**, and *fb_sparse_object_det* when we wanted to solve the task of **object detection**. We have made this choice based on the results in Table 2.1 and Table 2.2, which show that networks using sparse convolution work more efficiently than networks using traditional convolution.

- **event_representation:** It defines the kind of representation that we are going to use, it can be *histogram* or *event_queue*, both of them defined in Section 2.6.1.

- **resume_training:** This parameter will only be used to test noisy datasets. By setting this parameter to *True*, we can load the model that has been previously trained with the original data, i.e. without noise.

In Figure 3.4 we can visualize an outline in a summary mode of the above-mentioned parameters. The rest of the parameters that appear in the *settings.yaml* are not relevant to our work and according to the authors of the paper, they are the optimal ones.

Chapter 4

Evaluation

4.1 Introduction

In this fourth section, we will evaluate the computer vision application based on deep learning that we have explained in Section 2.6.1, *Event-based Asynchronous Sparse Convolutional Networks* [1].
We will start by explaining the four datasets we have used and then proceed to explain the different tests we have done. This second part is divided into three steps: in the first step, we have replicated the original results of the paper, where for the image classification task we have obtained better results than for the object detection task. In the second step, we have created the different noisy versions of each of the datasets used, except for the case of Prophesee Gen1 Automotive dataset, where we have had problems in the *.txt* to *.dat* format cover. Finally, in the third step, we have analyzed the results obtained when running the application with the different noisy datasets for both tasks, image classification and object detection.

4.2 Datasets

Before explaining the experiments that we have done and analyze the results of each of them, we will study the different datasets that this application [1] uses.
As we have commented in Section 2.6.1, the application that we are working with is able to solve two different tasks:

- Image classification
- Object detection

4.2.1 NCaltech101

The NCaltech101 or Neuromorphic-Caltech101 [66] is a dataset that is used in the task of image classification. This dataset is based on the original Caltech-101 dataset [69, 70], which was dedicated to frame-based cameras. To create the N-Caltech dataset, an ATIS camera [71] was used, which was attached to a pan-tilt monitoring unit that ensured that the sensor was in motion while recording the original Caltech-101 images, which were displayed on an LCD monitor. In Figure 4.1, we can visualize how the ATIS camera and the monitor were positioned to record the sequence of events.

Figure 4.1: In this figure, we can see how the ATIS camera and the LCD monitor were placed to capture the event images that appear in the NCaltech101 dataset. (Source: [66]).

The images of the Caltech101 dataset were of different sizes, so, in order to make the event images more similar in size, after they were captured, the event images were resized to make them as large as possible while maintaining the width-to-height ratio and ensuring that the maximum width was 240 pixels and the maximum height was 180 pixels, as it is explained in [66].

The NCaltech101 dataset is made up of 101 classes that correspond to different elements such as, for example, airplanes, lamps, ants, etc. the distribution of event sequences over these 101 classes do not follow a uniform distribution and makes up a total of 8.709 event sequences. Of the total number of event sequences, 4457 are stored in the train set, 2410 in the validation set and the remaining 1842 in the test set. All of the event sequences that conform to this dataset have a sequence length of 300 ms. The format that is used in this first dataset for storing all the events is a NumPy file (*.npy*). We can some of the event sequences that appear in this dataset in Figure 4.2

Object Detection NCaltech101

There is another version of the NCaltech dataset called Object Detection NCaltech101 [66] that is used for the task of object detection. It is exactly the same data as the previous one, but in this case, we also have information on the 2-dimensional bounding boxes of the objects. In Figure 4.3, we can see some examples of the datasets that appear in the Object Detection NCaltech101 with their bounding boxes.

Other difference between both versions of the dataset is that in this case, the events are stored in a *.bin* file. Each event occupies 40 bits as it is described below:

- **X coordinate:** Correspond to the bits 39-32 (in pixels).
- **Y coordinate:** Correspond to the bits 31-24 (in pixels).

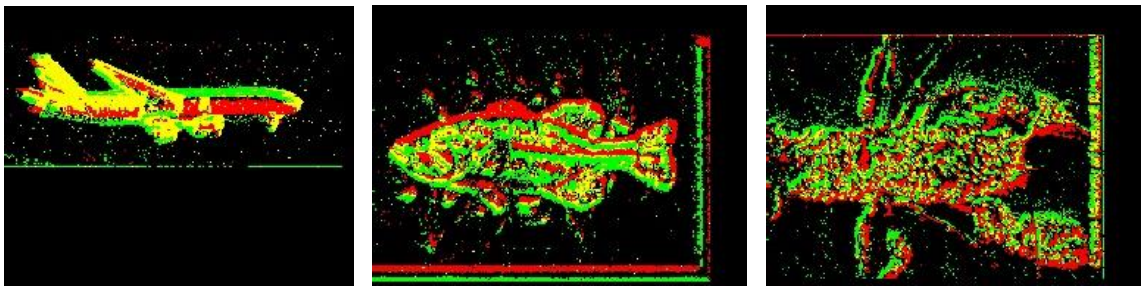


Figure 4.2: Examples of event sequences that compose the NCaltech101 dataset, they correspond to the classes *airplane*, *fish* and *scorpion*. The red points correspond to the positive events, while the green ones correspond to the negative ones. In addition, some yellow points correspond to a fusion of both previous cases, that is, both positive and negative events have been detected at these points.

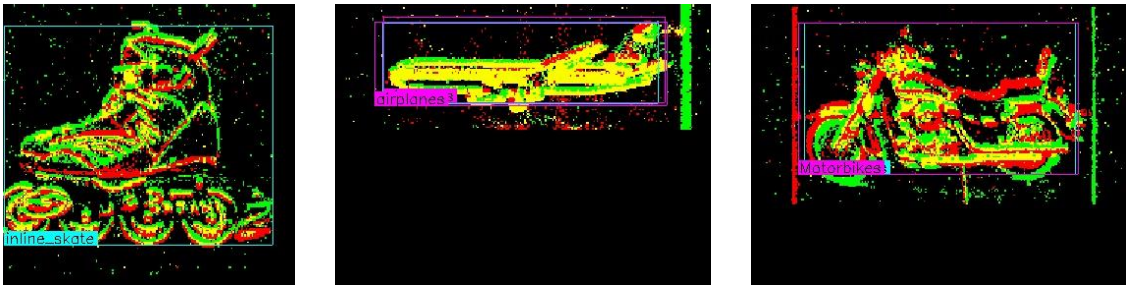


Figure 4.3: Examples of event sequences that compose the NCaltech101 for object detection dataset, they correspond to the classes *inline_skate*, *airplane* and *motorbike*. The color-coding is the same as explained in Figure 4.2, with red dots being positive events, green dots being negative events and yellow dots being the merger of the two.

- **Polarity:** Correspond to the bit 24 (0 for negative and 1 for positive).
- **Timestamp:** Correspond to the bits 23-0 (in microseconds).

4.2.2 NCars

The NCars dataset [67] is dedicated to solving the task of image classification. In this case, the dataset is formed by a total of 24.029 event sequences which are more or less uniformly distributed between two classes, *car* (with 12,336 event sequences) and *non-car* (with 11,693 event sequences). Following this distribution between classes, the total distribution of event sequences between the different sets is as follows: The train set is composed of 15,422 event sequences and the test set is composed of 8,607 event sequences.

