



*ugr* | Universidad  
de **Granada**

TRABAJO FIN DE GRADO  
INGENIERÍA EN INFORMÁTICA

# Técnicas de detección de objetos y segmentación usando retinas artificiales

---

**Autor**

Antonio José Aroca Aguilar

**Directores**

Francisco Barranco Expósito



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

—  
Granada, julio de 2021



# Técnicas de detección de objetos y segmentación usando retinas artificiales

Antonio José Aroca Aguilar

**Palabras clave:** aprendizaje automático (machine learning - ML), visión por computador (computer vision - CV), red neuronal, red neuronal convolucional (convolutional neural network - CNN), red neuronal convolucional dispersa(sparse convolutional neural network - SparseCNN), cámara basada en eventos, sensor de visión dinámica, ROS

## Resumen

En este proyecto de fin de grado se presenta el desarrollo de un sistema de reconocimiento de objetos a partir de la información de los eventos que nos proporcionan los sensores de visión dinámica, que son sensores activos que únicamente perciben cambios en la intensidad. Estos dispositivos ofrecen información que no es totalmente densa. De ahí que utilicemos redes neuronales dispersas (Sparse CNN) para tratar estos datos y mejorar el rendimiento.

También exponemos un análisis de los resultados obtenidos y una comparación con uno de los sistemas tradicionales de reconocimiento de objetos más utilizados en su versión más reciente, YOLOv5. Ante el elevado coste que suponen las cámaras basadas en eventos, utilizamos uno de los últimos y mejores simuladores software de código libre que se pueden encontrar en la bibliografía, v2e.

El proyecto será liberado tras la finalización del presente trabajo contribuyendo a la comunidad de software libre. Ante la novedad de estos sensores y el escaso código completamente abierto que se puede encontrar en la web al respecto, esperamos que el trabajo aquí desarrollado sea de utilidad para estudiantes, investigadores y desarrolladores interesados en la materia.



# Técnicas de detección de objetos y segmentación usando retinas artificiales

Antonio José Aroca Aguilar

**Keywords:** machine learning(ML), computer vision(CV), neural network, convolutional neural network(CNN), sparse convolutional neural network(SparseCNN), event based cameras, dynamic vision sensor, ROS

## Abstract

This bachelor thesis presents the development of an object recognition system based on the events information provided by the dynamic vision sensors, which are active sensors that only perceive changes in intensity. These devices offer information that is not totally dense. Hence, we use sparse neural networks (Sparse CNN) to process this data and improve performance.

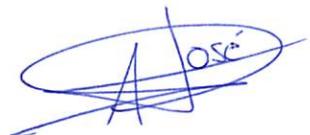
We also presents an analysis of the results obtained and a comparison with one of the most used traditional object recognition systems in its most recent version, YOLOv5. Due to the high cost of event-based cameras, we use one of the latest and best open source software simulators that can be found in the literature, v2e.

The project will be released after its finalization contributing to the open source community. Due to the novelty of these sensors and the scarce of completely open code that can be found on web in this field, it is expected the work developed here to be useful for students, researchers and developers interested in the matter.



---

Yo, **Antonio José Aroca Aguilar**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 26968797D, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.



Fdo: Antonio José Aroca Aguilar

Granada a 01 de septiembre de 2021 .





### Declaración de Originalidad del TFG

(Este documento debe adjuntarse cuando el TFG sea depositado para su evaluación)

D./Dña. Antonio José Aroca Aguilar, con DNI  
(NIE o pasaporte) 26968797D, declaro que el presente Trabajo de Fin de Grado es original, no habiéndose utilizado fuente sin ser citadas debidamente. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la [Normativa de Evaluación y de Calificación de los estudiantes de la Universidad de Granada](#) de 20 de mayo de 2013, esto *conllevará automáticamente la calificación numérica de cero [...] independientemente del resto de las calificaciones que el estudiante hubiera obtenido. Esta consecuencia debe entenderse sin perjuicio de las responsabilidades disciplinarias en las que pudieran incurrir los estudiantes que plagie.*

Para que conste así lo firmo el 01 de septiembre de 2021 (FECHA)

Firma del alumno



---

**D. Francisco Barranco Expósito**, Profesor del Área de Arquitectura y Tecnología de Computadores del Departamento Arquitectura y Tecnología de Computadores de la Universidad de Granada.

**Informa:**

Que el presente trabajo, titulado *Técnicas de detección de objetos y segmentación usando retinas artificiales*, ha sido realizado bajo su supervisión por **Antonio José Aroca Aguilar**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 01 de septiembre de 2021.

**El director:**

Firmado por BARRANCO EXPOSITO  
FRANCISCO - 75145818B el día  
02/09/2021 con un certificado  
emitido por AC FNMT Usuarios

**Francisco Barranco Expósito**



# Agradecimientos

Quisiera dar las gracias en primer lugar a mi tutor del proyecto, Francisco, por confiar en mi para su realización, y ayudarme en todo momento. Cabe destacar su plena disposición para cualquier duda durante todo el desarrollo, así como su cercanía e interés por los avances conseguidos en todo momento.

También me gustaría dar las gracias a José Antonio Díaz, mi amigo y compañero de clase, por ser mi persona de confianza, con la que me he podido desahogar en los momentos de agobio y con la que he podido celebrar las alegrías por las metas conseguidas. Ofreciendo siempre su ayuda, escuchando mis problemas y dando buenos consejos en todo momento.

Finalmente, pero no menos importante por ello, dar las gracias a toda mi familia. Mis padres, por trabajar sin descanso tanto en casa como fuera de ella para darme la oportunidad de estudiar lo que más me gusta, y dedicarme en cuerpo y alma a ello en todo momento, sin tener que preocuparme de nada más. A mi hermana mayor, por ser hermana y amiga, por ofrecerme su apoyo y experiencia. En definitiva, gracias a toda mi familia por estar siempre ahí apoyándome, sin poner malas caras, haciéndome reír en los momentos más difíciles y ofreciendo toda la ayuda posible a pesar de no ser entendidos en la materia.



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Estado del arte . . . . .	2
1.1.1. Cámaras basadas en eventos . . . . .	2
1.1.2. V2E . . . . .	8
1.1.3. Aprendizaje automático . . . . .	10
1.1.4. Redes Convolucionales Dispersas: SparseConvNet . . .	14
1.1.5. Visión por computador . . . . .	16
1.2. Motivación . . . . .	18
1.3. Objetivos . . . . .	19
<b>2. Gestión del proyecto</b>	<b>21</b>
2.1. Metodología de desarrollo . . . . .	21
2.2. Planificación . . . . .	22
2.3. Hitos . . . . .	25
2.4. Presupuesto . . . . .	26
2.4.1. Recursos Hardware . . . . .	27
2.4.2. Recursos Software . . . . .	28
2.4.3. Recursos Humanos . . . . .	28
2.4.4. Coste total . . . . .	28
<b>3. Entorno Operacional</b>	<b>29</b>
3.1. Ubuntu 18.04 . . . . .	29
3.2. Anaconda . . . . .	29
3.3. ROS . . . . .	30
3.3.1. Esquema de funcionamiento . . . . .	31
3.3.2. Herramientas . . . . .	32
3.4. PC y Servidor . . . . .	32
<b>4. Diseño</b>	<b>35</b>
4.1. Casos de uso . . . . .	35
4.2. Diagrama de flujo . . . . .	46

<b>5. Implementación</b>	<b>49</b>
5.1. Simulador de eventos: v2e . . . . .	49
5.2. ROS: dvs_ros . . . . .	51
5.3. Reconocimiento de objetos a partir de eventos: asynet . . . . .	53
5.3.1. Inferencia sobre la base de datos N-Caltech101 . . . . .	57
5.3.2. Inferencia sobre mis propios datos . . . . .	62
5.4. Reconocimiento de objetos a partir de imágenes: YOLOv5 . . . . .	67
<b>6. Problemas durante el desarrollo</b>	<b>71</b>
6.1. Restricciones sanitarias . . . . .	71
6.2. Restricciones hardware . . . . .	71
6.3. Restricciones software . . . . .	72
6.4. Problemas técnicos . . . . .	72
6.4.1. Virtualización . . . . .	72
6.4.2. Simulador con limitado número de resoluciones . . . . .	73
6.4.3. La base de datos NCars parseada no estaba disponible en el repositorio . . . . .	73
6.4.4. La lectura de eventos para la detección de objetos es mediante un archivo aedadat . . . . .	74
6.4.5. Mezcla de eventos no relacionados entre sí . . . . .	75
6.4.6. Requerimiento de un elevado espacio libre en disco . . . . .	75
6.5. Otros problemas . . . . .	76
6.6. Resumen de problemas y soluciones abordadas . . . . .	77
<b>7. Experimentación</b>	<b>79</b>
7.1. Métricas utilizadas . . . . .	79
7.2. Pruebas del simulador v2e . . . . .	80
7.3. Pruebas de segmentación implícita . . . . .	82
7.4. Pruebas de detección de objetos basada en eventos: asynet . . . . .	83
7.4.1. Selección de la base de datos para el entrenamiento . . . . .	83
7.4.2. Entrenamiento del modelo . . . . .	88
7.4.3. Inferencia sobre el conjunto test de N-Caltech101 para reconocimiento de objetos . . . . .	91
7.4.4. Pruebas variando los valores de threshold del detector e IoU en la supresión de no máximos . . . . .	94
7.4.5. Inferencia sobre datos obtenidos mediante el simulador v2e . . . . .	95
7.5. Pruebas de detección de objetos basada en imágenes: YOLOv5 . . . . .	100
7.6. Comparación entre asynet y yolov5 . . . . .	102
<b>8. Manual de usuario</b>	<b>105</b>
8.1. Guía de instalación . . . . .	105
8.1.1. rpg_dvs_ros . . . . .	105
8.1.2. v2e . . . . .	106

8.1.3. rpg_asynet . . . . .	107
8.1.4. YOLOv5 . . . . .	108
8.2. Manual de uso . . . . .	108
8.2.1. Obtención de nuevos datos . . . . .	108
8.2.2. Entrenamiento del modelo basado en eventos e inferencia . . . . .	109
8.2.3. Entrenamiento del modelo basado en imágenes e inferencia . . . . .	111
8.3. Enlaces de interés . . . . .	112
8.3.1. Enlaces de descarga del proyecto . . . . .	112
8.3.2. Enlaces de descarga de los distintos subsistemas utilizados . . . . .	112
<b>9. Conclusiones y futuras mejoras</b>	<b>113</b>
9.1. Conclusiones . . . . .	113
9.2. Futuras mejoras . . . . .	115
<b>Bibliografía</b>	<b>125</b>



# Índice de figuras

1.1. Ejemplos de usos de la visión por computador en la sociedad y la industria. Fuentes: [43], [44] y [45] . . . . .	1
1.2. Principio de funcionamiento de una cámara de eventos. Cuando el cambio en la intensidad luminosa del campo de visión de un píxel determinado supera cierto umbral emite un evento. Este evento puede ser positivo(ON) si la intensidad aumenta, o negativo(OFF) en caso contrario. Fuente: [129] . . . . .	3
1.3. Imagen obtenida por una cámara de eventos. Fuente: [127] . . . . .	4
1.4. resolución temporal y latencia de las cámaras tradicionales vs resolución temporal y latencia de las cámaras basadas en eventos. Fuente: [101] . . . . .	5
1.5. Diferencia en el rango dinámico entre una cámara basada en fotogramas y un sensor de eventos. Fuente: [100] . . . . .	6
1.6. Ejemplo de los 25 primeros eventos devueltos por el simulador v2e de un vídeo de prueba en formato txt. . . . .	8
1.7. Proceso de conversión de vídeo con fotogramas a eventos por el simulador v2e. Fuente: [118] . . . . .	9
1.8. Comparación de los eventos generador por una cámara DVS y los obtenidos por el simulador v2e a partir de un vídeo de fotogramas. Fuente: [118] . . . . .	10
1.9. Estructura básica de una red neuronal artificial. Fuente: [51] . . . . .	13
1.10. Estructura básica de una red neuronal artificial convolucional para clasificación. Fuente: [52] . . . . .	14
1.11. CNN vs SparseCNN. Fuentes: [27] y [130] . . . . .	15
1.12. La visión por computador se sirve de los algoritmos de machine learning para sintetizar e interpretar la información obtenida a partir de imágenes. Fuente: [85] y [86] . . . . .	16
1.13. Comparación entre las principales aplicaciones de la visión por computador. Fuente: [34], [33] y [32] . . . . .	18
2.1. Esquema de funcionamiento de la metodología incremental. Fuente: [60] . . . . .	21

2.2. Distintas páginas creadas en la hoja de cálculo de Google Docs utilizada para la planificación del proyecto. . . . .	22
2.3. Ejemplo de cómo se ha llevado a cabo la planificación y organización del proyecto. . . . .	23
2.4. Diagrama de Gantt del proyecto. . . . .	23
2.5. Diagrama de Gantt simplificado del proyecto. . . . .	24
3.1. Esquema de comunicación entre nodos ROS. Fuente: [67] . . .	32
4.1. Representación de los distintos elementos que intervienen en un diagrama de casos de uso. Fuente: [47] . . . . .	35
4.2. Diagrama UML de casos de uso. . . . .	36
4.3. Diagrama de flujo que representa el proceso a seguir por el usuario para utilizar el sistema correctamente. . . . .	47
5.1. Entradas, salidas y relación entre los distintos sistemas. . . . .	49
5.2. Líneas comentadas en el archivo v2ecore/emulator.py para permitir una salida con cualquier resolución. . . . .	50
5.3. Archivos que nos produce como salida el simulador v2e al haber eliminado el formato aedat. . . . .	51
5.4. Función principal del script utilizado para conseguir un archivo bag con todos los eventos generados en el simulador tras haber realizado algunas modificaciones a la versión disponible en [78]. . . . .	52
5.5. Script de ubuntu implementado para facilitar la generación completa de eventos y del archivo bag. . . . .	53
5.6. Líneas a cambiar en el archivo training/trainer.py. . . . .	55
5.7. Ejemplo que muestra los eventos obtenidos durante la formación de la base de datos N-Caltech101 a partir de las imágenes de Caltech101. Fuente: [131]. . . . .	56
5.8. Ejemplo que muestra una imagen formada a partir de acumulación de eventos de la base de datos Automotive y los bounding boxes creados por los autores. Fuente: [11]. . . . .	57
5.9. Atributos añadidos en el archivo testing/tester.py para realizar la inferencia. . . . .	58
5.10. Creación del dataset y el dataloader para el conjunto test en el archivo testing/tester.py. . . . .	58
5.11. Función test en el archivo testing/tester.py. . . . .	59
5.12. Función para cargar el último modelo que se haya guardado en el proceso de entrenamiento. . . . .	60
5.13. Función test en el archivo testing/tester_sin_estadisticas.py. .	61
5.14. Diferencia en collate_events en dataloader/loader.py y dataloader/loader_sin_estadisticas.py. . . . .	61
5.15. Función createDataset() del archivo dataloader/dataset_mio.py.	62

5.16. Función loadEventsFile() del archivo <code>dataloader/dataset_mio.py</code> que nos permite leer los eventos a partir de un archivo de texto.	63
5.17. Función <code>aniadirFiles</code> en el archivo <code>dataloader/dataset_mio.vio.py</code> .	65
5.18. Constructor de la clase <code>Loader</code> en el que no se realiza un barajado del dataset.	66
5.19. Script <code>crearVideo.py</code> basado en el código obtenido en [25].	67
5.20. Archivo de configuración <code>yaml</code> para entrenar el modelo YOLOv5 con la base de datos Caltech101.	68
5.21. Script implementado para poner las anotaciones de las imágenes de la base de datos Caltech101 en el formato requerido por YOLOv5.	70
7.1. Captura del nodo <code>roslaunch</code> mostrando los eventos que nos ha producido <code>v2e</code> con distintas resoluciones de salida.	80
7.2. Gráficas con la comparación de distintas características de las salidas producidas por <code>v2e</code> en función de la resolución.	82
7.3. Capturas de pantalla del vídeo a partir del cual se han sacado los eventos y se ha realizado la segmentación implícita.	83
7.4. Muestra del overfitting sufrido por un clasificador binario. El objetivo es separar las muestras rojas de las azules y todas ellas representan el subconjunto <code>train</code> . Mientras que la línea negra representa la función de predicción ideal, la curva verde clara representaría la función aprendida con el sobreajuste. Fuente: [56]	85
7.5. Gráficas con las métricas obtenidas por el modelo durante el entrenamiento y la validación de las primeras 100 épocas con las distintas bases de datos para clasificación de imágenes.	86
7.6. Gráficas con las métricas obtenidas por el modelo durante el entrenamiento y la validación de las primeras 100 épocas con las distintas bases de datos para detección de objetos.	87
7.7. Gráficas con las métricas obtenidas por el modelo en el entrenamiento y la validación durante 1000 etapas con la base de datos N-Caltech101 para clasificación.	89
7.8. Gráficas con las métricas obtenidas por el modelo en el entrenamiento y la validación durante 1000 etapas con la base de datos N-Caltech101 para reconocimiento de objetos.	90
7.9. Gráficas con las métricas obtenidas por el modelo en la validación durante 500 etapas más a raíz de las 1000 anteriores con la base de datos N-Caltech101 para reconocimiento de objetos, y otras 500 más a partir de las 1500 anteriores.	91
7.10. Ejemplos de predicciones realizadas sobre el subconjunto <code>test</code> con los mejores pesos obtenidos tras 2000 etapas de entrenamiento.	92

7.11. Más ejemplos de predicciones realizadas sobre el subconjunto test con los mejores pesos obtenidos tras 2000 etapas de entrenamiento . . . . .	93
7.12. Definición de IoU. El recuadro verde representa el bounding box real, mientras que el rojo es la predicción hecha por el modelo. Fuente: [17] . . . . .	95
7.13. Gráfica que relaciona la resolución del vídeo de salida del simulador v2e con el porcentaje de acierto en la inferencia por el modelo descrito. . . . .	96
7.14. Ejemplos de predicciones realizadas sobre nuestro propio conjunto de datos utilizando distintas resoluciones de salida en la obtención de los eventos a partir de vídeos con el simulador v2e. . . . .	97
7.15. Más ejemplos de predicciones realizadas sobre nuestro propio conjunto de datos utilizando distintas resoluciones de salida en la obtención de los eventos a partir de vídeos con el simulador v2e. . . . .	98
7.16. Ejemplos de predicciones realizadas sobre una escena grabada de lejos por nosotros mismos cuyos eventos hemos obtenido mediante el simulador v2e con una resolución de 720x1280 píxeles. El modelo asynet utilizado presenta los mejores pesos obtenidos tras 2000 épocas de entrenamiento, un threshold = 0.7 y un IoU = 0.5. . . . .	99
7.17. Funciones de pérdida y mAP del entrenamiento y la validación de 1000 épocas con YOLOv5 y la base de datos Caltech101.100	100
7.18. Ejemplos de predicciones realizadas sobre el subconjunto test por el modelo YOLOv5 con los mejores pesos obtenidos en el entrenamiento anterior. . . . .	101
7.19. Ejemplos de predicciones realizadas sobre imágenes sacadas de los vídeos grabados por mi mismo, en los que se muestran objetos cotidianos, por el modelo YOLOv5 con los mejores pesos obtenidos en el entrenamiento anterior. . . . .	101
7.20. Gráfica que compara la eficacia del modelo YOLOv5 basado en imágenes con el modelo Asynet que trabaja con eventos. . . . .	102
7.21. Gráficas que comparan el número de capas y la cantidad de operaciones de coma flotante que realizan el modelo YOLOv5 basado en imágenes y el modelo Asynet que trabaja con eventos.103	103

# Índice de cuadros

1.1. tipos de cámaras de eventos. Fuente: [95] . . . . .	7
2.1. costes hardware del proyecto . . . . .	27
2.2. costes software del proyecto . . . . .	28
2.3. costes totales del proyecto . . . . .	28
4.1. Caso de Uso CU-001: Crear el entorno . . . . .	37
4.2. Caso de Uso CU-002: Lanzamiento nodo máster . . . . .	37
4.3. Caso de Uso CU-003: Lanzamiento lector de bag . . . . .	38
4.4. Caso de Uso CU-004: Lanzar renderer . . . . .	38
4.5. Caso de Uso CU-005: Entrenamiento de un modelo . . . . .	39
4.6. Caso de Uso CU-006: Reconocimiento de objetos a partir de eventos . . . . .	39
4.7. Caso de Uso CU-007: Detección de objetos a partir de eventos . . . . .	40
4.8. Caso de Uso CU-008: Carga de un modelo preentrenado . . . . .	40
4.9. Caso de Uso CU-009: Lectura de eventos a partir de archivo . . . . .	41
4.10. Caso de Uso CU-010: Creación de vídeo a partir de las imáge- nes generadas . . . . .	41
4.11. Caso de Uso CU-011: Generación de eventos . . . . .	42
4.12. Caso de Uso CU-012: Escritura eventos en archivo texto . . . . .	42
4.13. Caso de Uso CU-013: Generación de vídeo con eventos gene- rados . . . . .	43
4.14. Caso de Uso CU-014: Generación del archivo bag . . . . .	43
4.15. Caso de Uso CU-015: Entrenamiento de un modelo . . . . .	44
4.16. Caso de Uso CU-016: Reconocimiento de objetos a partir de fotogramas . . . . .	44
4.17. Caso de Uso CU-017: Carga de un modelo preentrenado . . . . .	45
4.18. Caso de Uso CU-018: Detección de objetos a partir de foto- gramas . . . . .	45
4.19. Caso de Uso CU-019: Carga de imágenes . . . . .	46
6.1. Principales problemas encontrados durante el desarrollo del proyecto y acciones realizadas para solucionarlos . . . . .	77

7.1. Comparación de distintas características de las salidas producidas por v2e en función de la resolución. . . . .	81
7.2. Bondad de las predicciones en función del valor del threshold e IoU. . . . .	95
7.3. Número y porcentaje de acierto al inferir sobre nuestros propios datos en función de la resolución utilizada para obtener los eventos. . . . .	96

# Capítulo 1

## Introducción

Desde hace muchos años la visión por computador, y particularmente la detección y reconocimiento de objetos, ha adquirido una gran importancia en una sociedad cada vez más robotizada y automatizada como es la nuestra, con aplicaciones como la conducción autónoma o la videovigilancia entre otras muchas. Las técnicas pertenecientes a esta disciplina han evolucionado rápidamente obteniendo unos resultados más que aceptables, de hecho, estos resultados se han visto limitados en muchos casos por el hardware existente, las cámaras fotográficas y de vídeo.

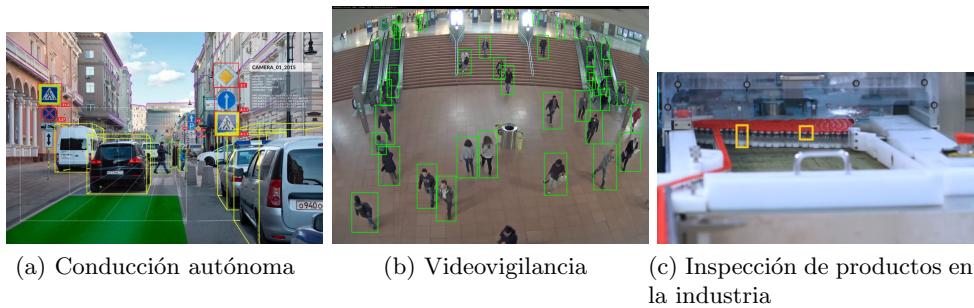


Figura 1.1: Ejemplos de usos de la visión por computador en la sociedad y la industria. Fuentes: [43], [44] y [45]

Sin embargo, desde hace algunos años se vienen desarrollando un nuevo tipo de cámaras, llamadas sensores de visión dinámica. Estos dispositivos no obtienen fotogramas con regularidad, sino que producen eventos de forma asíncrona en función de los cambios en la intensidad luminosa de las escenas. Las cámaras basadas en eventos ofrecen una serie de propiedades muy ventajosas respecto a las cámaras tradicionales basadas en fotografías, entre las que destacan una mayor resolución temporal, menor latencia,

menor desenfoque, etc. Estas y otras ventajas se explicarán en posteriores secciones(1.1.1)

Esto hace que hoy en día haya surgido un nuevo campo de investigación consistente en la aplicación y rediseño de los algoritmos de visión por computador tradicionales para adaptarlos a los eventos proporcionados por estos nuevos sensores. Todo esto motivado porque los métodos clásicos basados en fotogramas síncronos no nos sirven y no permiten sacar partido de las ventajas antes mencionadas.

Cada vez están tomando más importancia, y de hecho, cada vez hay más empresas dedicadas a la fabricación y comercialización de los sensores de visión dinámica, entre las que se encuentran Inivation([106]), Prophesse([107]), e incluso Samsung([108]). Además, cada vez más desarrolladores y grupos de investigación están involucrándose en la materia, como el department of NeuroInformatics of the University of Zurich and ETH Zurich. Se cree que estas cámaras basadas en eventos pueden marcar un antes y un después en el desarrollo de robots autónomos debido a sus propiedades únicas que suponen una serie de ventajas, que se verán en apartados posteriores(1.1.1), respecto a las cámaras tradicionales, pero para llegar a esto, primero tenemos que solucionar el problema de la adaptación de las técnicas de visión por computador a la información proporcionada por estos sensores.

Por esto en este proyecto de fin de grado se persigue la implementación y el estudio de un software para la detección y reconocimiento de objetos utilizando la información espacio temporal de los eventos proporcionados por estas cámaras. Además, aprovechamos las características intrínsecas de este tipo de sensores para realizar una segmentación de objetos implícita. Si tanto la escena, como la cámara permanecen estáticas, y sólo se mueven los cuerpos que haya en dicha escena, estaremos realizando una segmentación de objetos de forma “natural” debido a las propiedades de estos sensores, tan solo tendríamos que cerrar las fronteras mostradas por la salida de la cámara de eventos.

## **1.1. Estado del arte**

### **1.1.1. Cámaras basadas en eventos**

Las cámaras basadas en eventos o neuromórficas ([124]), también conocidas como sensores de visión dinámica son unos dispositivos que aportan información visual a partir de los cambios en la intensidad luminosa de la escena. Esto es, que a diferencia de las cámaras tradicionales que proporcionan una imagen completa de la escena, estos sensores tratan cada píxel de forma independiente, y dispararán un evento en aquellos en los que se

detecte un cambio de brillo mayor a cierto umbral determinado. El resto de píxeles en los que no se detecte dicho cambio simplemente permanecerán inactivos.

Por lo tanto, podemos decir que cada píxel actúa de forma autónoma, independiente y asíncrona. Además solo emitirán un evento cuando se produzca un cambio de luminancia superior a cierto umbral. La Real Academia Española define la luminancia ([\[8\]](#)) como la magnitud que expresa el flujo luminoso en una dirección determinada. Normalmente este umbral específico que se debe superar para que se produzca un evento es de un 15 % en contraste tal y como se afirma en [\[29\]](#).

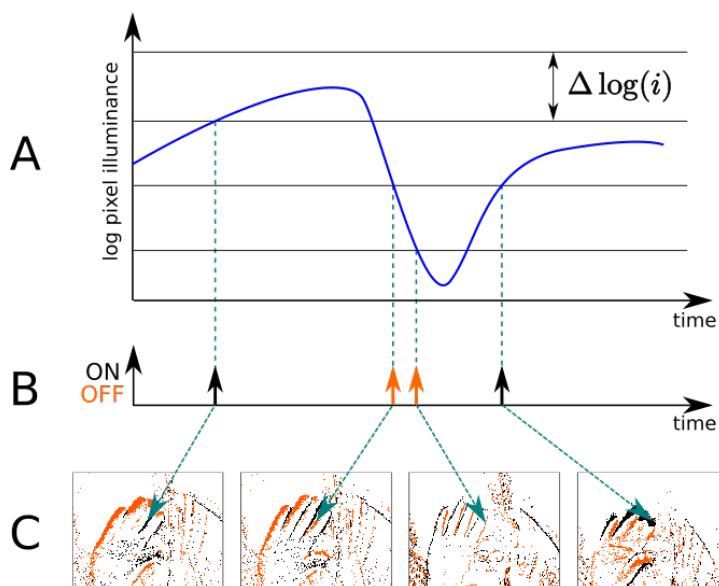


Figura 1.2: Principio de funcionamiento de una cámara de eventos. Cuando el cambio en la intensidad luminosa del campo de visión de un píxel determinado supera cierto umbral emite un evento. Este evento puede ser positivo(ON) si la intensidad aumenta, o negativo(OFF) en caso contrario. Fuente: [\[29\]](#)

Cuando un evento es disparado por un píxel, este nos proporciona información de tres tipos distintos:

- **Posición:** se refiere a la coordenada 'x' e 'y' del píxel que ha detectado el cambio.
- **Marca de tiempo(timestamp):** el tiempo que ha pasado desde que se empezó a capturar la imagen hasta que se ha detectado el cambio que ha provocado el evento.

- **Polaridad:** nos indica si ha habido un aumento del brillo, o por contra, una disminución de este en la escena.

El valor de la polaridad es un número entero comprendido en el conjunto  $\{0,1\}$  o  $\{-1,1\}$  dependiendo del fabricante. Esto quiere decir, que solamente nos está informando del sentido que ha tenido el cambio de brillo, pero no de la cantidad en que ha variado. Solamente sabemos que ha sido mayor al umbral determinado por el sensor.

Por tanto, tal como se describe en [129], podemos expresar el evento  $i$ -ésimo dentro del flujo de eventos de salida que nos proporciona la cámara como la tripleta  $evento_i = (x_i y_i, t_i, p_i)^T$  donde  $x_i y_i$  es la posición espacial del evento en el plano de la imagen,  $t_i$  es la marca de tiempo en la que se ha producido, y  $p_i$  es su polaridad.

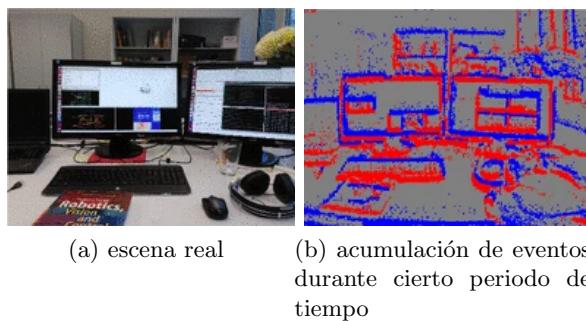


Figura 1.3: Imagen obtenida por una cámara de eventos. Fuente: [127]

En la imagen [1.3b] lo que estamos viendo realmente son los eventos producidos durante un periodo corto de tiempo colapsados y representados en una única imagen. Esto es, en realidad, una forma artificial de mostrar la salida de las cámaras de eventos, pero nunca debemos usar este método para procesar y trabajar con eventos puesto que estaríamos trabajando con imágenes nuevamente.

Estos sensores ofrecen unas características que van a ser muy ventajosas respecto a las cámaras tradicionales en algunas circunstancias. Estos beneficios son debidos principalmente a que mientras que los eventos proporcionados por las cámaras neuromórficas son asíncronos y dispersos espacialmente, los fotogramas ofrecidos por las cámaras tradicionales son síncronos y densos. Las principales ventajas obtenidas por estos sensores son las siguientes:

1. **Alta resolución temporal:** las cámaras tradicionales tienen una velocidad de fotograma que limita la frecuencia con la que se va a realizar una imagen, de esta forma sufren un retardo de entre 33ms y 16 ms en

función de si trabajan a 30fps o 60fps. Además estás cámaras deben esperar cierto tiempo hasta haber recogido suficiente información, lo que va a provocar que, si el objetivo que estamos filmando se mueve a gran velocidad en este tiempo de almacenamiento de fotones, se producirá un desenfoque. En los sensores de visión dinámica, por contra, debido a su naturaleza asíncrona no será necesario un tiempo de acumulación de fotones. Además podemos detectar eventos con una marca de tiempo del orden de microsegundos, por lo que no sufriremos retrasos similares a los de las cámaras tradicionales, ni el desenfoque, normalmente llamado motion blur por su traducción inglesa, comentado al capturar objetivos en rápido movimiento.

2. **Baja latencia:** debido a la asincronicidad de los píxeles de los sensores de visión dinámica, tan pronto como se detecte un cambio en la intensidad luminosa en un único píxel, este transmitirá un evento. Este comportamiento está en contraposición con las cámaras fotográficas tradicionales en las que es necesario esperar un tiempo de exposición global, como ya se ha dicho anteriormente. Las cámaras de eventos tienen una latencia cercana a los 10 microsegundos.

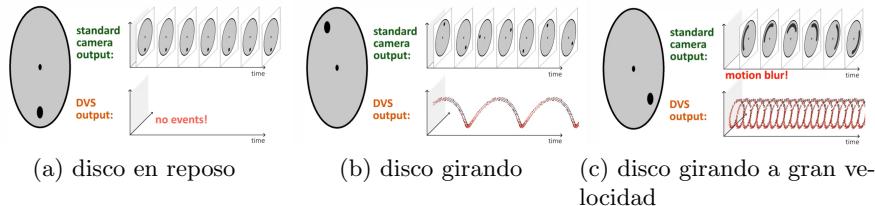


Figura 1.4: resolución temporal y latencia de las cámaras tradicionales vs resolución temporal y latencia de las cámaras basadas en eventos. Fuente: [101]

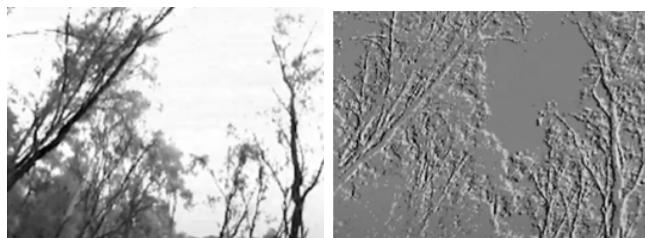
3. **Baja energía consumida:** aunque existen algunos modelos que consumen menos, lo normal es que una cámara basada en eventos utilice unos 10mW. Este consumo tan reducido de energía es debido principalmente a que sólo transmiten los cambios de brillo en la escena, eliminando todos los datos.
4. **Alto rango dinámico:** en estos sensores el receptor de fotones de cada píxel trabaja de forma independiente, y además lo hace en escala logarítmica, como se ve en la imagen 1.2. Esto les permite capturar información con rangos muy distintos de luz, desde la oscuridad de la noche hasta la luz del medio día. Al detectar simplemente cambios, obviando el valor que posee, somos capaces de adaptarnos a estas condiciones tan variantes. Mientras que las cámaras tradicionales de alta calidad tienen alrededor de 60dB como máximo, una cámara

neuromórfica supera los 120dB con facilidad. Esto es crítico en aplicaciones como la conducción autónoma, pensemos por ejemplo en los cambios de intensidad luminosa producidos en las entradas y salidas de los túneles. Podemos ver éste y otro ejemplo en la figura 1.5.

En las distintas imágenes que aparecen en la ilustración 1.5 lo importante es ver la diferencia que hay entre los elementos apreciables en las zonas claras, de alta intensidad luminosa, entre la imagen normal y la acumulación de eventos. En la imagen 1.5a vemos como el gran contraste producido entre el interior y el exterior del túnel nos impide observar los objetos que hay en la zona de alta luminosidad. En cambio, en la imagen 1.5b, vemos como sí se han producido suficientes eventos como para apreciar los distintos elementos que hay. Algo similar ocurre en las ilustraciones 1.5c y 1.5d, solo que ahora son los detalles de los árboles lo que no podemos ver bien debido a la claridad del cielo. En la acumulación de eventos podemos observar muchos más detalles y ramas que no se aprecian en la otra figura.



(a) imagen de la salida de un tunel obtenida con la cámara de un móvil  
(b) acumulación de eventos de la salida de un tunel



(c) imagen del cielo en un bosque obtenida con la cámara DAVIS  
(d) acumulación de eventos del cielo en un bosque

Figura 1.5: Diferencia en el rango dinámico entre una cámara basada en fotogramas y un sensor de eventos. Fuente: [100]

En resumen, las cámaras basadas en eventos son unos sensores que nos proporcionan un flujo asíncrono de eventos basados en los cambios en el brillo de la escena. Estas cámaras, aunque con precios altos, poseen una

serie de propiedades que las hacen de especial interés.

### Tipos de cámaras de eventos

Nombre	Evento generado	IMU	Fabricante	¿Se comercializa?
DVS 128	polaridad	No	Inivation	No
sDVS 128	polaridad	No	CSIC	No
DAVIS 240	polaridad	Sí	Inivation	Sí
DAVIS 346	polaridad	Sí	Inivation	Sí
SEES	polaridad	Sí	Insightness	Sí
Samsung DVS	polaridad	Sí	Samsung	No
Onboard	polaridad	Sí	Prophesee	Sí
Celex	intensidad	Sí	CelePixel	Sí

Cuadro 1.1: tipos de cámaras de eventos. Fuente: [95]

En la tabla 1.1 cuando hablamos de IMU nos referimos a un sensor de unidad inercial. Esto no es más que un pequeño dispositivo que nos informa acerca de la fuerza que sufre un cuerpo, su velocidad angular y orientación.

### Simuladores de cámaras de eventos

Los sensores de visión dinámica son dispositivos muy caros, cuestan varios miles de euros. Esto hace que no sean accesibles para todo el mundo, y que sea necesario otro método de obtención de eventos similares a los que proporcionaría una cámara de este tipo, sin tener que comprar dicho sensor, para que todo el mundo pueda realizar trabajos de investigación libremente.

En este sentido surgen los simuladores, que no son más que aplicaciones software que reciben un vídeo normal como entrada, y nos proporcionan como salida los eventos que se hubieran producido de haber sido filmada la escena con una cámara de eventos en lugar de con una normal.

