Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Monitorización de los ocupantes del vehículo y reconocimiento de la actividad del conductor basada en puntos clave

ESCUELA POLITECNICA SUPERIOR

Autor: Pablo Pardo Decimavilla

Tutor: Luis M. Bergasa Pascual

2023

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Monitorización de los ocupantes del vehículo y reconocimiento de la actividad del conductor basada en puntos clave

Autor: Pablo Pardo Decimavilla

Tutor: Luis M. Bergasa Pascual

Tribunal:

Presidente: Miguel González Herráez

Vocal 1º: Mª Soledad Escudero Hernanz

Vocal 2º: Luis M. Bergasa Pascual

Suplente: Ignacio Fernández Lorenzo

Fecha de depósito: 3 de julio de 2023

A mis padres, amigos y familia ...

"Siempre parece imposible... hasta que se hace"
Nelson Mandela

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que me han brindado su apoyo y han contribuido directa e indirectamente de manera significativa a la realización de este Trabajo Fin de Grado.

En primer lugar, quiero agradecer a mis padres, quienes han estado a mi lado durante toda mi formación académica. Su amor, dedicación y constante apoyo han sido fundamentales para mi éxito. Siempre han creído en mí y han sido mi fuente de inspiración para alcanzar mis metas. No puedo expresar con palabras lo agradecido que estoy por tenerlos como mis padres.

A mis amigos, quiero agradecerles por su incondicional amistad y por estar siempre presentes en mi vida. Sus palabras de aliento, su compañía y su motivación han sido esenciales en los momentos difíciles. Juntos hemos superado desafíos y celebrado éxitos, y no puedo imaginar este viaje sin su apoyo.

A mis compañeros de RobeSafe; Santiago, Miguel, Fabio, Navil, Carlos, Rodri, Javier y Felipe, quiero expresar mi gratitud por el valioso tiempo que hemos compartido a lo largo de este proyecto. Han sido una parte fundamental en el desarrollo de este trabajo. Cada uno ha colaborado de una forma u otra y estoy agradecido por su apoyo.

A mis profesores, quiero agradecerles por su dedicación y orientación durante mi trayectoria académica. Su pasión por la enseñanza, su conocimiento experto y su disposición para ayudarme han sido esenciales. Han estimulado mi curiosidad y me han brindado las herramientas necesarias para desarrollarme como estudiante y como profesional en el campo de estudio.

Además, quiero expresar un agradecimiento especial a mi tutor, Luis M. Bergasa Pascual, por su dedicación, orientación y apoyo a lo largo de este trabajo. Su experiencia y conocimientos han sido fundamentales en la estructuración y desarrollo de mi investigación. Siempre estuvo dispuesto a escuchar mis ideas, brindarme retroalimentación constructiva y guiarme en cada etapa del proceso. Estoy profundamente agradecido por su tiempo y dedicación, y por haberme brindado la oportunidad de crecer académicamente bajo su tutela

Finalmente, quisiera extender mi agradecimiento a todas las personas que, de una forma u otra, han contribuido a este logro. Su apoyo moral y consejos han sido fundamentales para mantenerme enfocado y motivado a lo largo de este camino.

¡Gracias!

Resumen

Se presenta una arquitectura modular basada en la fusión de puntos clave y detección de objetos para detectar distracciones al volante. Utilizando grabaciones multicámara, se extrae la pose del conductor, y se fusiona con la detección de objetos de interés para generar características. Un algoritmo de clasificación basado en *Machine Learning* procesa estas características, aplicando filtros para agregar contexto temporal. La propuesta se valida en dos *datasets* sobre distracciones al volante, demostrando mejoras en la inferencia de acciones y adaptabilidad al dominio. La arquitectura es ligera, explicativa y de baja latencia, permitiendo actualizaciones modulares y eliminación de cámaras sin modificar la red completa.

Palabras clave: Machine Learning, Detección de Objetos, Estimación de Pose, Detección de Distracciones del Conductor, Coches Autónomos.

Abstract

A modular architecture based on key point fusion and object detection is presented to detect distractions while driving. Using multi-camera recordings, the driver's pose is extracted, fusing them with the detection of objects of interest to generate features. A classification algorithm based on Machine Learning processes these features, applying filters to add temporal context. The proposal is validated on two datasets on driving distractions, demonstrating improvements in action inference and domain adaptability. The architecture is lightweight, explanatory and low latency, allowing modular updates and camera removal without modifying the entire network.