For capturing the events of this dataset, an ATIS camera [71] mounted behind the windshield of a car was used. The event sequences that make up the NCars dataset have a length of 100ms and the sensor width and height values are 120 and 100 respectively. Originally, the format of the events that compose the NCars dataset is a binary file (*.bin*), where each event has a total length of 8 bytes or, in other words, 64 bits, that are filled in the following way:

- **Timestamp:** Is encoded using 32 bits.
- **X coordinate:** Is encoded using 14 bits.
- **Y coordinate:** Is encoded using 14 bits.
- **Polarity:** Is encoded using 1 bit (it is encoded as 1/-1).
- **Unused bits:** There are 3 bits left that are not used.

As we have said, this format is used in the original dataset, but for this project, we have used a replica of the original dataset in text file format (*.txt*), that have been developed by the authors of the application [1]. In addition to the fact that the data are in text format, there is another difference concerning the original dataset and that is that a validation data set is added, leaving the distribution between the 3 sets as follows: The training set has 12.961 event sequences, the validation set has 2.462 event sequences and finally, the test set has the remaining 8606 event sequences. In Figure 4.4 we can see some examples of the event sequences that conform to the NCars dataset.

4.2.3 Prophesee Gen1 Automotive

Finally, in this fourth dataset that is called Prophesee Gen1 Automotive [68], we can find a total of 39 hours of recording on different scenarios (urban, highway, suburbs, etc.) which are traduced in a total of 255.781 event sequences that correspond to two different classes *cars* (228.123 event sequences) and *pedestrians* (27.658 event sequences). All the events have been captured with a Prophesee Gen1 sensor [72] mounted on a car, and for this case, the sensor dimensions are 304×240 pixels. This dataset is designed, to solve the task of object detection.

Each data file is stored in a binary file (*.dat*) in a total of 8 bytes following the same

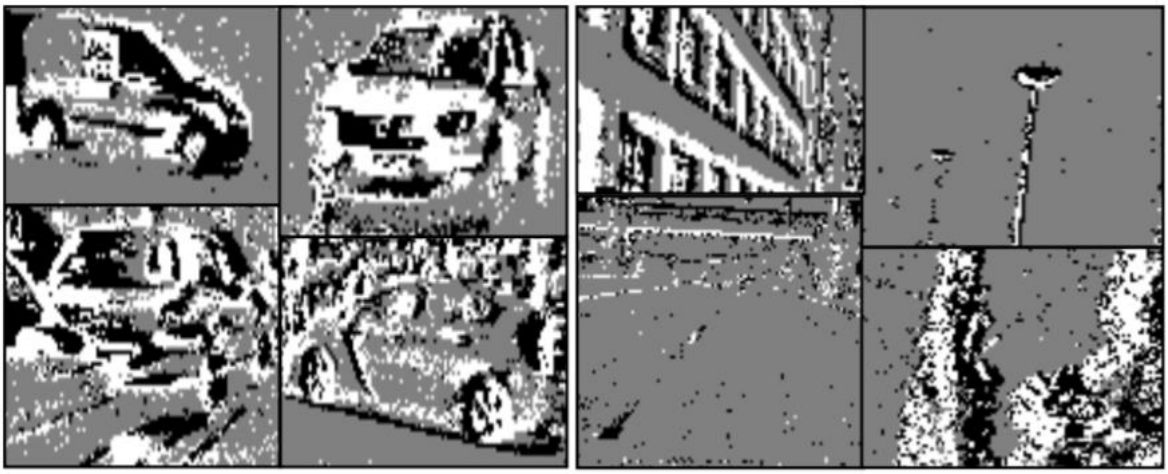


Figure 4.4: Examples of event sequences that compose the NCars dataset, the images that are in the left part correspond to the class: *car*, and the ones that are in the right part of the image correspond to the class: *no-car*. In this case, the white pixels correspond to positive events and the black ones to negative events. (Source: [67]).

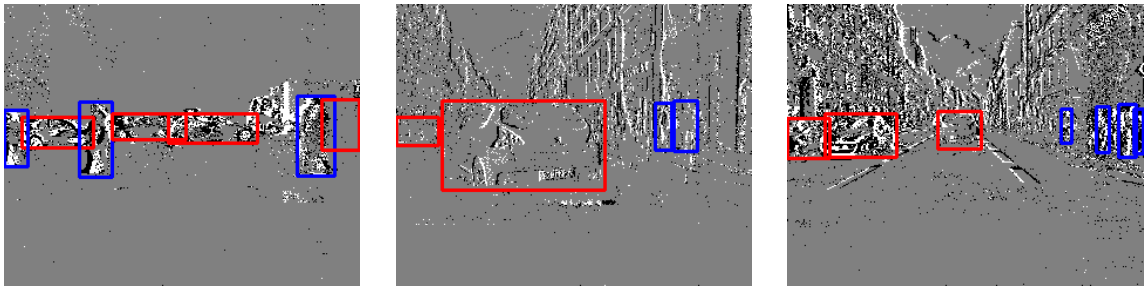


Figure 4.5: Examples of event sequences that compose the Prophesee Gen1 Automotive, in which we can visualize the two different classes that this dataset contains, the detected objects whose bounding boxes are red correspond to the class *car* and the ones whose bounding boxes are blue correspond to the class *pedestrians*. As happen in NCars dataset the white pixels correspond to positive events while the black ones correspond to negative events. (Source [68]).

structure that we have explained in Section 4.2.2, as both datasets have been developed by the same technological company, Prophesee. In addition, as it is a dataset for object detection, it contains for each sequence of events its corresponding *.npy* file in which the bounding boxes in 2 dimensions of the objects that appear. In Figure 4.5 we will find some examples of the kind of events that we will find in this last dataset.

Because the original dataset is too large (200 GB compressed and 700 GB uncompressed) to work on our computers, we have used a smaller version of it. The reduced version we have used has the following distribution: 251 event sequences for both the train and validation sets and 22 event sequences for the test set.

4.3 Experiments and results

In this section, we show you the different experiments and test that we have developed in order of, at the end of the process, analyze the effects on the performance of the application [1] of the different noises that we have explained to you in Section 3.3.
We have divided our experiments into three main steps:

1. Replicate the original results to verify whether the results indicated in the paper are achievable.

2. Introduce the different types of noises in the dataset that we are working with.

3. Run the application with the noisy dataset and analyze the results.

4.3.1 Step 1: Replicate original results

The main goal of this first step is to develop a baseline model whose performance is as close as possible to that achieved by the original authors of the paper [1]. For achieving this first objective we have to use the code that the authors of the paper themselves provide on GitHub [73]. We have divided this section into two to clarify the training and the results for both proposed tasks, image classification, and object recognition.

Image Classification

We have trained a total of four models, two of them use the NCaltech dataset, and the other two the NCars dataset. We have done this because we are interested in analyse both types of event representations available (*histogram* representation and *event_queue* representation). For all the cases we have selected the model *fb_sparse_vgg*.