Estos simuladores, a parte de la posibilidad, ya comentada, que nos brindan a todos de realizar trabajos de investigación libremente con estos eventos y este tipo de información, también presenta mucho interés la capacidad que nos dan para traducir o transformar las imágenes que se usan en los principales conjuntos de datos para cámaras convencionales, como pueden ser CIFAR-10([113]), MNIST([116]), ImageNet([115]), COCO([114]), etc. Lo cual nos permitiría comparar las prestaciones entre estas cámaras tradicionales basadas en fotogramas y los sensores de eventos.

Además, la mayoría de estos simuladores no se limitan únicamente a

devolver un archivo de texto con los eventos, sino que también proporcionan esta información en archivo binario, o nos pueden proporcionar también un vídeo en el que se muestre la escena compuesta por los distintos eventos que van apareciendo, etc.

```
#!events.txt
# This is a text DVS created by v2e (see https://github.com/SensorsINI/v2e)
# Format is time (float s), x, y, polarity (0=off, 1=on) as specified at http://rpg.ifl.uzh.ch/davis_data.html
# Creation time: 01:02PM March 29 2021
# Creation time: System.currentTimeMillis() 1617015722056
# User name: antonioaroca
# 0.0013915732270106673 27 146 1
# 0.0013915732270106673 27 146 1/0 1
# 0.0013915732270106673 202 204 1
# 0.0013915732270106673 71 73 1
# 0.0013915732270106673 313 52 1
# 0.0013915732270106673 345 200 1
# 0.0013915732270106673 305 46 1
# 0.0013915732270106673 320 57 1
# 0.0013915732270106673 313 208 1
# 0.0013915732270106673 302 47 1
# 0.0013915732270106673 317 169 1
# 0.0013915732270106673 261 10 1
# 0.0013915732270106673 284 32 1
# 0.0013915732270106673 264 219 1
# 0.0013915732270106673 315 221 1
# 0.0013915732270106673 316 168 1
# 0.0013915732270106673 316 168 1
# 0.0013915732270106673 199 99 1
# 0.0013915732270106673 267 8 1
# 0.0013915732270106673 303 47 1
# 0.0013915732270106673 345 194 1
# 0.0013915732270106673 294 33 1
# 0.0013915732270106673 102 179 1
# 0.0013915732270106673 261 38 1
```

Figura 1.6: Ejemplo de los 25 primeros eventos devueltos por el simulador v2e de un vídeo de prueba en formato txt.

Sin embargo, tienen un gran inconveniente, y es que existe una pérdida de eventos inherente por estar trabajando sobre una representación de la realidad, con todos los problemas que ya se han comentado que presentan las cámaras convencionales, en lugar de trabajar directamente sobre la escena.

Existen varios simuladores de eventos, entre los que destacan davis\_simulator([89]), ESIM([90]) que es una mejora del anterior, V2E([28]), y Microsoft AirSim([91]).

Uno de los más conocidos y utilizados es ESIM([34]). Este simulador nos permite publicar los eventos en ROS o guardar los datos en archivos rosbag. Además nos ofrece una serie de propiedades muy deseables como son la simulación del desenfoque producido por el movimiento, soporte para distorsión de la cámara, la simulación de la IMU, y muchas más. Sin embargo, nosotros en este trabajo utilizaremos v2e([36]) debido a su simplicidad y otros motivos que serán explicados más detenidamente en capítulos posteriores, 5.

### 1.1.2. V2E

V2E ([36]) es un simulador de eventos desarrollado por el department of NeuroInformatics of the University of Zurich and ETH Zurich ([117]). Es una herramienta software desarrollada sobre la plataforma python, que permite obtener la información generada por una cámara de eventos a partir de un vídeo grabado con una cámara normal, que solamente es capaz de obtener fotogramas.

En la figura 1.7 podemos apreciar un esquema realizado por los propios desarrolladores del simulador, que nos muestra el proceso que se realiza para poder realizar la conversión. De este proceso hay varios pasos que merece la pena destacar. El primero es la transformación de los fotogramas de RGB a Luma. Este modo de imagen es similar a la escala de grises (monocolor, un único canal), pero lo que codifica realmente es la luminosidad o claridad que presenta la foto. Este paso parece lógico puesto que según se ha explicado con anterioridad en este mismo documento, la información de la que se nutren los sensores de visión dinámica es precisamente los cambios en el brillo de la escena. A continuación, aunque de manera opcional (se debe indicar mediante un parámetro en la ejecución), se realiza una interpolación de fotogramas. Esto resulta de gran utilidad. Uno de los grandes problemas que presentaban las cámaras basadas en fotogramas es que debían esperar un tiempo de exposición global y tenían una latencia mucho mayor a las de las cámaras de eventos. Al interpolar varios frames entre dos fotogramas consecutivos estamos obteniendo de cierta forma una aproximación de lo que realmente ocurrió entre esos instantes, estamos generando una posible versión de la información perdida debido al tiempo de exposición global. Esta interpolación se realiza mediante SuperSloMo([\[88\]](#)), un framework que dados dos fotogramas tiene la capacidad de generar hasta 7 nuevos frames intermedios para formar secuencias de vídeo.

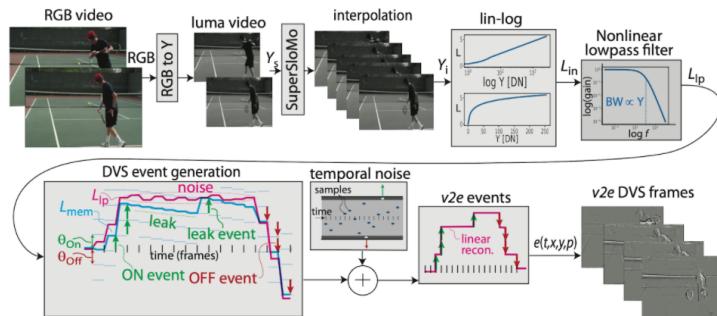


Figura 1.7: Proceso de conversión de vídeo con fotogramas a eventos por el simulador v2e. Fuente: [\[118\]](#)

Tal y como describen los autores en su web [\[118\]](#), este simulador v2e es capaz de modelar varios efectos de los píxeles DVS reales:

- Ancho de banda finito y dependiente de la intensidad del fotorreceptor.
- Eventos provocados por el ruido de fondo.
- Ruido temporal dependiente de la intensidad.

Aunque con la interpolación y otras técnicas usadas hemos conseguido aumentar significativamente la calidad de la simulación, seguimos teniendo

una diferencia de eventos significativa con respecto a los sensores reales. Y a pesar de que v2e proporciona un modelo de píxeles DVS mucho más realista que otros simuladores, como ESIM, solo puede procesar películas. Además, la gran limitación que presenta v2e ocurre ante vídeos con un gran desenfoque.

Como ya dijimos, los simuladores tienen, normalmente, la capacidad de devolver la información resultante en varios formatos. V2e nos ofrece distintos archivos con los eventos: txt, hdf5 y aedat en su versión 2.0, formato personalizado y creado por Inivation para almacenar la información de los eventos. Además nos proporciona varios archivos .avi en los que podemos ver el vídeo original, el vídeo con la interpolación superSloMo y el vídeo realizado con eventos en lugar de con fotogramas. En la figura 1.6 podemos ver un pequeño segmento de los eventos producidos en un vídeo de prueba en formato txt.

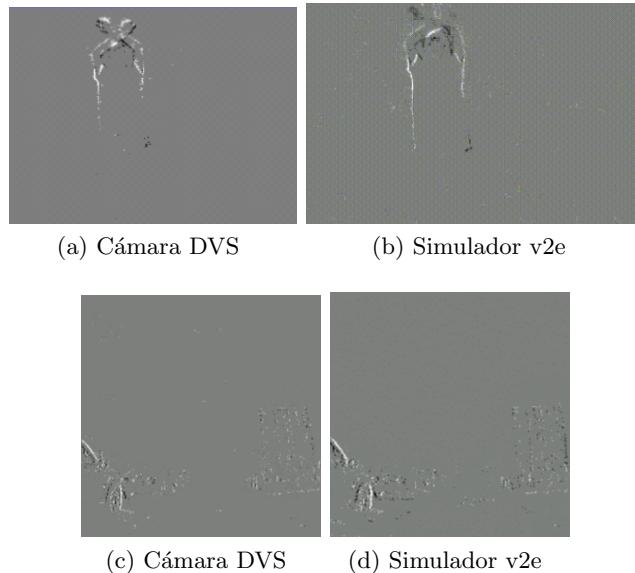


Figura 1.8: Comparación de los eventos generados por una cámara DVS y los obtenidos por el simulador v2e a partir de un vídeo de fotogramas. Fuente: [118]

### 1.1.3. Aprendizaje automático

El aprendizaje automático, o machine learning ([\[38\]](#)), se refiere al proceso por el cual un ordenador o una máquina es capaz de mejorar con la experiencia, aprender a partir de datos. Es una rama de la inteligencia artificial que crea modelos analíticos automáticamente a partir de la identificación de patrones, y tiene como objetivo final realizar predicciones en base a nuevos

conjuntos de datos. Lo que llamamos aprendizaje es realmente la actualización de un algoritmo en función de los datos. Se dice que este aprendizaje es automático porque es un proceso llevado a cabo de manera autónoma, sin la intervención de un programador humano.

Desde el comienzo de la informática, los científicos han perseguido alcanzar el día en que los ordenadores tengan la capacidad de aprender y ser más inteligentes que los propios seres humanos. Aunque todavía estamos muy lejos de este propósito, desde hace varios años sí se está consiguiendo dotar a las máquinas de cierta funcionalidad comparable con el razonamiento humano, y hoy en día el aprendizaje automático está más a la moda que nunca. Este auge del machine learning se ha visto potenciado por varios factores, como son que los ordenadores y servidores sean más potentes y económicos que nunca, pero sobre todo por la gran cantidad de datos que generan y están a disposición de las empresas.

Thomas Hayes Davenport Jr, autor estadounidense especializado en análisis y gestión del conocimiento e inteligencia artificial dice en un artículo para el periódico The Wall Street Journal ([\[3\]](#)):

*“Los humanos pueden crear, por lo general, uno o dos buenos modelos por semana; el machine learning puede crear miles de modelos por semana.”*

Esta afirmación, unida a las innumerables aplicaciones que estas técnicas presentan, y la necesidad de acertar en la toma de decisiones explican por qué es un campo que está a la orden del día y en el que se está invirtiendo tanto dinero ya sea en investigación, o en formación de personal cada vez más y mejor cualificado.

Dentro del aprendizaje automático tenemos una gran cantidad de algoritmos y técnicas diferentes, aunque todas ellas se pueden clasificar dentro de tres grandes grupos principalmente:

- **Aprendizaje supervisado:** los algoritmos son entrenados a partir de datos ya etiquetados correctamente, es decir, ya conocemos el resultado para estos datos que vamos a utilizar para que el modelo aprenda. El aprendizaje lo realiza comparando los resultados que va obteniendo con los reales que deberían dar para cambiar y corregir errores. Posteriormente se utilizará el modelo final que se ha conseguido para predecir la etiqueta en nuevos datos no etiquetados previamente.

En función de como sean estas etiquetas que debemos predecir, el problema puede ser de clasificación, cuando se debe de establecer una clase de entre un número finito de categorías; o de regresión, cuando el valor a predecir es un valor continuo.

- **Aprendizaje no supervisado:** en este caso los algoritmos no dispondrán de las etiquetas correspondientes a los datos utilizados para entrenar. El objetivo principal es que el sistema descubra patrones de información o estructuras en el interior de los datos. Estos métodos son los más parecidos al funcionamiento real de la psique humana.

A menudo, cuando se dispone de una gran cantidad de datos y solamente una pequeña proporción de ellos están etiquetados correctamente, se suelen utilizar algoritmos de aprendizaje mixtos, que reciben como entrada para el entrenamiento tanto datos etiquetados como no clasificados.

- **Aprendizaje por refuerzo:** son métodos que aprenden mediante castigos y recompensas, es decir, ensayo y error. Estos algoritmos aprenden que acciones deben realizar para alcanzar cierto objetivo beneficioso en función de cuál sea el estado actual en el que se encuentre el entorno. Realizar este proceso correctamente requiere un gran intercambio de información entre el mundo, o entorno, y el propio algoritmo o sistema.

## Redes Neuronales Convolucionales

Dentro de estas tres categorías que hemos visto existen numerosos algoritmos entre los que debemos escoger en función del objetivo final que queramos alcanzar. Muchos de los métodos no son específicos para una única categoría, sino que en función de como los utilicemos, y del problema a resolver, estaremos dentro de un grupo u otro

Entre los algoritmos más famosos y utilizados encontramos las llamadas redes neuronales, y particularmente las redes neuronales convolucionales ([\[135\]](#)), CNN por su nombre en inglés. Estas últimas son las que más nos interesan por el trabajo que se expone en este documento, sin embargo, no podemos hablar sobre las CNN sin saber antes qué son y en qué consisten las redes neuronales tradicionales.

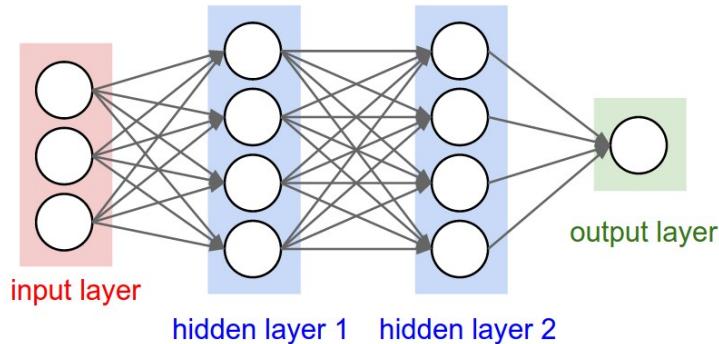


Figura 1.9: Estructura básica de una red neuronal artificial. Fuente:[\[51\]](#)

Una red neuronal ([\[119\]](#)) no es más que un conjunto de neuronas artificiales conectadas entre sí que comparten información entre ellas, siendo una neurona artificial una unidad de computo que intenta realizar el mismo proceso al seguido por las neuronas humanas. En función de la información de entrada que tenga la red, las distintas capas de neuronas obtendrán cierta información y extraerán patrones que les permitirán proporcionar unos resultados de salida.

En la ilustración [1.9](#) podemos observar la estructura general de las redes neuronales artificiales, en esta imagen vemos como los datos de entrada y de salida están conectados con una capa llamada oculta. Esta última capa es la que está formada por neuronas artificiales, representadas por los distintos círculos. Aunque en la imagen solo se aprecian dos únicas capas de neuronas, en la práctica tendremos muchas capas ocultas, cada una de ellas constituida por neuronas. Todas las neuronas artificiales pertenecientes a la misma capa realizaran la misma operación, ya sea extracción de características, simplificación de estas, etc, mientras que una capa sí puede realizar una operación distinta a otras capas, de hecho, esto será lo más corriente. Una de las características más importantes de las redes neuronales es que las neuronas pertenecientes a la misma capa nunca estarán relacionadas unas con otras, tal y como se aprecia en la imagen [1.9](#).

Una CNN no es más que un tipo de red neuronal artificial en la que las entradas son imágenes. Esto hace que se modifique levemente su estructura para hacerlas más eficientes y eficaces ante este tipo de datos de entrada. Uno de los cambios más significativos es la aparición de los tensores, es decir, a diferencia de las redes normales que trabajan con vectores de números, las CNN van a trabajar con unas estructuras de datos de 3 dimensiones. Esto es debido, entre otros motivos, para aprovechar la localidad de la información perteneciente a las distintas capas de las imágenes (RGB o RGBA).

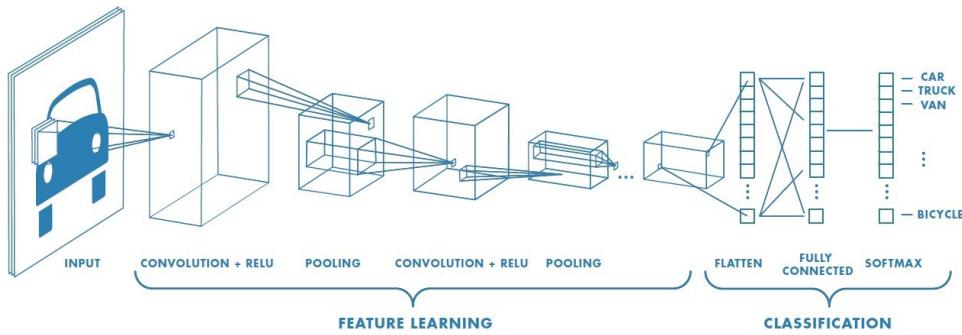


Figura 1.10: Estructura básica de una red neuronal artificial convolucional para clasificación. Fuente: [52]

En la figura 1.10 observamos precisamente como son los tensores. En esta imagen apreciamos también como hay una primera etapa de extracción de características utilizando estos tensores, y al llegar a un momento dado los aplana en un único vector y empieza una nueva etapa de clasificación en este caso, dependiendo del problema a resolver. Las redes neuronales convolucionales van a ir reduciendo la cantidad de información, para finalmente quedarnos con la más importante y la que nos va a ayudar a realizar nuestra tarea. Para ello las distintas capas de neuronas van a ir realizando operaciones diferentes.

Una de las principales características de estas redes es que están jerarquizadas, esto es, en las primeras capas se van a detectar patrones simples, como líneas rectas y curvas que pueden aparecer en la imagen, y conforme vamos avanzando en profundidad, irán extrayendo patrones cada vez más complejos.

Las CNN han ido evolucionando a lo largo de los años desde su aparición, teniendo cada vez un mayor número de capas. También han ido apareciendo capas con funcionalidades nuevas que se han demostrado útiles en mayor o menor medida en la extracción de características. Además, han surgido distintos tipos de CNN en función de cómo son las capas que presentan y la forma que tienen de tratar la información. En función del problema que estemos solucionando tendremos que escoger cuál es la mejor opción para nosotros, lo que sí está claro es que las redes neuronales convolucionales son una herramienta indispensable hoy en día a la hora de trabajar y procesar imágenes.

#### 1.1.4. Redes Convolucionales Dispersas: SparseConvNet

SparseConvNet ([27]) es la biblioteca de pytorch necesaria para el entrenamiento y uso de redes neuronales convolucionales dispersas submanifold.

Ya hemos hablado de las redes neuronales convolucionales, pero, ¿qué significa el sobrenombrado dispersas?

Las CNN funcionan bien con imágenes en 2D, pero cuando tenemos nubes de puntos en tres dimensiones nos encontramos una gran cantidad de véxeles (equivalente al píxel pero en 3D) vacíos, sin información. Además, al trabajar con estos nuevos volúmenes 3D el cálculo aumenta considerablemente. Con la necesidad de aligerar las operaciones necesarias y de no tratar de la misma forma aquellos lugares dónde sí hay información, de los que están vacíos, surgen las redes convolucionales dispersas ([\[120\]](#)). Aunque estas redes surgieron y se empezaron a utilizar para trabajar sobre estos volúmenes 3D, esto no quiere decir que no puedan ser aplicadas a imágenes 2D.

Con las CNN convencionales encontramos que la cantidad de sitios no nulos crece rápidamente, distorsionando esta información. Sin embargo, con las redes dispersas la cantidad de sitios activos no va a aumentar. Adicionalmente, con estas redes solamente vamos a considerar estos sitios activos, vamos a obviar los nulos, lo que nos va a ahorrar operaciones y va a impedir distorsionar la información, ya que solo vamos a tener en cuenta los activos a la hora de realizar convoluciones. Este comportamiento se puede ver en la figura [\[1.11\]](#). Al trabajar con redes dispersas solamente se realizan las convoluciones en aquellos píxeles en los que hay información, y solo tenemos en cuenta a la hora de realizar estas convoluciones estos mismos píxeles activos (verdes en la ilustración [\[1.11c\]](#)) obviando los nulos (rojos).

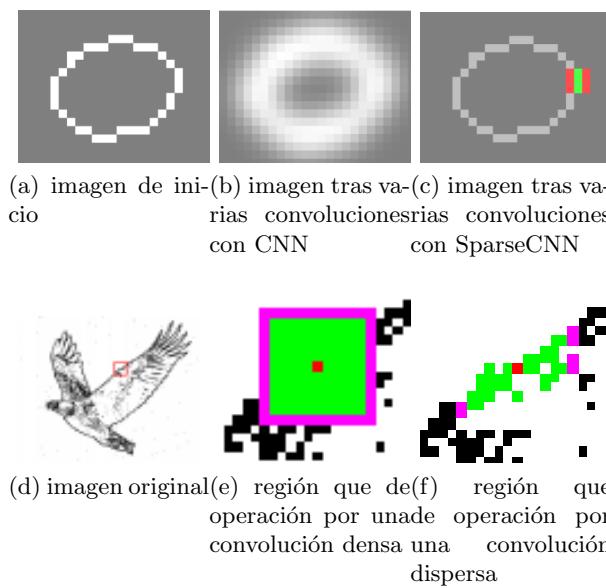


Figura 1.11: CNN vs SparseCNN. Fuentes: [\[27\]](#) y [\[130\]](#)

En las imágenes [1.11e](#) y [1.11f](#) los píxeles negros son los activos, mientras que los blancos están nulos o apagados. El píxel rojo es en el que estamos centrando la operación de convolución actualmente, el píxel en el que hemos centrado el filtro o kernel a aplicar para realizar esta operación. Y, finalmente, los que están representados en violeta y verde, son aquellos píxeles que van a formar parte de la operación de convolución. Vemos como con el modelo denso operamos con todos los píxeles que entran dentro del kernel, mientras que con el modelo disperso solamente operamos con los que entran dentro del kernel y están activos.

La biblioteca SparseConvNet nos permite trabajar tanto con VGG, como con ResNet, que no son más que varios tipos distintos de redes, con diferente orden y cantidad de capas ocultas. También proporciona una serie de ejemplos prácticos.

### 1.1.5. Visión por computador

La visión por computador o CV ([\[69\]](#)) es una rama de la inteligencia artificial que tiene como objetivo permitir a una computadora “ver”, “observar”, e incluso “comprender” las imágenes. Para ello, lo que hacemos realmente es dotar a los ordenadores de la capacidad de obtener información significativa de las imágenes, e interpretarla con el fin de realizar alguna predicción sobre dichos datos.

La visión por computador está estrechamente ligada con el machine learning, y particularmente con las redes neuronales convolucionales de las que ya hemos hablado en este documento. La relación entre estos campos es tan estrecha, que llegan a entremezclarse en muchos casos, y es que la CV utiliza los algoritmos del aprendizaje automático para extraer la información de las imágenes y procesarla correcta y eficientemente.

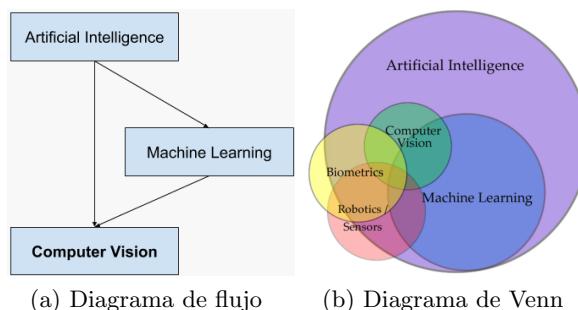


Figura 1.12: La visión por computador se sirve de los algoritmos de machine learning para sintetizar e interpretar la información obtenida a partir de imágenes. Fuente: [\[85\]](#) y [\[86\]](#)

Aunque el objetivo de aplicar la visión por computador puede ser obtener cualquier tipo de información a partir de una o varias imágenes de entrada, son cinco las aplicaciones básicas para las que normalmente se va a utilizar la CV.

- **Clasificación de imágenes:** consiste en clasificar una imagen en una clase determinada, o lo que es lo mismo, etiquetar dicha imagen. Vamos a tratar de describir lo que representa esa ilustración en una única palabra.
- **Detección de objetos:** se utiliza un nuevo objeto llamado bounding box para hallar los distintos objetos que hay en la imagen. El bounding box no es más que un cuadro que se dibuja por encima de la imagen y que contendrá al objeto detectado en su interior. Por lo tanto este cuadro nos servirá para apreciar rápidamente dónde se ha detectado el objeto u objetos, y es un requisito indispensable que solo contenga el objeto detectado. Esto quiere decir que no sirve de nada hacer bounding boxes que cubran toda la imagen, queremos que se ajuste lo más posible al objeto localizado.
- **Reconocimiento de objetos:** es una técnica que combina las dos anteriores. El objetivo ahora no es detectar un objeto solamente, sino que además queremos saber qué elemento es el que se ha hallado. Para ello se suele añadir en uno de los laterales del bonuding box la clase o etiqueta que se cree que presenta el individuo encontrado. Este es el propósito que se persigue en el trabajo que se describe en esta memoria.
- **Segmentación de imágenes:** vamos un paso más allá, ya no nos sirve sólo con detectar y clasificar los distintos objetos de la imagen, sino que deseamos indicar a qué clase pertenece cada píxel. Este no es el tipo de segmentación que nosotros realizamos en este trabajo de fin de grado, sino que como ya se ha mencionado anteriormente, realizamos una versión simplificada, inherente a la forma en que funcionan los sensores de eventos.
- **Seguimiento de objetos:** esta es una técnica usada en fotogramas obtenidos a partir de vídeos, consistente en detectar cada objeto que aparece en la grabación y en caso de que se mueva entre fotogramas dibujar una línea o algún rasgo que nos permita conocer cuál ha sido el camino seguido por dicho objeto.



Figura 1.13: Comparación entre las principales aplicaciones de la visión por computador. Fuente: [34], [33] y [32]

## 1.2. Motivación

Como ya se ha discutido en esta memoria, los sensores de visión dinámica tienen una serie de propiedades muy ventajosas sobre las cámaras tradicionales que están basadas en fotogramas. A pesar de este hecho, no ha sido hasta hace algunos años cuando se ha empezado a realizar investigaciones, y ha desarrollar técnicas de visión por computador aplicadas a la información proporcionada por estos sensores.

El hecho de que sea un campo de investigación joven, con pocos años de existencia, hace que todavía se estén intentando adaptar técnicas de CV tradicionales para poder ser aplicadas a estos eventos, asíncronos y dispersos, tanto temporal como espacialmente. Este hecho no hace más que aumentar el valor de cualquier proyecto que se pueda realizar sobre esta temática. Además, estas cámaras basadas en eventos se siguen desarrollando cada día más, y consecuentemente, con cierta frecuencia se descubren nuevas aplicaciones en las que pueden cobrar vital importancia. Hoy en día ya se está demostrando un gran interés en la incorporación de estos sensores en robots móviles, sobre todo drones. El poder incorporar técnicas de reconocimiento de objetos a drones equipados con esta nueva tecnología, les proporcionará una gran autonomía, ya sea para la exploración de zonas inaccesibles para un humano, para la búsqueda de supervivientes en un edificio derrumbado tras una catástrofe natural, o simplemente aprovechar su alta resolución temporal y baja latencia para detectar objetos inesperados que vayan a colisionar con el dron, y modificar la trayectoria rápidamente para esquivarlos.

En concreto, para la realización de este proyecto, se ha pensado que si las Sparse CNN proporcionan una gran ganancia computacional, manteniendo un correcto funcionamiento cuando trabajamos con nubes de puntos de tres dimensiones, y esta ganancia es debida principalmente a la gran cantidad de véxeles vacíos que hay en estos volúmenes. Por qué no aplicar estas técnicas a la información devuelta por estos sensores de eventos, siendo los eventos proporcionados asíncronos y muy reducidos en cantidad en comparación con

la información de un fotograma tradicional. De esta manera no desaprovechamos la latencia reducida y la alta resolución temporal de los sensores en los algoritmos de detección, porque no nos sirve de nada disponer de la información rápidamente si este tiempo que ganamos con el nuevo hardware lo desaprovechamos en las técnicas software.

Además, el uso de un simulador de eventos permite realizar una comparación más fácilmente con estos procesos realizados sobre imágenes convencionales. Así como su implantación en un sistema robótico, en lugar del sensor físicamente, evitaría tener que adquirir este nuevo hardware que, como hemos visto, es muy costoso y puede no ser sencilla su incorporación al resto del robot.

Finalmente, la liberación de código es una práctica bastante habitual en este tipo de proyectos de investigación, prácticamente una obligación ética, debido a que es un método que permite involucrar a muchos desarrolladores beneficiándose unos del trabajo de otros. Además, esto fomenta la innovación y que más gente se interese por los temas tratados. En este sentido el repositorio de github en el que se ha liberado este proyecto está disponible en [87].

### 1.3. Objetivos

La finalidad principal de este proyecto de investigación es la implementación de un software para el reconocimiento de objetos a partir de eventos, y el estudio de su eficacia. Además, también se pretende conseguir otra serie de objetivos como los que se exponen a continuación:

- Estudiar el estado del arte sobre los sensores de visión dinámica, simuladores de éstos y las técnicas de visión por computador aplicadas a los eventos devueltos por estos.
- Estudiar el estado del arte de las redes neuronales convolucionales dispersas y su funcionamiento.
- Implementar algoritmos de aprendizaje automático y visión por computador para la detección y el reconocimiento de objetos a partir de eventos que son asíncronos y dispersos en el espacio y el tiempo.
- Testear la segmentación de objetos implícita al trabajar con sensores de eventos, inherente a las propiedades y características de estas cámaras.
- Testear los algoritmos implementados ejecutándolos con distintos parámetros y distinta cantidad de etapas en el entrenamiento de la red.

- Testear la inferencia realizada por un modelo que trabaje con la información proporcionada por las cámaras basadas en eventos, y la inferencia realizada con un modelo que trabaje sobre las imágenes convencionales.
- Liberar el código en GitHub ([\[87\]](#)), redactando además un manual de usuario, para contribuir a la comunidad y que pueda ser estudiado y utilizado por cualquier otro estudiante e investigador libremente.
- Aprender nuevas metodologías de desarrollo de software y planificación a largo plazo.
- Aprender nuevos programas de comunicación entre miembros del equipo.

## Capítulo 2

# Gestión del proyecto

### 2.1. Metodología de desarrollo

Para la realización de este trabajo se ha realizado una metodología de desarrollo *Incremental*, que nos permite ir construyendo el proyecto en distintas etapas, agregando nuevas funcionalidades en cada una de ellas. Este modelo combina elementos de la metodología en cascada tradicional, con otros de la creación de prototipos, como es la filosofía interactiva.

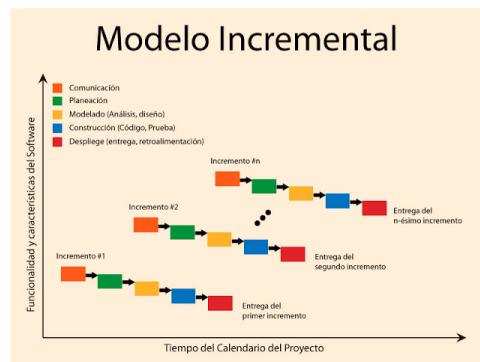


Figura 2.1: Esquema de funcionamiento de la metodología incremental.  
Fuente: [60]

En la imagen 2.1 vemos el esquema de funcionamiento de este tipo de metodologías, aunque existen distintas versiones que difieren en cuanto al comienzo de la siguiente etapa. Mediante este modelo vamos a generar software funcional de forma rápida y constante, desde etapas tempranas en el desarrollo. Además es una metodología flexible, por lo que posibles cambios en los requisitos no conlleva un gran coste, es fácil probar y depurar las versiones entre etapas, y en cada iteración estamos abordando y consiguien-

do hitos y objetivos alcanzables. Finalmente, también nos va a permitir ir obteniendo realimentación útil con cada pequeña funcionalidad añadida.

Por todos estos motivos, hemos creído conveniente adoptar esta metodología de desarrollo teniendo en cuenta que en este proyecto vamos a combinar distintos software con distintas funcionalidades cada uno de ellos, permitiéndonos conocer en cada momento el estado en el que nos encontramos y dejándonos la flexibilidad que necesitamos para añadir nuevos requisitos y adoptar pequeños cambios en estos, según vayan siendo detectados.

## 2.2. Planificación

Para la planificación y organización del proyecto se ha hecho uso de una hoja de cálculo de Google Docs. En este documento se han creado varias páginas (las podemos ver en la figura 2.2). La primera página, llamada principal, contiene las distintas tareas a realizar a alto nivel de abstracción. Mientras tanto, el resto de páginas incluyen las distintas subtareas que hay que realizar en cada uno de estos objetivos principales.

Principal 0.- Planificación 1.- Estudio del estado del arte 2.- Instalación de ROS 3.- Instalación de un simulador de eventos 4.- Instalación del software para el reconocimiento de objetos 5.- Selección de la base de datos 6.- Implementación de la inferencia	✓ 4.- Instalación del software para el reconocimiento de objetos 5.- Selección de la base de datos 6.- Implementación de la inferencia 7.- Pruebas, experimentos y mejoras 8.- Extracción de resultados 9.- Redacción de la memoria 10.- Preparación de la presentación	Gantt
(a)	(b)	

Figura 2.2: Distintas páginas creadas en la hoja de cálculo de Google Docs utilizada para la planificación del proyecto.

Cada tarea, tendrá cuatro columnas asociadas, siendo la primera de estas el nombre o breve descripción de la tarea a realizar. En las dos siguientes casillas, vamos a indicar la fecha en la que comenzamos a realizar la tarea y la fecha en la que la hemos acabado, en caso de que ya la hayamos finalizada. La última columna que nos indica el estado de la tarea, es simplemente una columna más visual para ver de un vistazo rápido las tareas pendientes, las que estamos realizando y las que ya se han acabado. Todo esto lo podemos ver en la ilustración 2.3. Este documento está hecho de forma que nosotros solamente tenemos que indicar las fechas de inicio y fin de cada subtarea. Las páginas principal y Gantt se llenan automáticamente, al igual que la columna del estado.

	A	B	C	D	E
1					
2	Tarea	Fecha Inicio	Fecha Finalización	Estado	
3	0.- Planificación	2021-01-11	2021-06-16	TERMINADO	
4	1.- Estudio del estado del arte	2021-01-15	2021-02-15	TERMINADO	
5	2.- Instalación de ROS	2021-02-15	2021-02-16	TERMINADO	
6	3.- Instalación de un simulador de eventos	2021-02-16	2021-06-15	TERMINADO	
7	4.- Instalación del software para el reconocimiento de objetos	2021-02-26	2021-03-15	TERMINADO	
8	5.- Selección de la base de datos para el entrenamiento	2021-03-15	2021-03-29	TERMINADO	
9	6.- Implementación de la inferencia	2021-03-29	2021-06-03	TERMINADO	
10	7.- Pruebas, experimentos y mejoras	2021-06-03	2021-06-17	TERMINADO	
11	8.- Extracción de resultados	2021-06-17	2021-07-02	TERMINADO	
12	9.- Redacción de la memoria	2021-06-29		EN PROCESO	
13	10.- Preparación de la presentación			NO COMENZADO	
14					

(a) página principal (tareas a alto nivel)

	A	B	C	D	E
1	Tarea	Fecha Inicio	Fecha Finalización	Estado	
2	Repetir experimentos anotando resultados	2021-06-29	2021-07-05	TERMINADO	
3	Redacción primera versión de la memoria	2021-06-30		EN PROCESO	
4	Revisión de la memoria por parte del profesor			NO COMENZADO	
5	Redacción de la versión definitiva de la memoria			NO COMENZADO	
6					

(b) página de las subtareas de la memoria (subtareas relacionadas con la redacción de la memoria)

Figura 2.3: Ejemplo de cómo se ha llevado a cabo la planificación y organización del proyecto.

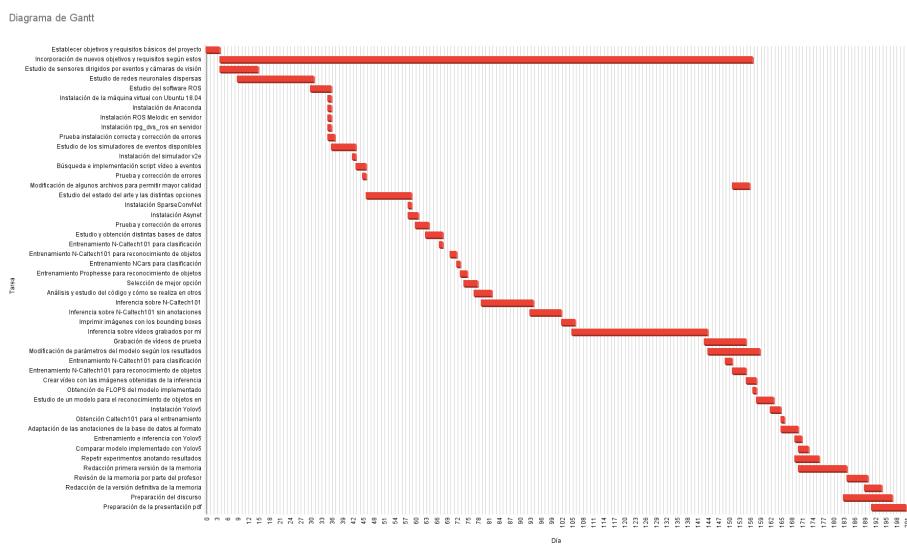


Figura 2.4: Diagrama de Gantt del proyecto.

El diagrama de Gantt es una herramienta gráfica que nos permite conocer el tiempo total dedicado a cada una de las tareas realizadas en el proyecto, y de esta forma conocer si se está avanzando a buen ritmo, o por contra nos estamos retrasando en la realización del proyecto. En las figuras 2.4 y 2.5 podemos ver el diagrama de Gantt y el diagrama de Gantt simplificado respectivamente, tras la finalización completa de este trabajo.