Keywords: Machine Learning, Object Detection, Pose Estimation, Driver Distractions Detection, Autonomous Cars.

Resumen extendido

El reconocimiento de las distracciones del conductor desempeña un papel fundamental en la seguridad vial. La Dirección General de Tráfico (DGT) afirma que uno de cada tres accidentes son causados por la inatención a la carretera. La Unión Europea (UE) es consciente de este problema y es por ello que ha regulado la obligatoriedad de incluir sistemas de alerta a las distracciones. En este marco se desarrolla este TFG presentando una arquitectura modular basada en la fusión de puntos clave y detección de objetos para predecir las acciones del conductor. A partir de grabaciones multicámara infrarrojas, detectaremos temporalmente una variedad de acciones que conducen a distracciones. Nuestro sistema detecta a las personas en el interior de un vehículo y los objetos de interés existentes, como los teléfonos móviles, y extrae puntos clave de la pose del conductor. Toda esta información se fusiona generando características que los relacionan y se procesan con un algoritmo de clasificación basado en *Machine Learning*. Se trata de distancias y ángulos entre los diferentes puntos clave y entre los objetos detectados y partes del cuerpo del conductor con probabilidad de interactuar entre ellos. Por último, se aplican filtros para añadir contexto temporal a las detecciones. Al analizar la secuencia de vídeo fotograma por fotograma se pierde mucha información acerca de las detecciones pasadas, es por ello que es necesario aplicarlos.

Nuestra propuesta se ha validado en dos conjuntos de datos del estado del arte sobre distracciones al volante. A través de varios experimentos demostramos que la fusión mejora sustancialmente la inferencia de acciones relacionadas y la adaptación al dominio respecto a otras propuestas de la literatura. Además, nuestro marco es ligero, explicable y tiene una baja latencia, ya que realiza la inferencia fotograma a fotograma. La modularidad de la red nos permite actualizar partes de forma independiente o eliminar una cámara sin tener que modificar toda la red. Se puede modificar el tipo de datos de entrada, como imágenes de profundidad o RGB, simplemente modificando los pesos del estimador de poses y el detector de objetos sin tener que volver a entrenar la red con un conjunto de datos especializado.

Índice general

Agradecimientos **vii**

Resumen **ix**

Abstract **xi**

Resumen extendido **xiii**

Índice general **xv**

Índice de figuras **xix**

Índice de tablas **xxi**

Índice de algoritmos **xxiii**

1 Introducción **1**

1.1 Motivación 1

1.1.1 Normativa europea 2

1.2 Niveles de autonomía 3

1.3 Visión Artificial y Deep Learning 5

1.4 Estructura general del proyecto 5

2 Estado del Arte **7**

2.1 Introducción 7

2.2 Detección de distracciones al volante 7

2.3 Detección de objetos 9

2.3.1 EfficientDet 9

2.3.2 RCNN 10

2.3.3 DETR 11

2.3.4 YOLO 12

2.3.5 Comparativa de detectores de objetos 14

2.4 Estimación de la pose 14

2.4.1 OpenPose 16

2.4.2 DensePose (R-CNN) 16

2.4.3 MoveNet 17

2.4.4 AlphaPose 18

2.4.5 Comparativa de estimadores de pose . . . 19
2.5 Clasificador XGBoost . . . 19

3 Herramientas software **21**
3.1 Datasets . . . 21
3.1.1 COCO . . . 21
3.1.2 SVIRO . . . 22
3.1.3 TICaM . . . 24
3.1.4 Track 3 *dataset* del AI City Challenge 2023 . . . 25
3.1.5 Cairo Distracted Driver Dataset . . . 26
3.2 Docker . . . 27
3.3 Librerías de Python . . . 27
3.3.1 PyTorch . . . 28
3.3.2 Scikit-learn . . . 28
3.3.3 OpenCV . . . 28
3.3.4 NumPy . . . 29
3.3.5 Matplotlib . . . 29
3.4 CUDA . . . 29
3.5 Otras herramientas complementarias . . . 30