In Table 4.1 we can see a comparison between the results that we have obtained and the ones that the authors achieved for this first task. When we analyze the results obtained, we can see that we have fulfilled the main objective of *Step 1*, which was to replicate, in an approximate way, the results obtained by the authors of the paper. A remarkable aspect that we may observe while analyzing the results obtained is that in both cases, for both datasets NCaltech101 and NCars, when we use the *histogram* representation we obtain a value equal to or slightly lower than that obtained by the original authors, but when we use the *event queue* representation the results we obtain exceed those obtained by the authors of the paper, using the same parameters for

Dataset	Event representation	Authors	Accuracy
NCaltech	Histogram	Original	0.745
		TFM	0.740
	Event queue	Original	0.741
		TFM	0.760
NCars	Histogram	Original	0.944
		TFM	0.944
	Event queue	Original	0.936
		TFM	0.949

Table 4.1: We can see the results of the original authors in comparison with the ones that we obtain in the task of image classification. We have evaluated both of the available event representations.

Dataset	Event representation	Authors	mAP
NCaltech101 Object Detection	Histogram	Original	0.615
		TFM	0.523
	Event queue	Original	0.643
		TFM	0.544
Gen1 Automotive	Histogram	Original	0.119
		TFM	0.048
	Event queue	Original	0.129
		TFM	0.030

Table 4.2: We can see the results of the original authors in comparison with the ones that we obtain in the task of object detection. We have evaluated both of the available event representations.

training, or at least the ones that they indicate in their paper.

Object recognition

Once that we have obtained our base models and we have analyzed the results obtained in the task of image classification, we will follow the same procedure for the second task that this application can cover, **object detection**. In this case, we have selected the sparse model that is dedicated to object recognition, *fb_sparse_object_det*. Analogously to the image classification task, we can visualize the results obtained in the object detection task in the following Table 4.2.

Analyzing the results that we have obtained in this second table, we can realise that we have not been able to achieve the results indicated in the original paper in any of the proposed cases. In the case of the NCaltech for Object Detection dataset, these results may be due to the fact that we had to use a different Pytorch function for its implementation due to a version problem. The function we have had problems with is *torchvision.ops.nms()* and the one we have used to replace it can be found in [74].

	Type of noise	Parameter	Value of parameter										
Dataset	Add noise xy	p_dg	0	0.025	0.005	0.0075	0.01	0.015	0.02	0.025	0.03	0.04	0.05
	Add noise ts	p_dt	0	0.05	0.1	0.15	0.2	0.25	0.3	0.4	0.5	0.6	0.7
	Add events	perct	0	0.1	0.2	0.25	0.3	0.4	0.5	0.6	0.7	0.8	0.9
	Remove events	perct	0	0.1	0.2	0.25	0.3	0.4	0.5	0.6	0.7	0.8	0.9

Table 4.3: In this table we show you the number of noisy dataset that are created for each of the original dataset. For each of the values that use p_dg, p_dt, or perct, a new noisy dataset is created.

In the case of the Prophesee Gen1 Automotive dataset, the drop in the mAP results, which are originally low, is since, as explained in Section 4.2.3, we have been forced to use a reduced version of this dataset due to its large size, while the results of the paper have been obtained using the full dataset for training.

4.3.2 Step 2: Introduce noise in the dataset

In this second step, we have created the noisy datasets that we will evaluate in the final *Step 3*. To do that, first, we must transform to text format (*.txt*) all the datasets that do not have this format in their original form, that is, all except NCars dataset. Then, using the noise generator, we will add the different types of noise to each dataset, and finally, we will use the format converter to return the datasets to their original format. Both concepts, the noise generator and the format converter, have been previously explained in Sections 3.3 and 3.4 respectively.

Next, as a reminder, we will describe very briefly the noises we have introduced in the four datasets:

- **Spatial noise:** It is also called *add noise xy* and its main objective is to introduce noise in the x and y coordinates that compose an event.
- **Temporal noise:** It is also called *add noise ts* and is responsible for introducing noise into the timestamp of each event.
- **Add events:** It has the function of introducing events that have been created randomly to each sequence of events.
- **Remove events:** It has the function to remove events from each sequence of events.

Each type of noise has a parameter that serves to regulate the amount of noise that is introduced into the dataset. In the case of spatial and temporal noise, the parameters *p_dg* and *p_dt*, respectively, are in charge of modulating the variance of the distribution of both noises, in the case of add and remove events, the parameter *perct* is in charge of defining the percentage of events that will be created or removed.
In order to analyze the influence of noise on the precision results, we have decided to create 10 different datasets of each type of noise, in which the parameters mentioned above will vary. In Table 4.3, we can see an example of how many noisy datasets are created for each of the original datasets.

In the case of the Prophesee Gen1 Automotive dataset, we have not been able to

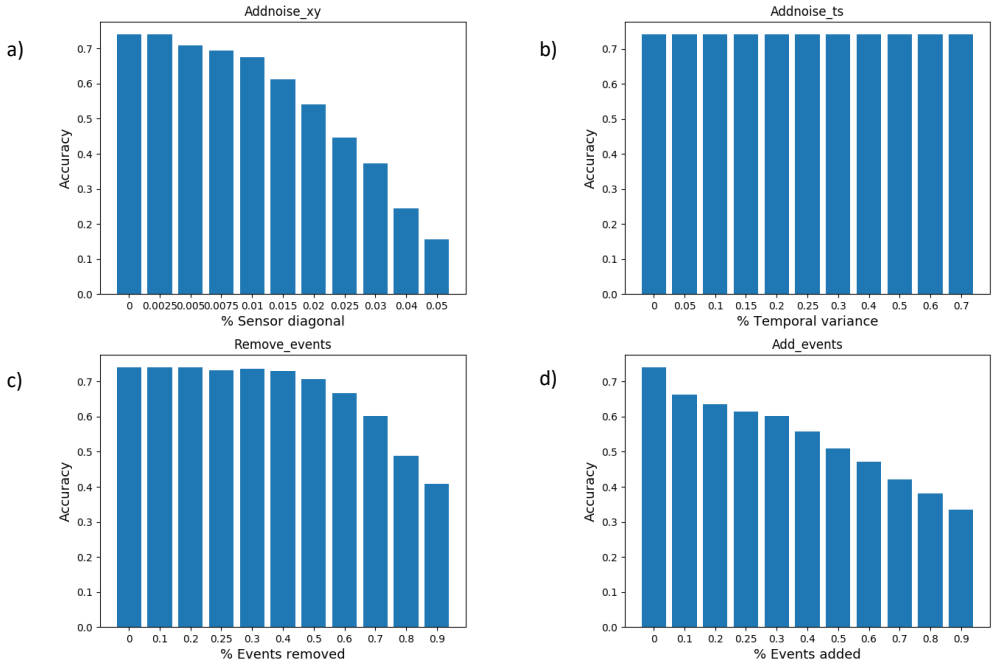


Figure 4.6: The figure shows the results that we have obtained using the different noisy versions that we have created from the NCaltech101 dataset. The vertical axis represents the accuracy and the horizontal axis represents the noise ratio that have been introduced.

overcome this second step. The problems have arisen when running the *.txt* to *.dat* format converter, described in Section 3.4, in which we get failures in the use of the RAM of the computer we are working with. This is why we have decided to propose as a future task to optimize this second converter in order to be able to make a complete analysis of this fourth dataset.