Diagrama de Gantt simplificado

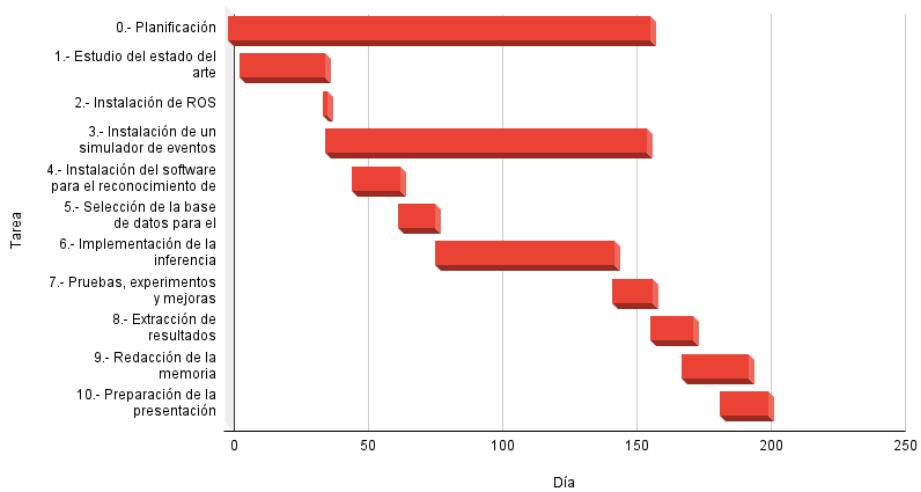


Figura 2.5: Diagrama de Gantt simplificado del proyecto.

Esta nos ha parecido una buena forma de llevar la planificación debido a su simplicidad y claridad. Además nos permite añadir o modificar fácilmente nuevas tareas, incluso en secciones que ya se creían terminadas, lo cual va muy acorde con la flexibilidad que nos daba la metodología incremental adoptaba.

Adicionalmente se ha utilizado la plataforma Slack para mantener una comunicación fluida con el tutor, y para ponernos de acuerdo los distintos alumnos con acceso al servidor utilizado acerca de quién usarlo en cada momento. Finalmente también se ha utilizado GitHub para mantener el proyecto en línea y accesible en todo momento.

Por lo general, nos hemos ido retrasando en la realización de cada una de las tareas principales respecto a la duración estimada que realicé conjuntamente con el tutor al inicio del proyecto. Este retraso lo podemos dividir en dos tipos bien diferenciados, un primer tipo motivado principalmente por los problemas en la comunicación debidos a la situación del COVID-19 y por la necesidad de dedicar tiempo a otras materias, y un segundo tipo de retraso ocasionado por los problemas técnicos y dificultades encontradas durante la

realización del proyecto. Mientras que el primer tipo era esperado y estaba controlado dentro de los márgenes que teníamos, el segundo tipo sí que ha causado algún que otro contratiempo importante respecto a la planificación inicial.

### 2.3. Hitos

Durante el desarrollo de este trabajo de fin de grado, como era previsible, aunque se han perseguido en todo momento unos objetivos globales claros y marcados, los más específicos han ido variando a medida que nos hemos ido encontrando con dificultades, puesto que estos inconvenientes nos hacían cambiar el punto de vista y enfoque planteado. Esto hace que los objetivos que nos habíamos marcado al principio del desarrollo, no coincidan con los finalmente alcanzados, que son los que aparecen expuestos en esta sección. Estas metas alcanzadas son las siguientes:

1. **Instalación de Ubuntu 18.04**(*Inicio en el día 35 del proyecto*): sistema operativo que posee el servidor sobre el que vamos a trabajar.
2. **Instalación de Anaconda**(*Inicio en el día 35 del proyecto*): distribución de python a utilizar.
3. **Instalación de ROS Melodic**(*Inicio en el día 35 del proyecto*): distribución de ROS compatible con el sistema operativo Ubuntu 18.04.
4. **Instalación de un simulador de eventos**(*Inicio en el día 36 del proyecto*): estudio de las distintas posibilidades, selección e instalación de un simulador que nos convierta un vídeo de fotogramas a un vídeo de eventos y nos permita obtener dichos eventos.
5. **Implementación script video a eventos**(*Inicio en el día 43 del proyecto*): búsqueda y adaptación de un script que nos permita obtener los eventos en formato rosbag (con el que trabaja ROS) a partir de un vídeo de eventos.
6. **Instalación software de reconocimiento de objetos a partir de eventos**(*Inicio en el día 46 del proyecto*): estudio de las técnicas existentes, selección e instalación de un software que nos permita realizar la detección y el reconocimiento de objetos a partir de los eventos.
7. **Instalación redes neuronales convolucionales dispersas**(*Inicio en el día 58 del proyecto*): instalación de la versión de facebook research para pytorch de las SparseCNN que vamos a utilizar para aprovechar la dispersión espacio-temporal de los eventos.

8. **Obtención de la base de datos** (*Inicio en el día 63 del proyecto*): estudio, selección y adaptación de la base de datos que vamos a utilizar para entrenar el modelo con eventos.
9. **Inferencia sobre la base de datos** (*Inicio en el día 79 del proyecto*): implementación de los scripts necesarios para realizar la inferencia sobre el conjunto de test de la base de datos utilizada para entrenar el modelo.
10. **Inferencia sobre imágenes de mi propio conjunto de datos** (*Inicio en el día 105 del proyecto*): implementación de los scripts necesarios para realizar la inferencia sobre imágenes de eventos obtenidas por mi.
11. **Inferencia sobre un vídeo obtenido por mi mismo** (*Inicio en el día 155 del proyecto*): implementación de los scripts necesarios para realizar la inferencia sobre vídeos de eventos obtenidos por mi.
12. **Instalación de un software para el reconocimiento de objetos en imágenes normales** (*Inicio en el día 158 del proyecto*): estudio, selección e instalación de un software para el reconocimiento de objetos en imágenes normales.
13. **Obtención de la base de datos** (*Inicio en el día 165 del proyecto*): búsqueda y adaptación de la misma base de datos utilizada para entrenar al modelo que recibe eventos, para utilizarlo en este modelo que funciona con imágenes.
14. **Validación de resultados obtenidos** (*Inicio en el día 170 del proyecto*): realización de pruebas con distintos valores de los parámetros, comparación entre el modelo basado en eventos y el modelo basado en fotogramas, y conclusiones ante los resultados obtenidos.

## **2.4. Presupuesto**

Para calcular el presupuesto total de este proyecto se han deducido los fondos necesarios para los distintos tipos de recursos utilizados de forma independiente, de esta forma el presupuesto global del proyecto puede ser obtenido fácilmente de la forma que se puede observar en [2.1](#), donde  $P_{total}$  hace referencia al presupuesto total que será necesario para la realización del proyecto, con  $P_{Hard}$  nos referimos a la cantidad necesaria para obtener los recursos hardware, con  $P_{Soft}$  nos referimos a los recursos software, con  $P_{Hum}$  nos referimos al presupuesto que tendremos que dedicar al pago del personal que lleve acabo el trabajo, y finalmente, con  $P_{extra}$  nos referimos a un amortiguador financiero para el pago de suministros, horas extra y reparaciones necesarias.

$$P_{total} = P_{Hard} + P_{Soft} + P_{Hum} + P_{extra} \quad (2.1)$$

Además hay que tener en cuenta que  $P_{Hum}$  se ha calculado en función del número de meses de trabajo que se ha realizado para la completa realización del proyecto y del sueldo medio que presenta un trabajador en el sector recién acabada la carrera. Por otro lado, tanto  $P_{Hard}$  como  $P_{Soft}$  se han calculado de la manera que nos muestra 2.2.

$$C_{aplicable} = (U_i/V_i) * C_i \quad (2.2)$$

Donde :

- $U_i$  es el número de meses que se ha utilizado el recurso i en el proyecto.
- $V_i$  es la vida útil del recurso i.
- $C_i$  es el coste medio del recurso i.

#### 2.4.1. Recursos Hardware

Artículo	$U_i$ (meses)	$V_i$ (meses)	$C_i(\text{€})$	$C_{aplicable}(\text{€})$
Ordenador personal	6	36	1200	200
Servidor con Nvidia RTX 2080 Ti	4	120	1800	60
Total				260

Cuadro 2.1: costes hardware del proyecto

En la tabla 2.1 se ha puesto el coste medio de un ordenador de sobremesa y la vida útil del mismo se ha establecido según lo indicado en 5. El coste del servidor se ha obtenido también como el coste medio añadiéndole el precio de la tarjeta gráfica según la web oficial [109], y la vida útil según nos indica 6.

### 2.4.2. Recursos Software

Artículo	$U_i$ (meses)	$V_i$ (meses)	$C_i(\text{€})$	$C_{aplicable}(\text{€})$
Ubuntu 18.04	6	120	0	0
ROS Melodic	2	-	0	0
Anaconda y Spider	6	-	0	0
Total				0

Cuadro 2.2: costes software del proyecto

### 2.4.3. Recursos Humanos

Teniendo en cuenta que se han requerido sobre unos 6 meses de trabajo, esto son aproximadamente 150 días, teniendo en cuenta que no se ha trabajado todos los días de la semana, en los que se han invertido una media de 4 horas diarias.

Además, un graduado en ingeniería informática cobra unos 22000 euros anuales en los primeros años tras finalizar su carrera según publica [4]. Esto supone un gasto cercano a los 1850€ mensuales, que al sumarle la seguridad social y demás pagos extra requeridos por parte de la empresa, supone un coste de 2850€ mensuales por un trabajador con jornada completa, es decir, 8 horas diarias y 5 días a la semana. Esto supone un coste de 17.81€/hora.

Considerando las 600 horas que he necesitado para completar el trabajo, se necesitaría reservar sobre unos 10686€.

### 2.4.4. Coste total

Recurso	Coste(€)
Hardware	260
Software	0
Humanos	10686
Total	10946
Total + amortiguador financiero(15 %)	12587.9

Cuadro 2.3: costes totales del proyecto

Por lo tanto, el presupuesto necesario para llevar a cabo un proyecto como el que se describe en este documento sería de unos 12587.9€

## Capítulo 3

# Entorno Operacional

En este capítulo del documento vamos a hablar de ROS (Robot Operating System), aunque solo daremos una breve introducción al marco de trabajo que representa, puesto que en el proyecto solo nos es de utilidad para mostrar los archivos rosbag y comprobar que los vídeos convertidos lo han hecho de forma correcta. También hablaremos del sistema operativo utilizado y de Anaconda, que nos va a proporcionar la distribución de python y el entorno que necesitamos. Finalmente daré las características técnicas más destacables de los distintos componentes hardware utilizados.

### 3.1. Ubuntu 18.04

Ubuntu 18.04 ([\[98\]](#)) es la distribución de Linux en la que se ha desarrollado el trabajo. Principalmente, el motivo por el que nos hemos decantado por utilizar esta versión es que el servidor utilizado funciona con este sistema operativo, y puesto que una parte importante del proyecto se ha tenido que realizar sobre dicho servidor (en el capítulo [\[6\]](#) se explican los motivos) consideramos que lo correcto es utilizar un único sistema operativo para todo el proyecto, y ya que utilizamos el servidor, realizar el trabajo entero en él, ahorrándonos así posibles problemas de compatibilidad entre versiones. Además la versión de ROS para Ubuntu 20.04, Noetic, no es tan robusta como las anteriores.

### 3.2. Anaconda

Anaconda ([\[2\]](#)) es una distribución libre del lenguaje de programación python desarrollada por Anaconda Inc., la cual nos aporta una gran cantidad de aplicaciones, librerías y un gestor de paquetes, Conda. Anaconda

cuenta con unas características que la hacen imprescindible para el desarrollo de aplicaciones de aprendizaje automático y visión por computador, entre las que se encuentran que es multiplataforma, cuenta con varios entornos de desarrollo, permite la instalación y administración de paquetes y sus dependencias de forma cómoda y rápida, permite acceder a recursos de aprendizaje avanzados, etc.

Además, una de las características que más nos interesan de Anaconda es la posibilidad de crear entornos virtuales. Estos entornos virtuales nos permiten aislar los recursos y librerías de los que haya instalados en el sistema principal o en otros entornos. Es muy recomendable trabajar con un entorno virtual por proyecto, o incluso un entorno distinto por funcionalidad, debido principalmente a que de este modo conseguimos evitar que existan problemas por incompatibilidades entre librerías, e incluso, entre distintas versiones de python. También nos ofrece claridad al evitar que tengamos instalados un montón de paquetes distintos en el sistema principal.

En nuestro caso, estos entornos son de vital importancia, puesto que vamos a trabajar con distintos programas, que necesitan distintas versiones de python, como son python2.7 y python3.8, con sus respectivos paquetes, que aunque proporcionen la misma funcionalidad, no son compatibles entre sí.

### 3.3. ROS

El sistema operativo robótico, más conocido como ROS ([\[74\]](#)) por su nombre en inglés, es un framework libre para el desarrollo de software para robots. Ofrece un conjunto de herramientas y librerías que nos permiten implementar comportamientos robóticos complejos de forma fácil y robusta.

Este sistema persigue la colaboración como marco de trabajo. Desarrollar una tarea para que un robot la realice es un proceso muy complejo, por muy fácil y trivial que sea una acción para un humano, a la hora de programar este comportamiento tenemos que describir detalladamente todo el proceso, lo cual no es fácil para acciones intuitivas. Es por esto que ROS permite integrar cómodamente varios sistemas, de esta forma un laboratorio podría descubrir un sistema de navegación mediante mapas muy eficiente y otro cómo elaborar estos mapas, o un sistema de navegación basado en la visión por computador, y mediante ROS podríamos integrarlos cómodamente.

ROS, por lo tanto, se divide en dos partes básicas. El sistema operativo y marco de trabajo común, o ros propiamente dicho, y ros-pkg, el conjunto de paquetes y funcionalidades distintas implementadas por toda la comunidad. Mientras que ros es libre, dentro de ros-pkg cada autor puede definir los derechos y licencias del paquete que él mismo o su equipo hayan desarrollado.

### 3.3.1. Esquema de funcionamiento

Existe una serie de elementos ([67]) que van a tomar parte en el proceso de comunicación peer to peer entre los distintos módulos que queremos integrar usando ROS. Estos elementos son los siguientes:

- **Nodos:** un nodo es un proceso, un programa ejecutable, que realiza una función concreta. Cada nodo se compila individualmente lo que aporta una gran modularidad a ROS. Ejs: mapeo del entorno, planificación de rutas, control de un telémetro láser, etc.
- **Pilas:** un conjunto de nodos que aportan una funcionalidad concreta al sistema forman una pila. Ej: navegación.
- **ROS Master:** proporciona un registro de nombres, almacena información sobre el registro de topics y servicios para los nodos. Los nodos se conectan entre ellos directamente, el ROS Master solamente proporciona información de búsqueda. Permite crear conexiones entre nodos dinámicamente a medida que estos se van ejecutando. Este nodo máster lleva implícito un servidor de parámetros que permite que los datos sean almacenados en función de una clave en el servidor central.
- **Mensajes:** el paso de mensajes es el sistema de comunicación entre nodos. Un mensaje es una estructura de datos, que puede estar formada por tipos de datos primitivos u otras estructuras más complejas definidas específicamente.
- **Topics:** los nodos envían mensajes pertenecientes a un tema determinado (los publican), y los nodos que estén interesados en dicho tema se suscriben a él para obtener los datos y particularidades del mensaje.
- **Servicios:** el modelo de publicar/suscribir mensajes no es apropiado para comunicaciones uno a muchos con interacción de solicitud y respuesta. Es por ello que este tipo de interacción se realiza mediante los servicios, un par de mensajes con una estructura muy determinada.
- **Bags:** son un tipo de archivo determinado que permite guardar y reproducir mensajes de ROS.

Vistos los distintos elementos que intervienen y cómo se realiza la comunicación entre dos sistemas mediante ROS, podemos esquematizarla de forma simple como se ve en la figura 3.1. En ella se puede apreciar fácilmente como un nodo publica distintos mensajes pertenecientes a un mismo topic, y otro nodo que esté interesado en ese tema se subscribe a él para obtener información más concreta. En la ilustración los nodos aparecen conectados directamente, no aparece reflejado el proceso de conexión y la

intervención del ROS master, el cual es fundamental y sin el cual no podría haber interacción entre nodos.

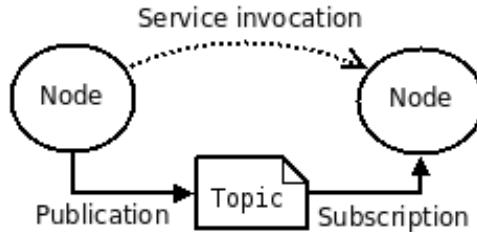


Figura 3.1: Esquema de comunicación entre nodos ROS. Fuente:[\[67\]](#)

### 3.3.2. Herramientas

A parte de las ventajas y las funciones que nos ofrece ros, podemos expandir sus aplicaciones mediante una serie de herramientas instalables del mismo modo que los paquetes. Las principales son las siguientes:

- **rviz:** herramienta para la visualización 3D. Posee distintos tipos de formatos y es configurable.
- **rosbag:** permite grabar y leer mensajes y datos de ROS. Para ello utiliza un tipo de archivo propio con extensión .bag en el que va escribiendo los topics y mensajes a medida que van siendo emitidos.
- **catkin:** permite la compilación de ROS. Es una versión posterior a rosbuild que permite la compilación multiplataforma.
- **rosbash:** aumenta las funcionalidades del bash de ros.
- **roslaunch:** nos permite la ejecución de múltiples nodos. Ofrece la capacidad de crear archivos de configuración que pueden ser usados para automatizar algunos procesos complejos.

## 3.4. PC y Servidor

El ordenador utilizado no tiene mayor importancia puesto que, como ya se ha comentado, el trabajo se ha realizado de forma íntegra sobre el servidor, a excepción de la redacción de esta memoria y el estudio del estado del arte.

Lo realmente importante son las especificaciones técnicas que presenta el servidor:

- **CPU:** Intel(R) Core(TM) i7-9700K @3.60GHz

- **GPU:** NVIDIA GeForce RTX 2080 Ti 11GB
- **RAM:** DDR4 de 64GB

Adicionalmente, se ha utilizado un teléfono móvil para grabar los vídeos con los que posteriormente hemos sacado los eventos e intentado detectar y reconocer los objetos que en ellos se muestran. Lo único relevante aquí, es por tanto, la resolución de la cámara. Sin embargo, debido al alto coste computacional que conlleva el simulador v2e, no hemos podido usar vídeos de gran resolución, la máxima utilizada ha sido 854x480.



# Capítulo 4

## Diseño

### 4.1. Casos de uso

Un diagrama de casos de uso representa un sistema desde el punto de vista de los actores y las interacciones entre los distintos subsistemas. Es muy importante diferenciar bien entre caso de uso y diagrama de caso de uso. Mientras que el primero es la descripción de una actividad o funcionalidad concreta, el segundo de ellos describe la interacción entre los actores y las distintas funcionalidades del sistema.

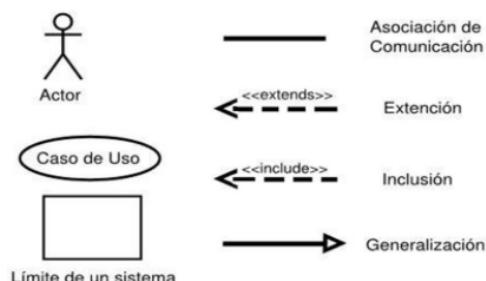


Figura 4.1: Representación de los distintos elementos que intervienen en un diagrama de casos de uso. Fuente: [47]

En un diagrama de casos de uso intervienen distintos elementos:

- **Actores:** cualquier entidad externa, persona o no, que interactúe con nuestro sistema e inicie o demande alguna actividad de este.
- **Casos de uso:** descripción de lo que hace una actividad o proceso, sin decir cómo lo hace.
- **Sistemas y subsistemas:** conjunto de casos de uso relacionados entre sí por formar parte de un sistema mayor con una funcionalidad

concreta.

- **Relaciones entre ellos:** la relación entre un actor y un caso de uso se llama comunicación e indica que un actor utiliza o demanda el uso de dicho proceso. En el caso de relaciones entre casos de uso podemos encontrarnos dos tipos distintos: include y extend. Cuando la relación entre dos casos de uso es de inclusión, el primero de ellos no se podrá realizar sin la ejecución del segundo, es decir, el primero necesita al segundo. En el caso de una relación de extensión, es similar a la anterior, pero la diferencia radica en que el primero no necesita la ejecución del segundo, es un extra.

En la figura 4.2 podemos ver el diagrama de casos de uso del sistema implementado en este trabajo. Podemos ver que intervienen procesos pertenecientes a cuatro subsistemas distintos, y algunos procesos independientes, que no pertenecen a ningún subsistema concreto. En este caso, tan solo interviene un actor, el usuario del sistema que es el que va a tener que ir completando el proceso y demandando el uso de los distintos procesos involucrados según corresponda.

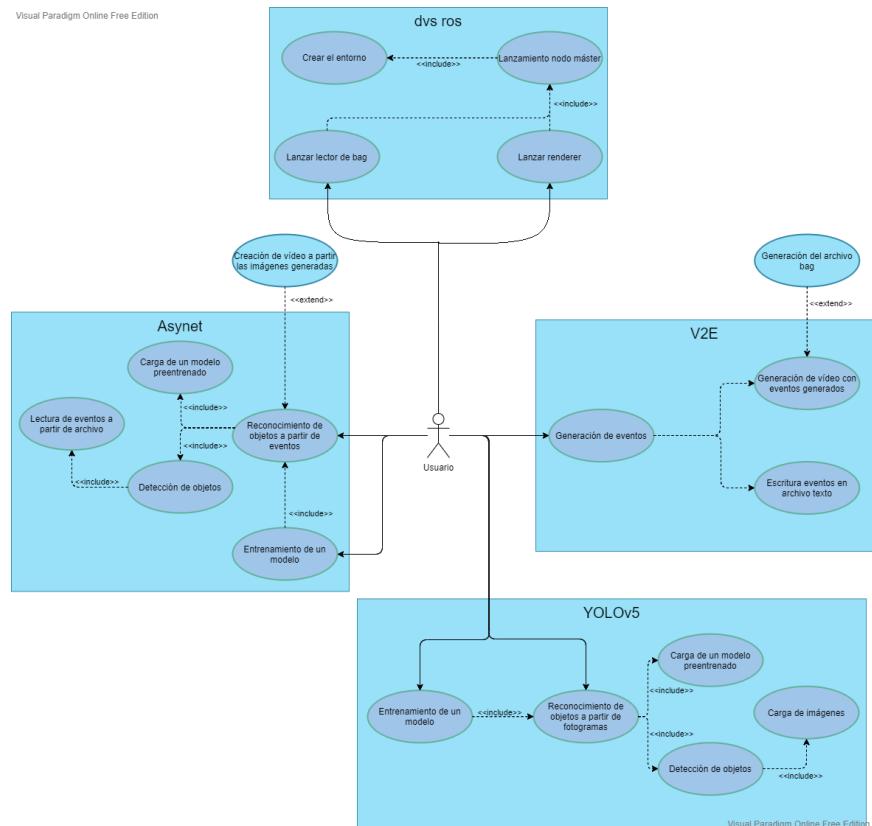


Figura 4.2: Diagrama UML de casos de uso.

Caso de Uso	Crear el entorno	CU-001
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual ros debe estar en activo.	
Post-condiciones	Podremos lanzar distintos nodos ROS.	
Propósito	Creación del entorno adecuado para el lanzamiento de distintos nodos ROS y que puedan establecer una comunicación entre ellos.	
Resumen	Se compilan y lanzan los paquetes y el renderer, y se configura el entorno adecuadamente.	

Cuadro 4.1: Caso de Uso CU-001: Crear el entorno

Caso de Uso	Lanzamiento nodo máster	CU-002
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual ros debe estar en activo y se tiene que haber creado el entorno adecuadamente.	
Post-condiciones	Se habrá creado correctamente el nodo máster, junto con el servidor de parámetros y el canal de registro de nodos y publicación de mensajes.	
Propósito	Crear el nodo máster y los requisitos necesarios para crear el sistema ROS al que podrán conectarse posteriormente nuevos nodos.	
Resumen	Creación del roscore (ROS Master + Servidor de parámetros + rosout logging node).	

Cuadro 4.2: Caso de Uso CU-002: Lanzamiento nodo máster

Caso de Uso	Lanzamiento lector de bag	CU-003
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual ros debe estar en activo, se tiene que haber creado el entorno adecuadamente y haber un nodo máster corriendo.	
Post-condiciones	Un nuevo nodo añadido al sistema estará publicando una serie de mensajes con la información de los eventos que hay en un archivo bag especificado.	
Propósito	Publicar los eventos alojados en un archivo bag.	
Resumen	Creación de un nodo rosbag encargado de publicar mensajes con los eventos alojados en un archivo con extensión bag indicado por parámetro.	

Cuadro 4.3: Caso de Uso CU-003: Lanzamiento lector de bag

Caso de Uso	Lanzar renderer	CU-004
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual ros debe estar en activo, se tiene que haber creado el entorno adecuadamente y haber un nodo máster corriendo.	
Post-condiciones	Un nuevo nodo añadido al sistema recogerá los mensajes con información acerca de eventos proporcionados por un nodo rosbag y permitirá la visualización de estos.	
Propósito	Tener información visual de los eventos.	
Resumen	Creación de un nodo roslaunch encargado de recoger y proyectar en formato de vídeo los mensajes con información de eventos.	

Cuadro 4.4: Caso de Uso CU-004: Lanzar renderer

<b>Caso de Uso</b>	Entrenamiento de un modelo	<b>CU-005</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual asynet debe estar en activo y debe haber una base de datos etiquetada lo suficientemente grande disponible.	
Post-condiciones	Tendremos un modelo capaz de realizar predicciones en el reconocimiento de objetos a partir de eventos.	
Propósito	Tener un modelo para el reconocimiento de objetos a partir de eventos entrenado.	
Resumen	Se van cargando eventos pertenecientes a una escena desde la base de datos hasta tener suficiente información de la escena. Con esta información el modelo realiza una predicción en el reconocimiento que comparará con la etiqueta real para actualizar una serie de pesos en función de cómo haya sido su predicción.	

Cuadro 4.5: Caso de Uso CU-005: Entrenamiento de un modelo

<b>Caso de Uso</b>	Reconocimiento de objetos a partir de eventos	<b>CU-006</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual asynet debe estar en activo y se debe haber detectado la presencia de un objeto en la región a reconocer.	
Post-condiciones	Se habrá dado una etiqueta al objeto detectado.	
Propósito	Identificar los objetos detectados en una escena a partir de eventos.	
Resumen	El modelo utiliza los pesos que tiene para comparar distintos patrones identificados en la escena y en función de estos decidir qué clase de objeto se ha detectado.	

Cuadro 4.6: Caso de Uso CU-006: Reconocimiento de objetos a partir de eventos

<b>Caso de Uso</b>	Detección de objetos a partir de eventos	<b>CU-007</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual asynet debe estar en activo.	
Post-condiciones	Se habrá detectado la presencia de un objeto en una región concreta de la imagen y generado un bounding box alrededor de este.	
Propósito	Detectar la presencia de objetos en la escena a partir de eventos.	
Resumen	El modelo utiliza los pesos que tiene para buscar y detectar ciertos patrones en los datos de la escena.	

Cuadro 4.7: Caso de Uso CU-007: Detección de objetos a partir de eventos

<b>Caso de Uso</b>	Carga de un modelo preentrenado	<b>CU-008</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual asynet debe estar en activo y tener un modelo entrenado previamente guardado en un archivo con formato pth.	
Post-condiciones	Tendremos un modelo con los pesos obtenidos en el entrenamiento que se realizó en algún momento anterior.	
Propósito	Obtener un modelo entrenado previamente permitiendo así su reutilización en cualquier momento.	
Resumen	Realizar la carga de un diccionario de pytorch con la información y los pesos referentes a un modelo preentrenado.	

Cuadro 4.8: Caso de Uso CU-008: Carga de un modelo preentrenado

<b>Caso de Uso</b>	Lectura de eventos a partir de archivo	<b>CU-009</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual asynet debe estar en activo y la existencia de al menos un archivo que leer.	
Post-condiciones	La información de los eventos ubicados en el archivo estará disponible para utilizarse.	
Propósito	Obtener la información de los eventos de una escena.	
Resumen	Leer todas las líneas del archivo ubicando en distintos arrays las coordenadas, la polaridad y el instante de tiempo de los eventos ubicados en el archivo. La posición x de los distintos arrays hará referencia al evento x, estando estos ordenados por anterioridad temporal.	

Cuadro 4.9: Caso de Uso CU-009: Lectura de eventos a partir de archivo

<b>Caso de Uso</b>	Creación de vídeo a partir de las imágenes generadas	<b>CU-010</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	Haber generado más de una imagen en el reconocimiento de objetos.	
Post-condiciones	Un vídeo formado por las distintas imágenes.	
Propósito	Crear un vídeo en el que aparezcan bounding boxes con el nombre de los objetos que aparecen en los distintos frames.	
Resumen	Se leen ordenadamente todas las imágenes de una carpeta determinada y se construye un vídeo con todas esas imágenes a velocidad de 10 fotogramas por segundo.	

Cuadro 4.10: Caso de Uso CU-010: Creación de vídeo a partir de las imágenes generadas

Caso de Uso	Generación de eventos	CU-011
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	En entorno virtual v2e debe estar en activo y debe haber al menos un vídeo con formato avi.	
Post-condiciones	Generación de eventos en función de los cambios de intensidad detectados en los frames.	
Propósito	Obtener los eventos que daría un sensor dinámico de visión si se hubiera grabado con él la escena mostrada en el vídeo.	
Resumen	En primer lugar se transforma el vídeo de RGB a Luma para poder detectar los cambios en la intensidad comodamente. A continuación se realiza una interpolación entre cada dos frames para añadir fotografías y reducir la cantidad de eventos generados en un mismo espacio de tiempo.	

Cuadro 4.11: Caso de Uso CU-011: Generación de eventos

Caso de Uso	Escritura eventos en archivo texto	CU-012
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	En entorno virtual v2e debe estar en activo y se deben haber generado los eventos de un archivo de vídeo.	
Post-condiciones	La información referente a los eventos estará disponible en un archivo de texto.	
Propósito	Escribir las coordenadas, polaridad e instante de tiempo de los distintos eventos generados en un archivo de texto.	
Resumen	Por cada evento escribir una línea con las coordenadas, polaridad e instante de tiempo.	

Cuadro 4.12: Caso de Uso CU-012: Escritura eventos en archivo texto

<b>Caso de Uso</b>	Generación de vídeo con eventos generados	<b>CU-013</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	En entorno virtual v2e debe estar en activo y se deben haber generado los eventos de un archivo de vídeo.	
Post-condiciones	Habrá un vídeo con todos los eventos.	
Propósito	Crear un vídeo en el que visualizar los eventos, como si la escena se hubiera grabado con un sensor de visión dinámica.	
Resumen	Crear distintos fotogramas en los que representar los eventos en función de sus instantes de tiempo y coordenadas para finalmente unirlos todos en un archivo de vídeo.	

Cuadro 4.13: Caso de Uso CU-013: Generación de vídeo con eventos generados

<b>Caso de Uso</b>	Generación del archivo bag	<b>CU-014</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	La existencia de un vídeo de eventos.	
Post-condiciones	Información de los eventos representada en archivo con formato bag.	
Propósito	Creación de un archivo bag con la información de los eventos de una escena para su posterior lectura mediante ROS.	
Resumen	Extraer la información de los eventos que aparecen en el vídeo y escribirlos en un archivo con extensión bag en el formato determinado.	

Cuadro 4.14: Caso de Uso CU-014: Generación del archivo bag

Caso de Uso	Entrenamiento de un modelo	CU-015
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual yolo debe estar en activo y debe haber una base de datos etiquetada lo suficientemente grande disponible.	
Post-condiciones	Obtendremos un modelo capaz de realizar predicciones en el reconocimiento de objetos en una imagen.	
Propósito	Tener un modelo para el reconocimiento de objetos en imágenes entrenado.	
Resumen	El modelo realiza una predicción en el reconocimiento que comparará con la etiqueta real para actualizar una serie de pesos en función de cómo haya sido su predicción.	

Cuadro 4.15: Caso de Uso CU-015: Entrenamiento de un modelo

Caso de Uso	Reconocimiento de objetos a partir de fotogramas	CU-016
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual yolo debe estar en activo y se debe haber detectado la presencia de un objeto en la región a reconocer.	
Post-condiciones	Se habrá dado una etiqueta al objeto detectado.	
Propósito	Identificar los objetos detectados en una imagen.	
Resumen	El modelo utiliza los pesos que tiene para comparar distintos patrones identificados en la escena y en función de estos decidir qué clase de objeto se ha detectado.	

Cuadro 4.16: Caso de Uso CU-016: Reconocimiento de objetos a partir de fotogramas

<b>Caso de Uso</b>	Carga de un modelo preentrenado	<b>CU-017</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual yolo debe estar en activo y tener un modelo entrenado previamente guardado en un archivo con formato pth.	
Post-condiciones	Tendremos un modelo con los pesos obtenidos en el entrenamiento que se realizó en algún momento anterior.	
Propósito	Obtener un modelo entrenado previamente permitiendo así su reutilización en cualquier momento.	
Resumen	Realizar la carga de un diccionario de pytorch con la información y los pesos referentes a un modelo preentrenado.	

Cuadro 4.17: Caso de Uso CU-017: Carga de un modelo preentrenado

<b>Caso de Uso</b>	Detección de objetos a partir de fotogramas	<b>CU-018</b>
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual yolo debe estar en activo.	
Post-condiciones	Se habrá detectado la presencia de un objeto en una región concreta de la imagen y generado un bounding box alrededor de este.	
Propósito	Detectar la presencia de objetos en la imagen.	
Resumen	El modelo utiliza los pesos que tiene para buscar y detectar ciertos patrones en los datos de la escena.	

Cuadro 4.18: Caso de Uso CU-018: Detección de objetos a partir de fotogramas

Caso de Uso	Carga de imágenes	CU-019
Actor	Usuario	
Tipo	Principal, real	
Pre-condiciones	El entorno virtual yolo debe estar en activo y la existencia de al menos un archivo que leer.	
Post-condiciones	La información de los tres canales de la imagen estará disponible para ser utilizada.	
Propósito	Obtener la información de la imagen.	
Resumen	Leer los tres canales de la imagen y representarlos en forma de matriz de 3 dimensiones.	

Cuadro 4.19: Caso de Uso CU-019: Carga de imágenes

## 4.2. Diagrama de flujo

Un diagrama de flujo es una representación gráfica de un proceso desde su inicio hasta el final. En un diagrama de flujo [4.3](#) aparecen reflejadas todos los distintos caminos que puede recorrer el algoritmo hasta completarse. El diagrama de flujo que aparece aquí reflejado describe el proceso que debe seguir el usuario del sistema para completar todo el proceso desde la grabación de un vídeo hasta el reconocimiento de objetos en él.

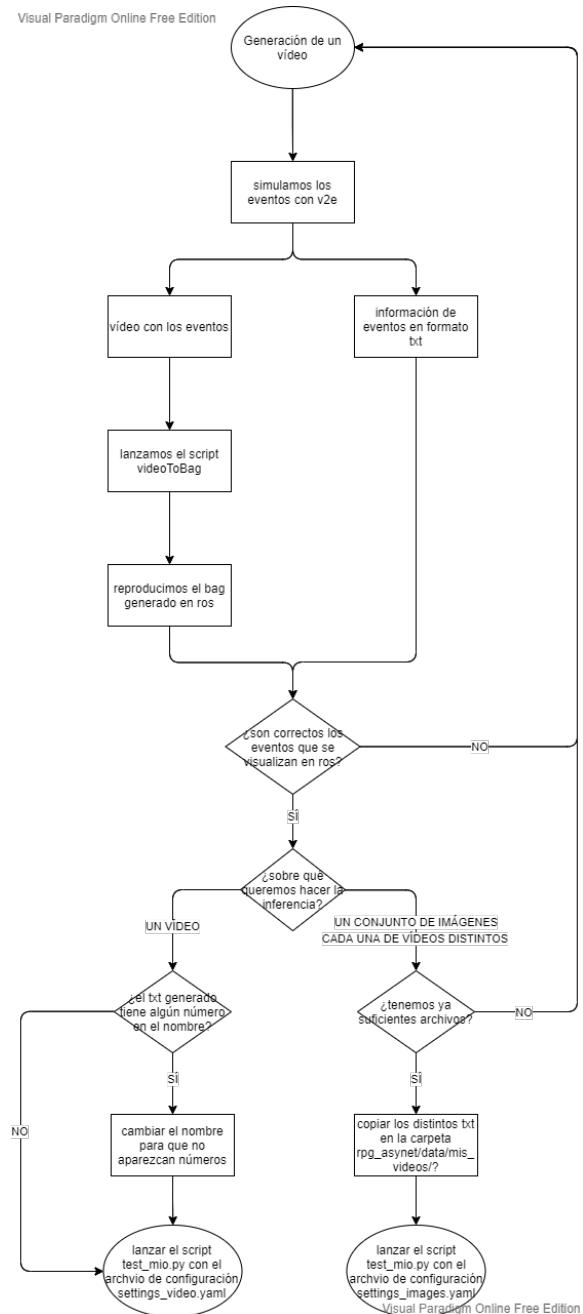


Figura 4.3: Diagrama de flujo que representa el proceso a seguir por el usuario para utilizar el sistema correctamente.

En la descripción del proceso que aparece reflejado en el diagrama 4.3 se ha partido de la base de que el modelo ya ha sido entrenado previamente. En [13] se proporcionan varios modelos ya entrenados. En caso de querer realizar

un entrenamiento propio, o simplemente entrenarlo un poco más, habría que lanzar el script `train.py` con el archivo de configuración `settings.yaml` e indicar que el nuevo modelo resultante en la salida de este script es el que queremos utilizar en el resto de archivos de configuración. Todo este proceso se explicará con mayor detenimiento en secciones posteriores ([5](#) y [7](#)).