4 Desarrollo **31**
4.1 Introducción . . . 31
4.1.1 Hipótesis e ideas de mejora . . . 32
4.2 Implementación de la detección de objetos . . . 33
4.2.1 Formato de las anotaciones . . . 33
4.2.2 Métricas . . . 36
4.2.3 Resultados de la validación . . . 38
4.2.4 Entrenamiento . . . 39
4.3 Estimador de poses . . . 40
4.3.1 Integración de YOLO con AlphaPose . . . 41
4.3.2 Creación de script combinado . . . 43
4.4 Integración de la estimación de la pose con KNDAR . . . 43
4.5 Generación de características . . . 44
4.6 Entrenamiento del clasificador . . . 45
4.7 Mejora postprocesado . . . 48
4.7.1 Filtro Paso Bajo . . . 49
4.7.2 Duración mínima de la acción . . . 50
4.8 Adaptación código a tiempo real . . . 51
4.9 Cumplimiento de la normativa europea . . . 52

5 Resultados **55**
5.1 Métricas . . . 55
5.2 Resultados globales . . . 56

5.3 Estudio de ablación 59
5.3.1 Cámaras 59
5.3.2 Fusión 59

6 Conclusiones y líneas futuras 61
6.1 Conclusiones 61
6.2 Líneas futuras 62

Bibliografía 65

Apéndice A Algoritmos 69
A.1 Non Max Suppression 69
A.2 Post-procesado de las confianzas de las predicciones de distracción 70

Apéndice B Características extraidas de los puntos clave del conductor 71

Apéndice C Código resumido del sistema en tiempo real 73

Apéndice D Requisitos técnicos específicos para un sistema ADDW 75
D.1 Control del sistema ADDW 75
D.2 Condiciones medioambientales 76
D.3 Control de las distracciones del conductor 76
D.4 Requisitos de la interfaz hombre-máquina 77
D.4.1 Advertencia visual 77
D.4.2 Aviso acústico 78
D.4.3 Advertencia háptica 78
D.5 Aviso de fallo del sistema ADDW 78
D.5.1 Fallos permanentes 78
D.5.2 Fallos temporales 78

Apéndice E Presupuesto 81
E.1 Recursos Hardware y Software 81
E.2 Recursos Humanos 81

Índice de figuras

1.1 Distracciones al volante por el uso del *smartphones* 2
1.2 Representación gráfica de los niveles de autonomía 4
1.3 Ejemplo de red neuronal ampliamente utilizada en visión por computador (AlexNet 2012) 6
1.4 Arquitectura de la solución propuesta 6

2.1 Estado del arte en clasificación de imágenes evaluado en Imagenet 9
2.2 Estructura de EfficientDet 10
2.3 Funcionamiento de R-CNN 10
2.4 Flujo de trabajo de Fast R-CNN 11
2.5 Comparación de tiempo de inferencia entre los tres componentes de la familia RCNN 11
2.6 Funcionamiento de DETR 12
2.7 Comparación de tiempo de inferencia entre los tres componentes de la familia RCNN 12
2.8 Ejemplo de aplicación del algoritmo NMS 13
2.9 Comparación entre las versiones 5, 6, 7 y 8 de YOLO 13
2.10 Diferentes tipos de modelos de estimación de poses 15
2.11 Representación gráfica de los puntos clave en un corredor 16
2.12 Flujo de trabajo de OpenPose 17
2.13 Ejemplos de inferencia de la red DensePose 17
2.14 Arquitectura de MoveNet 18
2.15 Arquitectura de AlphaPose 19
2.16 Arquitectura general del clasificador XGBoost 20

3.1 Ejemplo de imágenes del *dataset* COCO para la detección de objetos 23
3.2 Ejemplo de imágenes del *dataset* COCO para la estimación de la pose 23
3.3 Ejemplo de imágenes del *dataset* SVIRO 24
3.4 Simulador para la captura de imágenes del *dataset* TICaM 24
3.5 Ejemplo de imágenes reales del *dataset* TICaM 25
3.6 Ejemplo de imágenes sintéticas del *dataset* TICaM 25
3.7 Ejemplo de imágenes del *dataset* Track 3 del AI City Challenge 2023 26
3.8 Ejemplo de imágenes reales y en simulador del *dataset* Cairo Distracted Driver . 27
3.9 Esquema del ciclo de trabajo con Docker 28
3.10 Ejemplos de gráficos creados con Matplotlib 29