4.3.3 Step 3: Run the noisy dataset and analyse the results

Once that we have all of our four noisy datasets in the correct formats to make the application performs correctly, we will advance to this final step.

The main objective of this *Step 3* is to test the models that we have trained in the *Step 1* (with the original dataset) using the noisy datasets that we have created. Once we have all our experiments done, we have analyzed the results obtained in order to understand how the different kinds of noises that we have introduced affect the performance of the application in both tasks that it is able to solve, image classification and object detection. Is important to add that, the default setting for these tests is to use the histogram representation, otherwise, if in some case we have to use the event queue representation we will make sure it is clear.

Results on image classification

We will start by analysing the results that we have obtained using the noisy versions of the NCaltech101 dataset (Figure 4.6) and the NCars dataset (Figure 4.7), which are the ones used in the image classification task.

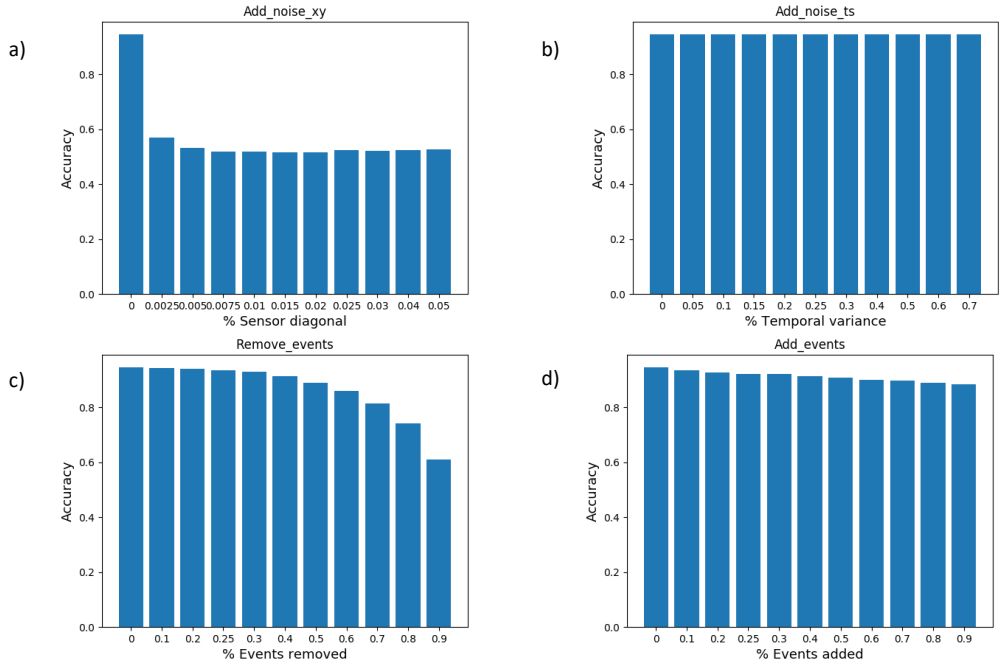


Figure 4.7: The figure shows the results that we have obtained using the different noisy versions that we have created from the NCars dataset. The vertical axis represents the accuracy and the horizontal axis represents the noise ratio that have been introduced.

In the case of spatial noise (*add noise xy*) we can see how, in the NCaltech101 graphic, the results tend to get worse as the value of the noise ratio, which in this case corresponds to the percentage of the diagonal of the sensor, increases. This noise is the one that suffers a greater drop in performance, starting, as we have seen in Table 4.1, with an accuracy of 0.741 and ending, with a value of 0.155.
When we analyze this type of noise in the case of NCars dataset, if we look once again at Table 4.1, is 0.944, we can see how this value drops drastically at the first non-zero value of the noise ratio. Also is remarkable that as the noise ratio increases to 0.0075, the accuracy value is stuck at 50%, which would be the analog of a random choice between two possible classes. The conclusion we can extract for this first noise is that spatial information plays a key role in the image classification task. In Figure 4.8, we can see how, although the noise introduced in both images is different, the effect is devastating in both, as it almost completely modifies the edges of the car being captured.

On graphic b), we can see the one that corresponds to the temporal noise (*add noise ts*). The impact that this kind of noise has on the performance in comparison to the previous one is totally different, because it is almost non-existent for both cases, for both NCaltech101 and NCars. The variation over the different values of the noise ratio is negligible. With these results, we can extract that, in comparison with the other noises, the variable of the timestamp has almost no impact on the performance of the task of image classification.
As we explain in Section 2.6.1, the *histogram* representation does not look at the timestamp information to divide the events, it just separates them using polarity information. In contrast, *event queue* representation uses event queues of a fixed length

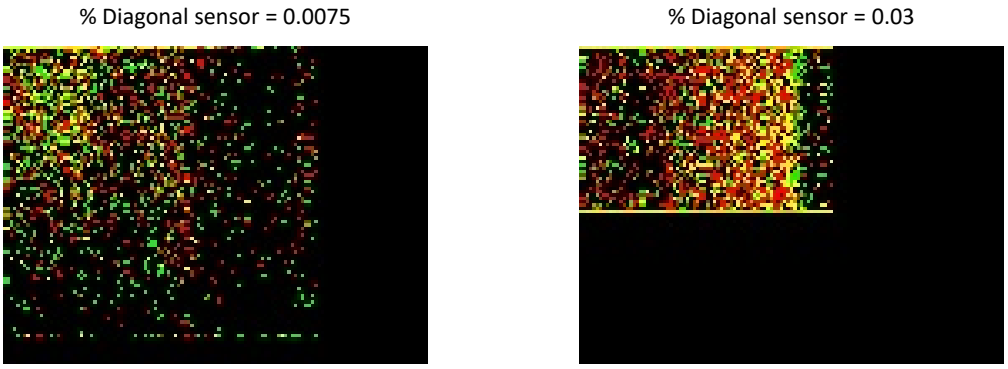


Figure 4.8: The figure shows two event sequences corresponding to the spatial noise added in the NCars dataset. Both event sequences correspond to the class *car*.

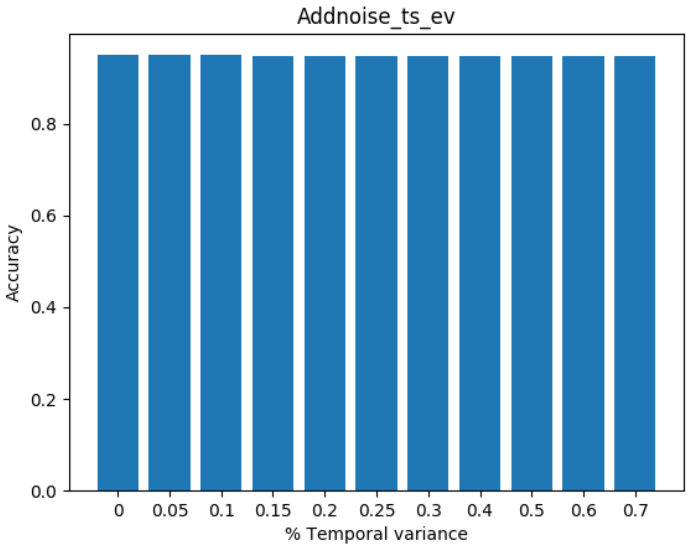


Figure 4.9: The figure shows the results of Addnoise ts using *event queue* representation on the NCars dataset.