# Capítulo 5

# Implementación

Para la realización de este proyecto de fin de grado se ha hecho uso de varios sistemas, integrándolos, adaptándolos cuando ha sido necesario, añadiendo nuevas funcionalidades, y adaptando las salidas de unos módulos a las entradas requeridas por otros. En esta sección explicaremos los principales cambios y añadidos realizados en cada sistema.

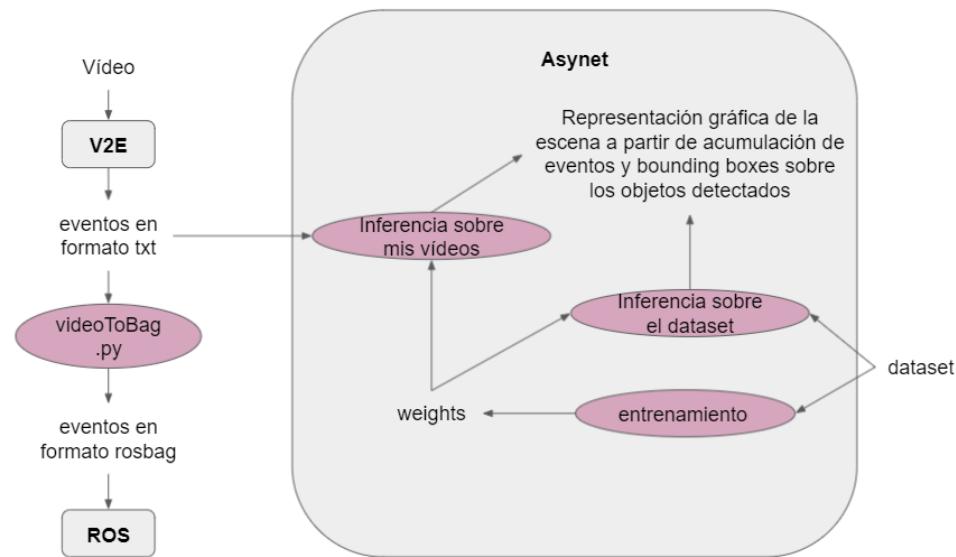


Figura 5.1: Entradas, salidas y relación entre los distintos sistemas.

### 5.1. Simulador de eventos: v2e

Siendo el tema principal del proyecto que en esta memoria se presenta el reconocimiento de objetos usando retinas artificiales, parece lógico que

lo primero en lo que nos centremos sea la generación de la información aportada por estos sensores. Las cámaras de eventos son muy caras como para que nosotros pensemos en obtener una, sin embargo, los simuladores de eventos nos proporcionan la capacidad de obtener información espacio-temporal equiparable a la que se hubiese obtenido con un sensor de visión dinámica, y por ello son una alternativa perfectamente válida y gratuita. Además podemos aprovechar la versatilidad que nos aportan al ofrecernos los eventos de salida en distintos formatos, así como resoluciones diferentes.

En concreto nos hemos decantado por la utilización de v2e. La elección de este simulador por delante de otros, como ESIM, se debe a que aunque solo es capaz de procesar archivos de vídeo, y solo los que openCV pueda leer, ofrece un modelo de píxel DVS más realista debido a su sistema de conversión. Además es un simulador más simple, mientras que ESIM es más complejo de entender y utilizar al admitir entradas de distintos tipos, no vídeos únicamente. En el apartado [I](#) de este documento se explica qué son los simuladores de eventos y para qué se utilizan con mayor amplitud, además del funcionamiento de v2e. En esta sección nos centraremos en explicar los cambios realizados en él y el motivo de estos.

```

183 #           if dvs_aedat2:
184 #               path = os.path.join(self.output_folder, dvs_aedat2)
185 #               path = checkAddSuffix(path, '.aedat')
186 #               logger.info('opening AEDAT-2.0 output file ' + path)
187 #               self.dvs_aedat2 = AEDat2Output(
188 #                   path, output_width=self.output_width,
189 #                   output_height=self.output_height)

```

(a) líneas comentadas en el constructor

```

201 #           if self.dvs_aedat2 is not None:
202 #               self.dvs_aedat2.close()

```

(b) líneas comentadas en la función cleanup()

```

621 #           if self.dvs_aedat2 is not None:
622 #               self.dvs_aedat2.appendEvents(events)

```

(c) líneas comentadas en la función generate\_events()

Figura 5.2: Líneas comentadas en el archivo v2ecore/emulator.py para permitir una salida con cualquier resolución.

En un principio el sistema solo aceptará 346x260 píxeles y 240x180 píxeles como resoluciones de salida válidas, produciendo cuando especificamos otra distinta el error “CAMERA type not found, add your camera to v2ecore/output/aedat2\_output.py”. Con el objetivo de obtener los eventos

con cualquier calidad que podamos necesitar teníamos dos posibles soluciones a aplicar. Implementar la salida para el resto de resoluciones que creamos convenientes, o no generar el archivo aedat, que es dónde ocurre el error. En este caso se ha abordado esta última alternativa.

Para evitar la generación del archivo aedat, simplemente tenemos que eliminar las llamadas a las funciones del archivo especificado por el mensaje de error que se producen desde otros ejecutables. Resulta que esto ocurre solamente en el script v2ecore/emulator.py, y en lugar de eliminar las líneas de código directamente se ha optado por comentarlas, evitando así que el intérprete de python las tenga en cuenta. En la imagen 5.2 se muestran todas las líneas que tuvimos que comentar y la función concreta a la que pertenecen.

Con estos pequeños cambios, los archivos generados por el simulador v2e son los que vemos en la figura 5.3. De todos estos archivos, tan solo nos interesan dos de ellos, dvs-video.avi y v2e-dvs-events.txt. El segundo es el archivo mediante el cual extraeremos la información de los eventos para que la red neuronal utilizada pueda aprender y realizar predicciones, y cuyo formato puede verse en la ilustración 1.6. Por otro lado, el primero nos servirá para comprobar que la simulación es adecuada.

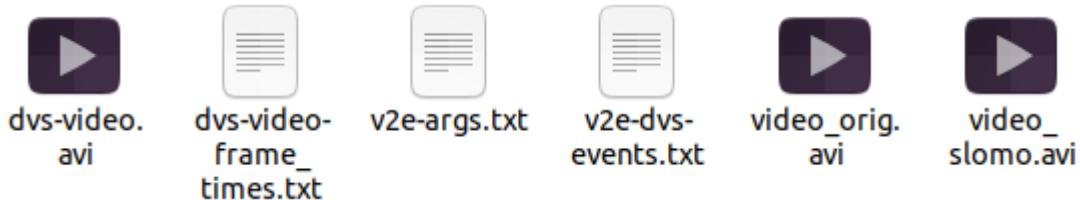


Figura 5.3: Archivos que nos produce como salida el simulador v2e al haber eliminado el formato aedat.

## 5.2. ROS: dvs\_ros

Además del simulador, también hemos tenido que instalar la versión de ROS para el sistema operativo utilizado, Melodic, y el paquete dvs\_ros, el cual nos proporciona una serie de controladores para los sensores de eventos DVS y DAVIS. La ventaja que presenta este paquete es que nos permite trabajar con archivos de datos de eventos, archivos rosbags, que hayan sido generados previamente, sin tener que disponer del sensor físicamente.

Para cumplir con todas las dependencias necesarias para la instalación del paquete dvs\_ros, nos hemos visto obligados en muchos casos a instalar dichas dependencias por separado, incluso a desinstalar la versión existente en el sistema previamente. Dicho esto, no hemos creído conveniente ni

necesario modificar nada en lo referido a este paquete y ROS Melodic.

El objetivo por el cual nos es necesario este sistema es poder visualizar los eventos generados con el simulador y comprobar de una manera más sencilla si son adecuados para el reconocimiento de objetos, o al menos, si los eventos se adaptan bien a la realidad, no son muy limitados y no existe demasiado ruido.

Sin embargo, ahora nos encontramos con un problema añadido, como ya sabemos los nodos de ROS trabajan con archivos rosbags, y nuestro simulador no tiene la capacidad de producir este tipo de archivo. ¿La solución? Utilizar alguna de las salidas ofertadas para la generación del formato requerido mediante un script externo. Esto es precisamente lo que nos pusimos a investigar, y rápidamente dimos con un código, [78], que nos daba esta posibilidad mediante el dvs-video.avi del simulador. Este código no estaba absento de errores y dependencias incumplidas. Los fallos más destacados se deben a la utilización de una versión anticuada del módulo OpenCV, por lo que no es complicado solucionarlos.

```

12 def CreateVideoBag(videopath, bagname):
13     '''Creates a bag file with a video file'''
14     bag = rosbag.Bag(bagname, 'w')
15     cap = cv2.VideoCapture(videopath)
16     cb = CvBridge()
17     prop_fps = cap.get(cv2.CAP_PROP_FPS)
18     if prop_fps != prop_fps or prop_fps <= 1e-2:
19         print ("Warning: can't get FPS. Assuming 24.")
20         prop_fps = 24
21     ret = True
22     frame_id = 0
23     while(ret):
24         ret, frame = cap.read()
25         if not ret:
26             break
27         stamp = rospy.rostime.Time.from_sec(float(frame_id) / prop_fps)
28         frame_id += 1
29         image = cb.cv2_to_imgmsg(frame, encoding='bgr8')
30         image.header.stamp = stamp
31         image.header.frame_id = "camera"
32         bag.write(TOPIC, image, stamp)
33     cap.release()
34     bag.close()

```

Figura 5.4: Función principal del script utilizado para conseguir un archivo bag con todos los eventos generados en el simulador tras haber realizado algunas modificaciones a la versión disponible en [78].

A parte de estos errores simples, también se han tenido que instalar algunos paquetes nuevos, e incluso reinstalar otros ya disponibles. También existe la necesidad de ejecutar el script mediante Python2.7, cuando la versión disponible era la 3.8. Esto nos obligó a descargar Python2 y reinstalar algunos paquetes para que puedan ser utilizados por ambas versiones de for-

ma correcta. El script utilizado, con todos los cambios realizados lo podemos ver en la imagen [5.4](#).

Con el objetivo de automatizar y facilitar todo el proceso de generación de eventos, desde el simulador v2e hasta la obtención del formato bag, se ha creado el script generarEventos.sh que se puede ver en la figura [5.5](#). En la tercera línea podemos ver el modo correcto en el que hay que ejecutarlo. Es importante hacerlo así, y no mediante los comandos “sh” ni “./” porque estos dos métodos producirían ciertos errores al tener que cambiar de entorno virtual mediante comandos “conda” dinámicamente, dentro del propio script.

```

1#!/bin/bash
2
3#Ejemplo de uso: bash -i generarEventos.sh video1 video2
4
5let CONTADOR=0
6
7for l in "$@"
8do
9    let CONTADOR=$CONTADOR+1
10   conda activate v2e
11   python v2e.py -l input/$l.avi --overwrite --timestamp_resolution=.003 --auto_timestamp_resolution=False --dvs_exposure_duration
0.005 --output_folder=output/$l --overwrite --pos_thres=.15 --neg_thres=.15 --sigma_thres=0.03 --output_width=854 --output_height=480 --
cutoff_hz=15
12   echo "Video $l COMPLETADO"
13   conda deactivate
14   conda activate ros
15   python2 eventsToBag.py output/$l/dvs-video.avi output/$l/$l.bag
16   conda deactivate
17   conda activate v2e
18   echo "Generado archivo bag correctamente"
19   mv output/$l/v2e-dvs-events.txt output/$l/$l.txt
20 done|
```

Figura 5.5: Script de ubuntu implementado para facilitar la generación completa de eventos y del archivo bag.

Este script permite la simulación de varios vídeos con formato avi uno detrás de otro cómodamente. La resolución de salida de todos ellos será de 854x480 píxeles, y se obtendrán los eventos del vídeo completo, aunque tanto este como otros parámetros es muy simple cambiarlos tal y como se explica en el repositorio del v2e([\[28\]](#)). También vemos que el archivo bag generado tiene el nombre del vídeo de entrada, del mismo modo que se modifica el nombre del txt con la información de los eventos para que tenga este mismo nombre. Un aspecto muy importante a tener en cuenta para la el posterior reconocimiento de objetos es que el nombre no tenga ningún número, dan igual los caracteres especiales, pero no los números. Esto es por el método implementado para la lectura de los eventos, como veremos más adelante en este mismo capítulo.

### 5.3. Reconocimiento de objetos a partir de eventos

Una vez hemos adquirido la capacidad, o al menos las herramientas, para comprobar que la salida de v2e es apta para realizar el reconocimiento de objetos sobre ella, tan solo nos faltaba una red neuronal que fuera capaz de trabajar con este tipo de datos. Debido a la novedad de estas cámaras y

que todavía se está investigando mucho en el asunto, no fue fácil encontrar un modelo capaz de realizar esta función. Todavía hay un abismo entre los innumerables modelos capaces de realizar la detección y reconocimiento de objetos sobre imágenes, a los extremadamente pocos que son capaces de trabajar con eventos. Finalmente encontramos y nos pareció adecuado Asynet. Este es un modelo implementado por el Department of Neuroinformatics of the University of Zurich and ETH Zurich, que hace uso de la implementación de Facebook Research de las Sparse CNN para aprovechar la asincronicidad espacio-temporal de los eventos.

La documentación del repositorio donde está disponible el sistema, [24], nos indica que tenemos cuatro modelos distintos a nuestra disposición:

- SparseVGG para clasificación
- SparseVGG para detección y reconocimiento de objetos
- DenseVGG para clasificación
- DenseVGG para detección y reconocimiento de objetos

Como ya se ha comentado en alguna ocasión, el que a nosotros realmente nos interesa es el modelo disperso para aprovechar al máximo las características de los sensores de visión dinámica. Sin embargo, ya que teníamos la posibilidad de utilizar un modelo denso pensamos que era una buena idea realizar una comparación entre ambos, aunque finalmente tuvimos que olvidarnos de esta idea. El problema está en que se trata de una red neuronal muy densa, con una gran cantidad de conexiones entre neuronas, y entrenar este modelo implica una carga computacional demasiado grande, inalcanzable con el servidor y el hardware que tenemos a nuestro alcance. Esto nos impide reentrenarla, y este hecho unido a que no tenemos a nuestra disposición los pesos obtenidos por los autores, nos imposibilita hacer dicho análisis. No obstante puede ser bastante interesante realizar este estudio en un futuro si alguna vez disponemos de la capacidad para ello.

A la hora de entrenar el modelo que nos parezca conveniente tenemos que hacer unas pequeñas modificaciones en ciertos archivos. Aunque el ejecutable para realizar el entrenamiento es proporcionado por los autores, éste no presenta ningún tipo de mecanismo de parada. El entrenamiento, tal y como los desarrolladores lo han subido, no acaba nunca. Esto tiene fácil corrección, basta con cambiar las líneas que se ven en la imagen [5.6] en el archivo training/trainer.py. Esta es la única modificación que hemos tenido que introducir para poder completar el entrenamiento satisfactoriamente. En estas líneas vemos que se han puesto los valores 1000 y 500 en cada una respectivamente, no hay que prestar especial interés a estos números puesto que ya explicaremos en capítulos posteriores, 7. *Experimentación* [7] para

ser más concretos, el por qué de ellos. Ahora mismo solo queremos destacar la modificación de las condiciones del bucle para que éste tenga un final.

232       **while iteracionActual < 1000:#True:**

(a) Entrenamiento del modelo sparseVGG para clasificación

492       **while iteracionActual < 500:#True:**

(b) Entrenamiento del modelo sparseVGG para reconocimiento de objetos

Figura 5.6: Líneas a cambiar en el archivo training/trainer.py.

Para realizar este entrenamiento tenemos que haber seleccionado una base de datos de entre las disponibles. En este caso podíamos elegir entre dos bases de datos para la clasificación y otras dos para el reconocimiento de objetos.

Entre estas cuatro bases de datos hay dos que son la misma, N-Caltech101, es decir, esta base de datos se puede utilizar tanto para la detección como para la clasificación, solamente cambian las anotaciones que hay sobre ella. Neuromorphic Caltech101 es una base de datos formada a partir de las imágenes que constituyen Caltech101 y la cámara de eventos ATIS. Para ello se han proyectado las distintas imágenes sobre una pantalla blanca de una en una, y con el sensor ATIS situado en una estructura mecánica en frente de dicha pantalla, se obtienen los eventos resultantes de realizar 3 sacudidas rápidas y pequeñas de la cámara. Tal y como dicen los autores en [131], se ha necesitado 500ms por imagen, 100ms por sacudida y 200ms para la transición entre imágenes. En total se han obtenido los eventos de 8709 imágenes, pertenecientes a 100 clases distintas, más una de background. Todas ellas con un tamaño de 245x302 píxeles. Los eventos se han publicado en un archivo binario por cada imagen, con su mismo nombre y en los que cada línea contiene la información relativa a un único evento. Cada una de las líneas contiene 40 bits de datos que se distribuyen de la siguiente manera:

- **bits 39 - 32:** posición x en píxeles.
- **bits 31 - 24:** posición y en píxeles.
- **bits 23:** polaridad siendo un 0 para OFF y un 1 para representar el ON.
- **bits 22 - 0:** marca de tiempo en microsegundos.

Además N-Caltech101 presenta anotaciones con la información de los bounding boxes. Un ejemplo de los eventos formados a partir de una imagen se puede ver en la figura 5.7.

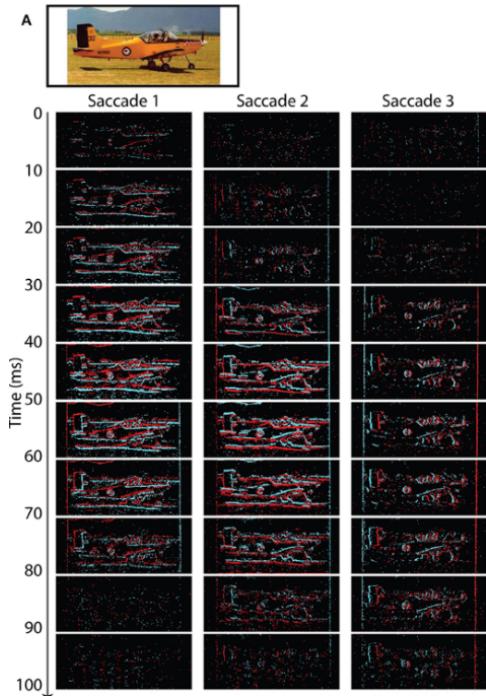


Figura 5.7: Ejemplo que muestra los eventos obtenidos durante la formación de la base de datos N-Caltech101 a partir de las imágenes de Caltech101. Fuente: [131].

Las otras dos bases son NCars para clasificación y Prophesee Gen1 Automotive para la detección de objetos. Estas dos bases de datos están relacionadas entre sí, ambas representan datos de coches y entornos grabados desde un vehículo. Los datos de NCars se han obtenido durante varias conducciones equipando una cámara ATIS en el parabrisas del coche. El dataset, centrado en clasificación, contiene 12336 ejemplos de automóviles y 11693 que no lo son, muestran el entorno simplemente. Cada ejemplo con una duración de 100ms. Por otro lado, Automotive se ha obtenido mediante una cámara Prophesee Gen1 equipada en el techo de un vehículo. Esta base de datos se ha construido mediante los datos obtenidos tras 39 horas de conducción y se han creado bounding boxes para un total de 228123 coches y 27658 peatones. Un ejemplo de este último dataset lo podemos ver en la imagen 5.8.

Finalmente, decidimos trabajar y seguir el proyecto sobre N-Caltech101 por varios motivos que serán explicados con detalle en 7 entre los que des-

tacan su simplicidad, la mayor cantidad de clases y su mayor utilidad en un entorno cotidiano.

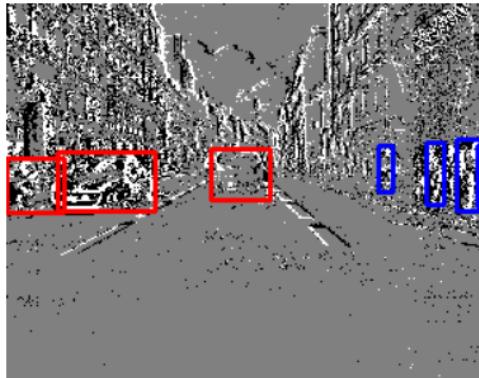


Figura 5.8: Ejemplo que muestra una imagen formada a partir de acumulación de eventos de la base de datos Automotive y los bounding boxes creados por los autores. Fuente: [11].

Para alcanzar nuestro objetivo final, inferir sobre datos obtenidos por nuestra cuenta, hemos decidido abordar una serie de etapas intermedias que comienzan infiriendo sobre los datos de la propia base de datos N-Caltech101 que no hemos usado para entrenar, y terminan con este objetivo que nos hemos propuesto inicialmente.

### 5.3.1. Inferencia sobre la base de datos N-Caltech101

Para aplicar el modelo preentrenado al conjunto test de la base de datos Neuromorphic Caltech101 hemos creado un nuevo archivo tester.py ubicado en la carpeta testing. Este nuevo script de python hace uso de algunas de las funcionalidades de la clase AbstractTrainer(), que se encuentra definida también en el archivo training/trainer.py, y que es la encargada de crear y cargar el dataset, así como de contener las variables donde se almacenan ciertas estadísticas de interés. También hacemos uso en nuestro nuevo archivo de la clase SparseObjectDetModel(), que recibe como parámetro un objeto de la clase anterior y es la encargada de crear el modelo que vamos a utilizar. Estas clases no se han añadido tal cual aparecen en el archivo anterior, hay que hacer ciertas variaciones para incluir la funcionalidad que queremos añadir, la inferencia, y eliminar aquellos aspectos que no nos sirven por ser específicos para el entrenamiento.

Lo primero que hemos tenido que hacer es añadir una serie de atributos para manejar ciertos datos y guardar las estadísticas que el conjunto test nos ofrece, sustituyéndolas por las del train. En concreto son las que aparecen en la imagen 5.9.

```

350     self.test_loader = None
351     self.nr_test_epochs = None

384     self.test_batch_step = 0
385     self.testing_loss = 0
386     self.testing_accuracy = 0
387     self.test_confusion_matrix = np.zeros([self.nr_classes, self.nr_classes])

```

Figura 5.9: Atributos añadidos en el archivo testing/tester.py para realizar la inferencia.

Además, un detalle importante que no se nos puede pasar por alto es la creación del dataloader y el dataset para este conjunto. Aunque están implementados los constructores de estos objetos en sus archivos correspondientes, no se hace la llamada a ellos en el script del entrenamiento puesto que no los necesitamos, sin embargo en esta ocasión no se nos pueden olvidar. Para ello bastaría con añadir las líneas que aparecen reflejadas en la figura 5.10.

```

423     test_dataset = self.dataset_builder(self.settings.dataset_path,
424                                         self.settings.object_classes,
425                                         self.settings.height,
426                                         self.settings.width,
427                                         self.settings.nr_events_window,
428                                         mode='testing',
429                                         event_representation=self.settings.event_representation)
430     self.nr_test_epochs = int(test_dataset.nr_samples / 1) + 1
431     self.nr_classes = test_dataset.nr_classes
432     self.object_classes = test_dataset.object_classes

440     self.test_loader = self.dataset_loader(test_dataset, batch_size=1,
441                                         device=self.settings.gpu_device,
442                                         num_workers=self.settings.num_cpu_workers, pin_memory=False)

```

Figura 5.10: Creación del dataset y el dataloader para el conjunto test en el archivo testing/tester.py.

Finalmente, para terminar con el archivo testing/tester.py, hemos implementado la función test(), cuyo objetivo es hacer la inferencia sobre el conjunto que recibe este mismo nombre. Esta función, que se muestra en la ilustración 5.11, carga el modelo preentrenado especificado y va infiriendo en los datos según estos van siendo cargados. Además esta función implementada nos generará una imagen por cada archivo de entrada formada por los distintos eventos, y dibujará sobre ella tanto el bounding box real, el que aparece en las anotaciones, como el que nuestro modelo ha predicho. Adicionalmente, también nos proporcionará información acerca de la bondad de sus predicciones.

```

657     def test(self):
658         self.loadCheckpointTest(self.settings.pretrained_sparse_vgg)
659
660         self.pbar = tqdm.tqdm(total=self.nr_test_epochs, unit='Batch', unit_scale=True)
661         self.resetTesting()
662         self.model = self.model.eval()
663         self.bounding_boxes = BoundingBoxes()
664         loss_function = yoloLoss
665
666         for i_batch, sample_batched in enumerate(self.test_loader):
667             event, bounding_box, histogram = sample_batched
668
669             # Convert spatial dimension to model input size
670             histogram = torch.nn.functional.interpolate(histogram.permute(0, 3, 1, 2),
671                                              torch.Size(self.model_input_size))
672             histogram = histogram.permute(0, 2, 3, 1)
673
674             # Change x, width and y, height
675             bounding_box[:, :, [0, 2]] = (bounding_box[:, :, [0, 2]] * self.model_input_size[1].float()
676                                         / self.settings.width).long()
677             bounding_box[:, :, [1, 3]] = (bounding_box[:, :, [1, 3]] * self.model_input_size[0].float()
678                                         / self.settings.height).long()
679
680             locations, features = self.denseSparse(histogram)
681
682             with torch.no_grad():
683                 model_output = self.model([locations, features, histogram.shape[0]])
684                 loss = loss_function(model_output, bounding_box, self.model_input_size)[0]
685                 detected_bbox = yoloDetect(model_output, self.model_input_size.to(model_output.device),
686                                             threshold=0.7)
687                 detected_bbox = nonMaxSuppression(detected_bbox, iou=0.6)
688                 detected_bbox = detected_bbox.cpu().numpy()
689
690             # Save validation statistics
691             self.saveBoundingBoxesTest(bounding_box.cpu().numpy(), detected_bbox)
692
693             batch_one_mask = locations[:, -1] == 0
694             vis_locations = locations[batch_one_mask, :]
695             features = features[batch_one_mask, :]
696             vis_detected_bbox = detected_bbox[detected_bbox[:, 0] == 0, 1:-2].astype(np.int)
697
698             # Visualization
699             image = visualizations.visualizeLocations(vis_locations.cpu().int().numpy(), self.model_input_size,
700                                                 features=features.cpu().numpy(),
701                                                 bounding_box=bounding_box[0, :, :].cpu().numpy(),
702                                                 class_name=[self.object_classes[i]
703                                                 for i in bounding_box[0, :, -1]])
704             image = visualizations.drawBoundingBoxes(image, vis_detected_bbox[:, :-1],
705                                                 class_name=[self.object_classes[i]
706                                                 for i in vis_detected_bbox[:, -1]],
707                                                 ground_truth=False, rescale_image=False)
708
709             path_name = os.path.join(self.settings.vis_dir, 'image_' + str(self.test_batch_step) + '.png')
710             fig, ax = plt.subplots()
711
712             ax.imshow(image.astype(np.float))
713             ax.axis('off')
714             fig.savefig(path_name)
715             plt.close()
716
717             self.pbar.set_postfix(Valloss=loss.data.cpu().numpy())
718             self.pbar.update(1)
719             self.test_batch_step += 1
720             self.testing_loss += loss
721
722             self.testing_loss = self.testing_loss / float(self.test_batch_step)
723             self.saveTestingStatisticsObjectDetection()
724
725             self.pbar.close()

```

Figura 5.11: Función test en el archivo testing/tester.py.

Una práctica habitual en visión por computador es inferir sobre el conjunto test de la base de datos utilizada para entrenar justo al finalizar este entrenamiento. De esta manera obtenemos más estadísticas y medidas de bondad ajenas al propio entrenamiento que las obtenidas simplemente con la validación, lo que nos permitirá realizar un mejor análisis de los resultados y de los mejores pesos obtenidos en el entrenamiento. Es por ello que he creado otro script de python, training/trainer\_and\_tester.py, que combina el proceso de entrenamiento con la posterior inferencia, cuando ésta se realiza sobre el subconjunto test de la misma base de datos, en un único archivo.

Para la creación de este script simplemente hemos tenido que incluir las funciones implementadas y los cambios realizados en el tester.py a las ya existentes en el trainer.py, gastando cuidado para que las nuevas funcionalidades añadidas no se pisen con las anteriores. Adicionalmente a esto, hemos

tenido que crear una nueva función para cargar el modelo a la hora de hacer inferencia para que este cargue los últimos pesos que se hayan guardado, puesto que estos serán los mejores.

En este caso no podíamos especificar cuáles son los pesos exactos que queremos cargar porque no lo sabemos. Aunque a priori podemos calcular y conocer en qué épocas se van a realizar las etapas de validación, no podemos saber cuál de todas ellas nos proporcionará mejores resultados. Lo que sí sabemos es que los últimos pesos que se hayan guardado son los que mejores métricas han proporcionado. Esto es lo que nos ha llevado a implementar la función `loadPretrainedWeights2()`, que se puede ver en 5.12 de esa manera, almacenando en un array todo el contenido del directorio donde se han almacenado los distintos archivos con los pesos y cargando el último elemento.

```
343     def loadPretrainedWeights2(self):
344         contenido = os.listdir(self.settingsckpt_dir)
345         file_path = os.path.join(self.settingsckpt_dir, contenido[len(contenido)-1])
346         self.loadCheckpointTest(file_path)
```

Figura 5.12: Función para cargar el último modelo que se haya guardado en el proceso de entrenamiento.

El siguiente paso lógico que debíamos dar antes de poder detectar objetos en nuestros propios archivos de eventos es eliminar la información de las anotaciones. Cuando tu haces inferencia, ya sea sobre imágenes o sobre eventos, tu quieres predecir dónde está un objeto que no sabes que está ahí. Normalmente este proceso se automatiza, y tú ni siquiera vas a ver las imágenes o los eventos. Por lo tanto no dispones de estas anotaciones, es por ello que hemos creado el script `testing/tester_sin_estadisticas.py`, que nos permite inferir sobre la base de datos utilizada para el entrenamiento obviando las anotaciones que tiene.

Para ello no hay que realizar grandes cambios con respecto al ejecutable `testing/tester.py` explicado anteriormente. Simplemente tenemos que eliminar todas las instrucciones que utilizan esta información ya sea para imprimir las imágenes con ambos bounding boxes, o para calcular ciertas métricas, como la pérdida o el mean average precision. Por lo tanto la única función que debemos modificar es la encargada de inferir, la función `test()`. Este método de la clase `SparseObjectDetModel` quedaría como se ve en la ilustración 5.13. Tras eliminar estas líneas nos queda una función mucho más simple.

```

343     def test(self):
344         self.loadCheckpointTest(self.settings.pretrained_sparse_vgg)
345
346         self.pbar = tqdm.tqdm(total=self.nr_test_epochs, unit='Batch', unit_scale=True)
347         self.resetTesting()
348         self.model = self.model.eval()
349         self.bounding_boxes = BoundingBoxes()
350
351
352         for i_batch, sample_batched in enumerate(self.test_loader):
353             event, histogram = sample_batched
354
355             # Convert spatial dimension to model input size
356             histogram = torch.nn.functional.interpolate(histogram.permute(0, 3, 1, 2),
357                                              torch.Size(self.model_input_size))
358             histogram = histogram.permute(0, 2, 3, 1)
359
360             locations, features = self.denseToSparse(histogram)
361
362             with torch.no_grad():
363                 model_output = self.model([locations, features, histogram.shape[0]])
364                 detected_bbox = yolodetect(model_output, self.model_input_size.to(model_output.device),
365                                              threshold=0.7)
366                 detected_bbox = nonMaxSuppression(detected_bbox, iou=0.6)
367                 detected_bbox = detected_bbox.cpu().numpy()
368
369             batch_one_mask = locations[:, -1] == 0
370             vis_locations = locations[batch_one_mask, :2]
371             features = features[batch_one_mask, :]
372             vis_detected_bbox = detected_bbox[detected_bbox[:, 0] == 0, 1:-2].astype(np.int)
373
374             # Visualization self.batch_step
375             file_path = os.path.join(self.settings.vis_dir, 'image_' + str(self.test_batch_step) + '.png')
376             visualizations.visualizeLocations(vis_locations.cpu().int().numpy(), self.model_input_size,
377                                              features=features.cpu().numpy(), path_name=file_path,
378                                              bounding_box=vis_detected_bbox[:, :-1],
379                                              class_name=[self.object_classes[i]
380                                              for i in vis_detected_bbox[:, -1]])
381
382             self.pbar.update(1)
383             self.test_batch_step += 1
384
385             self.writer.add_image('Testing/Input Histogram', test_images, self.epoch_step, dataformats='NHWC')
386
387             self.pbar.close()

```

Figura 5.13: Función test en el archivo testing/tester\_sin\_estadisticas.py.

Aparte de este cambio, debemos hacer otra pequeña modificación, aunque es una cuestión de eficiencia, no de necesidad. Para qué queremos cargar la información de las anotaciones si no la vamos a utilizar. En la imagen 5.13 ya se puede apreciar que al iterar sobre los distintos archivos, en la línea 356, no obtenemos el bounding box que nos indican los autores, pero para que esto sea así y siga siendo funcional el código debemos de introducir un pequeño cambio en un nuevo archivo: dataloader/loader\_sin\_informacion.py. Este script es el que nos va a ir proporcionando los datos una vez han sido leídos. No hay grandes diferencias con respecto al loader.py que nos proporcionan los autores, de hecho simplemente tenemos que tocar levemente la función collate\_events() tal y como vemos en la figura 5.14.

<pre> 29 def collate_events(data): 30     labels = [] 31     events = [] 32     histograms = [] 33     for l, d in enumerate(data): 34         labels.append(d[0]) 35         histograms.append(d[1]) 36         ev = np.concatenate([d[0], l*np.ones([len(d[0]), 1], dtype=np.float32)], 1) 37         events.append(ev) 38     events = torch.from_numpy(np.concatenate(events, 0)) 39     labels = default_collate(labels) 40 41     histograms = default_collate(histograms) 42 43     return events, labels, histograms </pre>	<pre> 32 def collate_events(data): 33     events = [] 34     histograms = [] 35     for l, d in enumerate(data): 36         histograms.append(d[1]) 37         ev = np.concatenate([d[0], l*np.ones([len(d[0]), 1], dtype=np.float32)], 1) 38         events.append(ev) 39     events = torch.from_numpy(np.concatenate(events, 0)) 40 41     histograms = default_collate(histograms) 42 43     return events, histograms </pre>
---	---

(a) loader.py

(b) loader\_sin\_estadisticas.py

Figura 5.14: Diferencia en collate\_events en dataloader/loader.py y dataloader/loader\_sin\_estadisticas.py.

### 5.3.2. Inferencia sobre mis propios datos

Centrándonos ahora en el objetivo principal de este proyecto de fin de grado, inferir sobre nuestros propios datos de entrada, ajenos a la base de datos utilizada para entrenar el modelo, lo primero que hay que considerar es que la inferencia en sí se realiza exactamente de la misma forma que en el testing/tester\_sin\_informacion.py por lo que realmente no hay ninguna diferencia entre este archivo y el testing/tester\_mio2.py creado para realizar el propósito que en este párrafo se describe. Que no haya ninguna diferencia entre la función test() de un archivo y del otro no quiere decir que ya esté todo el trabajo hecho, al contrario, permitir el reconocimiento de objetos sobre eventos creados por nosotros mismos nos ha supuesto un verdadero dolor de cabeza.

En primer lugar, debíamos de sustituir la forma en que se crea el dataset. Puesto que a partir de ahora no vamos a necesitar conjuntos train ni validation, todos los archivos estarán destinados al test para realizar la inferencia sobre ellos. Para ello modificamos la función createDataset() del archivo dataloader/dataset.py creando un nuevo script llamado dataloader/dataset\_mio.py, dicha función la podemos ver en la figura 5.15. En ella vemos como nos da igual que conjunto estemos creando, añadimos a él todos los archivos. Esto no es problema al no tener los conjuntos de validación y entrenamiento ningún uso. Esta función no sería válida si queremos entrenar al modelo con nuestra propia base de datos, en este caso sí habría que dividir el total de archivos de eventos entre los tres conjuntos.

```

405     def createDataset(self):
406         """Does a stratified training, testing, validation split"""
407         np_random_state = np.random.RandomState(42)
408         # training_ratio = 0.7
409         testing_ratio = 1
410         # Validation Ratio will be 0.1
411
412         for i_class, object_class in enumerate(self.object_classes):
413
414             dir_path = os.path.join("data/mis_videos", object_class)
415             image_files = listdir(dir_path)
416             nr_samples = len(image_files)
417
418             random_permutation = np_random_state.permutation(nr_samples)
419             nr_samples_test = int(nr_samples*testing_ratio)
420
421             if self.mode == 'testing':
422                 start_idx = 0
423                 end_idx = start_idx + nr_samples_test
424             elif self.mode == 'training':
425                 start_idx = 0
426                 end_idx = start_idx + nr_samples_test
427             elif self.mode == 'validation':
428                 start_idx = 0
429                 end_idx = start_idx + nr_samples_test
430
431             for idx in random_permutation[start_idx:end_idx]:
432                 filepath = os.path.join("data/mis_videos", object_class, image_files[idx])
433                 self.files.append(filepath)
434                 self.class_labels.append(i_class)

```

Figura 5.15: Función createDataset() del archivo dataloader/dataset\_mio.py.

Asynet, tal y como está implementada por sus autores Davide Scara-

muzza y Antonio Loquercio entre otros, obtiene los eventos a través de un archivo binario. Este documento, según se indica, tiene el formato aedat. Sin embargo nos encontramos con dos problemas. El primero es que no se indica cuál de las cuatro versiones existentes de aedat requiere. Este problema no es demasiado importante puesto que bastaría con comprobar a base de ensayo y error que versión necesita, y una vez comprobado esto, existen mecanismos que nos permiten pasar de una versión a otra automáticamente. El verdadero problema es el segundo, y es que aunque v2e era capaz de proporcionar como salida los eventos en este formato, tuvimos que eliminarlo para permitir obtener una mayor resolución, y por tanto, un mayor número de eventos.