4.1 Flujo de trabajo de KNDAR 32
4.2 Estructura de la propuesta de mejora de KNDAR 33
4.3 Representación del formato de las cajas en YOLO 34
4.4 Comparación entre el formato YOLO y el formato COCO para detección de objetos 34
4.5 Ejemplo de *bounding box* de la detección de un gato 35
4.6 Impacto del umbral de IOU para la elección de un verdadero o falso positivo . . 36
4.7 Ejemplo de una curva Precision-Recall para cálculo de mAP 38
4.8 Curvas del re-entrenamiento de YOLOv8 40
4.9 Matriz de confusión para la clase *person* en el modelo YOLOv8-m afinado. 40
4.10 Relación mAP@.5:.95 y tiempo de inferencia para cada una de los modelos validados y entrenados 41
4.11 Posible localización de conductor y pasajero en un coche 42
4.12 Esquema gráfico del flujo de datos dentro del script 43
4.13 Representación gráfica y valores de los puntos clave en formato COCO. 44
4.14 Distancias del centro del volante y del teléfono móvil a muñecas y orejas (sólo para el teléfono móvil). 45
4.15 Distancias del centro de la botella de agua a las muñecas del conductor. 46
4.16 Resultados de F1 en testing para diferentes valores de *n_estimators* y *max_depth* 48
4.17 Gráficas de a) la función de pérdidas y b) el error del modelo a lo largo del entrenamiento 48
4.18 Salida de la señal de la acción 2 antes y después del filtro de post-procesado. . . 50
4.19 Salida de la señal de todas las acciones excepto la 2 antes y después del filtro de post-procesado. 50
4.20 Fotograma de la demo de la implementación del TFG en tiempo real 52
4.21 Diagrama de flujo de la implementación del proyecto en tiempo real. 53
4.22 Diagrama de estados del sistema ADDW propuesto 54
4.23 Esquema de la implementación del sistema ADDW propuesto 54

5.1 Histograma comparativo de la puntuación OS en las diferentes acciones entre la línea de base y nuestro método. 56

Índice de tablas

2.1 Estado del arte de los principales detectores de objetos evaluados en el *dataset* COCO 14
2.2 Estado del arte de estimación de la pose evaluados en en *dataset* COCO 19

3.1 Información del formato de los *datasets* 22
3.2 Anotaciones incluidas en los *datasets* 22
3.3 Distracciones 26

4.1 Tabla comparativa de las versiones 3, 5 y 8 de YOLO evaluados en SVIRO y TiCAM 38
4.2 Tabla comparativa de los modelos s, m y l de YOLOv8 entrenados y validados en SVIRO y TiCAM 39
4.3 Evaluación del modelo para diferentes parámetros del clasificador 47
4.4 Tabla comparativa de las longitudes de cada acción en el conjunto de datos de prueba 51

5.1 Puntuación OS para acciones relevantes entre la línea de base y nuestra propuesta. 57
5.2 Resultado de los 15 primeros en el *AI City Challenge Track3* en el año 2022 y 2023. 58
5.3 Diferentes métodos de clasificación de distracciones validados en el *Dataset* II . . 58
5.4 Comparación de la generalización de modelos 59
5.5 OS score para el *set* de *testing* del Dataset I. 59
5.6 Comparación del rendimiento de las acciones que intervienen en la fusión de la detección de objetos y puntos clave con el filtro original 60

E.1 Componentes y precio del equipo de trabajo. 81
E.2 Desglose de precio y dedicación a las principales tareas del proyecto. 82
E.3 Coste económico total del proyecto 82

Índice de algoritmos

A.1 Algoritmo Non Max Suppression 69
A.2 Filtro de contexto para predicciones de acción 70

Capítulo 1

Introducción

Todo logro comienza con la decisión de intentarlo

Gail Devers

1.1 Motivación

En 2021 se produjeron 921 accidentes mortales en las carreteras españolas en los que fallecieron 1.004 personas y 3.728 resultaron heridas graves [1]. Se calcula que en Europa murieron 19.800 personas en accidentes de tráfico [2]. Uno de cada tres accidentes se debe a la falta de atención al volante, por ejemplo al utilizar el teléfono móvil, manipular el navegador, comer, fumar o por fatiga o estrés. Se podrían salvar muchas vidas si se redujeran las distracciones. El uso de teléfonos móviles, especialmente los *smartphones*, ha generado preocupación en cuanto a la seguridad vial. Estos dispositivos ofrecen diversas aplicaciones que pueden distraer la atención de los conductores, como mapas, información del tráfico y comunicaciones. Un estudio realizado por el Institute of Advanced Motorists [3] demostró los efectos negativos de utilizar *smartphones* mientras se conduce. Se encontró que el tiempo de distracción ha aumentado significativamente, pudiendo estar entre el 40 % y 60 % al utilizar el teléfono. Además, situaciones como la que se muestra en la Figura 1.1 aumentan en un 38 % el tiempo de reacción, provocando dificultad para responder rápidamente a cambios en la velocidad del vehículo de adelante. Los coches completamente autónomos solucionan drásticamente esta problemática, pero todavía no son una realidad y será necesario una tecnología que de solución en la evolución hasta la autonomía completa.