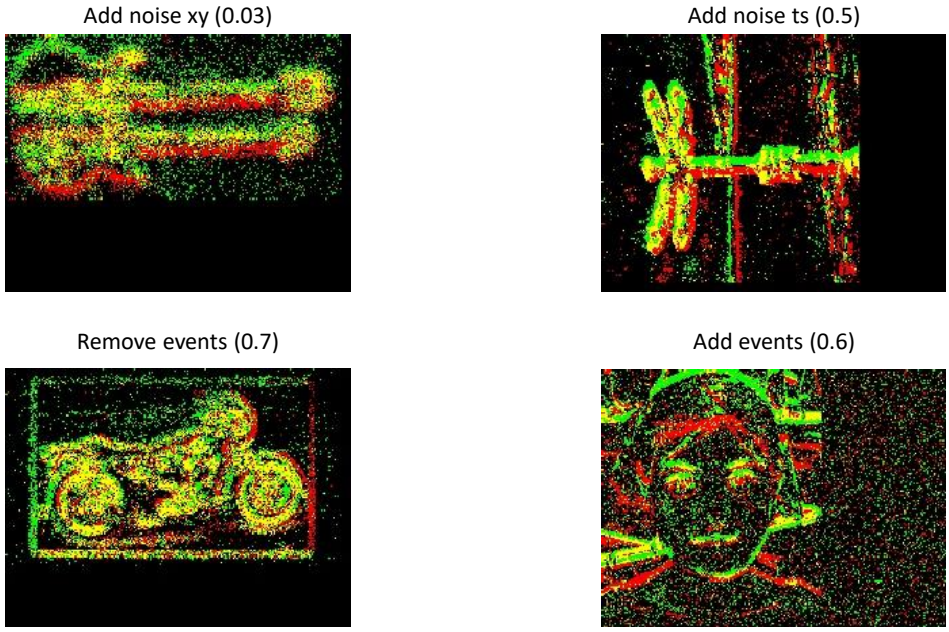


Figure 4.10: The figure shows some examples of the event sequences that we have created from the original NCaltech101 dataset for each type of noise, the images correspond to the classes: *electric guitar*, *ceiling fan*, *motorbike* and *face easy*.

of 15 events, where events are sorted by their timestamps. Therefore, we decided to test this noise on the models that had been trained using the event queue representation. In the case of NCaltech101 dataset, we have not been able to obtain any result due to programming problems that we have not been able to solve. However, in Figure 4.9 we can see the results obtained for NCars dataset, and, again, we can see that although the *event queue* representation does take into account the timestamps of the events, the modification of these has no influence on the accuracy results.

The following noise c), correspond to *remove events*, as a little reminder, this noise was in charge of removing events. We can see that until the value of the noise ratio is not high enough (approximately 0.4 in both cases), the performance of the algorithm does not decrease, actually, in some cases, it does improve a little bit. On the other hand, when the noise ratio gets higher values, the performance starts to decrease unquestionably. It is remarkable that, in the NCaltech101 dataset, even though we remove the 90% of the events, that correspond to the last case, the value of the accuracy is higher than the one that we obtain in the worst case of *add noise xy*. As a conclusion of this type of noise, we can say that, until the percentage of events that are eliminated does not reach a value of approximately 40%, it does not have a great influence on the image classification task.

Finally, the last noise that we can find on the image is *add events*, whose main objective is contrary to the one that we just have analyzed. In this case, when we look at the NCaltech101 results, the drop of the accuracy along the noise ratio values is similar to the one that happens in *add noise xy*, but in this case, it does not as worse as in the add noise xy one. The way of how the performance drops make a lot of sense because we are adding random events, so when we have a higher value of the noise ratio, you will

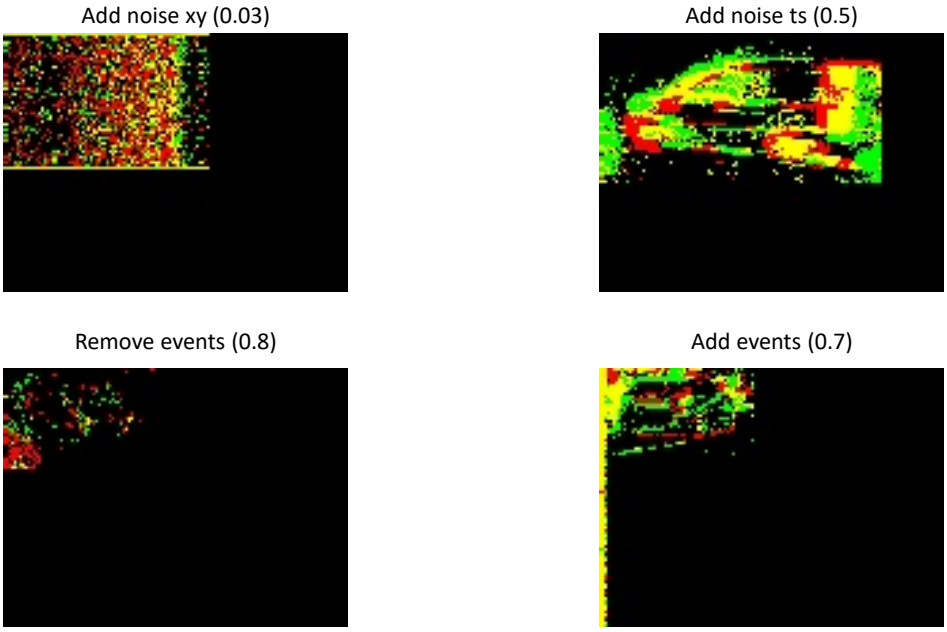


Figure 4.11: The figure shows some examples of the event sequences that we have created from the original NCars dataset for each type of noise, all of the figures corredpond to the class *car*.

create more random events and more difficult will be the task of image classification. But when we look at the results of NCars dataset, we can see some differences, obviously, the performance of the algorithm get worse when we increase the noise ratio and we add more events, but the slope of the fall is much less steep than in the previous case, in fact, when the noise ratio adopts its maximum value (0.9), the algorithm returns an accuracy of 0.883 which is only one point less than the initial accuracy, 0.944.

Additionally, in Figures 4.10 and 4.11, we show some examples of event sequences belonging to the different noisy versions of NCaltech101 and NCars respectively. In both figures, we can see how the spatial noise modifies the edges of the objects being displayed, while spatial noise does not seem to modify the event sequences too much. Additionally, you can see how both add events and remove events add or remove events.

Results on Object Detection

In this part, we have only included the results of NCaltech101 Object Detection dataset due to the memory problems that we have with the Prophesee Gen1 Automotive dataset while converting the formats.

In Figure 4.12 we can see how the results that we have obtained look like. We will start analyzing graph a) of spatial noise (*add noise xy*) which has a very familiar aspect to us, as it suffers a huge drop along with the different values of the noise ratio, like in the image classification task. In comparison with the other three graphics, it is the one that shows a higher influence on the performance of the algorithm. Once again, we can conclude by saying that for the object detection task it is very important that the spatial localization of the events is correct, i.e. not distorted.