La única opción que nos queda es crear nosotros una nueva función capaz de leer los eventos, y la información requerida de éstos desde un formato que sí tengamos disponible en todo momento, y no dependa de la resolución establecida para la salida, el archivo v2e-dvs-events.txt. Para ello debíamos de sustituir la función loadEventsFile() del archivo dataloader/dataset\_mio.py. Esta función es la que podemos ver en la ilustración 5.16. En ella vamos a observar que solamente nos quedamos con los eventos producidos en los primeros 70 milisegundos del vídeo que transformamos mediante el simulador.

```

502     def loadEventsFile(self, file_name):
503
504         all_x = []
505         all_y = []
506         all_p = []
507         all_ts = []
508
509         intervalo = 0.07
510         inicio = 0.0|
511         with open(file_name, 'r') as f:
512
513             for i, line in enumerate(f.readlines()):
514
515                 if i >= 6: #las 6 primeras son cabecera
516
517                     data = line.split() #obtenemos los elementos separado.
518 #                     all_ts.append(np.uint32(float(data[0])*1000000))
519 #                     if float(data[0]) > 2.0 and float(data[0]) < 2.07:
520
521                         if i == 6:
522                             inicio = float(data[0])
523                         elif inicio + intervalo < float(data[0]):
524                             break
525
526                         all_ts.append(np.uint32(float(data[0])))
527                         all_x.append(np.uint32(data[1]))
528                         all_y.append(np.uint32(data[2]))
529                         all_p.append(np.uint32(data[3]))
530 #
531                         if float(data[0]) > 1:
532                             break
533
534                         all_x = np.uint32(all_x)
535                         all_y = np.uint32(all_y)
536                         all_p = np.uint32(all_p)
537                         all_ts = np.uint32(all_ts)
538
539                         all_p = all_p.astype(np.float64)
540                         all_p[all_p == 0] = -1
541
542         return np.column_stack((all_x, all_y, all_ts, all_p))

```

Figura 5.16: Función loadEventsFile() del archivo dataloader/dataset\_mio.py que nos permite leer los eventos a partir de un archivo de texto.

Solamente nos quedamos con un intervalo de siete centésimas de segundo porque asynet acumula todos los eventos que lee desde un mismo archivo. El problema si consideramos la totalidad de información que nos proporcionan archivos de gran tamaño es que estamos juntando eventos producidos en tiempos muy diferentes y que por lo tanto, es muy poco probable que estén relacionados entre sí. Esto nos obliga a establecer un marco temporal máximo de eventos a considerar, que en nuestro caso hemos decidido que sea de 70ms. En posteriores secciones, [7](#) veremos esto con más detalle.

A pesar de haber conseguido inferir sobre nuestros propios datos, obtenidos con ayuda del simulador v2e, cambiando la forma de leer los eventos tal y como se ha explicado, no terminábamos de estar del todo conformes con el hecho de que solamente se tuvieran en cuenta las primeras siete centésimas de segundo de cada vídeo. Es por ello que consideramos otra alternativa, mejor a nuestro juicio, ir dividiendo los datos de 70 ms en 70 ms para poder inferir correctamente y aprovechar así todos los eventos de cada archivo que tengamos.

Para manejar esta opción hemos creado un nuevo archivo dataloader/dataset\_mio\_video.py en el que hemos cambiado un poco algunos procedimientos. Para empezar se ha modificado la función createDataset(). En lugar de añadir cada archivo como se ve en la línea 433 de la imagen [5.15](#), llamamos a una nueva función implementada llamada aniadirFiles().

Esta función se puede ver en la ilustración [5.17](#) y está encargada de ir leyendo los eventos del archivo original, pasado por parámetro, y crear tantos archivos nuevos como cantidad de segmentos de siete centésimas de segundo podamos dividir estos eventos. Todos estos nuevos archivos contendrán la misma cabecera que el archivo original, y a continuación, la información de los eventos que entren en el intervalo de tiempo que le haya correspondido a cada uno. Finalmente devolverá el nombre que le ha dado a todos estos archivos porque éstos serán los que constituyan el dataset con el que vamos a trabajar.

Todos estos nuevos documentos que se han creado dinámicamente se alojarán en la misma ubicación que el archivo original a partir del cuál han sido producidos. Su nombre será también el mismo que el nombre del documento inicial, seguido de una barra baja(\_) y un número. Gracias a esta nomenclatura la función createDataset() sabe qué archivos tienen que formar parte del dataset y cuáles no. Por ello es muy importante que inicialmente los archivos que nosotros introduzcamos no contengan ningún número en el nombre.

```

505     def anadirFiles(self, filepath):
506         print(filepath)
507         archivosDevolver = []
508         numeroArchivo = 1
509         tiempoInicioActual = 0.0
510         intervalo = 0.07
511
512         tiempoTotal = 0.0
513         with open(filepath, 'r') as f:
514             for i, line in enumerate(f.readlines()):
515                 if i >= 6: #las 6 primeras son cabecera
516                     data = line.split() #obtenemos los elementos separados y los pasamos al tipo de dato
517                     tiempoTotal = float(data[0])
518
519         tiempoTotal = math.ceil(tiempoTotal)
520
521         archivos = []
522         for i in range(math.ceil(tiempoTotal/intervalo)):
523             archivo = []
524             archivos.append(archivo)
525
526         print('tiempo total: {}'.format(tiempoTotal))
527         print('tiempo total / intervalo: {}'.format(tiempoTotal/intervalo))
528         print('len archivos : {}'.format(len(archivos)))
529         cabecera = []
530         archivoActual = 0
531
532         with open(filepath, 'r') as f:
533             for i, line in enumerate(f.readlines()):
534                 if i < 6: #cabecera
535                     cabecera.append(line)
536
537                 if i >= 6: #las 6 primeras son cabecera
538
539                     data = line.split() #obtenemos los elementos separados y los pasamos al tipo de dato que necesitamos
540                     if float(data[0]) > tiempoInicioActual and float(data[0]) < (tiempoInicioActual+intervalo):
541                         archivos[archivoActual].append(line)
542                     elif float(data[0]) > (tiempoInicioActual+intervalo):
543                         archivoActual += 1
544                         tiempoInicioActual += intervalo
545                         archivos[archivoActual].append(line)
546
547         for numeroArchivo,archivo in enumerate(archivos, start=1):
548             if len(archivo) > 0:
549                 filepathParcial = filepath.split('.')
550                 num = str(numeroArchivo)
551                 if len(num) == 1:
552                     num = '0000' + num
553                 elif len(num) == 2:
554                     num = '000' + num
555                 elif len(num) == 3:
556                     num = '00' + num
557                 elif len(num) == 4:
558                     num = '0' + num
559                 filepath1 = filepathParcial[0] + '_' + num + '.txt'
560
561                 if os.path.exists(filepath1):
562                     os.remove(filepath1)
563                 fic = open(filepath1, 'w')
564                 fic.writelines('%s\n' % s for s in cabecera)
565                 fic.writelines('%s\n' % s for s in archivo)
566                 fic.close()
567                 archivosDevolver.append(filepath1)
568
569         archivosDevolver.sort()
570     return archivosDevolver

```

Figura 5.17: Función anadirFiles en el archivo dataloader/dataset\_mio\_vio.py.

De esta forma podemos inferir sobre cada vídeo al completo. En la función loadEventsFile() no ha sufrido grandes cambios con respecto a la versión anterior que se podía ver en la imagen 5.16. Al contener los archivos ahora eventos pertenecientes a un intervalo de tiempo que nosotros hemos considerado correcto, podemos leerlos todos, por lo que debemos eliminar las líneas [509,510] y [520,523], las que nos obligaban a detener la lectura al sobrepasar un margen de tiempo.

Una vez conseguido esto, nos pareció una idea interesante añadir la po-

sibilidad de crear un vídeo con todas estas imágenes que nuestro programa está generando, en las que se aprecian los eventos junto con el bounding box y la clase del objeto detectado. Así podríamos comparar el vídeo original con éste, en el que vamos viendo remarcados los objetos que aparecen en él.

Antes de intentar si quiera crear el vídeo con dichas imágenes, necesitábamos que éstas fueran producidas y guardadas en orden, cosa que no ocurre. Esto requiere eliminar el barajado que los autores hacen con el dataset. El motivo por el que se hace el shuffle es para que el conjunto train y validation no sea siempre el mismo en las distintas etapas del entrenamiento y validación respectivamente. En este caso, en el que solamente queremos inferir, no tiene ninguna influencia por lo que podemos eliminarlo tranquilamente. Este barajado se hace en el constructor de la clase Loader, del archivo dataloader/loader.py, y tras eliminarlo quedaría tal y como se ve en la figura 5.18.

```

10 class Loader:
11     def __init__(self, dataset, batch_size, num_workers, pin_memory, device, shuffle=False): #par
12         self.device = device
13
14         self.loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size,
15                                         num_workers=num_workers, pin_memory=pin_memory,
16                                         collate_fn=collate_events)

```

Figura 5.18: Constructor de la clase Loader en el que no se realiza un barajado del dataset.

Una vez conseguido que las imágenes generadas sigan el mismo orden que los fotogramas del vídeo original de partida, tan solo queda generar una película con todas ellas. Con este propósito hemos creado el script crear-Video.py basándonos en un código encontrado en [25]. Este script, que se puede ver en la imagen 5.19, carga todas las imágenes de un directorio dado, y con ayuda del modulo OpenCV genera un vídeo incluyendo tantas imágenes por segundo como le especifiquemos. En nuestro caso le hemos indicado que incluya 10 fps para que la velocidad con que aparecen las imágenes sea suficiente como para verlas con claridad pero no tan lenta como para que el vídeo sea largo y tedioso.

Para finalizar con esta sección, cabe mencionar que se han creado una serie de scripts para facilitar la ejecución de las distintas tareas implementadas. Estos scripts reciben un archivo de configuración como entrada y a partir de él realizan la función correspondiente. En el apartado 8 veremos con más detalle estos ejecutables y cuál debemos de utilizar en función de nuestras intenciones o intereses.

```

10 def convertToVideo(pathIn, pathOut, fps, time):
11     frame_array = []
12     files = [f for f in os.listdir(pathIn) if isfile(join(pathIn, f))]
13     files.sort()#REORDENA FRAMES
14     print(files)
15     for i in range(len(files)):
16         filename = pathIn+files[i]
17         print(filename)
18         img=cv2.imread(filename)
19         height, width, layers = img.shape
20         size = (width,height)
21
22 #         for k in range (time):
23 #             frame_array.append(img)
24         frame_array.append(img)
25
26     out = cv2.VideoWriter(pathOut, cv2.VideoWriter_fourcc(*'mp4v'), fps, size)
27     for i in range(len(frame_array)):
28         out.write(frame_array[i])
29     out.release()
30     print("TASK COMPLETED")
31
32 #EJECUTAMOS FUNCIÓN.
33
34 def crearVideo(filepath):
35     directory = filepath
36     fps = 10
37     time = int(len(os.listdir(filepath)) / 10)
38     pathIn = directory + '/'
39     pathOut=pathIn + 'video.avi'
40     convertToVideo(pathIn, pathOut, fps, time)

```

Figura 5.19: Script crearVideo.py basado en el código obtenido en [25].

## 5.4. Reconocimiento de objetos a partir de imágenes

Finalmente, a modo de comparación con un sistema que trabaje con imágenes, se ha hecho uso del detector YOLO en su quinta versión, la última de la que tenemos constancia. Los motivos por los que finalmente nos hemos decantado por utilizar esta CNN no son pocos, entre los que destaca su gran relación  $\frac{precision}{tiempoEjecucion}$ .

La instalación se ha llevado a cabo atendiendo a las instrucciones que aparecen en su repositorio de GitHub, [26], sin mayor dificultad. En esta dirección se nos aporta también un modelo preentrenado con la base de datos COCO. Sin embargo, esto no nos sirve puesto que no es correcto comparar los resultados que nos aportan dos redes neuronales si estas no han sido entrenadas con la misma base de datos. Por lo tanto, y puesto que para el entrenamiento de Asynet hemos utilizado N-Caltech101, nos hemos descargado la base de datos Caltech101 de [7].

Aunque hemos obtenido la base de datos, para poder completar el entrenamiento del modelo tenemos que realizar ciertas modificaciones en ella.

Estas alteraciones se indican en [97]. En primer lugar hemos tenido que crear un archivo de configuración yaml en el que indicar la ruta a la base de datos y a las imágenes que se van a utilizar para el entrenamiento y validación. A continuación, en este mismo archivo se tienen que especificar el número total de clases que hay, así como el nombre o etiqueta de cada una de estas categorías. El archivo de configuración creado se puede ver en la imagen 5.20.

```

1 # download command/URL (optional)
2 download: https://drive.google.com/u/1/uc?export=download&confirm=fszI&id=137RyRjvTBkBlFeYBNZBtVlDHQ6_Ewsp
3
4 # train and val data as 1) directory: path/images/, 2) file: path/images.txt, or 3) list: [path1/images/, path2/images/]
5 train: caltech101/images/train
6 val: caltech101/images/val
7
8 # number of classes
9 nc: 101
10
11 # class names
12 names: ['accordion', 'airplanes', 'anchor', 'ant', 'barrel', 'bass', 'beaver', 'binocular',
13 'bonsai', 'brain', 'brontosaurus', 'buddha', 'butterfly', 'camera', 'cannon', 'car_side',
14 'ceiling_fan', 'cellphone', 'chain', 'chandelier', 'cougar_body', 'cougar_face', 'crab',
15 'crayfish', 'crocodile', 'crocodile_head', 'cup', 'dalmatian', 'dollar_bill', 'dolphin',
16 'dragonfly', 'electric_guitar', 'elephant', 'emu', 'euphonium', 'ewer', 'Faces', 'Faces_easy',
17 'ferry', 'flamingo', 'flamingo_head', 'gaffield', 'gerenuk', 'lamophone', 'grand_piano',
18 'hawksbill', 'headphone', 'hedgehog', 'helicopter', 'ibis', 'inline_skate', 'joshua_tree',
19 'kangaroo', 'kite', 'lamp', 'laptop', 'Leopards', 'llama', 'lobster', 'lotus', 'mandolin',
20 'menorah', 'metronome', 'minaret', 'Motorbikes', 'nautilus', 'octopus', 'okapi',
21 'pagoda', 'panda', 'pigeon', 'pizza', 'platypus', 'pyramid', 'revolver', 'rhino', 'rooster',
22 'saxophone', 'schooner', 'scissors', 'scorpion', 'sea_horse', 'snopy', 'soccer_ball',
23 'stapler', 'starfish', 'stegosaurus', 'stop_sign', 'strawberry', 'sunflower', 'tick',
24 'trilobite', 'umbrella', 'watch', 'water_lilly', 'wheelchair', 'wild_cat', 'windsor_chair',
25 'wrench', 'yin_yang']

```

Figura 5.20: Archivo de configuración yaml para entrenar el modelo YOLOv5 con la base de datos Caltech101.

Una vez creado el archivo anterior, nos damos cuenta de que la forma en la que se encuentran las anotaciones de los bounding boxes y las clases, no coinciden entre como aparecen en la database descargada y como YOLO trabaja con ellas. Mientras que Caltech101 especifica las anotaciones proporcionándote las coordenadas de la esquina superior izquierda del bounding box, el ancho, y el alto que este presente, YOLO exige que la disposición de estas sea:

1. Un número entero representando la clase a la que pertenece. Esta numeración empieza en el 0 y se corresponde con el orden especificado en el archivo yaml.
2. La coordenada X del centro del bounding box.
3. La coordenada Y del centro del bounding box.
4. El ancho del bounding box.
5. El largo del bounding box.

Además para YOLO todas las coordenadas están comprendidas en el intervalo [0,1] siendo la esquina superior izquierda la (0,0) y la esquina inferior derecha la (1,1). Por cada objeto que aparezca en la imagen hay que escribir

una línea con toda esta información en las anotaciones. Cambiar todas las notas de los objetos imagen por imagen a mano puede ser una tarea muy tediosa e ineficiente, así que hemos implementado un script de python para automatizar el proceso. El código es el que vemos en la figura 5.21.

Finalmente, no podemos dejar estas anotaciones en cualquier sitio, sino que debemos ponerlas en una disposición concreta. Este paso es muy simple y solamente hay que crear las carpetas necesarias y disponer la base de datos tal y como nos indican los autores. Este paso también se hace en el script de la figura 5.21. En ella vemos que hemos dejado un 70 % de las imágenes para el train, un 20 % para el test, y un 10 % para la validación. El motivo por el que se han escogido estos porcentajes es para que la proporción de imágenes en cada grupo sea equivalente a la del modelo de Asynet.

Una vez modificada la base de datos de acuerdo a las especificaciones que se requieren para el entrenamiento y creado el archivo de configuración yaml, simplemente tenemos que utilizar los distintos scripts de python que se nos aportan y que nos van a permitir realizar tanto el entrenamiento y validación guardando los mejores pesos encontrados, como la inferencia sobre nuevas imágenes.