Es en este marco donde se encuadra este Trabajo Fin de Grado (TFG). Se tendrá una monitorización constante del conductor a través de cámaras. Será una arquitectura de tiempo real la encargada de procesar todas las imágenes y estimar la posible distracción en cada momento. En el caso de coches con un nivel de autonomía bajo se avisará al conductor, ya sea con señales luminosas o visuales, de que debe cesar la distracción. Para el caso de vehículos con un nivel de autonomía medio-alto esta tecnología también aplica. Por un lado servirá para realizar una transición autónoma-manual segura. Por otro lado, si se detecta una situación peligrosa y el

Figura 1.1: Distracciones al volante por el uso del *smartphones*
Fuente: www.dgt.es

conductor no se encuentra atento a la vía, el vehículo podrá tomar el control y evitar un posible accidente.

1.1.1 Normativa europea

En la página oficial de la Unión Europea [4] encontramos el borrador del Reglamento Delegado de la Comisión (UE) que complementa el Reglamento (UE) 2019/2144 del Parlamento Europeo y del Consejo en lo que respecta a los sistemas avanzados de advertencia de distracción del conductor. El borrador establece las reglas detalladas sobre los procedimientos de prueba específicos y los requisitos técnicos para la homologación de ciertos vehículos con respecto a sus sistemas avanzados de advertencia de distracción del conductor. El sistema ADDW (Advanced Driver Distraction Warning) debe determinar cuándo la atención visual del conductor no está dirigida hacia las tareas de conducción y alertar al conductor a través de la interfaz hombre-máquina del vehículo. Teniendo en cuenta la gran variedad de características de los conductores, la variabilidad de las posiciones de los asientos y la relativa inmadurez del mercado de las tecnologías existentes, los requisitos de rendimiento de los sistemas ADDW se han fijado en un nivel realista y alcanzable con respecto a la limitada experiencia acumulada de los sistemas en el mercado y el margen necesario para una mayor innovación de estos sistemas. El reglamento establece que los vehículos de las categorías M[1] y N[2] deben estar equipados con sistemas avanzados de advertencia de distracción del conductor (ADDW) a partir del **7 de julio de 2024** para los nuevos diseños y a partir del **7 de julio de 2026** para todos los vehículos nuevos.

En el anexo del borrador se detallan los requisitos técnicos generales y específicos para cumplir con la normativa europea. En relación a los generales se establecen los siguientes:

- El sistema ADDW determinará cuándo la atención visual del conductor no está dirigida a las tareas de conducción y alertará al conductor a través de la interfaz hombre-máquina del vehículo.
- El proyecto estará diseñado para minimizar la tasa de error del sistema (falsos positivos) en condiciones reales de conducción.

[1] vehículo convencional para el transporte de personas y su equipaje
[2] vehículo para transporte de mercancías

- Privacidad y protección de datos:
  - El sistema ADDW funcionará sin basarse en datos biométricos personales resultantes de un tratamiento técnico específico relativo a las características físicas, fisiológicas o de comportamiento de una persona, que permitan o confirmen la identificación única de dicha persona, como imágenes faciales o datos dactiloscópicos, de cualquiera de los ocupantes del vehículo. Este requisito no prohíbe que el sistema ADDW utilice datos de la cámara o cámaras equipadas en el vehículo, sino que prohíbe la identificación de la persona por el sistema ADDW.
  - El sistema ADDW se diseñará de forma que sólo registre y conserve continuamente los datos necesarios para que el sistema funcione y opere dentro de un sistema en bucle cerrado.
  - Todo tratamiento de datos personales se llevará a cabo de conformidad con la legislación de la UE en materia de protección de datos.