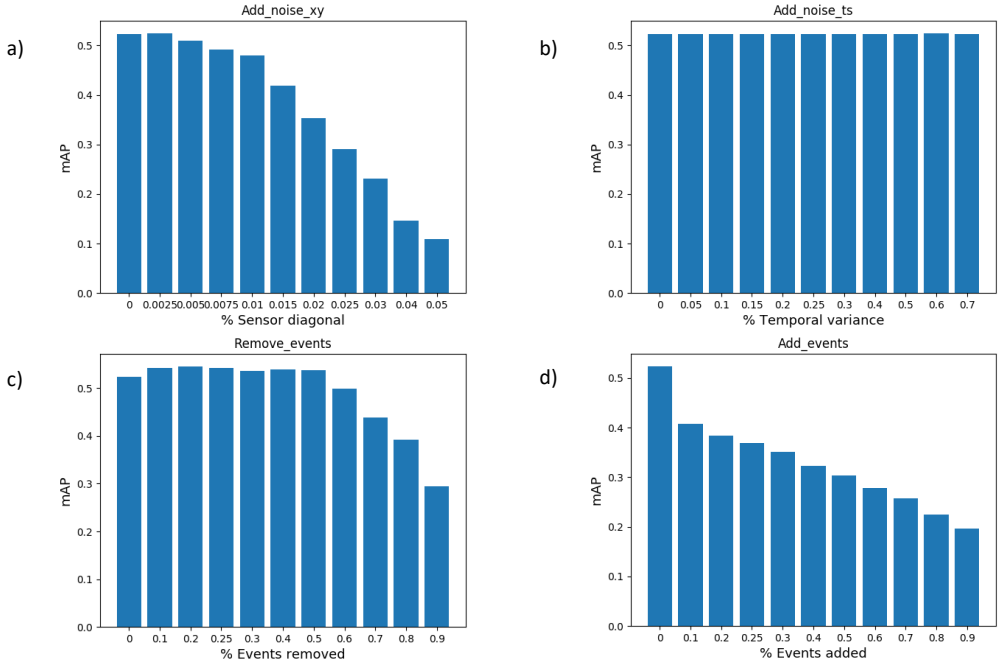


Figure 4.12: The figure shows the results that we have obtained using the different noisy versions that we have created from the NCaltech101 Object Detection dataset. The vertical axis represents the accuracy and the horizontal axis represents the noise ratio that have been introduced.

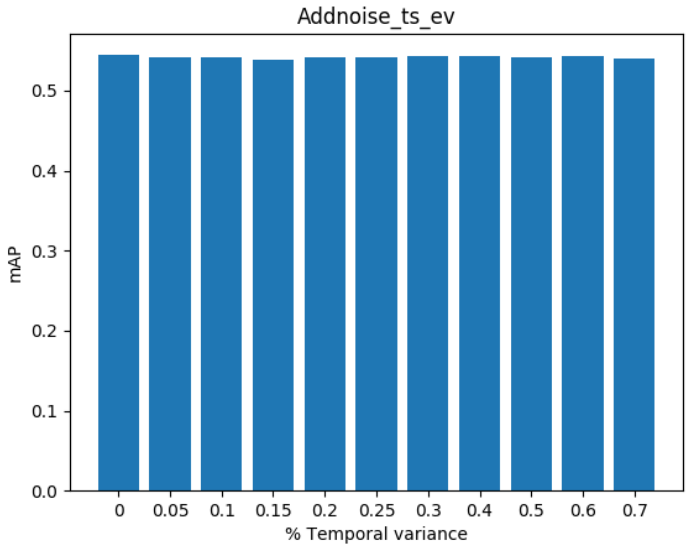


Figure 4.13: The figure shows the results of Addnoise ts using *event queue* representation on the NCaltech101 Object Detection dataset dataset.

In graphic b) of temporal noise (*add noise ts*), we can see how the behavior of the performance of the algorithm along with the different values of the noise ratio still very similar to the other cases that we have previously seen. During the eleven different executions, the value of mAP is stacked around the initial value, that as we have seen in Table 4.2, is 0.523.
In the same way, as we have done in the case of image classification, we have tried to test this same noise on a model that has been trained using the event queue representation. In Figure 4.13 we can see the results we have obtained, and as in the image classification task, we can see how the temporal noise has a minimal effect on the mAP results of the application.

The next noise that we are going to analyze is *remove events*, that is figure c). We can see how for the first values of the noise ratio, the performance of the algorithm improves a little bit, but, as was also the case in the image classification task, when the noise ratio reaches a value of 0.5, the performance of the algorithm starts to decrease, reaching, in the last case, a value of 0.293.

Ultimately, the last noise of this dataset that we are going to analyze is *add events*, which has a behavior very similar to the case of *add noise xy*, with a progressive drop of the mAP along the values adopted by the noise ratio. The most remarkable thing about this last graph is that in the first value of noise ratio, which means the inclusion of a small number of random events, the drop in performance is much greater than in the rest, causing a notable step in the graph we are analyzing.

Chapter 5

Conclusions and future work

5.1 Conclusions

The main objective of this work was the analysis of the influence of different types of noise on the results in the tasks of image classification and object detection on the application, *Event-based Asynchronous Sparse Convolutional Networks* [1]. The key points we can highlight from this study are:

- First of all, we would like to emphasize the knowledge acquired in the study of the state of the art, where we have defined event cameras, thus learning about their fundamentals and the advantages they have over traditional cameras. In addition, we have also studied different examples of computer vision and deep learning applications that use this technology.

- In Section 3.4, we have seen how, since there is no specific format in which to store the events, it was required to develop different format converters to be able to work with the different datasets proposed.

- The experiments we have carried out have been divided into three main steps. In *Step 1*, which consisted of replicating the results obtained by the original authors in their paper, we have seen how, in the case of the image classification task, the results have been closer to those published in the paper than in the case of object detection. This is because for the NCaltech for Object Detection dataset we have used different versions in some libraries as we have seen in Section 4.3.1. On the other hand, in the case of the Prophesee Gen1 Automotive dataset, was because we were forced to use a smaller version of the original dataset due to its large size (700 GB).

- In Section 4.3.3, where we have analyzed the results obtained when testing the application with the noisy datasets, we have seen how the variation of the location of the events (*temporal noise*), has a very big influence on the results obtained in the image classification task as well as in the object detection task.
  In contrast, the temporal information provided by the timestamps has hardly any influence on the performance of the two types of tasks with either of the two event representations used, *histogram* representation and *event queue* representation, because none of them prioritised temporal data over other types of data such as spatial data.
  In addition, we have seen that for the two proposed tasks, removing events does

not affect performance until 40%/50% of the events in each sequence are deleted. Finally, we have observed that the addition of events worsens the performance of the algorithm, however, depending on each dataset, this drop in performance has a more or less pronounced slope. For example, the performance drop in the NCars dataset is less pronounced than in the case of NCaltech101 and NCaltech101 Object Detection datasets.