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Jun 20 00:15:52 2021
4
5 @author: Equipo
6 """
7
8 from scipy.io import loadmat
9 import os
10 import cv2 as cv
11
12 def parseDataBase(pathImage,pathAnnotation):
13
14     clasesImagenes = os.listdir(pathImage)
15     clasesAnotaciones = os.listdir(pathAnnotation)
16     clasesImagenes.sort()
17     clasesAnotaciones.sort()
18     print(clasesImagenes)
19     print(clasesAnotaciones)
20     numeroImagen = 0
21     numeroClase = -1
22     for i,j in zip(clasesImagenes,clasesAnotaciones):
23         numeroClase += 1
24         imagenes = os.listdir(os.path.join(pathImage,i))
25         anotaciones = os.listdir(os.path.join(pathAnnotation,j))
26
27         for z in range(len(imagenes)):
28
29             if z <= int(len(imagenes)*0.7):
30
31                 imagen = cv.imread(os.path.join(pathImage,i,imagenes[z]))
32
33                 alto, ancho, canales = imagen.shape
34                 coordenadas = loadmat(os.path.join(pathAnnotation,j,anotaciones[z]))["box_coord"]
35                 y,h,x,w = coordenadas[0]
36
37                 xCenter = ((x+w)/2) / ancho
38                 yCenter = ((y+h)/2) / alto
39                 width = w/ancho
40                 height = h/alto
41
42                 cv.imwrite(os.path.join('images','train','im'+str(numeroImagen)+'.jpg'),imagen)
43                 file = open(os.path.join('labels','train','im'+str(numeroImagen)+'.txt'), "w")
44                 file.write(str(numeroClase) + ' ' + str(xCenter) + ' ' + str(yCenter) + ' ' + str(width) + ' ' + str(height))
45                 file.close()
46                 numeroImagen += 1
47
48             elif z<= (int(len(imagenes)*0.7) + int(len(imagenes)*0.2)):
49
50                 imagen = cv.imread(os.path.join(pathImage,i,imagenes[z]))
51                 alto, ancho, canales = imagen.shape
52
53                 coordenadas = loadmat(os.path.join(pathAnnotation,j,anotaciones[z]))["box_coord"]
54                 y,h,x,w = coordenadas[0]
55
56                 xCenter = ((x+w)/2) / ancho
57                 yCenter = ((y+h)/2) / alto
58                 width = w/ancho
59                 height = h/alto
60
61                 cv.imwrite(os.path.join('images','test','im'+str(numeroImagen)+'.jpg'),imagen)
62                 file = open(os.path.join('labels','test','im'+str(numeroImagen)+'.txt'), "w")
63                 file.write(str(numeroClase) + ' ' + str(xCenter) + ' ' + str(yCenter) + ' ' + str(width) + ' ' + str(height))
64                 file.close()
65
66                 numeroImagen += 1
67
68             else:
69
70                 imagen = cv.imread(os.path.join(pathImage,i,imagenes[z]))
71                 alto, ancho, canales = imagen.shape
72
73                 coordenadas = loadmat(os.path.join(pathAnnotation,j,anotaciones[z]))["box_coord"]
74                 y,h,x,w = coordenadas[0]
75
76                 xCenter = ((x+w)/2) / ancho
77                 yCenter = ((y+h)/2) / alto
78                 width = w/ancho
79                 height = h/alto
80
81                 cv.imwrite(os.path.join('images','val','im'+str(numeroImagen)+'.jpg'),imagen)
82                 file = open(os.path.join('labels','val','im'+str(numeroImagen)+'.txt'), "w")
83                 file.write(str(numeroClase) + ' ' + str(xCenter) + ' ' + str(yCenter) + ' ' + str(width) + ' ' + str(height))
84                 file.close()
85                 numeroImagen += 1
86
87             # coordenadas = loadmat(filepath)["box_coord"]
88             # print(coordenadas)
89             # y,h,x,w = coordenadas[0]
90             # print('x: {} y:{} , finX:{} , finY:{}'.format(x,y,w,h))
91
92
93 pathAnnotation = 'Annotations'
94 pathImage = '101_ObjectCategories'
95 parseDataBase(pathImage, pathAnnotation)

```

Figura 5.21: Script implementado para poner las anotaciones de las imágenes de la base de datos Caltech101 en el formato requerido por YOLOv5.

# Capítulo 6

## Problemas durante el desarrollo

### 6.1. Restricciones sanitarias

Desde principios del año 2020 hasta la actualidad nos vemos afectados por una pandemia a nivel mundial. Esta situación provocada por el COVID-19 ha creado una serie de restricciones que han afectado a la sociedad en todos los ámbitos, y la realización de este trabajo no ha sido una excepción. Algunos de los inconvenientes más destacables motivados por esta plaga son los que se exponen a continuación:

- No han sido posibles tantas reuniones presenciales con el tutor del proyecto como hubiera sido conveniente, ya sea para aclaración de dudas o la discusión de puntos importantes del proyecto.
- No ha sido posible trabajar con la cámara basada en eventos de la que dispone el departamento, hecho por el cual hemos tenido que buscar una solución alternativa como ha sido trabajar con un simulador.

### 6.2. Restricciones hardware

- Necesidad de una tarjeta gráfica nvidia que sea compatible con CUDA para ciertas operaciones.
- Al utilizar una cámara basada en fotogramas y un simulador de eventos, en lugar de un sensor de visión dinámica, las condiciones de iluminación deben ser adecuadas.

Al no disponer de una tarjeta gráfica que cumpla las especificaciones requeridas, nos hemos visto obligados a utilizar un servidor propiedad del departamento. Este servidor no ha estado dedicado en exclusividad a mí, sino que han sido varios los alumnos y profesores que han tenido acceso a él a lo largo del semestre. Esto, junto con la gran carga computacional de los trabajos que estamos realizando todos, ha hecho que en algunos momentos el servidor esté inaccesible. Solamente una persona podía ejecutar su programa en cierto momento, en función de la carga que este necesitara. No obstante, como ya se ha comentado en algún momento en este documento, [2] a través de la plataforma Slack hemos podido mantener una comunicación fluida y organizar el tiempo en el que alguno necesitara el servidor en exclusividad.

### 6.3. Restricciones software

- El servidor utilizado trae instalado el sistema operativo Ubuntu 18.04 por lo que debemos trabajar sobre él.
- Debido al sistema operativo utilizado, la versión de ROS ha de ser Melodic.
- Necesitamos distintas versiones de Python para la ejecución completa del proyecto, en concreto hablamos de Python2.7 y Python3.8. Debido a esto se recomienda utilizar los entornos virtuales de Anaconda para aislar cada funcionalidad y no tener incompatibilidades.

### 6.4. Problemas técnicos

#### 6.4.1. Virtualización

Inicialmente se pensó realizar el proyecto en una máquina virtual mediante Oracle VirtualBox para aumentar la seguridad y portabilidad del sistema. Además, nos permite la creación de entornos en los que realizar pruebas sin que éstas afecten al resto de la infraestructura.

Sin embargo, al virtualizar nos encontramos con un gran inconveniente que ya se ha comentado en apartados anteriores [6.2], y es que necesitamos una tarjeta gráfica compatible con CUDA. Si ya de por sí, no podemos aprovechar toda la capacidad del anfitrión desde la máquina virtual, en el caso de la tarjeta gráfica con VirtualBox es imposible, el sistema operativo huésped no tiene acceso directo al hardware del anfitrión, sino que se realiza a través de una GPU virtualizada.

Este inconveniente hizo que abandonáramos por completo la idea y nos decidíramos por trabajar directamente sobre el anfitrión. No obstante mi

ordenador personal tampoco cuenta con una GPU con las especificaciones requeridas, por lo que finalmente, ante las limitaciones económicas para comprar una GPU válida, o alquilar un servidor, decidimos pedir prestado uno de los que son propiedad del departamento. Esto, aunque fue la mejor opción que podíamos abordar, requirió volver a replicar e instalar todo el trabajo realizado con antelación en el servidor.

#### 6.4.2. Simulador con limitado número de resoluciones

En este documento ya se han discutido tanto las ventajas como los inconvenientes que presentan los simuladores de eventos. A pesar de la importancia que presenta el uso del simulador v2e ya que sin él no hubiera sido posible la realización de este TFG, nos encontramos con un inconveniente de gran relevancia. Aunque en un principio da igual la resolución del vídeo de entrada basado en fotogramas, para el vídeo de salida solo existen una serie muy limitada de resoluciones disponibles y estas son muy bajas, 346x260 y 240x180 píxeles.

A este hecho no se le prestó demasiada importancia inicialmente. Es cierto que cuanta menor resolución, menor cantidad de eventos generados, sin embargo hasta etapas más avanzadas en el desarrollo no se descubrió que este factor fuera a tener tanta influencia en la detección de los objetos como se ha demostrado posteriormente.

Analizando el código se descubrió que el error no estaba tanto en la transformación del vídeo ni en la obtención de eventos, sino en la generación del archivo aedat. Este tipo de archivos es una de las salidas que presenta este simulador y nos ofrecen la información de los eventos en un formato estandarizado, en concreto v2e utiliza la segunda versión de aedat. Habiendo localizado el foco del error, se intentaron replicar las salidas válidas con nuevas resoluciones. Sin embargo, no tuvimos éxito y finalmente se optó por una solución válida a priori, pero que posteriormente iba a ocasionar nuevos problemas. Se eliminó el código que nos proporcionaba este archivo en la salida de tal forma que, despreciando el aedat, obtenemos todos los otros archivos y formatos generados sin problema.

#### 6.4.3. La base de datos NCars parseada no estaba disponible en el repositorio

En el repositorio del cual se ha obtenido el modelo utilizado para el reconocimiento de objetos ([24]) se hace referencia a distintas bases de datos disponibles para el entrenamiento de la red. Uno de los objetivos que nos propusimos al conocer este hecho es realizar el entrenamiento con las distintas bases de datos para comprobar con cuál se obtiene mejor precisión y

trabajar sobre ella. Sin embargo, al comenzar el entrenamiento con una de las cuatro bases de datos disponibles, NCars, ocurría un error en la lectura de datos.

Tras un tiempo sin encontrar el por qué de este error, descubrimos que uno de los issues hacía referencia a él. Al parecer los autores tuvieron que parsear la base de datos antes de poder utilizarla y se les olvidó subirla al repositorio. La solución obvia estaba clara, hacer nosotros los mismos cambios que los autores, sin embargo no dicen en ningún lado cuáles fueron las modificaciones realizadas y descubrirlas por nuestra cuenta podría conllevar más tiempo del deseado, por lo tanto optamos por dejar un comentario nuevo en el issue en cuestión pidiendo a los autores la base de datos parseada por correo, al igual que ya había hecho una persona antes que nosotros.

Así que decidimos continuar trabajando por otro lado a la espera de si obteníamos respuesta, y así fue. A los pocos días obtuvimos el enlace de descarga de la base de datos lista para trabajar sobre ella, y pudimos retomar el trabajo por donde lo dejamos.

#### 6.4.4. La lectura de eventos para la detección de objetos es mediante un archivo aedat

En la sección [6.4.2 Simulador con limitado número de resoluciones](#) explicamos que la solución adoptada ante el bajo número de resoluciones disponibles fue eliminar la salida que producía dicha limitación, el archivo aedat. Sin embargo, al seguir avanzando en el proyecto, nos encontramos con que el software utilizado para la detección y reconocimiento de objetos lee los eventos precisamente desde este tipo de archivos.

Este problema nos lleva a que quizá deberíamos de conseguir este formato en el simulador de eventos, o quizá implementar un nuevo script que nos lo genere por nuestra cuenta. Pero antes de esto, decidimos probar con la resolución 240x180 que sí podíamos obtenerla directamente con v2e y ver si la lectura de datos en el reconocimiento de objetos era correcta. No lo fue.

Al principio no entendíamos por qué esto era así ya que en la documentación se indicaba claramente que la lectura se realizaba a partir de archivos aedat. La única conclusión razonable que llegamos a encontrar era que, siendo cierto que se requería el formato dicho anteriormente, se exige una versión distinta a la disponible en ese momento. En concreto existen cuatro versiones principales, aedat, aedatv2, aedatv3 y aedatv4, aunque hay algunas subclases. Sabemos que v2e nos ofrece la segunda, pero no tenemos ninguna información acerca de la requerida por el otro sistema.

Viendo la disposición de la información en los archivos de las distintas bases de datos, que si se podían leer correctamente, tampoco conseguimos

obtener ningún resultado. Así que, finalmente, decidimos implementar nuevas funciones que nos permitieran leer los datos de los eventos requeridos a partir de uno de los archivos que el simulador siempre nos ofrece como salida, los eventos en formato txt.

#### 6.4.5. Mezcla de eventos no relacionados entre sí

Asynet lee los eventos almacenados en un archivo y los acumula todos ellos antes de clasificar o detectar objetos con esa información. Esto es válido cuando los datos almacenados son cercanos en el tiempo. Sin embargo, si almacenamos en un mismo archivo todos los eventos generados en un intervalo temporal amplio, estaremos entrelazando información independiente entre sí y que no tiene por qué estar relacionada. Cuando esto ocurre el modelo no es capaz de realizar su predicción correctamente como es esperable.

La solución parece clara, que los archivos que contengan información de eventos estén limitados a cierto intervalo de tiempo. En caso de disponer del sensor físicamente podría incluirse un nodo ROS dedicado a la escritura de los eventos, y que cada X unidad de tiempo cierre el archivo actual y empiece la escritura en uno nuevo.

Sin embargo no tenemos la opción de utilizar la cámara de eventos, sino que estamos utilizando el simulador v2e y éste nos ofrece todos los eventos de un mismo vídeo en un mismo documento de texto. Por ello lo que se ha decidido es que sea el propio programa de Asynet el que en el momento en que está construyendo el dataset con el que trabajar, divida la información de estos archivos en tantos documentos como sea necesario, cada uno centrado en un marco temporal corto y único. De esta manera podrá leer los eventos de cada archivo y acumularlos tranquilamente, ya que al ser la diferencia temporal entre el primero y el último evento reducida, podemos suponer que todos los eventos contenidos en ese mismo archivo estarán relacionados entre sí, y representarán una misma escena desde una misma posición focal.

#### 6.4.6. Requerimiento de un elevado espacio libre en disco

Ya se ha explicado en secciones anteriores que uno de los métodos implementados nos divide todos los eventos de un archivo dado en distintos documentos nuevos, cada uno con la misma cabecera y con los eventos pertenecientes a un intervalo temporal de siete centésimas de segundo. Esto presenta un gran inconveniente, la gran cantidad de espacio en memoria que requiere.

La simulación de un vídeo de 20 segundos de duración, estableciendo una resolución de salida de 854x480 píxeles, genera un archivo de eventos que ocupa alrededor de 500MB. Si pensamos que entre todos los documen-

tos generados a partir de los eventos que este contiene van a ocupar como mínimo otros 500MB, realmente es algo más porque la cabecera la estamos arrastrando en todos y cada uno de los archivos, y que en muchas ocasiones realizaremos la inferencia sobre varios archivos en la misma ejecución, nos damos realmente cuenta de que vamos a necesitar un gran espacio libre en memoria para que el programa pueda ejecutarse correctamente.

Ante este inconveniente la única solución es ir realizando ejecuciones con una cantidad limitada de archivos e ir liberando espacio según van finalizando los programas. Así dejamos suficiente memoria disponible para futuras ejecuciones con otros datos.

## 6.5. Otros problemas

- **Lugar de residencia:** debido a que vivo en un pueblo pequeño, con conexión a internet inestable en los días de lluvia y viento, he tenido algunos problemas de conexión a internet al principio del trabajo, lo que me impedía tanto acceder al servidor como obtener nuevos papers e información.
- **Transición desde el adsl a la fibra óptica:** debido a que se ha realizado el paso de adsl a fibra óptica esta primavera en mi zona, he estado unos tres o cuatro días sin conexión a internet, ocasionando los mismos inconvenientes que en el caso anterior. Por suerte, a raíz de esta nueva instalación se solucionó el problema anterior.

## 6.6. Resumen de problemas y soluciones abordadas

Problema encontrado	Fuente del problema	Solución óptima o ideal	Solución adoptada
Imposibilidad de reuniones presenciales con el tutor académico	Situación de COVID-19	Vacunación y reuniones minimizando los riesgos	Reuniones online y comunicación fluida mediante otras plataformas
Imposibilidad de utilizar el sensor	Situación de COVID-19 y no tener suficientes cámaras para que cada alumno tenga una	Adquirir sensores suficientes como para proporcionar uno a cada alumno	Utilización de un simulador
Necesidad de tarjeta gráfica compatible con CUDA	Software Asynet	Adquirir un PC con gráfica Nvidia compatible	Préstamo de un servidor por parte del departamento
Simulador con limitado número de resoluciones	Generación del archivo aedat	Programar el resto de resoluciones necesarias	Omitir la generación de este archivo
No funciona el entrenamiento con la base de datos NCARS	No esta disponible la base de datos parseada	Parsearla nosotros mismos de la misma manera	Solicitud de la base de datos parseada a los autores
Lectura de los eventos errónea	Se debe hacer desde un archivo aedat y no disponemos de él	Adquirir el archivo aedat por medio del simulador	Implementación de una nueva función para leer los eventos desde otro tipo de formato disponible
Mezcla de eventos no relacionados entre sí	Un archivo demasiado grande puede contener eventos no relacionados entre sí y Asynet combina todos los eventos que estén en un mismo archivo	El simulador v2e debe dividir la información en archivos enfocados en cortos margenes de tiempo	Asynet, mientras construye el dataset, se encarga de dividir la información de un archivo en documentos diferentes, cada uno centrado en un intervalo pequeño de tiempo

Cuadro 6.1: Principales problemas encontrados durante el desarrollo del proyecto y acciones realizadas para solucionarlos



# Capítulo 7

## Experimentación

### 7.1. Métricas utilizadas

Para evaluar las prestaciones de los distintos modelos entrenados y pruebas realizadas tenemos las siguientes medidas de bondad:

- **Función de pérdida (loss function):** mide cómo de lejos está un valor estimado del real. En aprendizaje automático buscamos minimizar el valor de esta función. Aunque existen innumerables funciones de pérdida diferentes, cuando nosotros nos refiramos a este término estaremos haciendo alusión a dos de ellas, crossentropyloss del módulo torch.nn ([\[57\]](#)) para clasificación y yololoss ([\[128\]](#)) para detección de objetos. La primera de ellas combina una variación del softmax, función muy utilizada en aprendizaje automático, logSoftmax ([\[58\]](#)) para reducir su inestabilidad numérica con NLLLoss, Negative Log Likelihood Loss ([\[59\]](#)), dos funciones muy útiles para problemas de clasificación con varias clases. La segunda penaliza especialmente la mala localización del centro de los bounding boxes y aquellos que son más pequeños sobre los mayores.
- **Precisión (accuracy):** porcentaje de acierto en las predicciones. Se calcula dividiendo el número de aciertos entre el total de predicciones realizadas.
- **Mean Average Precision (mAP):** definiremos la precisión como el porcentaje de verdaderos positivos que hemos predicho, es decir, de entre todos los que hemos identificado como positivos, cuántos lo son realmente. Por otro lado, el recall es el porcentaje que representan los verdaderos positivos identificados entre todos los posibles positivos. Sabiendo esto, el área bajo la curva de la gráfica que relaciona la precisión en el eje ‘y’, y el recall en el eje ‘x’ para una clase determina-

da, se llama Average Precision o AP. Finalmente denotaremos como mAP, Mean Average Precision, a la media del AP de todas las clases. La implementación utilizada es la de [133].

## 7.2. Pruebas del simulador v2e

Nuestro proyecto no se centra en la obtención de eventos a partir de vídeos, sino que es una simple herramienta que nos vemos obligados a utilizar al no disponer de la cámara de eventos. Es por ello que no consideramos adecuado invertir más tiempo del estrictamente necesario en la realización de pruebas y experimentos con este sistema.

Ya se explicó en apartados anteriores que ignoramos el formato aedat2 para poder obtener distintas resoluciones de salida. Pero, ¿qué es lo que hemos ganado al aumentar esta resolución? En la imagen 7.1 podemos ver una captura de la salida que nos proporciona el nodo roslaunch de ROS leyendo los eventos producidos por v2e con un mismo vídeo de entrada y distintas resoluciones de salida.

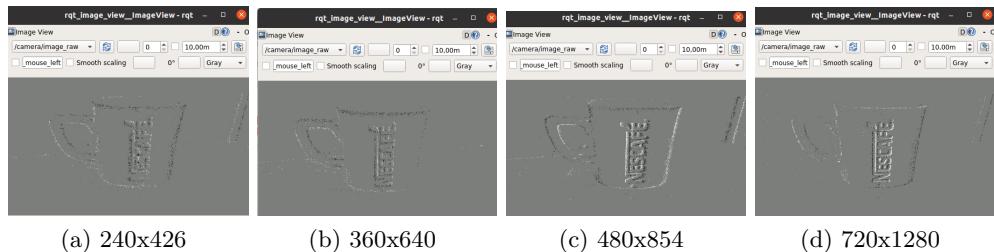


Figura 7.1: Captura del nodo roslaunch mostrando los eventos que nos ha producido v2e con distintas resoluciones de salida.

También se ha aumentado la calidad más aún, probando con 1920x1080 píxeles. Sin embargo, tanto con esta resolución, como con mayores, agotamos la memoria de la GPU del servidor y no podíamos realizar la conversión completa. Con respecto a las que sí hemos podido probar con los recursos que teníamos a nuestro alcance, podemos apreciar como, aunque mínima entre etapas consecutivas, excepto en el último paso, hay una diferencia de calidad. Esta diferencia no hay que apreciarla en la intensidad en que se visualiza la taza, puesto que las capturas son de instantes de tiempo distintos. El aumento de calidad lo apreciamos en la definición de los bordes de los objetos.

Aunque estas capturas que apreciamos en la ilustración 7.1 nos permiten observar gráficamente la diferencia en la generación de eventos en función de la resolución de salida, simplemente estamos viendo imágenes puntuales,

capturas de pantalla, no tenemos ningún dato objetivo que justifique el aumento o decremento de eventos producidos en función de la resolución. Para ello hemos extraído distintas características de la conversión, como son la cantidad de eventos generados, el tamaño del archivo txt que contiene estos eventos, y el tiempo que hemos tardado en conseguirlos. Toda esta información se ha dispuesto en la tabla 7.1.

característica \ resolución	240x426	360x640	480x854	720x1280
<b>Eventos generados</b>	892234	2288598	4415983	10163323
<b>Tamaño del archivo (MB)</b>	25.3	66	127.5	295
<b>Tiempo requerido (s)</b>	299	322	338	408

Cuadro 7.1: Comparación de distintas características de las salidas producidas por v2e en función de la resolución.

No sólo se ha presentado esta información en forma de tabla, sino que en la figura 7.2 podemos ver los mismos datos en una representación gráfica que nos permite apreciar las diferencias cómodamente con un simple vistazo rápido. Tanto la cantidad de eventos generados como, consecuentemente, el tamaño de los archivos de salida con la información de estos eventos, aumentan exponencialmente con la resolución de salida. También vemos como el tiempo necesario para las tres primeras resoluciones es muy parecido, mientras que el primer salto considerable se produce al llegar a 720x1280 píxeles.

Es importante tener en cuenta estos datos de cara a la utilización del simulador en nuestro sistema. Aunque a priori lo más adecuado e idóneo es utilizar un sensor de visión dinámica directamente, si por temas de presupuesto no nos es posible, o simplemente nos interesa trabajar con un simulador, debemos tener muy presente esta información de cara a utilizar una resolución u otra en función de la memoria disponible, el tiempo de respuesta máximo permitido, etc.

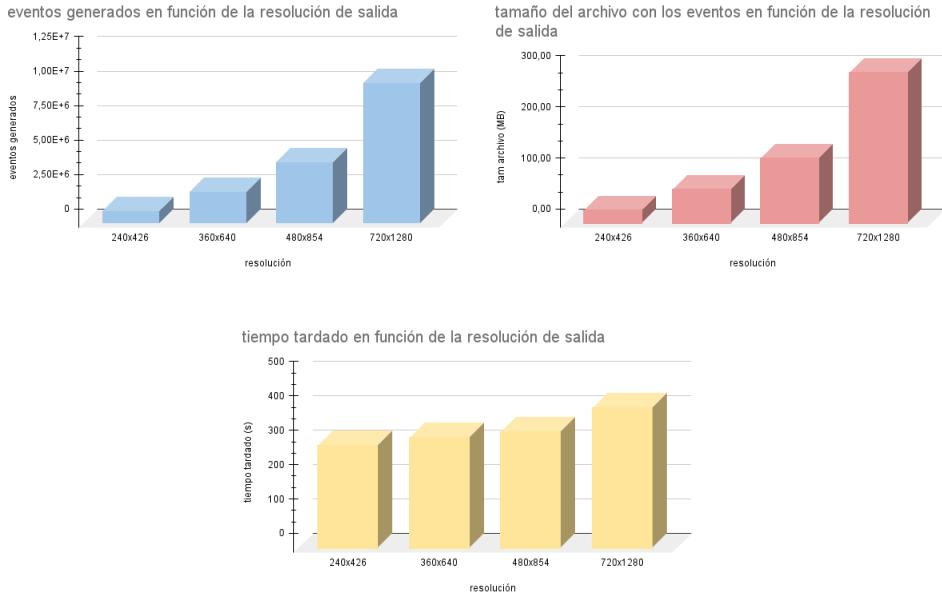


Figura 7.2: Gráficas con la comparación de distintas características de las salidas producidas por v2e en función de la resolución.

### 7.3. Pruebas de segmentación implícita

Como ya se ha comentado a lo largo de esta memoria en alguna ocasión, no se pretende realizar una segmentación de objetos como tal, en la que se clasifica y se pone una etiqueta a cada uno de los píxeles que componen una imagen, sino que se pretenden aprovechar las características intrínsecas de las cámaras de eventos. Esto es, ante una escena estática en la que solo se mueven los distintos objetos que haya, manteniendo el sensor también sin movimiento, la cámara proporcionará eventos únicamente de los bordes de los objetos que estén en movimiento. Estos eventos ya formarían una segmentación (cerrando las fronteras) del objeto de forma automática.

Para probar esto, se han grabado una serie de vídeos en los que, como se ha dicho, filmamos una escena estática con una cámara fija, en la que aparecen varios objetos en movimiento. Al obtener los eventos de estos vídeos mediante el simulador v2e y visualizarlos con ROS y la ayuda del paquete dvs\_ros podemos observar esta segmentación descrita. En la figura 7.3 vemos capturas de pantalla del vídeo original y de la visualización que ROS nos ofrece de los eventos generados en las que apreciamos perfectamente a qué nos referimos cuando hablamos de esta segmentación implícita.

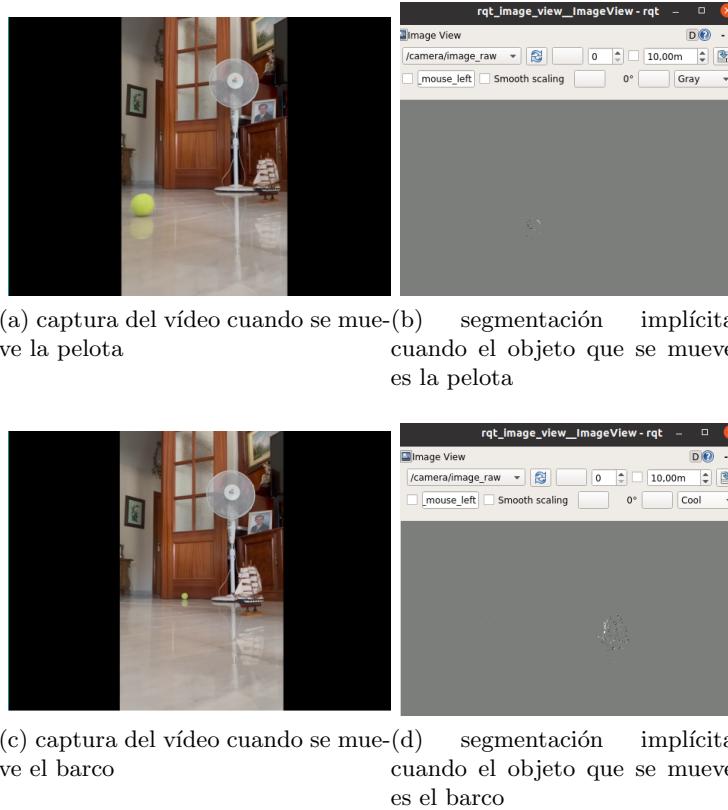


Figura 7.3: Capturas de pantalla del vídeo a partir del cual se han sacado los eventos y se ha realizado la segmentación implícita.

Por lo tanto estamos realizando una segmentación “natural” debido al tipo de sensor utilizado y sus propiedades. Se podría decir que estamos realizando una segmentación temprana o pseudo-segmentación, no una segmentación completa, ante las condiciones que aquí se describen (escena y cámara estáticas).

## 7.4. Pruebas de detección de objetos basada en eventos

### 7.4.1. Selección de la base de datos para el entrenamiento

Uno de los primeros, y más importantes, procesos que tenemos, ya vayamos a realizar clasificación, detección de objetos, segmentación, o cualquier otro propósito, es seleccionar una base de datos adecuada con la que hacer el entrenamiento. Ya se ha comentado que nosotros teníamos cu-

tro conjuntos de datos distintos a nuestra disposición (N-Caltech101, N-Caltech101\_object\_detection, NCars y Prophesee Gen1 Automotive), aunque éstas estaban relacionadas dos a dos, es decir, teníamos una base de datos para clasificación y su equivalente para la detección de objetos, y una segunda para clasificación y, de nuevo, su igual para el reconocimiento de objetos.

A continuación se describen los distintos experimentos realizados y relacionados con la implementación de una detección de objetos a partir de eventos eficaz y eficiente. Comprobaremos experimentalmente qué conjunto de datos, de los disponibles, nos puede ofrecer mejores resultados, el número óptimo de etapas para el entrenamiento del modelo, cuáles son los mejores valores posibles para los distintos parámetros, y comprobaremos también los resultados que obtenidos al inferir sobre archivos de eventos que nosotros mismos generemos con ayuda del simulador v2e.

Antes de comenzar, cabe mencionar que, siempre y cuando no se indique expresamente un cambio, se han empleado los valores que vienen por defecto al descargar el proyecto asynet desde su repositorio ([\[24\]](#)) para los distintos parámetros. Esto es, un threshold y un IoU de 0.3 y 0.6 respectivamente, posteriormente se explica cada uno de ellos con mayor detenimiento. También se ha respetado la proporción de datos que especifican los autores al dividir el total de la base de datos en los distintos subconjuntos a utilizar. Estos son train, validation y test con una de las distribuciones más extendidas, 70 %, 10 % y 20 % de los datos para cada uno respectivamente. Del mismo modo, se mantiene el hecho de no realizar data augmentation, es decir, no se manipulan las imágenes para añadir más diversidad.

Cada uno de estos subconjuntos en los que se divide la base de datos original tiene una función concreta. Mientras que los dos primeros van a utilizarse durante el entrenamiento, el tercero y último es sobre el que vamos a realizar la posterior inferencia. El subconjunto train es el que realmente se utiliza para entrenar el modelo, actualizar los pesos de las neuronas, el modelo ve y aprende de estos datos. Por otro lado, el subconjunto val o validation sirve para ajustar los hiperparámetros del modelo. Cada cierto número de etapas en las que usamos el conjunto anterior, se realiza una de validación. El modelo ve estos datos cada cierto tiempo, pero nunca aprende de ellos.

El subconjunto de validación será necesario también para prevenir el overfitting o sobreajuste del modelo a los datos de entrenamiento. El sobreajuste ocurre cuando durante el proceso de entrenamiento el modelo aprende los detalles y el ruido de los datos pertenecientes al subconjunto train, de tal forma que esto tiene un impacto negativo en la capacidad del modelo para generalizar y, por tanto, predecir sobre futuros datos de entrada. Cuando ocurre este overfitting el valor de la función objetivo sigue mejorando a me-

dida que se procesan datos del entrenamiento, pero empeora ante nuevas muestras ajenas a este conjunto.

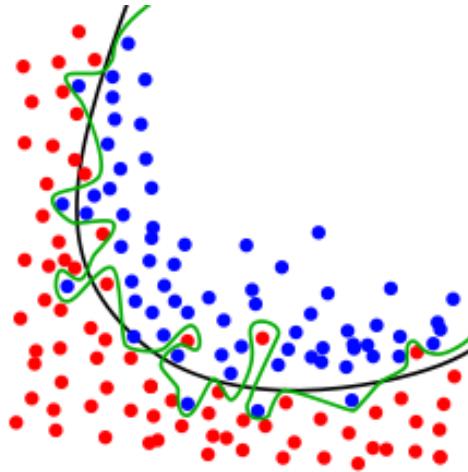


Figura 7.4: Muestra del overfitting sufrido por un clasificador binario. El objetivo es separar las muestras rojas de las azules y todas ellas representan el subconjunto train. Mientras que la línea negra representa la función de predicción ideal, la curva verde clara representaría la función aprendida con el sobreajuste. Fuente: [\[56\]](#)

### Entrenamiento con las distintas bases de datos para clasificación

Se ha entrenado el modelo FBSparsseCNN un total de 100 épocas con N-Cars y N-Caltech101 para clasificación. De esta forma se puede comparar los resultados que somos capaces de obtener en ambas. Las distintas gráficas resultantes tras la finalización de los distintos entrenamientos realizados las podemos ver en la figura 7.5.

En estas gráficas mostradas en 7.5 podemos ver la función de pérdida y la precisión obtenida, tanto sobre el conjunto utilizado para entrenar, como en el subconjunto de validación. Hay varios aspectos que nos llaman la atención a primera vista. En primer lugar la gran precisión que obtenemos en el entrenamiento, con ambas bases de datos observamos valores muy cercanos a 1 al final de las cien épocas. Aunque podemos pensar que esto es algo positivo, puede ser un indicio de sobreentrenamiento y overfitting. De forma equivalente, llama la atención el valor final de la función de pérdida, prácticamente 0. Sin embargo, nos resulta todavía más curioso que con NCars desde las primeras etapas, a partir de la décima aproximadamente, tenemos ya unos valores inferiores a 0.1.

Al fijarnos en los subconjuntos de validación de ambos datasets vemos como las líneas ya son más irregulares, presentan muchos más máximos y

mínimos, y con mayor diferencia entre picos consecutivos. Aunque esperado al tener tan solo dos clases distintas, nos sigue resultando bastante llamativo que con la segunda base de datos hayamos conseguido mantener los buenos datos obtenidos con el conjunto train. Aunque como ya hemos dicho, esta base de datos solamente tiene dos clases, cars(coches) y background(fondo) en función de si aparecen coches o no en la imagen. Es más corriente el efecto observable en N-Caltech, mantener la tendencia según van aumentando las épocas, pero quedándonos lejos del entrenamiento, teniendo en cuenta que solo hemos realizado 100 etapas.

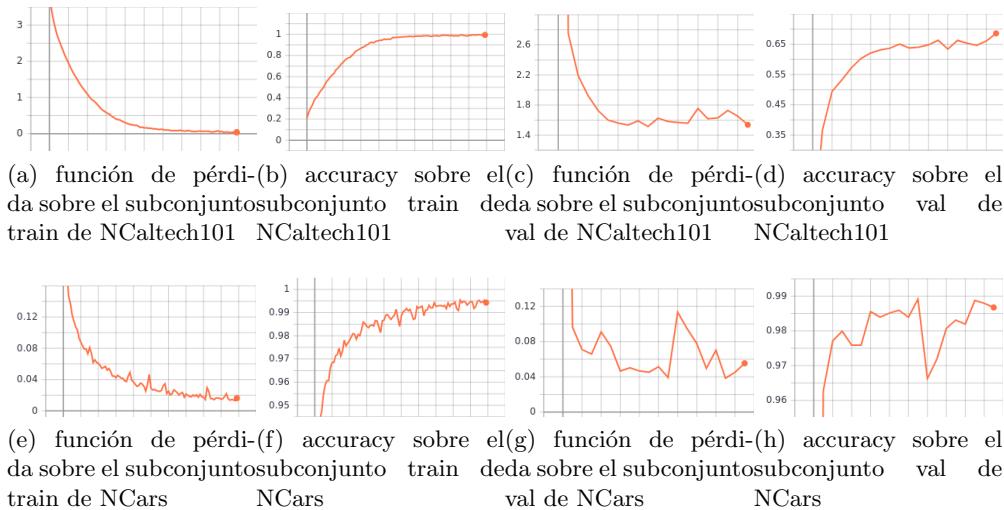


Figura 7.5: Gráficas con las métricas obtenidas por el modelo durante el entrenamiento y la validación de las primeras 100 épocas con las distintas bases de datos para clasificación de imágenes.

Finalmente, destacamos que la base de datos NCars presenta mejores valores que NCaltech101 en la precisión y en la función de pérdida. Además, aunque a simple vista podemos llegar a pensar lo contrario debido a que las gráficas están definidas en rangos distintos, NCaltech101 es más irregular. Todo esto motivado por la cantidad de clases de los conjuntos de datos. Sin embargo, esta última base de datos, precisamente por tener esa mayor cantidad de categorías distintas, y observando la curva y la tendencia de las gráficas, es la que suponemos que en caso de aumentar el número de épocas de entrenamiento tenderá a mejorar mucho más sus resultados que el otro conjunto, NCars.

### Entrenamiento con las distintas bases de datos para reconocimiento de objetos

Repetimos el mismo experimento, entrenando con los datasets centrados en la detección y reconocimiento de objetos, que en el fondo es la tarea que a nosotros más nos preocupa. En sus gráficas, que podemos ver en la figura 7.6, observamos a simple vista como la pérdida en el subconjunto train, a pesar de mantener una tendencia descendente, acabando con valores cercanos a 0, es mucho más irregular en ambas bases de datos que cuando la tarea a realizar era clasificación de imágenes. Observamos mucho ruido y oscilaciones. Sin embargo, los resultados realmente sorprendentes son los que encontramos en el conjunto de validación.

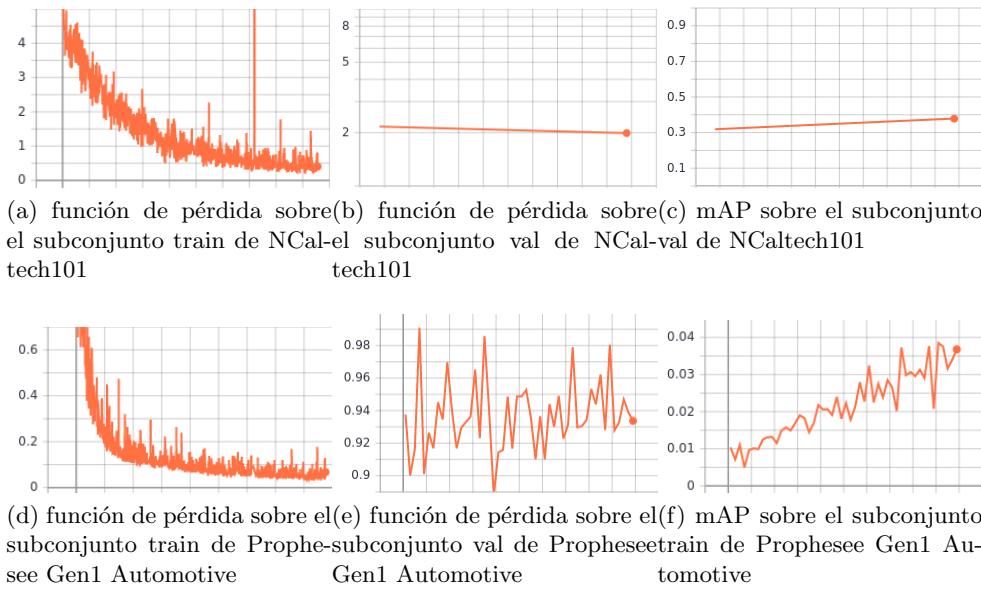


Figura 7.6: Gráficas con las métricas obtenidas por el modelo durante el entrenamiento y la validación de las primeras 100 épocas con las distintas bases de datos para detección de objetos.

En las gráficas referidas a esta categoría observamos comportamientos muy diferentes en función de la base de datos utilizada. La primera diferencia apreciable es que mientras Prophesee tiene mucha oscilación y picos, es muy irregular, N-Caltech101 es muy suave, prácticamente una línea recta. Este comportamiento se ve claramente reflejado en la función de pérdida. Es cierto que presenta menores valores el loss sobre Prophesee, sin embargo, está todo el rato oscilando y no se estabiliza en ningún momento, de hecho, tiene una tendencia ascendente. Por otra parte, el mean average precision ya no es tan irregular en Prophesee y sí tiene una buena tendencia, pero

vemos que los valores van del 0.01 al 0.04, siendo el rango de la función [0,1] y lo idóneo un valor de 1. Esto nos dice que el modelo no está siendo capaz de predecir nada bien dónde se encuentran los coches, en este caso. Sobre N-Caltech101 vemos todo lo contrario, el loss es más alto pero nada irregular y mantiene una buena tendencia, y el mAP sigue sin ser irregular y manteniendo la tendencia que deseamos, y además, aunque no podemos decir que sean buenos valores, éstos son diez veces mejor que los del dataset anterior.

#### 7.4.2. Entrenamiento del modelo

Para la realización del resto de experimentos realizados en este trabajo nos hemos decantado por la utilización de N-Caltech101 debido principalmente a todo lo comentado en apartados anteriores y a su versatilidad, no está enfocada solamente a la conducción y sus entornos, con dos posibles clases únicamente, como pasa con N-Cars y Prophesee Gen1 Automotive.

A continuación, centrándonos en el conjunto de datos N-Caltech101, entrenaremos de nuevo al modelo con una mayor cantidad de épocas y, una vez completado este proceso, realizaremos una serie de experimentos centrados en la inferencia tanto sobre el subconjunto test del dataset como sobre datos obtenidos por nosotros mismos con ayuda del simulador v2e.