Seguidamente en el anexo del reglamento se detallan los requisitos técnicos específicos, correspondientes a la implementación y diseño. Velocidad a la que se activa el sistema, forma de aviso, ángulo de situación de la cámara, son ejemplos de este apartado. En el borrador se estudia únicamente la mirada del conductor para determinar si está atento a la vía o no. En nuestro proyecto se realiza una re-interpretación mediante el estudio de las acciones a través del análisis de la posición corporal completa del conductor. En muchas ocasiones, esta forma de abordar el problema ofrece una perspectiva más completa. En la ejecución de este proyecto se tomará en consideración la normativa correspondiente con el fin de establecer una propuesta de implementación acorde a las recomendaciones y regulaciones europeas en este campo. En el Apéndice D de este TFG se detallan los requisitos técnicos específicos de una manera resumida. Además en el capítulo 4.9 se propondrá la integración de nuestro sistema según esos estándares.

1.2 Niveles de autonomía

Los niveles de autonomía en los coches autónomos se definen según la capacidad del vehículo para operar sin intervención humana en diferentes situaciones de conducción. Estos niveles se establecieron por la Sociedad de Ingenieros de Automoción (SAE) [5] y son ampliamente aceptados en la industria automotriz. Estos definen el nivel de intervención que puede tener un sistema dentro del vehículo. Además una misma tecnología en un vehículo con un bajo nivel de autonomía puede servir simplemente de aviso. Por el contrario en uno con alto nivel puede tomar decisiones importantes. Encontramos un total de seis niveles representados en la Figura 1.2 y definidos a continuación:

- **Nivel 0**: Sin automatización: En este nivel, el conductor es responsable de todas las tareas de conducción. El vehículo no tiene ninguna función de asistencia avanzada y todas las acciones, como el frenado y la dirección, deben ser realizadas por el conductor.

- **Nivel 1**: Asistencia al conductor: En este nivel, el vehículo tiene ciertas funciones de asistencia, como control de crucero adaptativo o asistencia de mantenimiento de carril. Sin embargo, el conductor sigue siendo responsable de la conducción y debe estar atento en todo momento.

- **Nivel 2**: Automatización parcial: Aquí, el vehículo puede llevar a cabo algunas tareas de conducción de manera autónoma, como acelerar, frenar y mantenerse dentro del carril. Aunque el sistema puede realizar estas funciones, el conductor debe supervisar activamente el entorno y estar preparado para intervenir si es necesario.

- **Nivel 3**: Automatización condicional: En este nivel, el vehículo es capaz de conducir de forma autónoma en ciertas circunstancias y entornos predefinidos. El conductor puede ceder temporalmente el control y realizar otras tareas, pero debe estar preparado para retomar el control si el sistema solicita intervención.

- **Nivel 4**: Alta automatización: En este nivel, el vehículo puede operar de manera autónoma en la mayoría de las situaciones de conducción, incluso si el conductor no puede retomar el control de inmediato. Sin embargo, el alcance de la autonomía puede estar limitado a áreas geográficas específicas o condiciones climáticas particulares.

- **Nivel 5**: Automatización completa: En el nivel más alto de autonomía, el vehículo es completamente autónomo y puede manejar todas las condiciones de conducción sin intervención humana. No se requiere la presencia de un conductor y el vehículo puede funcionar en cualquier entorno o situación de tráfico.