5.2 Future work

Once we have analysed and drawn conclusions from the results we have obtained throughout the different experiments carried out during the project, we are going to propose a series of tasks that could be done in the future to make a broader study of how noise in event signals affects the performance of different computer vision applications based on deep learning:

- First of all, it would be important to solve the memory problems that we have encountered in the *.txt* to *.dat* converter for the Prophesee Gen1 Automotive dataset (explained in Section 4.3.2, to be able to make a more exhaustive analysis of the application. In the same way, it would be necessary to train the model using the complete dataset and not the reduced version we have had to work with, in order to obtain results more similar to those obtained by the authors of the paper.

- Another proposition is to use other computer vision applications that make use of event signals and that are based on deep learning in order to perform more sophisticated analysis. To do so, we propose to carry out the same procedure as the one we have done in this project. The applications we would like to use are the ones we have explained in Sections 2.6.2 and 2.6.3, corresponding to *Convolutional spiking neural networks for spatio-temporal feature extraction* [40] and *Inceptive Event Time-Surfaces for Object Classification using Neuromorphic Cameras* [75] respectively.

- Another line of future work would be to compare these results with the results obtained from the metrics developed by VPULab, whose aim is to serve as a predictor of the behavior of the signals generated with the simulators, to analyze the level of correlation between the event distortion metric and the performance.

- As we have commented in Section 3.3, when adding noise both in the x and y coordinates (spatial noise) and when adding noise on the timestamp (temporal noise), we could choose to adopt two types of distributions, *Gaussian* and *Beta*, in our case we have only focused on the *Gaussian* type, so it could be interesting to create noise using exactly the same parameters except for the distribution, selecting the*Beta* distribution this time. In this way we could compare the results and analyse the influence of the distribution that the noise adopts when it is introduced in the performance of the algorithm.

- Finally, in order to further analyse of this application, another possible future work would be to carry out the experiments with the second type of event representation of the application, as we have seen in Section 2.6.1, we could choose

between the *histogram* representation or the *event queue* representation and, although in step 1 we have analysed the results using both types of representations, in step 3, in which we analysed the influence of noise on the performance of the algorithm, we have test the *event queue* representation in the case of the temporal noise. Therefore, it would be interesting to obtain the results with the other noises using the *event queue* representation and compare them with those we have obtained using *histogram* representation.

Bibliography

[1] N. Messikommer, D. Gehrig, A. Loquercio, and D. Scaramuzza, "Event-based asynchronous sparse convolutional networks," 2020. 1, 12, 14, 16, 17, 18, 23, 27, 29, 32, 34, 45

[2] G. Indiveri and R. Douglas, "Neuromorphic vision sensors," *Science*, vol. 288, no. 5469, pp. 1189–1190, 2000. 3

[3] Anonymous, "How the eye works." `https://www.essiloriraq.com/`, 2020. Accessed June 2020. 4

[4] P. K. Brown and G. Wald, "Visual pigments in single rods and cones of the human retina," *Science*, vol. 144, no. 3614, pp. 45–52, 1964. 3

[5] H. Rebecq, D. Gehrig, and D. Scaramuzza, "Esim: an open event camera simulator," in *Conference on Robot Learning*, pp. 969–982, PMLR, 2018. 4, 15

[6] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "Events-to-video: Bringing modern computer vision to event cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 3, 9

[7] S. Dai and Y. Wu, "Motion from blur," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2008. 5, 6

[8] G. Gallego, T. Delbruck, G. M. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. Davison, J. Conradt, K. Daniilidis, and et al., "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, p. 1–1, 2020. 5, 7, 14

[9] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128 128 120 db 15 s latency asynchronous temporal contrast vision sensor," 2006. 5

[10] C. Scheerlinck, N. Barnes, and R. Mahony, "Continuous-time intensity estimation using event cameras," 2018. 7

[11] S. K. Moore, "Prophesee's event-based camera reaches high resolution." `https://spectrum.ieee.org/`, February 2020. Accessed June 2020. 7

[12] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006. 7

[13] U. Frese, "Treemap: An o (log n) algorithm for indoor simultaneous localization and mapping," *Autonomous Robots*, vol. 21, no. 2, pp. 103–122, 2006. 7

[14] M. Milford, E. Vig, W. Scheirer, and D. Cox, "Vision-based simultaneous localization and mapping in changing outdoor environments," *Journal of Field Robotics*, vol. 31, no. 5, pp. 780–802, 2014. 7

[15] X. Yuan, J.-F. Martínez, M. Eckert, and L. López-Santidrián, "An improved otsu threshold segmentation method for underwater simultaneous localization and mapping-based navigation," *Sensors*, vol. 16, no. 7, p. 1148, 2016. 7

[16] A. Nemra and N. Aouf, "Robust airborne 3d visual simultaneous localization and mapping with observability and consistency analysis," *Journal of Intelligent and Robotic Systems*, vol. 55, no. 4, pp. 345–376, 2009. 7

[17] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high-speed scenarios," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 994–1001, 2018. 8

[18] J. Xu, M. Jiang, L. Yu, W. Yang, and W. Wang, "Robust motion compensation for event cameras with smooth constraint," *IEEE Transactions on Computational Imaging*, vol. 6, pp. 604–614, 2020. 7

[19] A. Bhoi, "Monocular depth estimation: A survey," 2019. 8

[20] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger, "Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8445–8453, 2019. 8

[21] S. Walz, T. Gruber, W. Ritter, and K. Dietmayer, "Uncertainty depth estimation with gated images for 3d reconstruction," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–8, 2020. 8

[22] J. Hidalgo-Carrió, D. Gehrig, and D. Scaramuzza, "Learning monocular dense depth from events," *arXiv preprint arXiv:2010.08350*, 2020. 8, 9

[23] T. Leroux, S.-H. Ieng, and R. Benosman, "Event-based structured light for depth reconstruction using frequency tagged light patterns," *arXiv preprint arXiv:1811.10771*, 2018. 8

[24] H. Rebecq, R. Ranftl, V. Koltun, and D. Scaramuzza, "High speed and high dynamic range video with an event camera," *IEEE transactions on pattern analysis and machine intelligence*, 2019. 8

[25] T. Stoffregen, C. Scheerlinck, D. Scaramuzza, T. Drummond, N. Barnes, L. Kleeman, and R. Mahony, "Reducing the sim-to-real gap for event cameras," in *Proc. Eur. Conf. Comput. Vis.*, Springer, 2020. 8

[26] S. Barua, Y. Miyatani, and A. Veeraraghavan, "Direct face detection and video reconstruction from event cameras," in *2016 IEEE winter conference on applications of computer vision (WACV)*, pp. 1–9, IEEE, 2016. 9

[27] G. Haessig, F. Galluppi, X. Lagorce, and R. Benosman, "Neuromorphic networks on the spinnaker platform," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pp. 86–91, IEEE, 2019. 9

[28] J. Nagata, Y. Sekikawa, K. Hara, and Y. Aoki, "Foe-based regularization for optical flow estimation from an in-vehicle event camera," *Electronics and Communications in Japan*, vol. 103, no. 1-4, pp. 19–25, 2020. 9

[29] C. Reinbacher, G. Munda, and T. Pock, "Real-time panoramic tracking for event cameras," in *2017 IEEE International Conference on Computational Photography (ICCP)*, pp. 1–9, IEEE, 2017. 9