#### Entrenamiento previo sobre N-Caltech101 para clasificación

Motivados por la naturaleza de la base de datos Caltech101, sobre la que está basada N-Caltech101, en la que solamente aparece un único objeto por imagen, y éste suele hallarse centrado en la escena, y ante la posibilidad, debido a que disponemos de N-Caltech101 para clasificación y detección, es buena práctica preentrenar el modelo para identificar qué se representa en la imagen, y posteriormente usar esos pesos para seguir entrenando, ya no sólo para saber qué representa la escena, sino dónde está ese objeto (recordamos que en la base de datos sobre la que estamos trabajando ahora mismo solamente contiene un objeto por archivo de datos). Estas dos tareas conjuntas van a constituir el reconocimiento de objetos. Esto es realizar transferencia de aprendizaje, comúnmente conocido como transfer learning por su nombre en inglés. Este proceso consiste en aprender, y almacenar conocimiento en la resolución de un problema, para posteriormente utilizar este saber adquirido para resolver un problema relacionado, pero distinto.

Entrenamos el modelo para clasificación un total de 1000 épocas, obteniendo las métricas y estadísticas que podemos observar en la ilustración 7.7. El motivo por el que nos hemos decidido por realizar un entrenamiento con este número de etapas, es porque los propios autores de asynet dicen

en el paper [I30] que en los experimentos que ellos mismos han realizado han conseguido los mejores porcentajes de acierto a partir de esta cantidad, sobre todo en la 1500, pero a partir de la 1000 los resultados ya se podían considerar buenos.

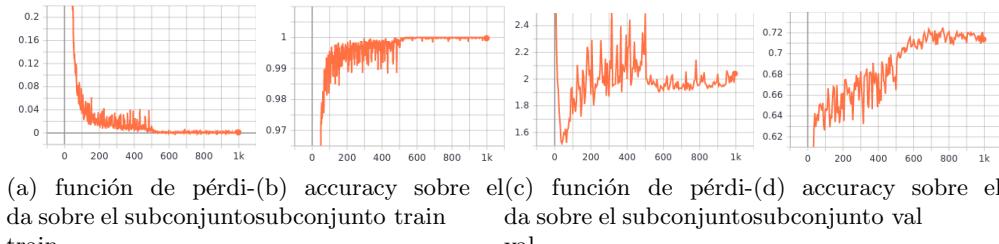


Figura 7.7: Gráficas con las métricas obtenidas por el modelo en el entrenamiento y la validación durante 1000 etapas con la base de datos N-Caltech101 para clasificación.

Con un entrenamiento de 1000 épocas para clasificación con N-Caltech101 ya vemos en la figura 7.7 como es más que suficiente para obtener unos valores de pérdida y precisión casi ideales (0 y 1 respectivamente) en el train. Sin embargo, no podemos afirmar lo mismo del conjunto de validación, que es en el que más nos deberíamos de fijar. En él, aunque a raíz de la mitad del proceso conseguimos estabilizar un poco las oscilaciones de la función de pérdida, no conseguimos tener una tendencia descendente durante todo el proceso ni bajar de 1.9. Pero también es cierto que en lo referente a la precisión, el accuracy, conseguimos llegar a un valor superior a 0.71 manteniendo una clara tendencia ascendente y controlando las oscilaciones, nuevamente a partir de las etapas centrales.

Si nuestro objetivo final fuera clasificar imágenes, está claro que esto no nos sirve, tendríamos que intentar cambiar algo y entrenar durante más etapas. Sin embargo, la motivación para hacer esta prueba no era más que el no partir de cero en el entrenamiento para el reconocimiento de objetos, tener unos pesos al comienzo que no fueran aleatorios y que nos puedan servir de utilidad. Al ser esto así, podemos aceptar los resultados obtenidos.

### Entrenamiento para reconocimiento de objetos de 1000 épocas sobre N-Caltech101

A partir de los pesos obtenidos en la clasificación entrenamos el modelo para el reconocimiento de objetos. En este caso se ha decidido por un entrenamiento de nuevo de 1000 épocas, cuyas métricas y estadísticas generadas podemos ver en la imagen 7.8. En las gráficas que se aprecian podemos ob-

servar como la pérdida en el entrenamiento, a pesar de presentar oscilaciones mantiene una clara tendencia a reducir su valor y consigue alcanzar valores cercanos a 0. Este comportamiento en el entrenamiento es el que vamos a observar en todas las gráficas que se muestren a partir de aquí en esta sección 7.4.2 por lo que no lo comentaremos más, nos quedamos con que presenta la tendencia deseada (descendente) y valores muy cercanos a 0 al final del entrenamiento, lo cual es bueno también.

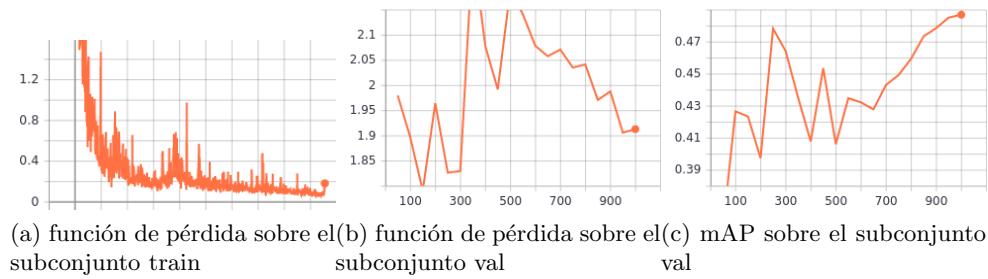


Figura 7.8: Gráficas con las métricas obtenidas por el modelo en el entrenamiento y la validación durante 1000 etapas con la base de datos N-Caltech101 para reconocimiento de objetos.

Completamente distintas son las figuras 7.8b y 7.8c que se refieren a la validación. Observándolas vemos como no ha sido hasta pasadas la mitad de las iteraciones que hemos empezado a estabilizar las curvas y a obtener un comportamiento más natural. De hecho, da la sensación de que si hubiéramos seguido entrenando, si hubiésemos establecido un mayor número de etapas, los pesos obtenidos hubieran sido mejores.

Por ello, hemos utilizado estos últimos pesos para entrenar otras 500 épocas adicionales a partir de los pesos que mejor resultados nos han proporcionado en el entrenamiento anterior, y a su vez éstos para otras 500 más. En las gráficas que hay en la imagen 7.9 podemos apreciar las métricas observables en estos dos últimos modelos que se han comentado.

Si bien es cierto que los resultados obtenidos son mejores que los resultantes tras solamente 1000 épocas de entrenamiento, el valor máximo de mAP conseguido ahora es de 0.56 por el 0.48 anterior, el verdadero problema es que no se observa la tendencia a mejorar que veíamos en la imagen 7.8. En estas dos últimas pruebas, sobre todo en la que ya hemos realizado 2000 etapas de entrenamiento en total, vemos como hemos llegado a un punto en el que no conseguimos mejorar y el mAP se ha descontrolado completamente, no deja de dar grandes saltos sin seguir una tendencia clara.

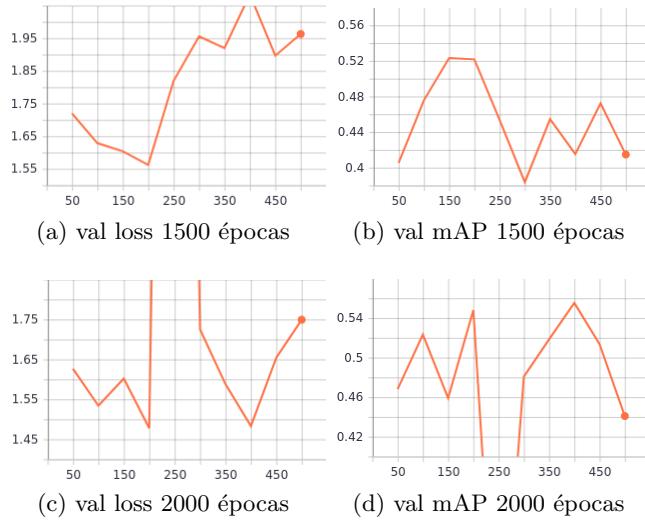


Figura 7.9: Gráficas con las métricas obtenidas por el modelo en la validación durante 500 etapas más a raíz de las 1000 anteriores con la base de datos N-Caltech101 para reconocimiento de objetos, y otras 500 más a partir de las 1500 anteriores.

#### 7.4.3. Inferencia sobre el conjunto test de N-Caltech101 para reconocimiento de objetos

Con este modelo entrenado 2000 etapas hemos utilizado los distintos scripts implementados, y comentados en secciones anteriores, *Implementación*(5), y posteriores, *Manual de Usuario*(8), de este mismo documento, para inferir sobre el subconjunto test de la propia base de datos.

En las figuras 7.10 y 7.11 se muestran algunos ejemplos de las predicciones realizadas. En concreto mostramos un ejemplo donde la predicción ha sido equivocada (7.10c), un ejemplo en el que hemos acertado con el objeto que se representa, pero no tenemos del todo claro su posición en la escena (7.10a) y otros muchos en los que hemos conseguido realizar la predicción correctamente. En estas imágenes el bounding box de color cian es el original, mientras que el magenta representa las predicciones del modelo.

Teóricamente, el valor de la función de pérdida y el valor del mean average presicion deben ser muy similares a los que obtuvimos en la validación ( $\text{loss} = 1.49$  y  $\text{mAP} = 0.55$ ). En el test hemos obtenido una pérdida de 1.429, y un mAP con un valor de 0.5597. Estos valores son similares (el valor absoluto de la diferencia entre ellos es un número menor a 0.075) a los ya comentados, que obtuvimos en la validación, lo que nos dice que el subconjunto val o validation está bien seleccionado, es un buen representante de la base de datos completa.

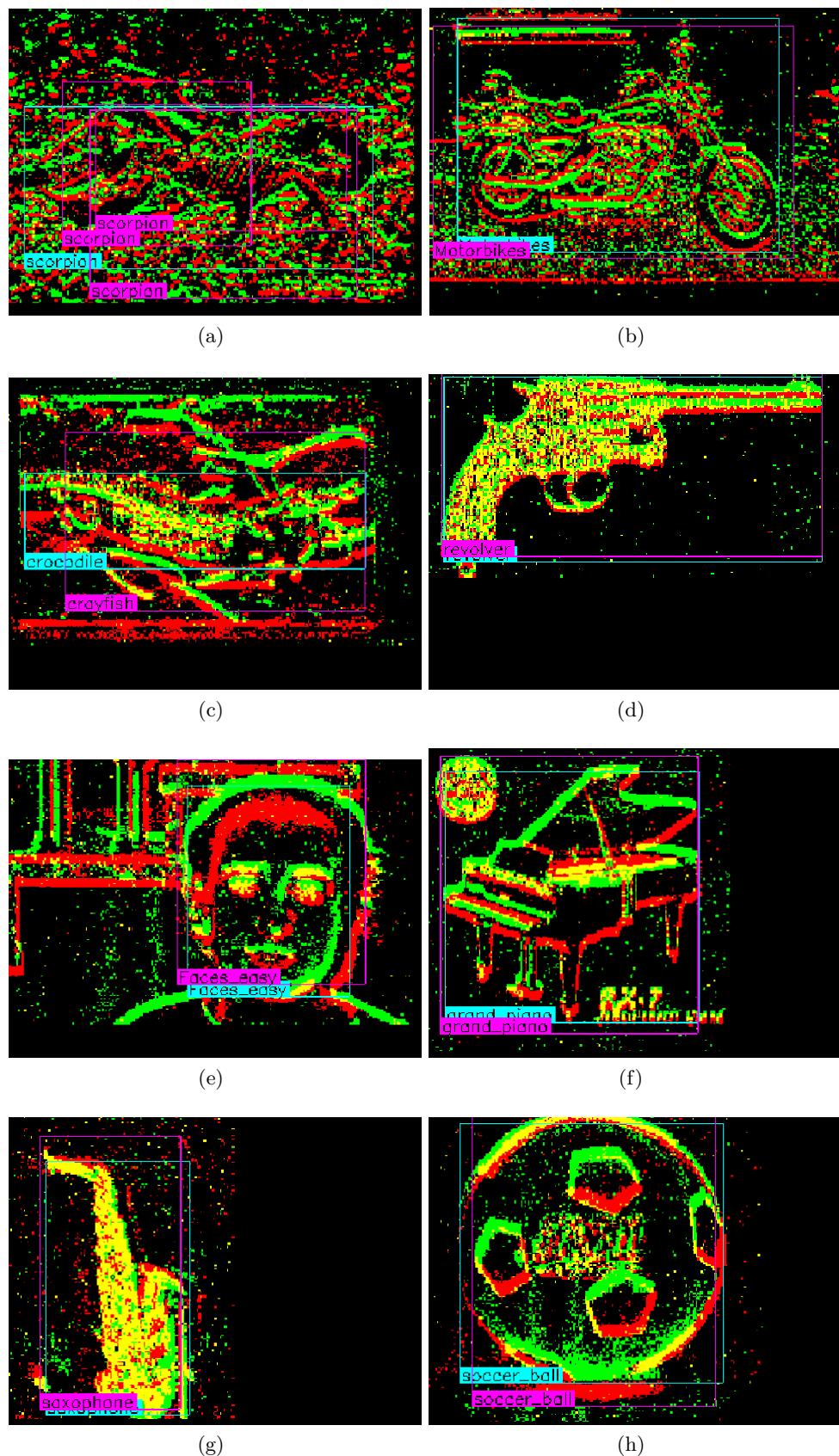


Figura 7.10: Ejemplos de predicciones realizadas sobre el subconjunto test con los mejores pesos obtenidos tras 2000 etapas de entrenamiento.

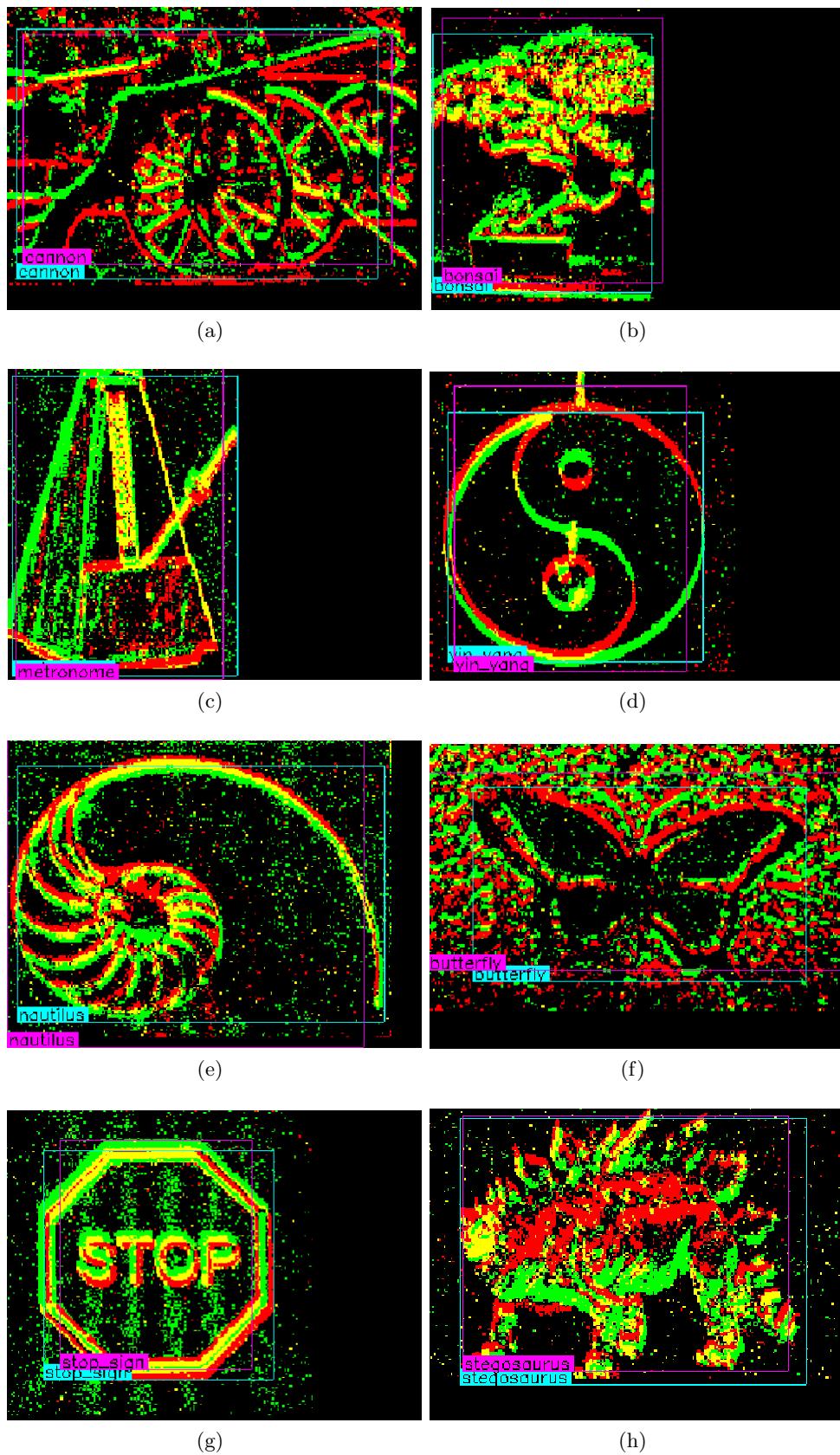


Figura 7.11: Más ejemplos de predicciones realizadas sobre el subconjunto test con los mejores pesos obtenidos tras 2000 etapas de entrenamiento.

Hacer la inferencia sobre el conjunto test, obteniendo imágenes con las predicciones y la etiqueta original, nos ha permitido observar una gran cantidad de imágenes en las que, al igual que pasa en [\[7.10a\]](#), aparecen varios bounding boxes para un mismo objeto.

En las imágenes que se han mostrado en las figuras [\[7.10\]](#) y [\[7.11\]](#) vemos como la representación que hacemos de los eventos es distinta a la usada hasta este momento. Ahora vemos algunos píxeles pintados en rojo, verde o amarillo, mientras que el fondo de la imagen es negro. Este cambio se debe a que ahora estamos trabajando con el paquete asynet, mientras que antes lo hacíamos sobre v2e o dvs\_ros. Sin embargo, a pesar de ser distintos paquetes, solamente cambia el color a la hora de representar los eventos. Todas las imágenes están formadas mediante la acumulación de eventos, en este caso se representan en una misma imagen todos los eventos que contenga el archivo leído en cada momento y se dibujan en verde aquellos eventos con polaridad positiva, mientras que los rojos son los de polaridad negativa. En el caso de aquellos píxeles que presentan color amarillo es porque coinciden dos o más eventos con polaridades opuestas en instantes de tiempo distintos pero con una misma posición espacial.

#### 7.4.4. Pruebas variando los valores de threshold del detector e IoU en la supresión de no máximos

En esta sección se presentan una serie de inferencias realizadas con un mismo modelo, unos mismos pesos, pero en las que vamos variando el valor de un conjunto de variables. Estos parámetros van a determinar si un posible objeto detectado se considera como tal o se descarta, y si dos mismos recuadros delimitadores pertenecen a un mismo objeto, y en caso de ser así, con cuál quedarnos. Las variables con las que vamos a ir jugando son las siguientes:

- **Threshold:** el umbral es el nivel mínimo de confianza que el modelo deberá tener en un objeto detectado para mostrar el bounding box a su alrededor.
- **IoU:** acrónimo de Intersection over Union, representa lo superpuesto que está el bounding box calculado por el modelo y el real. Si el IoU vale uno entonces el cuadro calculado será igual al real. En la imagen [\[7.12\]](#) podemos ver una definición gráfica. En este caso, el IoU no se calcula respecto al bounding box real, sino en relación a todos los que hayamos calculado para unos mismos datos de entrada, y será el parámetro que marque la supresión de no máximos.

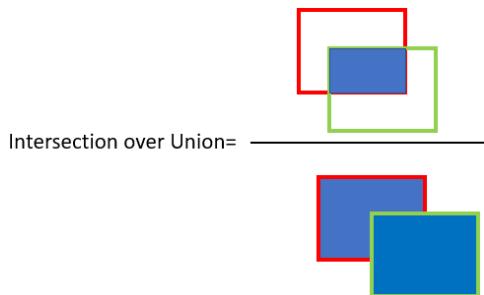


Figura 7.12: Definición de IoU. El recuadro verde representa el bounding box real, mientras que el rojo es la predicción hecha por el modelo. Fuente: [\[17\]](#)

En la tabla [\[7.2\]](#) vamos a observar el mAP obtenido sobre el conjunto test en función del valor que presenten estos parámetros que se acaban de describir. No se muestra la pérdida obtenida en función de los distintos parámetros porque el loss se mantiene constante en 1.429, estas variables no determinan esta métrica. La tabla es simplemente una pequeña muestra de algunas de las combinaciones probadas, y en ella vamos a ver como seleccionar adecuadamente el valor que presentan estos parámetros va a suponer una mejora o empeoramiento casi del 10 % en la bondad de las predicciones.

<b>threshold</b>	0.25	0.25	0.3	0.7	0.75
<b>IoU</b>	0.5	0.8	0.6	0.5	0.5
<b>mAP</b>	0.5542	0.5161	0.5597	0.5832	0.5701

Cuadro 7.2: Bondad de las predicciones en función del valor del threshold e IoU.

#### 7.4.5. Inferencia sobre datos obtenidos mediante el simulador v2e

En este apartado, al igual que en el anterior, presentamos varios experimentos realizados con un mismo modelo, con las mismas condiciones. La diferencia radica en que ahora vamos a presentar los resultados obtenidos al realizar inferencia usando nuestro propio conjunto de datos, es decir, la entrada ahora son distintos archivos txt con los eventos que hemos obtenido nosotros mismos con ayuda del simulador v2e.

Los vídeos que constituyen el dataset, a partir de los cuales se han obtenido los distintos eventos, han sido grabados por mi mismo, y en ellos se muestran distintos objetos corrientes que se pueden encontrar en cualquier casa. Estos objetos que se han filmado y han sido utilizados para realizar los

experimentos son: una cámara de vídeo digital, una fotografía tipo carnet, el lateral de un coche, una lámpara, un ordenador portátil, un reloj despertador, una taza de café, un teléfono inalámbrico y unas tijeras de costura.

Todos los vídeos, a excepción del que muestra un coche, han sido grabados con las mismas condiciones de luminosidad. Todos tienen una duración aproximada de dos segundos y muestran un único objeto centrado en el objetivo de la cámara y ocupando casi todo el alto y ancho de la escena. Estos vídeos utilizados se pueden encontrar en la carpeta compartida en drive para la liberación del proyecto ([\[13\]](#)).

Los experimentos que aquí se presentan no tienen como único objetivo mostrar si el modelo es capaz de detectar e identificar correctamente todos los objetos, sino que pretendemos al mismo tiempo ver cómo de determinante es la resolución del vídeo de salida en la generación de eventos. Para ello se ha generado la tabla [7.3](#) que muestra información de las salidas que nos ha proporcionado el modelo estudiado, entre la que se encuentra el porcentaje de aciertos.

La inferencia se ha realizado separando los eventos de un mismo archivo cada 70 ms tal y como se explicó en el capítulo 5. *Implementación* ([5](#)), y con un modelo cuyos pesos son los mejores obtenidos durante el entrenamiento de 2000 épocas. Además se ha utilizado un threshold = 0.7 y un IoU = 0.5 en todos ellos.

Resolución utilizada	240x426	360x640	480x854	720x1280
Número de aciertos	73	98	109	115
Porcentaje de acierto	22.18 %	29.78 %	33.13 %	34.95 %
Ejemplos	7.14a	7.14b	7.14c y 7.14c	7.15a, 7.15b y 7.15c

Cuadro 7.3: Número y porcentaje de acierto al inferir sobre nuestros propios datos en función de la resolución utilizada para obtener los eventos.

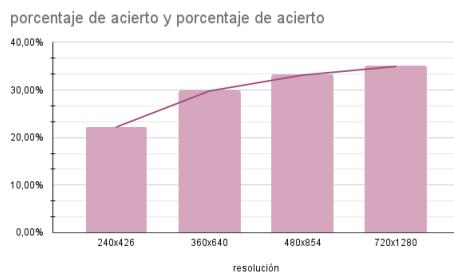


Figura 7.13: Gráfica que relaciona la resolución del vídeo de salida del simulador v2e con el porcentaje de acierto en la inferencia por el modelo descrito.

En este caso no hemos podido hablar de mAP porque los datos de entrada no están etiquetados. Consideramos entonces que una predicción es acertada cuando se ha identificado correctamente el objeto que representan los datos, y se ha establecido un bounding box alrededor de este, sin necesidad de que se ajuste a la perfección al tamaño y posición del objeto en cuestión.

También se ha elaborado una gráfica que enfrenta el porcentaje de acierto con la resolución de salida establecida en el simulador v2e. En la gráfica, que podemos verla en la imagen 7.13, apreciamos como la mejora obtenida con el aumento de calidad es cada vez menor. La línea que se muestra tiene forma logarítmica. Esta información, unida a la que se proporcionó en el apartado 7.2 nos va a permitir establecer la resolución más adecuada en función de los recursos disponibles y el uso que se quiera hacer del sistema.

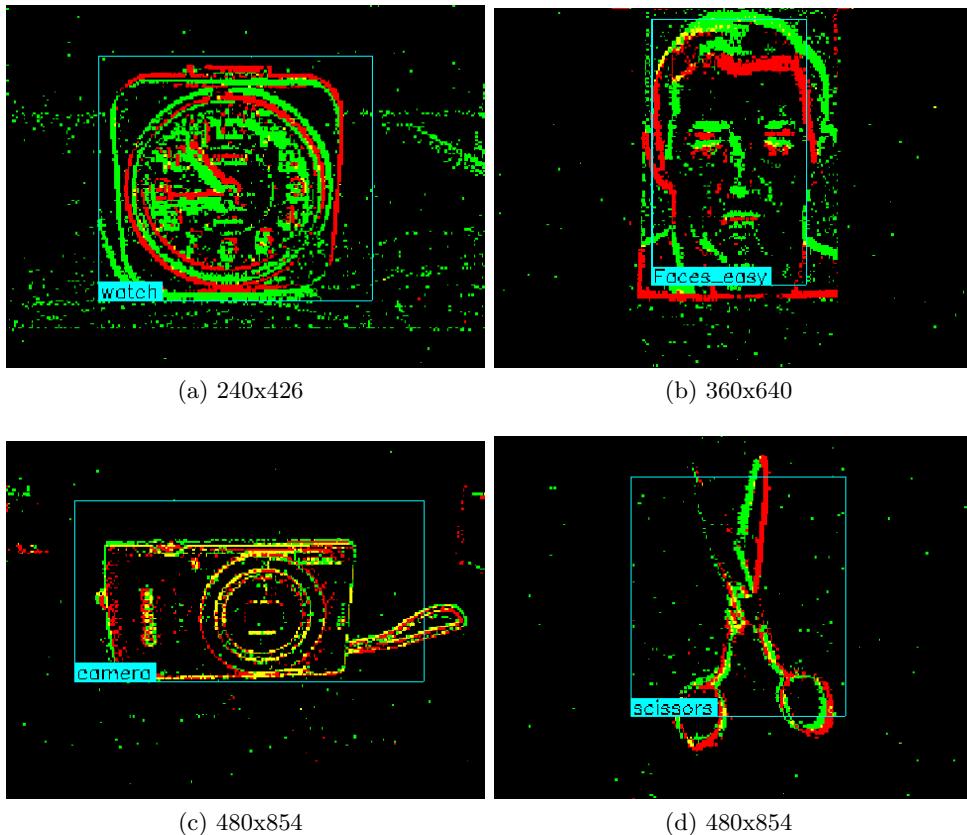


Figura 7.14: Ejemplos de predicciones realizadas sobre nuestro propio conjunto de datos utilizando distintas resoluciones de salida en la obtención de los eventos a partir de vídeos con el simulador v2e.

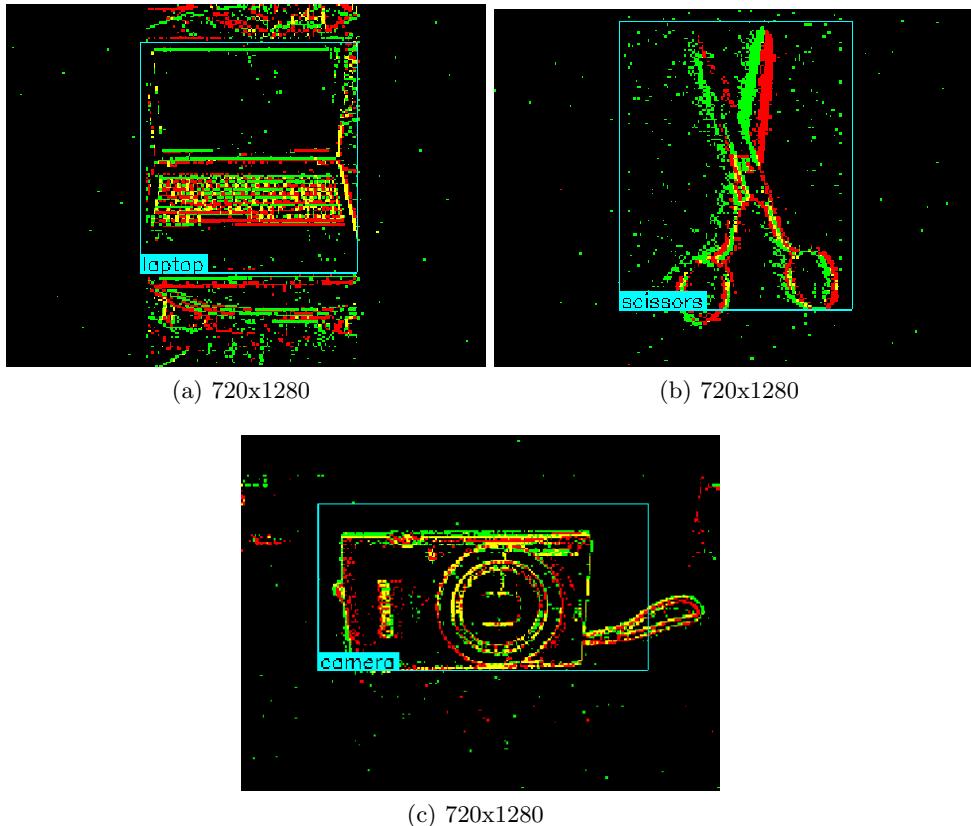


Figura 7.15: Más ejemplos de predicciones realizadas sobre nuestro propio conjunto de datos utilizando distintas resoluciones de salida en la obtención de los eventos a partir de vídeos con el simulador v2e.

### Inferencia y generación de un vídeo con las imágenes resultantes

Para terminar con la inferencia utilizando asynet, se han hecho varias pruebas intentando obtener un vídeo con las imágenes de salida que se nos proporciona. La creación de un vídeo a partir de imágenes no es algo complicado, sin embargo, no conseguimos obtener un vídeo en el que se detecten correctamente los objetos y tenga sentido crear dicho vídeo. Es decir, aunque conseguimos crear el vídeo, para que un vídeo así tenga sentido se debe mostrar la escena relativamente alejada de los objetos, para así poder ver dónde están y observar cómo se mueven en relación a la posición de la cámara, según ésta, o los propios objetos, se desplazan.

El problema encontrado radica en la base de datos utilizada. Como ya se ha mencionado en alguna ocasión, en N-Caltech101 todos los objetos están centrados en la imagen y ocupan gran parte de ella, no se aprecia nada

más en la foto, y estamos pretendiendo encontrar objetos en un escenario radicalmente distinto al del entrenamiento. El modelo no consigue hacerlo correctamente, tal y como vemos en la figura 7.16.

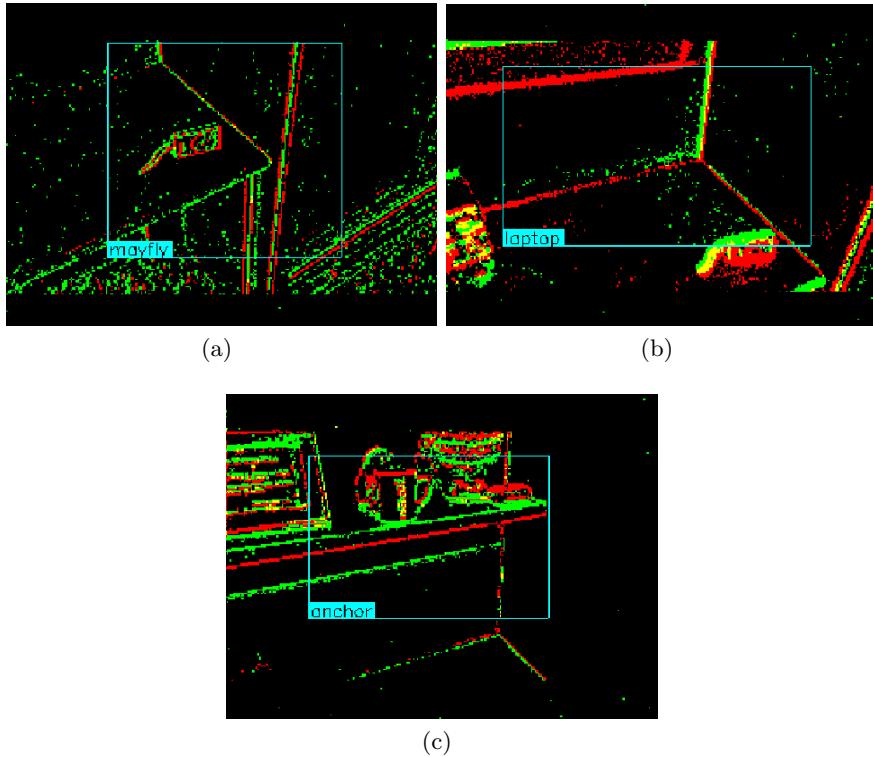


Figura 7.16: Ejemplos de predicciones realizadas sobre una escena grabada de lejos por nosotros mismos cuyos eventos hemos obtenido mediante el simulador v2e con una resolución de 720x1280 píxeles. El modelo asynet utilizado presenta los mejores pesos obtenidos tras 2000 épocas de entrenamiento, un threshold = 0.7 y un IoU = 0.5.

En las imágenes 7.16a y 7.16b lo que se aprecia realmente es una cámara fotográfica digital en un escritorio, mientras que la imagen 7.16c muestra una taza en un estante por encima del escritorio anterior. Tal y como hemos dicho anteriormente, el modelo no es capaz de detectar y reconocer estos objetos por los motivos que también se han explicado ya en este apartado.

Realizando el entrenamiento con una base de datos más adecuada para este fin, como puede ser COCO, quizás conseguíramos unos resultados aceptables. Sin embargo, esto no está a nuestro alcance hoy en día puesto que esta base de datos, al igual que la inmensa mayoría, todavía no se han adaptado, como sí lo ha hecho Caltech101.

## 7.5. Pruebas de detección de objetos basada en imágenes

Al no ser este detector el objetivo de nuestro proyecto, sino que la utilización de éste es para poder realizar una comparación entre asynet y un modelo de detección de objetos tradicional, no hemos realizado con él tantas pruebas como con el sistema anterior. Nos hemos centrado en realizar un entrenamiento con la misma base de datos utilizada en el modelo basado en eventos, y una posterior inferencia tanto con el subconjunto test de la base de datos, como con imágenes equivalentes a las escenas representadas por los eventos que se le han pasado a asynet.

Tanto para la realización del entrenamiento, como para la inferencia hemos mantenido todos los parámetros del modelo por defecto, tal y como vienen en su repositorio original [26].

Tras la realización de un entrenamiento para el reconocimiento de objetos con un total de 1000 etapas, se han generado y almacenado en wandb las gráficas que podemos ver en la figura 7.17. En todas ellas la tendencia es la correcta desde el principio, alcanzando muy buenos valores pronto y conservando dichos resultados hasta el final. De hecho, en la pérdida de la validación y en el mAP observamos como a partir de la etapa número 400 la mejora es prácticamente nula, insignificante. En cualquier caso, al final del entrenamiento tenemos una pérdida inferior a 0.005 y un mean average precision superior a 0.80, lo que nos sugiere que las predicciones que realicemos con este modelo y estos pesos van a ser muy buenas.

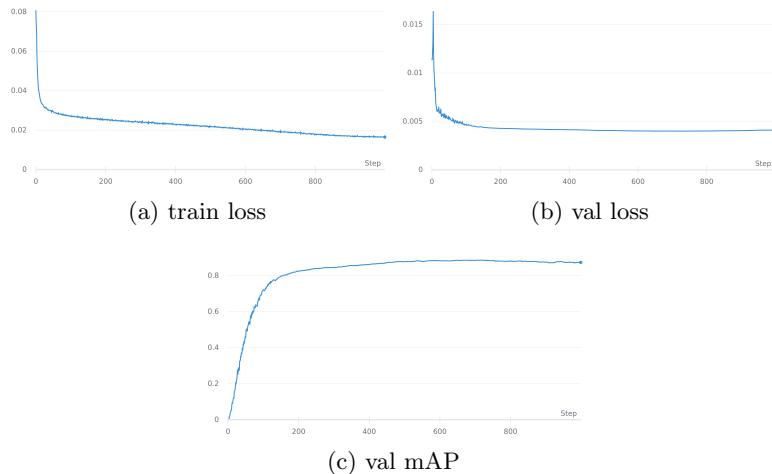


Figura 7.17: Funciones de pérdida y mAP del entrenamiento y la validación de 1000 épocas con YOLOv5 y la base de datos Caltech101.

Con los mejores pesos encontrados durante el entrenamiento anterior, hemos inferido sobre el subconjunto test de la propia base de datos obteniendo un mAP de 0.891. Algunos ejemplos, tanto correctos, como fallidos, de las predicciones realizadas los podemos ver en la ilustración 7.18. Este valor en el mean average precision es aún mejor que el esperado y confirma que este modelo con estos pesos es capaz de realizar predicciones muy acertadas.

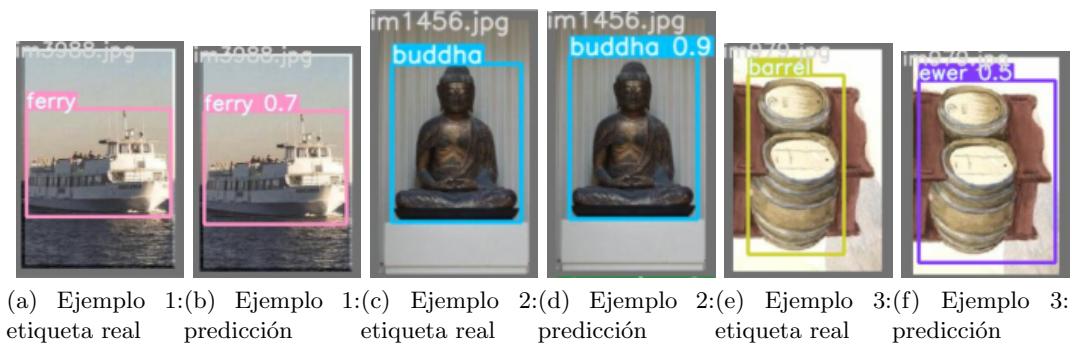


Figura 7.18: Ejemplos de predicciones realizadas sobre el subconjunto test por el modelo YOLOv5 con los mejores pesos obtenidos en el entrenamiento anterior.

Más importante aún que los resultados que podamos obtener sobre la base de datos, que no deja de ser, una situación ideal, preparada, es observar si el mismo modelo, con los mismos parámetros y pesos, es capaz de detectar y reconocer los objetos correctamente en las imágenes sacadas de los vídeos con los que posteriormente hemos generado los eventos y probado con el modelo anterior, asynet. Se han sacado fotogramas de esos distintos vídeos y se ha inferido sobre ellos obteniendo un acierto del 45.72 %, una cifra muy inferior al porcentaje de acierto obtenido en esa situación ideal que describimos anteriormente.



Figura 7.19: Ejemplos de predicciones realizadas sobre imágenes sacadas de los vídeos grabados por mi mismo, en los que se muestran objetos cotidianos, por el modelo YOLOv5 con los mejores pesos obtenidos en el entrenamiento anterior.

Al igual que se hizo con el modelo anterior, inferimos sobre las imágenes en las que se aprecian los objetos de lejos, aquellas creadas con el objetivo de construir un vídeo posteriormente, obteniendo el mismo resultado. No conseguimos detectar correctamente ninguno de los objetos que se muestran en dichas imágenes.

## 7.6. Comparación entre asynet y yolov5

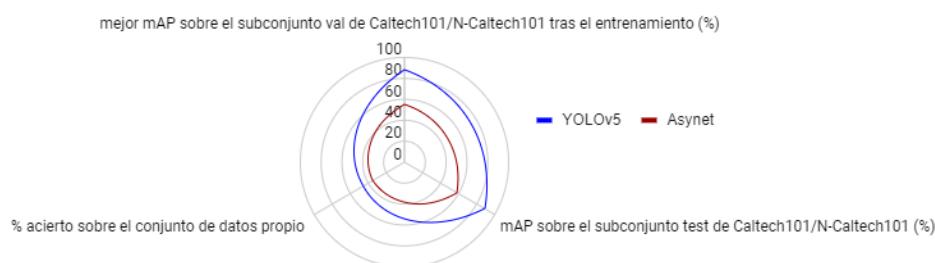


Figura 7.20: Gráfica que compara la eficacia del modelo YOLOv5 basado en imágenes con el modelo Asynet que trabaja con eventos.

Los distintos comportamientos que han ido adoptando la función de pérdida y el mean average precision durante el entrenamiento, y la validación se sobreentiende, (figuras 7.8, 7.9 y 7.17) son radicalmente distintos según el modelo utilizado. Mientras que YOLOv5 ha convergido muy rápidamente y ha presentado valores muy buenos, prácticamente 0 en el loss y más de 0.8 en el mAP, el modelo Asynet se ha mantenido mucho más irregular durante todas y cada una de las etapas. Este modelo basado en eventos no ha conseguido estabilizar la curva en ningún momento, y aun así, tan solo en la función de pérdida del conjunto train conseguimos acercarnos a los valores proporcionados por el otro modelo. De hecho, la diferencia es tal que, el mAP pasa de superar claramente el 0.80 a no alcanzar el 0.56 en función del modelo utilizado.

Al inferir sobre el conjunto test de la propia base de datos, como cabría esperar, esta diferencia en la bondad de las predicciones entre ambos modelos se mantiene. Mientras que el modelo denso basado en imágenes, YOLOv5, obtenía un  $mAP = 0.891$ , el mejor valor conseguido con asynet era de 0.5832. Hay una diferencia superior a 0.3 trabajando sobre la base de datos Caltech101, y su análoga N-Caltech101.

Sin embargo, esta diferencia se ve reducida cuando inferimos sobre fotogramas sacados de vídeos, y sus correspondientes eventos, grabados por mi. Aunque ambos modelos reducen su eficacia, la de YOLOv5 se ve reducida una cantidad mayor que la de asynet. Con la resolución de vídeo más

adecuada que tenemos a nuestro alcance a la hora de obtener eventos, este último modelo obtiene un porcentaje de acierto del 34.95 % en sus predicciones. Mientras tanto, YOLOv5 es capaz de acertar en el 45.72 % de las imágenes con las que se ha probado. Como vemos, la diferencia entre modelos se ha reducido a un tercio aproximadamente. Ha pasado de 0.3078 a 0.1077. Esta reducción se debe, entre otros factores, a que al presentar más información las imágenes que los eventos, el modelo que trabaja con ellas es más fácil que se sobreajuste a la base de datos con la que se entrena. Es por ello que al inferir sobre imágenes con condiciones de luminosidad distintas, que no están tan preparadas, ni se han preprocesado, vamos a tener más dificultades a la hora de realizar la detección y reconocimiento de objetos correctamente.

En el único aspecto que no hay diferencia entre los dos modelos, es que ninguno es capaz de detectar los objetos cuando éstos se encuentran alejados del objetivo de la cámara, cuando solo representan una pequeña parte de la escena y no están centrados en ella. Como ya dijimos en secciones anteriores, esto es debido a que la base de datos no es apropiada, por lo tanto, no tiene importancia el utilizar un modelo u otro. No vamos a ser capaces de detectarlos y reconocerlos correctamente.

Hasta ahora hemos estado comparando la eficacia obtenida con cada modelo en los experimentos realizados. Sin embargo, a la hora de seleccionar qué modelo nos conviene utilizar no debemos fijarnos únicamente en esto.

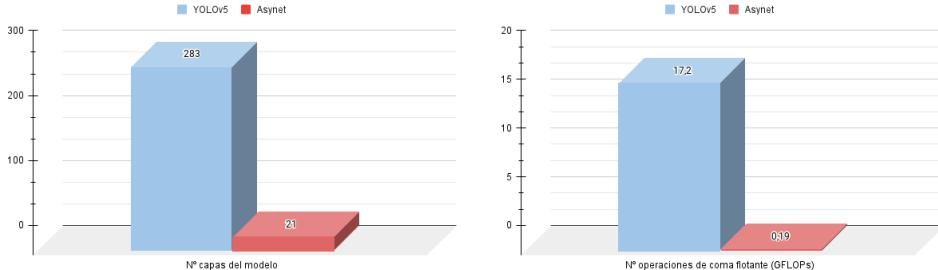


Figura 7.21: Gráficas que comparan el número de capas y la cantidad de operaciones de coma flotante que realizan el modelo YOLOv5 basado en imágenes y el modelo Asynet que trabaja con eventos.

Otro aspecto a tener en cuenta es el tiempo necesario para procesar los datos de entrada. Como ya se ha mencionado en alguna ocasión, los eventos solamente contienen información espacio temporal y de polaridad. En el caso opuesto encontramos las imágenes, que por lo general presentan o bien tres canales, RGB, o un único canal si la imagen está en escala de grises. En cualquier caso aquí encontramos mucha más información, lo que supone un

aumento en el tiempo requerido para su procesado y en el espacio necesario para su almacenamiento, aunque éste sea temporal.

Además, YOLOv5 es un modelo que supone mucha más carga computacional que asynet. Esto es algo lógico puesto que este último es una red dispersa, y éstas, tal y como se explica en el capítulo 1: *Introducción* (1), son más simples que las densas porque eliminan aquellas conexiones entre neuronas cuyos pesos valen 0. Esto lo vemos reflejado en la cantidad de operaciones en coma flotante que requieren estos dos modelos. Mientras que YOLO en su quinta versión tiene una arquitectura de 283 capas que precisa de 17.2 GFLOPs, el modelo asynet, con 21 capas, tan solo necesita el 1.10 % de estas operaciones, 0.19 GFLOPs.

Por lo tanto, podemos decir que el modelo YOLOv5 aprovecha toda esa información extra que recibe para extraer más características de las imágenes. Esto le permite aprender mejor y realizar predicciones más acertadas que asynet, un modelo que solamente recibe eventos con información espacio temporal. Sin embargo, este último modelo, supone una carga computacional 90 veces inferior y permite aprovechar todas las ventajas que se presentan al trabajar con eventos, ya sea obteniéndolos por medio de un sensor de visión dinámica, o por medio de un simulador. En definitiva estamos perdiendo precisión y calidad en las predicciones a cambio de tener un modelo más sencillo, que además es capaz de aprovechar las ventajas que nos ofrecen los eventos. Estamos estableciendo un compromiso entre complejidad, recursos necesarios consecuentemente, y la precisión del modelo.

# Capítulo 8

## Manual de usuario

Este apartado pretende servir como guía para futuros desarrolladores interesados en el uso del sistema implementado en este trabajo de fin de grado. Es por ello que presentamos un manual en el que se explica qué scripts ejecutar y cómo ejecutarlos en función de la tarea que se persiga en cada momento. Además, con motivo de facilitar la instalación de cada subsistema utilizado, tarea que no ha sido nada fácil, y la integración de estos, ofrecemos también una guía. En ésta indicaremos el repositorio desde el que descargar e instalar cada parte, así como las soluciones encontradas a los principales problemas que hemos tenido que solventar y que más tiempo nos han llevado durante la instalación.

### 8.1. Guía de instalación

Debido a que cada subsistema instalado es independiente del resto, es posible instalarlos en cualquier orden o incluso obviar alguno de ellos. Sin embargo, se recomienda la instalación de todos ellos y seguir el orden descrito en este manual por ser la estructuración más lógica.

Además, antes de empezar la instalación debemos asegurarnos que cumplimos con todos los requisitos hardware y software descritos en la sección *Problemas durante el desarrollo* (6). También es recomendable que cada instalación se haga en un entorno virtual distinto para no provocar conflictos entre paquetes. Solo así podremos garantizar el correcto funcionamiento de todos los sistemas empleados.

#### 8.1.1. rpg\_dvs\_ros

En primer lugar debemos instalar la versión de ROS que soporte el sistema operativo con el que se vaya a trabajar. En nuestro caso, como ya se

ha explicado en algún que otro momento, el sistema operativo empleado ha sido la distribución 18.04 de Ubuntu y por lo tanto instalamos ROS Melodic siguiendo los pasos que se nos explica en la web oficial [74]. En un principio no deberíamos de encontrar ningún problema siempre y cuando no nos salgamos del guión establecido.

Una vez hemos completado la instalación de alguna de las tres versiones de ROS (Kinetic, Melodic o Noetic) con las que es compatible el siguiente sistema que debemos conseguir, ya podremos instalar correctamente los controladores para las cámaras de eventos y la lectura de archivos rosbag pregrabados, el paquete dvs\_ros.

La instalación de este paquete se puede hacer cómodamente según como se indica en el repositorio de github [75]. Durante la instalación de este módulo es posible que nos encontramos con algunos paquetes que no se encuentran instalados en nuestro ordenador, o en el entorno virtual que estemos utilizando en ese momento. Esto no tiene por qué suponer un gran inconveniente, podemos instalar los paquetes requeridos utilizando “pip install”. Sin embargo, si el error persiste, como nos ha pasado a nosotros con rospkg, una buena alternativa es utilizar anaconda para instalar dicho paquete. Podemos hacerlo mediante la orden “conda install -c conda-forge rospkg”.

### 8.1.2. v2e

A continuación debemos instalar el simulador v2e siguiendo las indicaciones del repositorio([28]). Al aparecer el archivo environment.yml en el repositorio tenemos la posibilidad de crear el entorno virtual de conda con todos los paquetes necesarios instalados en él mediante la instrucción “conda env create -f environment.yml”. Esta es la manera más simple y rápida de cumplir con todos los requisitos sin inconvenientes.

Se recomienda descargar el contenido de v2e-master del repositorio en el que hemos subido este proyecto, [87], en lugar de clonar el de la web oficial porque así ya estarían modificados los archivos que hemos tenido que retocar, y tendríamos descargados los ejecutables nuevos creados. Si por algún motivo se prefiere clonar el original y posteriormente añadir uno mismo los cambios, simplemente tendría que realizar los siguientes pasos:

1. En el archivo v2ecore/emulator.py hay que comentar las líneas que se ven en la imagen 5.2.
2. Descargar el archivo eventsToBag.py del repositorio de github del proyecto. Para poder lanzar este script de forma correcta tendremos que instalar algunos módulos adicionales. De nuevo podemos hacerlo mediante “pip install”. Sí tenemos que tener en cuenta que ante el error

“no module named cryptodome”, el paquete que tenemos que instalar no se llama “cryptodome”, sino “pycryptodomex”. Siempre que no sepamos el nombre exacto del módulo a instalar podemos buscarlo en la web y habrá una gran comunidad, con muchos foros que nos proporcionaran esta información.

3. Descargar el archivo generarEventos.sh del repositorio de github del proyecto.

### 8.1.3. rpg\_asynet

Llegados a este momento, en el que ya tenemos la posibilidad de obtener los eventos de una escena, debemos instalar el software que nos permitirá realizar la detección y reconocimiento de objetos a partir de esta información. Debemos realizar la instalación de Asynet siguiendo los pasos descritos en el repositorio [24] por los autores del sistema.

Si, como ha sido nuestro caso, al descargar eigen desde el enlace proporcionado aparece el mensaje de error “404 That link has no power here”, existe un enlace alternativo que se especifica en uno de los issues del repositorio ([77]). Este link sí está disponible actualmente y tan solo tendremos que descargar el directorio principal eigen y ubicarlo en la ruta especificada, rpg\_asynet/async\_sparse\_py/include.

Alternativamente podemos descargarnos el directorio rpg\_asynet desde el repositorio de nuestro proyecto. Aquí ya está el paquete eigen situado correctamente y nos ahorraríamos tener que realizar los siguientes cambios con respecto a la versión original de asynet:

1. Sustituir las carpetas training, dataloader y config por las que se pueden encontrar en el repositorio de nuestro proyecto, [87].
2. Copiar la carpeta testing y los archivos ejecutables de python que se encuentran en el directorio principal de asynet en el repositorio de nuestro proyecto, [87].
3. Descargar la base de datos que vayamos a utilizar del enlace proporcionado en el repositorio de asynet [24]. Aunque la base de datos NCars ha sido modificada para su utilización y deberíamos descargarla desde el siguiente issue en el que he tenido que realizar un comentario para obtenerla [10]. Podemos descargar los directorios en los que debemos incluir los archivos que nosotros mismos obtengamos mediante el simulador, o el sensor, directamente desde una carpeta compartida en drive [13]. En ella hemos publicado algunos recursos del proyecto.

4. (OPCIONAL) Añadir unos modelos preentrenados que se encuentran en la carpeta drive del proyecto [13]. Estos modelos podemos incluirlos en una nueva carpeta con el nombre que nos parezca conveniente.

### 8.1.4. YOLOv5

Finalmente, el software utilizado para realizar una comparación de resultados, YOLO en su quinta versión, podemos instalarlo, en caso de que nos sea de interés, siguiendo las indicaciones de su repositorio [26].

Aunque se recomienda descargar el directorio yolo del github de nuestro proyecto [87] ya que de esta forma obtenemos todo directamente, excepto la base de datos Caltech101 utilizada para el entrenamiento parseada, que la podemos obtener en la carpeta compartida de drive [13]. En el repositorio de github también encontramos algunos modelos de prueba ya preentrenados. Tras la descarga de yolo desde nuestro repositorio simplemente debemos ejecutar el comando “pip install -r requirements.txt” situados en la carpeta yolov5 para que se instalen todas las dependencias sin problema y descomprimir la base de datos en el directorio principal.

Si por cualquier motivo se opta por instalarlo desde el enlace original, tendríamos que situar el directorio yolov5 obtenido dentro de una nueva carpeta llamada yolo. También tendríamos que realizar los siguientes cambios:

1. Copiar todo el contenido del directorio yolo, a excepción de yolov5, del repositorio de nuestro proyecto [87].
2. Copiar en yolov5 el ejecutable test.py y el archivo de configuración caltech101.yaml del repositorio de nuestro proyecto [87].

## 8.2. Manual de uso

### 8.2.1. Obtención de nuevos datos

En primer lugar debemos explicar cómo obtener los eventos a partir de un vídeo obtenido mediante una cámara de fotogramas. Para esto, situados en la carpeta donde hayamos realizado la instalación de v2e, debemos situar el vídeo, o vídeos, sobre los que obtener los eventos en la carpeta input. A continuación debemos lanzar el ejecutable generarEventos.sh indicándole el nombre de los vídeos deseados, y éste será el encargado de llamar al simulador y de generar el archivo rosbag automáticamente.

Toda la salida del simulador y la bolsa con los eventos se alojarán en una nueva carpeta, con el mismo nombre que el vídeo, y alojada en el directorio output.

Para ejecutar este script correctamente no debemos hacerlo mediante los comandos “./” ni “sh” porque algunas órdenes cambian el entorno operacional conda y no son admitidas por dichos métodos. Es por ello que debemos ejecutarlos mediante “bash -i generarEventos.sh <lista de vídeos>”. Al tener que cambiar el entorno operacional de conda es imprescindible abrir el script y especificar el nombre del entorno donde se ha realizado la instalación de v2e en las líneas 10 y 17. Además del nombre donde se ha instalado el paquete dvs\_ros en la línea 14.