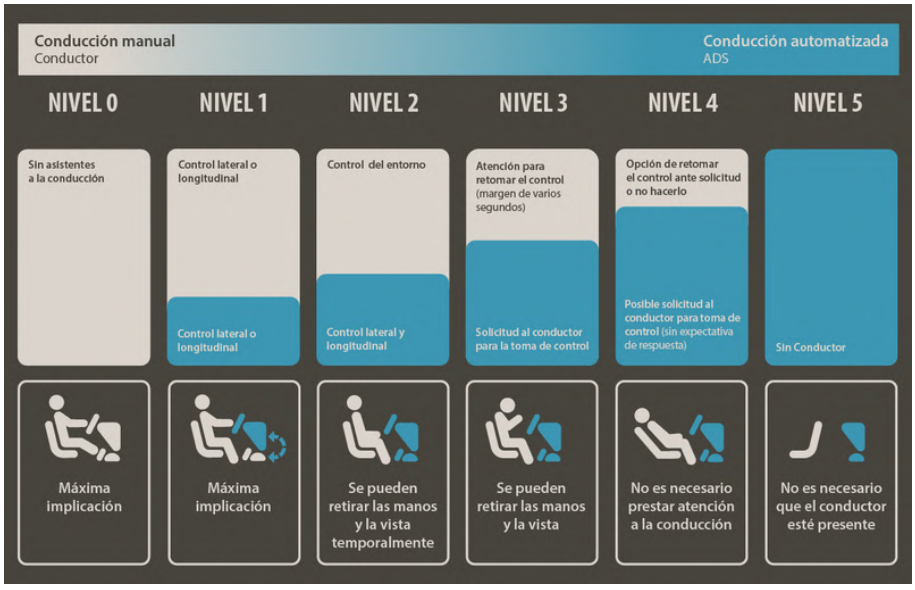


Figura 1.2: Representación gráfica de los niveles de autonomía

Para el nivel 0 y 1 el sistema servirá únicamente de aviso, ya que el conductor está estrictamente obligado a prestar atención a la vía. En ningún momento el control es tomado por el vehículo. Las distracciones son comunes, por ello el sistema ayudará a reducir drásticamente accidentes derivados de la inatención. Cuando sea detectado se avisará al conductor para que

cese la distracción. Para los niveles 2, 3 y 4 el sistema se encargará de conocer la atención del conductor para saber si es posible que éste retome el control del vehículo. Cuando no sea posible continuar con la conducción autónoma se le avisará al conductor de que es necesario que tome el control, pero hasta que el vehículo no se cerciora de que el conductor se encuentra atento para tomar el control no se procederá al relevo. Si el conductor ignora las indicaciones el vehículo tomará la decisión más segura en ese momento, o detenerse o continuar la marcha. Para el nivel 4, a pesar de no ser obligatorio que el conductor reaccione, si decide continuar con los mandos habrá que conocer su nivel de atención. Además, durante la conducción manual se usará el sistema como aviso. Finalmente para el nivel 5 al no haber conductor este sistema no tendría cabida.

1.3 Visión Artificial y Deep Learning

Visión artificial es una disciplina que se enfoca en desarrollar sistemas informáticos capaces de comprender y procesar información visual de manera similar a como lo hace el ser humano. Utilizando algoritmos y técnicas de procesamiento de imágenes, la visión artificial busca extraer características significativas de las imágenes y vídeos para su análisis y toma de decisiones automatizada. Este campo ha experimentado avances significativos en los últimos años, gracias al desarrollo de técnicas de aprendizaje automático, especialmente el *Deep Learning*.

El *Deep Learning*, o aprendizaje profundo, es una rama del *Machine Learning* o aprendizaje automático que se basa en redes neuronales artificiales con múltiples capas. Estas redes neuronales están diseñadas para aprender y representar patrones y relaciones complejas a partir de grandes conjuntos de datos. En el contexto de la visión artificial, el *Deep Learning* ha demostrado ser una herramienta poderosa para la extracción automática de características y el reconocimiento de objetos en imágenes y vídeos. Al entrenar redes neuronales profundas con grandes cantidades de datos etiquetados, se pueden lograr resultados impresionantes en tareas como la detección de objetos, estimación de la pose humana o la clasificación de imágenes. El *Deep Learning* ha revolucionado la visión artificial al permitir que los sistemas aprendan automáticamente características y patrones sin la necesidad de una programación manual detallada. En la Figura 1.3 se observa un ejemplo de red neuronal denominada AlexNet.

El TFG está fuertemente basado en estos dos conceptos. Se aplicarán técnicas de visión artificial basadas en *Deep Learning* y *Machine Learning* para conocer la posición del conductor, detectar objetos de interés y estimar las distracciones.

1.4 Estructura general del proyecto

A continuación se enuncia la estructura general del TFG. Basándose en el proyecto KNDAR [6] hemos construido una arquitectura que fusiona la detección de objetos multiclase y los puntos clave del conductor para inferir acciones. Como se observa en la Figura 1.4 se trata de una arquitectura modular que procesa en tiempo real tres imágenes correspondientes a tres puntos