[30] A. Glover and C. Bartolozzi, "Robust visual tracking with a freely-moving event camera," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3769–3776, IEEE, 2017. 9

[31] I. Alonso and A. C. Murillo, "Ev-segnet: Semantic segmentation for event-based cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019. 9

[32] J. Zhang, K. Yang, and R. Stiefelhagen, "Issafe: Improving semantic segmentation in accidents by fusing event-based data," *arXiv preprint arXiv:2008.08974*, 2020. 9

[33] S. G. Wysoski, L. Benuskova, and N. Kasabov, "Adaptive learning procedure for a network of spiking neurons and visual pattern recognition," in *International conference on advanced concepts for intelligent vision systems*, pp. 1133–1142, Springer, 2006. 9, 10

[34] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Communications*, vol. 9, Jun 2018. 9, 11

[35] D. Soni, "Spiking neural networks, the next generation of machine learning." `https://towardsdatascience.com/`, January 2018. Accessed June 2020. 10

[36] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Networks*, vol. 111, pp. 47–63, 2019. 10

[37] J. N. Allen, H. S. Abdel-Aty-Zohdy, and R. L. Ewing, "Cognitive processing using spiking neural networks," in *Proceedings of the IEEE 2009 National Aerospace Electronics Conference (NAECON)*, pp. 56–64, 2009. 10

[38] S. Lobov, V. Mironov, I. Kastalskiy, and V. Kazantsev, "A spiking neural network in semg feature extraction," *Sensors*, vol. 15, no. 11, pp. 27894–27904, 2015. 10

[39] R. Vaila, J. Chiasson, and V. Saxena, "Deep convolutional spiking neural networks for image classification," *arXiv preprint arXiv:1903.12272*, 2019. 10

[40] A. Samadzadeh, F. S. T. Far, A. Javadi, A. Nickabadi, and M. H. Chehreghani, "Convolutional spiking neural networks for spatio-temporal feature extraction," 2020. 10, 18, 19, 20, 46

[41] M. Hopkins, G. Pineda-García, P. A. Bogdan, and S. B. Furber, "Spiking neural networks for computer vision," *Interface focus*, vol. 8, no. 4, p. 20180007, 2018. 10

[42] A. Ardakani, C. Condo, and W. J. Gross, "Sparsely-connected neural networks: Towards efficient vlsi implementation of deep neural networks," 2017. 11

[43] S. Dey, K.-W. Huang, P. A. Beerel, and K. M. Chugg, "Characterizing sparse connectivity patterns in neural networks," *2018 Information Theory and Applications Workshop (ITA)*, Feb 2018. 11

[44] B. Graham, M. Engelcke, and L. van der Maaten, "3d semantic segmentation with submanifold sparse convolutional networks," 2017. 11, 12, 14

[45] X. Li, J. Guivant, N. Kwok, Y. Xu, R. Li, and H. Wu, "Three-dimensional backbone network for 3d object detection in traffic scenes," 2019. 12

[46] B. Graham and L. van der Maaten, "Submanifold sparse convolutional networks," 2017. 12

[47] Z. Zhou, "How does sparse convolution work?." `https://towardsdatascience.com/`, December 2020. Accessed June 2020. 13

[48] C. Scheerlinck, H. Rebecq, T. Stoffregen, N. Barnes, R. Mahony, and D. Scaramuzza, "Ced: Color event camera dataset," 2019. 15

[49] E. Mueggler, H. Rebecq, G. Gallego, T. Delbruck, and D. Scaramuzza, "The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam," *The International Journal of Robotics Research*, vol. 36, p. 142–149, Feb 2017. 15

[50] D. Gehrig, M. Gehrig, J. Hidalgo-Carrió, and D. Scaramuzza, "Video to events: Recycling video datasets for event cameras," 2020. 15

[51] "Event-based vision resources." `https://github.com/uzh-rpg/event-based_vision_resources`. Accessed June 2020. 15

[52] Y. Bi and Y. Andreopoulos, "Pix2nvs: Parameterized conversion of pixel-domain video frames to neuromorphic vision streams," in *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 1990–1994, 2017. 15

[53] S. Abu-El-Haija, N. Kothari, J. Lee, P. Natsev, G. Toderici, B. Varadarajan, and S. Vijayanarasimhan, "Youtube-8m: A large-scale video classification benchmark," 2016. 15

[54] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li, "Yfcc100m: The new data in multimedia research," *Commun. ACM*, vol. 59, p. 64–73, Jan. 2016. 15

[55] Y. Hu, S.-C. Liu, and T. Delbruck, "v2e: From video frames to realistic dvs events," *arXiv e-prints*, pp. arXiv–2006, 2020. 15

[56] M. Cannici, M. Ciccone, A. Romanoni, and M. Matteucci, "Asynchronous convolutional networks for object detection in neuromorphic cameras," 2019. 18

[57] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 4489–4497, 2015. 18

[58] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *arXiv preprint arXiv:1506.04214*, 2015. 18

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. 18

[60] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. 19

[61] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015. 19

[62] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "Cifar10-dvs: an event-stream dataset for object classification," *Frontiers in neuroscience*, vol. 11, p. 309, 2017. 19

[63] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza, *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7243–7252, 2017. 19

[64] R. W. Baldwin, M. Almatrafi, J. R. Kaufman, V. Asari, and K. Hirakawa, "Inceptive event time-surfaces for object classification using neuromorphic cameras," in *International Conference on Image Analysis and Recognition*, pp. 395–403, Springer, 2019. 21, 22

[65] I. Alzugaray and M. Chli, "Asynchronous corner detection and tracking for event cameras in real time," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3177–3184, 2018. 21

[66] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in Neuroscience*, vol. 9, p. 437, 2015. 24, 29, 30

[67] A. Sironi, M. Brambilla, N. Bourdis, X. Lagorce, and R. Benosman, "Hats: Histograms of averaged time surfaces for robust event-based object classification," 2018. 24, 32, 33

[68] P. de Tournemire, D. Nitti, E. Perot, D. Migliore, and A. Sironi, "A large scale event-based detection dataset for automotive," 2020. 24, 32, 33

[69] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *2004 conference on computer vision and pattern recognition workshop*, pp. 178–178, IEEE, 2004. 29

[70] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006. 29

[71] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470–1484, 2014. 29, 32

[72] C. Posch, D. Matolin, and R. Wohlgenannt, "A qvga 143 db dynamic range frame-free pwm image sensor with lossless pixel-level video compression and time-domain cds," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2010. 32

[73] N. Messikommer, D. Gehrig, A. Loquercio, and D. Scaramuzza, "Event-based asynchronous sparse convolutional networks." `https://github.com/uzh-rpg/rpg_asynet`, 2020. Accessed June 2020. 34

[74] "Torchvision support for nms." `https://github.com/gdlg/pytorch_nms`, January 2020. Accessed June 2020. 35

[75] R. W. Baldwin, M. Almatrafi, J. R. Kaufman, V. Asari, and K. Hirakawa, "Inceptive event time-surfaces for object classification using neuromorphic cameras," *Image Analysis and Recognition*, p. 395–403, 2019. 46