El script por defecto funciona con archivos de entrada .avi, aunque si tenemos algún otro formato se puede indicar fácilmente en la línea 11. Además, este script que hemos implementado obtiene los eventos del vídeo entero y la resolución de salida es 854x480 por defecto. Sin embargo ambas cosas pueden cambiarse añadiendo los parámetros de v2e adecuados en la línea 11 nuevamente. Podemos ver todas las opciones disponibles en el repositorio de v2e, [28].

Finalmente se aconseja que los nombres de los vídeos no contengan números debido al funcionamiento de algunos ejecutables de sistemas posteriores.

Para comprobar que los eventos se han generado correctamente nos basta con visualizar el archivo rosbag con ayuda de ROS. Para ello tendremos que abrir tres consolas con el entorno de ROS activado. En las distintas terminales ejecutaremos los siguientes comandos:

1. “catkin build davis\_ros\_driver”
2. “catkin build dvs\_renderer”
3. “source /home/<user\_name>/catkin\_ws/devel/setup.bash”
4. Ejecutar uno de los siguientes en cada terminal:
  - “roscore”
  - “rosbag play -l path-to-file.bag”
  - “roslaunch dvs\_renderer renderer\_mono.launch”

De esta forma veremos una película con los distintos eventos que se han generado con ayuda del simulador y podremos comprobar si son correctos y si pensamos que pueden ser suficientes para nuestro propósito.

### 8.2.2. Entrenamiento del modelo basado en eventos e inferencia

Para entrenar correctamente un modelo con una base de datos de eventos entre las disponibles tendremos que indicar en primer lugar el modelo que

queremos utilizar y dónde se encuentra ubicada la base de datos. Esto lo haremos modificando el archivo settings.yaml situado en el directorio config. En este documento podremos indicar incluso si queremos partir desde cero, o de un modelo ya preentrenado y cuyos pesos tenemos almacenados en un archivo con extensión pth.

Los distintos scripts y ejecutables que nosotros hemos implementado solo serán válidos para el modelo fb\_sparsse\_object\_det que es el que nosotros hemos utilizado, aunque es fácilmente ampliable al resto de modelos en caso de ser necesario. Esto hace que en el archivo de configuración tengamos que especificar este modelo sí o sí para poder lanzar correctamente cualquiera de los archivos y las funciones realizadas por nosotros mismos. Esto no ocurre así en el archivo train.py, aquí podemos seleccionar cualquier modelo, por lo que una buena práctica puede ser entrenar el modelo fb\_sparsse\_vgg para clasificación, y posteriormente hacer lo propio con el equivalente dedicado a la detección de objetos partiendo de los pesos anteriores.

Si vamos a realizar un entrenamiento, ya sea mediante trainer.py o a través de trainer\_and\_tester.py, debemos especificar el número de épocas dedicadas a entrenar. Para ello tenemos que abrir los archivos mencionados y escribir la cantidad requerida en las líneas 232 y 492 para el primer archivo, y la línea 376 en el segundo archivo.

Una vez hemos hecho esto ya podemos realizar la tarea que queramos mediante la ejecución del comando “sh asynet.sh <opcion>”, donde opcion hace referencia a alguno de los scripts que se encuentran en el directorio principal y cuyas funciones se explican a continuación. Todo esto teniendo activado el entorno virtual donde se haya instalado el paquete asynet.

- **<opcion> = 0:** En este caso se ejecutará el script train.py que nos sirve para entrenar un modelo con el número de épocas que se especifiquen, realizando algunas etapas de validación y salvando los pesos del modelo cuando se mejoren a los últimos almacenados.
- **<opcion> = 1:** En este caso se ejecutará el script train\_and\_test.py que nos sirve para entrenar un modelo con el número de épocas que se especifiquen, realizando algunas etapas de validación y salvando los pesos del modelo cuando se mejoren a los últimos almacenados. Una vez concluido el entrenamiento, cargará los mejores pesos, los últimos que se hayan guardado, y realizará inferencia sobre el conjunto test de la base de datos. Aportándonos así más información que sólo realizando la validación.
- **<opcion> = 2:** En este caso se ejecutará el script test\_con\_estadisticas.py que nos sirve para realizar la inferencia sobre una de las distintas bases de datos que admite el modelo, calculando y guardando estadísticas y

métricas de bondad en función de las anotaciones de la propia base de datos.

- **<opcion> = 3:** En este caso se ejecutará el script test\_sin\_estadisticas.py que nos sirve para realizar la inferencia sobre una de las distintas bases de datos que admite el modelo, sin calcular ni guardar estadísticas ni métricas de bondad.
- **<opcion> = 4:** En este caso se ejecutará el script test\_mis\_datos\_primeros\_eventos.py que nos sirve para realizar la inferencia sobre los datos que nosotros hayamos generado mediante el simulador de eventos. Debido a la gran cantidad de eventos que se generan, aquí solo consideraremos los primeros 0.07 segundos de cada archivo.
- **<opcion> = 5:** En este caso se ejecutará el script test\_mis\_datos\_todos\_eventos.py que nos sirve para realizar la inferencia sobre los datos que nosotros hayamos generado mediante el simulador de eventos. Se tendrán en cuenta todos los eventos generados, pero los iremos agrupando en intervalos de 0.07 segundos.
- **<opcion> = 6:** En este caso se ejecutará el script test\_mis\_datos\_todos\_eventos\_con\_video.py que es equivalente al caso anterior, solo que ahora generamos un vídeo al final con las distintas imágenes resultantes de la inferencia.

Es importante saber que para el correcto funcionamiento de las últimas tres opciones debemos de copiar los archivos de eventos con extensión txt que nos devuelve el simulador en el interior de los directorios que se encuentran en data/mis\_videos. Esta carpeta es desde la que se realiza la lectura de eventos.

En algunas de las opciones mencionadas se dice que hay un guardado de estadísticas y métricas de bondad. Este salvado se realiza a través de tensorflow. Esto implica que para visualizar las distintas gráficas y datos recogidos debemos hacerlo desde una plataforma web llamada tensorboard. Una vez hemos accedido a este portal es muy intuitivo, sin embargo acceder a él no es algo tan trivial. En primer lugar debemos de situarnos con una terminal en la carpeta donde se haya realizado la salida del script, ésta estará contenida en el directorio log. Una vez ubicados en ella ejecutamos el siguiente comando: “tensorboard - - logdir=.”. Ahora tan solo tenemos que dirigirnos a la dirección web que nos indique, normalmente será <http://localhost:6006/>.

### 8.2.3. Entrenamiento del modelo basado en imágenes e inferencia

Aunque en apartados anteriores se ha comentado cómo se ha llevado a cabo la adaptación de la base de datos Caltech101 para que yolov5 trabaje

sobre ella, la base de datos parseada se puede obtener directamente del enlace a la carpeta compartida en drive([\[13\]](#)), por lo que al no tener que realizar este paso pasamos directamente a explicar cómo entrenar e inferir con este modelo.

En primer lugar, antes de nada, debemos registrarnos en la web weights and biases([\[110\]](#)). Este paso es importante porque, al igual que asynet utilizaba tensorboard para la visualización de gráficas y datos, yolo va a almacenar toda esta información y nos va a permitir visualizarla y obtenerla a través de esta plataforma.

Trabajar con esta versión de yolo es algo muy sencillo. Para entrenar el modelo simplemente ejecutaremos la orden “python train.py - - data <yaml\_file> - - cfg yolov5s.yaml - - weights” - - batch-size <tam\_batch>”. Tras esto, y antes de comenzar el primer entrenamiento que hagamos, nos pedirá que introduzcamos los datos de nuestra cuenta de wandb.

Con el modelo entrenado ya podemos realizar la inferencia sobre el conjunto test de la base de datos utilizada para entrenar mediante “python test.py - - data <yaml\_file> - - weights ‘<archivo pt generado tras el train>’ - - img <tam\_images>”. O por el contrario, inferir sobre una serie de imágenes independientes que especifiquemos, utilizando para ello “python detect.py - - <path\_image.jpg> - - weights ‘<archivo pt generado tras el train>’ - - img <tam\_images>”.

## 8.3. Enlaces de interés

### 8.3.1. Enlaces de descarga del proyecto

- Repositorio de github del proyecto: [\[87\]](#).
- Carpeta compartida en drive con algunos recursos: [\[13\]](#).

### 8.3.2. Enlaces de descarga de los distintos subsistemas utilizados

- Web de descarga de ROS Melodic: [\[74\]](#).
- Repositorio de github de dvs\_ros: [\[75\]](#).
- Repositorio de github de v2e: [\[28\]](#).
- Repositorio de github de asynet: [\[24\]](#).
- Repositorio de github de yolov5: [\[26\]](#).

## **Capítulo 9**

# **Conclusiones y futuras mejoras**

### **9.1. Conclusiones**

En este proyecto hemos estudiado y desarrollado algoritmos para la detección y el reconocimiento de objetos, así como la segmentación de éstos, a partir de los eventos y la información asíncrona en espacio y tiempo que nos proporcionan las cámaras basadas en eventos. Esta adaptación de los algoritmos de visión por computador tradicionales a las nuevas tecnologías permite su inclusión en sistemas robóticos, lo que supone un sistema de visión robótica más autónomo y eficiente. También se ha estudiado y explorado la utilización de simuladores de eventos, lo cual permite ahorrar costes y no tener que incluir nuevo hardware, lo que supondría un rediseñamiento del robot.

En cuanto a los objetivos que fueron planteados inicialmente, a continuación se expresa el estado en que se encuentran tras la finalización del proyecto.

Los primeros objetivos que se establecieron estaban referidos al estudio del estado del arte tanto de los sensores de visión dinámica, como de las técnicas de visión por computador aplicadas a los eventos proporcionados por éstos, los simuladores disponibles, y las redes neuronales dispersas. Con este estudio hemos puesto de manifiesto qué son, cómo funcionan y qué propiedades tienen los sensores dinámicos, así como sus ventajas con respecto a las cámaras convencionales basadas en fotogramas. También hemos indagado en los tipos de sensores de este tipo que existen hoy en día, si se comercializan o no y qué empresas son las desarrolladoras de cada uno de ellos. A continuación, en este documento expusimos qué son los simuladores de eventos y profundizamos en aquellos que consideramos que tienen especial

interés. Finalmente hemos realizado un repaso que nos lleva desde el aprendizaje automático y la visión por computador en general, hasta qué son y con qué propósito surgieron las redes neuronales convolucionales dispersas.

Como cuarto objetivo planteamos testear la segmentación de objetos que se realiza de forma implícita con la utilización de los sensores de eventos. En este sentido obtenemos una segmentación que podemos llamar temprana, o pseudo-segmentación, que solo funciona cuando la cámara permanece estática y son los distintos objetos los que se mueven. No podemos decir que sea una segmentación completa, puesto que solo funciona en esas situaciones concretas y habría que realizar un cierre de fronteras, pero es una muy buena aproximación que no requiere carga computacional adicional, ni consume ningún recurso, nos la proporciona la cámara de eventos implícitamente.

También nos planteamos implementar algoritmos que posibiliten realizar el reconocimientos de objetos a partir de estos eventos. Para tal propósito hemos implementado una serie de scripts y hemos adaptado el paquete asynet ([\[130\]](#)) desarrollado por el Department of Neuroinformatics of the University of Zurich and ETH Zurich. De esta forma hemos posibilitado no solo el entrenamiento de un modelo disperso, capaz de aprovechar la naturaleza asíncrona de las cámaras de eventos, sino también la inferencia tanto sobre los conjuntos de datos a los que se hace alusión en el repositorio [\[24\]](#) como sobre nuevos datasets que creemos nosotros mismos. También se ha realizado un estudio del sistema para comprobar experimentalmente qué resultados nos aporta utilizando una de las bases de datos disponibles, N-Caltech101([\[131\]](#)), para entrenar la red, y seleccionar la mejor combinación de valores para los distintos parámetros configurables.

Adicionalmente, ya que hemos hecho uso de un simulador para la obtención de los eventos a partir de un vídeo de entrada, y este simulador nos permite seleccionar la resolución de salida deseada, hemos comprobado experimentalmente cómo influye esta resolución en la generación de eventos y en la bondad de las predicciones que se realicen sobre ellos. Finalmente, implementamos un script capaz de crear un vídeo a partir de imágenes confeccionadas mediante acumulación de eventos y en las que aparecen resaltados los objetos detectados por el modelo.

A continuación nos propusimos probar y testear un modelo de reconocimiento de objetos convencional, basado en imágenes. Concretamente, hemos hecho uso del modelo YOLOv5 para comprobar los resultados que podemos obtener con él. Adicionalmente, se ha expuesto una comparación entre este modelo y el utilizado anteriormente para trabajar con eventos, en el que miramos tanto la eficacia como la eficiencia.

Como séptimo objetivo planteamos lo que para nosotros es más una obligación ética y moral que un acto de buena voluntad, la liberación de código para contribuir a la comunidad de software libre. En este sentido el proyecto

se encuentra alojado en un repositorio de GitHub, al que podemos acceder mediante el siguiente enlace [87] y se ha creado una carpeta compartida en drive, disponible para todo el que la necesite en [13], con la base de datos utilizada y algunos de los pesos utilizados.

Finalmente, nos propusimos dos objetivos orientados a la planificación a largo plazo y la comunicación entre miembros de un mismo equipo, dos aspectos completamente nuevos para mi. Esto lo hemos conseguido mediante el empleo de herramientas como las hojas de cálculo de Google y Slack, que permiten realizar estas tareas de formas sencilla y fluida.

## 9.2. Futuras mejoras

Como posibles futuras mejoras proponemos varias alternativas que ya se han mencionado brevemente en algún momento a lo largo de esta memoria.

Como primera alternativa planteamos la realización de un estudio comparativo entre el modelo sparse, que nosotros hemos utilizado, y el denso de VGG que se encuentra disponible en el propio repositorio de asynet [24]. Para ello sí es cierto que necesitaríamos un hardware que incluya una GPU con mayor capacidad de procesamiento y memoria que la que nosotros hemos tenido disponible.

Otra futura ampliación sería entrenar el modelo con una base de datos más adecuada para la generación del vídeo, como puede ser COCO, Common Object in Context. COCO es un dataset muy interesante porque nos ofrece más de 330000 imágenes pertenecientes a distintas clases, todas ellas representantes de objetos cotidianos, como puede ser una persona, un coche, una corbata, etc. Además permite reconocer múltiples objetos en una misma imagen y a distintas escalas. Sin embargo, esta base de datos no está disponible en eventos, solo en imágenes. Habría que generar los eventos de cada una de las distintas fotos de la misma manera que se ha hecho con Caltech101 y N-Caltech101, lo cual es un proceso lento que requiere del equipo adecuado.

Una futura ampliación que no se ha abordado en este TFG por falta de tiempo, pero que considero de interés y me gustaría realizar en un futuro, es la integración del software de reconocimiento de objetos mediante sensores de visión dinámica en un sistema robótico. Esto se haría mediante distintos nodos en ROS y dotaría al robot de un sistema de visión con una serie de propiedades únicas que le permitirían adaptarse a condiciones complicadas en cuanto a luminosidad.

Relacionado con el caso anterior, también proponemos integrar ,mediante nodos ROS nuevamente, un sistema de reconstrucción 3D de la escena a partir de la información proporcionada por los eventos. Esta funcionalidad

permitiría que una vez un objeto de nuestro interés ha sido detectado, podamos situarlo espacialmente y crear una especie de maqueta, o mapa de tres dimensiones a escala, del entorno que rodea al objeto.

Finalmente, siempre se debe estar atento a los nuevos avances que se puedan realizar en el estado del arte y en el desarrollo de cualquier sistema que mejore en rendimiento o eficiencia a los que se integran en este proyecto. Debido a la naturaleza modular, tanto de este trabajo, como de ROS, en caso de querer realizar alguna de las futuras mejoras aquí descritas, podremos sustituir un subsistema por otro sin alterar el resto de bloques.

# Bibliografía

- [1] *Añadir repositorios inivation:* <https://inivation.gitlab.io/dv/dv-docs/docs/getting-started.html#ubuntu-linux>.
- [2] *Anaconda:* <https://www.anaconda.com/products/individual>.
- [3] *Artículo en The Wall Street Journal de Thomas H. Davenport:* <https://www.wsj.com/articles/BL-CIOB-2765>.
- [4] *Artículo sueldo ingeniero informático:* [https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-informatico#:~:text=En%202019%20el%20salario%20de,profesionales%20mejor%20pagados%](https://universidadeuropea.com/blog/cuanto-gana-un-ingeniero-informatico#:~:text=En%202019%20el%20salario%20de,profesionales%20mejor%20pagados%20)
- [5] *Artículo vida útil de un PC:* <https://www.computerworld.es/archive/el-ciclo-de-vida-util-de-un-pc-ha-quedado-reducido-a-tan-solo-tres-anos-segun-idc>.
- [6] *Artículo vida útil de un servidor:* <https://www.curvature.com/es/resources/blog/giving-old-servers-a-new-lease-of-life#:~:text=Sin%20embargo%20seg%C3%A9n%20el%20informe,de%20los%20>
- [7] *Base de datos Caltech-101:* [http://www.vision.caltech.edu/Image\\_Datasets/Caltech101/](http://www.vision.caltech.edu/Image_Datasets/Caltech101/).
- [8] *Base de datos N-Caltech101 para clasificación:* [http://rpg.ifi.uzh.ch/datasets/gehrig\\_et\\_al\\_iccv19/N-Caltech101.zip](http://rpg.ifi.uzh.ch/datasets/gehrig_et_al_iccv19/N-Caltech101.zip).
- [9] *Base de datos N-Caltech101 para detección de objetos:* <https://www.garrickorchard.com/datasets/n-caltech101>.
- [10] *Base de datos NCars para clasificación:* [https://github.com/uzh-rpg/rpg\\_asynet/issues/20](https://github.com/uzh-rpg/rpg_asynet/issues/20).
- [11] *Base de datos Prophesee-Gen1 para detección de objetos:* <https://www.prophesee.ai/2020/01/24/prophesee-gen1-automotive-detection-dataset/>.

- [12] Cambiar resolución a un video: [https://video.online-convert.com/es/convertir-a-avi?external\\_url=https%3A%2F%2Fwww.online-convert.com%2Fes%2Fdownloadfile%2F4e237633-b0cb-496b-afe5-e54fe0943c19%2F6a5289f5-bb9a-4eb3-9421-429fe2c5325f](https://video.online-convert.com/es/convertir-a-avi?external_url=https%3A%2F%2Fwww.online-convert.com%2Fes%2Fdownloadfile%2F4e237633-b0cb-496b-afe5-e54fe0943c19%2F6a5289f5-bb9a-4eb3-9421-429fe2c5325f).
- [13] Carpeta compartida de drive con enlaces para el proyecto: <https://drive.google.com/drive/folders/1WubXaQkZuon3FBLn5Bcu9HsPq9f3PZHT?usp=sharing>.
- [14] Conexión con servidor: <https://blog.desdelinux.net/x11-forwarding-a-traves-de-ssh/>.
- [15] Convolución dispersa: <https://ichi.pro/es/como-funciona-la-convolucion-dispersa-3459496185809>.
- [16] Cuda visible device: <https://www.programmersought.com/article/3991738646/>.
- [17] Definición de IoU: <https://towardsdatascience.com/evaluating-performance-of-an-object-detection-model-137a349c517b>.
- [18] Definición de luminancia: <https://dle.rae.es/luminancia>.
- [19] Duda con asynet a la hora de hacer inferencia sobre un dataset entero: [https://github.com/uzh-rpg/rpg\\_asynet/issues/9](https://github.com/uzh-rpg/rpg_asynet/issues/9).
- [20] Duda con asynet al haber un bucle infinito al entrenar: [https://github.com/uzh-rpg/rpg\\_asynet/issues/6](https://github.com/uzh-rpg/rpg_asynet/issues/6).
- [21] Duda instalando ROS: [https://answers.ros.org/question/62589/problem-with-the-terminal-ros\\_distro/](https://answers.ros.org/question/62589/problem-with-the-terminal-ros_distro/).
- [22] Error con el script para pasar un video de eventos a bag con el modulo cv2: <https://stackoverflow.com/questions/45215755/opencv-python-cv2-cv-cap-prop-fps-error/45216118>.
- [23] Error instalando v2e con el paquete cryptodome: <https://github.com/IdentityPython/pysaml2/issues/389>.
- [24] github asynet: [https://github.com/uzh-rpg/rpg\\_asynet](https://github.com/uzh-rpg/rpg_asynet).
- [25] Github crear video a partir de imágenes: [https://github.com/antonioam82/ejercicios-python/blob/master/frames\\_to\\_video.py](https://github.com/antonioam82/ejercicios-python/blob/master/frames_to_video.py).
- [26] Github de yolov5 con pytorch: <https://github.com/ultralytics/yolov5/issues/36>.
- [27] github sparseConvNet: <https://github.com/facebookresearch/SparseConvNet>.
- [28] github v2e: <https://github.com/SensorsINI/v2e>.

- [29] *Guardar y cargar modelos con pytorch:* [https://pytorch.org/tutorials/beginner/saving\\_loading\\_models.html#what-is-a-state-dict](https://pytorch.org/tutorials/beginner/saving_loading_models.html#what-is-a-state-dict).
- [30] *Guardar y cargar modelos tensorflow:* [https://www.tensorflow.org/tutorials/keras/save\\_and\\_load](https://www.tensorflow.org/tutorials/keras/save_and_load).
- [31] *Guardar y serializar modelos tensorflow:* [https://www.tensorflow.org/guide/keras/save\\_and\\_serialize?hl=es-419](https://www.tensorflow.org/guide/keras/save_and_serialize?hl=es-419).
- [32] *Imagen de reconocimiento de objetos:* <https://hackernoon.com/how-visual-object-detection-can-transform-manufacturing-industries-8b6698cc0a47>.
- [33] *Imágenes de detección de objetos y segmentación:* [https://www.reddit.com/r/learnmachinelearning/comments/kt0hov/difference\\_in\\_image\\_classification/](https://www.reddit.com/r/learnmachinelearning/comments/kt0hov/difference_in_image_classification/)
- [34] *Imágenes de seguimiento y clasificación:* <https://blog.g2vp.com/the-state-of-computer-vision-f8c95401b462>.
- [35] *Información general aprendizaje automático:* [https://es.wikipedia.org/wiki/Aprendizaje\\_autom%C3%A1tico](https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico).
- [36] *Información general aprendizaje automático:* <https://www.apd.es/que-es-machine-learning/>.
- [37] *Información general aprendizaje automático:* <https://www.hpe.com/lamerica/es/what-is/machine-learning.html>.
- [38] *Información general aprendizaje automático:* [https://www.sas.com/es\\_es/insights/analytics/machine-learning.html](https://www.sas.com/es_es/insights/analytics/machine-learning.html).
- [39] *Información general aprendizaje automático:* <https://www.xataka.com/robotica-e-ia/machine-learning-y-deep-learning-como-entender-las-claves-del-presente-y-futuro-de-la-inteligencia-artificial>.
- [40] *Información general aprendizaje automático:* <http://www.cs.us.es/~fsancho/?e=75>.
- [41] *Información general cámaras basadas en eventos:* <https://medium.com/@nabil.madali/introduction-to-event-based-vision-d9cfa1d98264>.
- [42] *Información general de las SparseCNN:* <https://towardsdatascience.com/how-does-sparse-convolution-work-3257a0a8fd1>.

- [43] *Información general de usos de la visión por computador:* <https://becominghuman.ai/computer-vision-applications-in-self-driving-cars-610561e14118>.
- [44] *Información general de usos de la visión por computador:* <https://blog.vsoftconsulting.com/blog/top-usecases-of-computer-vision-in-manufacturing>.
- [45] *Información general de usos de la visión por computador:* <https://www.multitel.eu/expertise/computer-vision/intelligent-video-surveillance/>.
- [46] *Información general diagrama casos de uso:* [https://es.wikipedia.org/wiki/Caso\\_de\\_uso](https://es.wikipedia.org/wiki/Caso_de_uso).
- [47] *Información general diagrama casos de uso:* <https://slideplayer.es/slide/10753055/>.
- [48] *Información general diagrama casos de uso:* <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagrama-de-casos-de-uso/>.
- [49] *Información general redes neuronales artificiales:* [https://es.wikipedia.org/wiki/Red\\_neuronal\\_artificial](https://es.wikipedia.org/wiki/Red_neuronal_artificial).
- [50] *Información general redes neuronales convolucionales:* <https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8>.
- [51] *Información general redes neuronales convolucionales:* <https://cs231n.github.io/convolutional-networks/>.
- [52] *Información general redes neuronales convolucionales:* <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.
- [53] *Información general redes neuronales convolucionales:* <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>.
- [54] *Información general redes neuronales convolucionales:* <https://www.juanbarrios.com/redes-neurales-convolucionales/>.
- [55] *Información general sobre Anaconda:* <https://www.tokioschool.com/noticias/ciencia-datos-anaconda-python/>.
- [56] *Información general sobre el overfitting:* <https://es.wikipedia.org/wiki/Sobreajuste>.

- [57] *Información general sobre la función de pérdida crossentropyloss de pytorch:* <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>.
- [58] *Información general sobre la función de pérdida logSoftmax de pytorch:* <https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html#torch.nn.LogSoftmax>.
- [59] *Información general sobre la función de pérdida NLLLoss de pytorch:* <https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html#torch.nn.NLLLoss>.
- [60] *Información general sobre la metodología incremental:* <http://isw-udistrital.blogspot.com/2012/09/ingenieria-de-software-i.html>.
- [61] *Información general sobre la metodología incremental:* <https://blog.becas-santander.com/es/metodologias-desarrollo-software.html>.
- [62] *Información general sobre la metodología incremental:* <https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf>.
- [63] *Información general sobre los entornos virtuales de Anaconda:* <https://www.devacademy.es/entornos-virtuales-en-python-anaconda>.
- [64] *Información general sobre ROS:* [https://en.wikipedia.org/wiki/Robot\\_Operating\\_System](https://en.wikipedia.org/wiki/Robot_Operating_System).
- [65] *Información general sobre ROS:* <https://jjromeromarras.wordpress.com/2014/08/27/conceptos-basicos-de-ros/>.
- [66] *Información general sobre ROS:* <https://www.ros.org/about-ros/>.
- [67] *Información general sobre ROS:* <http://wiki.ros.org/es/ROS/Conceptos>.
- [68] *Información general visión por computador:* <https://blog.enzymeadvisinggroup.com/computer-vision-inteligencia-artificial>.
- [69] *Información general visión por computador:* <https://www.ibm.com/topics/computer-vision>.
- [70] *Información general visión por computador:* <http://www.ehu.eus/ccwintco/uploads/d/d4/PresentacionMundoVirtual.pdf>.
- [71] *Información Thomas H. Davenport:* [https://en.wikipedia.org/wiki/Thomas\\_H.\\_Davenport](https://en.wikipedia.org/wiki/Thomas_H._Davenport).
- [72] *Inicialización a tensorboard:* <https://pytorch.org/docs/stable/tensorboard.html>.
- [73] *Instalación Anaconda:* <https://www.digitalocean.com/community/tutorials/como-instalar-anaconda-en-ubuntu-18-04-quickstart-es>.

- [74] *Instalación de ROS*: <http://wiki.ros.org/melodic/Installation/Ubuntu>.
- [75] *Instalación rpg\_dvs\_ros*: [https://github.com/uzh-rpg/rpg\\_dvs\\_ros](https://github.com/uzh-rpg/rpg_dvs_ros).
- [76] *Introducción a tensorflow*: [https://www.tensorflow.org/tensorboard/get\\_started](https://www.tensorflow.org/tensorboard/get_started).
- [77] *Issue donde se proporciona un enlace alternativo para la descarga de eigen*: [https://github.com/uzh-rpg/rpg\\_asynet/issues/5](https://github.com/uzh-rpg/rpg_asynet/issues/5).
- [78] *Pasar un video de eventos a .bag*: <https://stackoverflow.com/questions/31432870/how-do-i-convert-a-video-or-a-sequence-of-images-to-a-bag-file>.
- [79] *Problema instalando rpg\_dvs\_ros con paquetes catkin-pkg*: <https://howtoinstall.co/es/python-catkin-pkg>.
- [80] *Problema instalando rpg\_dvs\_ros con paquetes catkin-pkg*: <https://stackoverflow.com/questions/61148849/unmet-dependencies-unable-to-install-python-catkin-pkg>.
- [81] *Problema instalando rpg\_dvs\_ros con paquetes catkin\_tools*: <https://answers.ros.org/question/353113/catkin-build-in-ubuntu-2004-noetic/>.
- [82] *Problema instalando rpg\_dvs\_ros con paquetes catkin\_tools*: [https://github.com/catkin/catkin\\_tools/issues/611](https://github.com/catkin/catkin_tools/issues/611).
- [83] *Problema instalando rpg\_dvs\_ros con paquetes ros anaconda*: <https://anaconda.org/conda-forge/ros-ros>.
- [84] *Recursos sobre las camaras de eventos*: [https://github.com/uzh-rpg/event-based\\_vision\\_resources](https://github.com/uzh-rpg/event-based_vision_resources).
- [85] *Relación entre la visión por computador y el machine learning*: <https://machinelearningmastery.com/what-is-computer-vision/>.
- [86] *Relación entre la visión por computador y el machine learning*: [https://www.researchgate.net/figure/Venn-diagram-showing-the-relationship-between-artificial-intelligence-and-other\\_fig4\\_336375517](https://www.researchgate.net/figure/Venn-diagram-showing-the-relationship-between-artificial-intelligence-and-other_fig4_336375517).
- [87] *Repositorio de github del proyecto*: <https://github.com/nonoaroca/Deteccion-de-objetos-utilizando-retinas-artificiales>.
- [88] *Repositorio de SuperSloMo*: <http://jianghz.me/projects/superslomo/>.
- [89] *Repositorio del simulador davis\_simulator*: [https://github.com/uzh-rpg/rpg\\_davis\\_simulator](https://github.com/uzh-rpg/rpg_davis_simulator).
- [90] *Repositorio del simulador ESIM*: [https://github.com/uzh-rpg/rpg\\_esim](https://github.com/uzh-rpg/rpg_esim).

- [91] *Repositorio del simulador Microsoft AirSim:* <https://github.com/microsoft/AirSim>.
- [92] *Slomo:* <http://jianghz.me/projects/superslomo/>.
- [93] *State\_dicts de pytorch:* [https://pytorch.org/tutorials/recipes/recipes/what\\_is\\_state\\_dict.html](https://pytorch.org/tutorials/recipes/recipes/what_is_state_dict.html).
- [94] *Submanifold sparse convolutional networks:* <https://towardsdatascience.com/submanifold-sparse-convolutional-networks-b0c54f07f4e3>.
- [95] *Tipos de cámaras de eventos:* [https://en.wikipedia.org/wiki/Event\\_camera](https://en.wikipedia.org/wiki/Event_camera).
- [96] *Tutorial on Event-based Vision for High-Speed Robotics:* <https://www.rit.edu/kgcoe/iros15workshop/papers/IROS2015-WASRoP-Invited-04-slides.pdf>.
- [97] *Tutorial yolov5 con pytorch con tu propia base de datos de entrenamiento:* <https://github.com/ultralytics/yolov5/issues/12>.
- [98] *Ubuntu18:* <https://releases.ubuntu.com/18.04.5/>.
- [99] *Unidad de medida inercial:* [https://en.wikipedia.org/wiki/Inertial\\_measurement\\_unit](https://en.wikipedia.org/wiki/Inertial_measurement_unit).
- [100] *Vídeo de High Speed and High Dynamic Range Video with an Event Camera del UZH Robotics and Perception Group:* <https://www.youtube.com/watch?v=eomALySSGVU>.
- [101] *Video diferencia cámaras tradicionales y de eventos:* <https://www.youtube.com/watch?v=LauQ6LWTkxM&t=35s>.
- [102] *VirtualBox:* <https://www.virtualbox.org/wiki/Downloads>.
- [103] *Visualizar datos y estadísticas con tensorboard:* [https://pytorch.org/tutorials/intermediate/tensorboard\\_tutorial.html](https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html).
- [104] *Web con datasets para davis:* [http://rpg.ifi.uzh.ch/davis\\_data.html](http://rpg.ifi.uzh.ch/davis_data.html).
- [105] *Web crear video a partir de imágenes:* <https://programacionpython80889555.wordpress.com/2020/10/08/creando-video-a-partir-de-imagenes-en-python-con-opencv/>.
- [106] *Web de Inivation:* <https://inivation.com/>.
- [107] *Web de Prophesse:* <https://www.prophesee.ai/>.
- [108] *Web de Samsung España:* <https://www.samsung.com/es/>.
- [109] *Web de venta oficial de Nvidia:* <https://shop.nvidia.com/es-es/geforce/store/gpu/?page=1&limit=9&locale=es-es&category=GPU&gpu=RTX%202080%20Ti>.

- [110] *Web de weights and biases: <https://wandb.ai/site>.*
- [111] *Web de Weights and Biases: <https://wandb.ai/site>.*
- [112] *Web de yolov5 con pytorch: [https://pytorch.org/hub/ultralytics\\_yolov5/](https://pytorch.org/hub/ultralytics_yolov5/).*
- [113] *Web del dataset CIFAR-10: <http://www.cs.toronto.edu/~kriz/cifar.html>.*
- [114] *Web del dataset COCO: <https://cocodataset.org/#home>.*
- [115] *Web del dataset ImageNet: <https://image-net.org/>.*
- [116] *Web del dataset MNIST: <http://yann.lecun.com/exdb/mnist/>.*
- [117] *Web del Department of Informatics, Institute of NeuroInformatics and Robotics Perception Group de la University of Zurich y el ETH Zurich: [http://rpg.ifi.uzh.ch/research\\_dvs.html](http://rpg.ifi.uzh.ch/research_dvs.html).*
- [118] *web v2e: <https://sites.google.com/view/video2events/home>.*
- [119] Amit D.Kachare A.D.Dongare, R.R.Kharde. Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2008.
- [120] Hassan Foroosh Marshall Tappen Baoyuan Liu, Min Wang and Ma-rianna Penksy. Sparse convolutional neural networks. 2015.
- [121] M. Yang S. C. Liu C. Brandli, R. Berner and T. Delbruck. A  $240 \times 180$  130 db 3 us latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 2014.
- [122] D. Scaramuzza E. Mueggler, B. Huber. Event-based, 6-dof pose tracking for high-speed maneuvers. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014.
- [123] G. Gallego T. Delbruck D. Scaramuzza E. Mueggler, H. Rebecq. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *International Journal of Robotics Research*, 2017.
- [124] Delbruck T. Orchard G. Bartolozzi C. Taba B. Censi A. Leutenegger S. Davison A. Conradt J. Daniilidis K. Scaramuzza D. Gallego, G. Event-based vision: A survey. *IEEE Trans. Pattern Anal. Machine Intell. (TPAMI)*, 2020.
- [125] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. *CVPR*, 2018.

- [126] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017.
- [127] Elias Mueggler Davide Scaramuzza Henri Rebecq, Guillermo Gallego. Emvs: Event-based multi-view stereo—3d reconstruction with an event camera in real-time. *International Journal of Computer Vision*, 2018.
- [128] Ross Girshick Ali Farhadi Joseph Redmon, Santosh Divvala. You only look once: Unified, real-time object detection.
- [129] Jean-Matthieu Maro. Event-based gesture recognition with dynamic background suppression using smartphone computational capabilities. 2018.
- [130] Nico Messikommer, Daniel Gehrig, Antonio Loquercio, and Davide Scaramuzza. Event-based asynchronous sparse convolutional networks. *European Conference on Computer Vision. (ECCV)*, 2020.
- [131] G.; Jayawant A.; Orchard, G.; Cohen and N. Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, vol.9, no.437, 2015.
- [132] T. Delbruck P. Lichtsteiner, C. Posch. A  $128 \times 128$  120db 15us latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid State Circuits*, 2008.
- [133] Eduardo A. B. da Silva Rafael Padilla, Sergio L. Netto. A survey on performance metrics for object-detection algorithms. *International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020.
- [134] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*, 2018.
- [135] Saad AL-ZAWI Saad ALBAWI, Tareq Abed MOHAMMED. Understanding of a convolutional neural network. 2017.
- [136] S-C. Liu Y. Hu and T. Delbruck. v2e: From video frames to realistic dvs events. *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021.