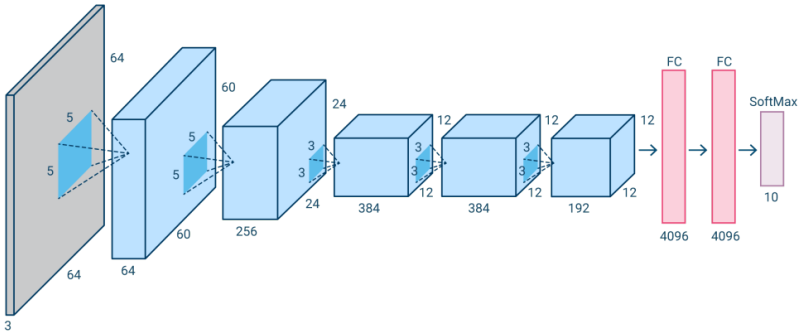


Figura 1.3: Ejemplo de red neuronal ampliamente utilizada en visión por computador (AlexNet 2012)
Fuente: www.run.ai

de vista diferentes del conductor. Primero se detectan el conductor y los objetos clave, importantes para determinadas acciones. A continuación, se infieren los puntos clave del conductor previamente detectados. Con toda esta información se construyen características que relacionen tanto los puntos clave como los objetos importantes. Después, mediante un algoritmo de *Machine Learning* clasificamos cada fotograma con la acción más probable para finalmente filtrarlo y obtener el instante de inicio y fin de cada acción detectada. Como veremos a continuación, el algoritmo será entrenado y evaluado sobre dos bases de datos del estado del arte para distracciones al volante.

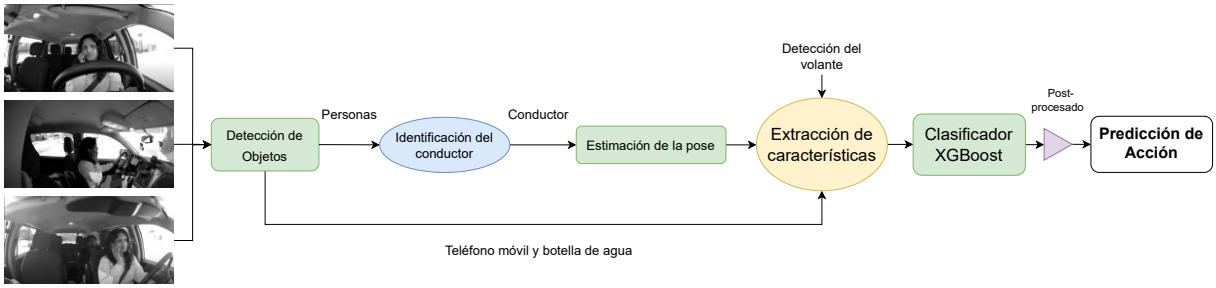


Figura 1.4: Arquitectura de la solución propuesta

Capítulo 2

Estado del Arte

El conocimiento del pasado es el fundamento del progreso

Charles Kettering

2.1 Introducción

Para solucionar un problema tan complejo como el descrito, los autores proponen alternativas muy variadas. Entender las acciones de un humano es un gran reto para la visión artificial y es un campo muy importante de estudio. Primeramente se expondrán las diferentes aproximaciones en la temática. Como ya se ha adelantado anteriormente se apostará por una solución modular basada en la fusión de objetos y poses del conductor. Se expondrá el estado del arte de cada uno de los módulos que componen la arquitectura.

2.2 Detección de distracciones al volante

En la actualidad, la detección de distracciones al volante es un tema de gran interés en el campo de la seguridad vial y se han desarrollado diversas técnicas para detectar y prevenir esta peligrosa conducta. Cada autor se centra en diferentes características del conductor. Entre las técnicas más utilizadas se encuentra el análisis de la atención con la mirada del conductor, la monitorización de rostros faciales y el análisis del cuerpo y rostro del conductor para inferir distracciones. Al analizar la mirada, se extrae la proyección en la parte frontal interior del vehículo. De esta forma se puede conocer si está atento o no a la vía. Con el estudio del rostro completo se puede conocer el estado y emociones del mismo, como la somnolencia, embriaguez, felicidad o enfado. Finalmente, al analizar el cuerpo completo del conductor, incluyendo la cara, se extrae acciones concretas que conllevan a distracciones. Estas son el uso del móvil, conversar con el copiloto, recoger objetos del suelo, entre otros. Es en este último enfoque en el que se centra este trabajo.

El objetivo final de este tipo de monitorización es inferir una acción entre unas dadas. Se trata de un problema de clasificación ya sea de imágenes, si hacemos la inferencia por fotograma,

o de secciones de vídeo, si lo hacemos de varios fotogramas. Los enfoques más comunes y con mejor rendimiento para esta temática son:

- Entrenar o afinar **redes de comprensión de vídeo** que se centran en caracterizar las diferentes acciones dentro de una secuencia [7] [8]. Redes como SlowFast [9], MMAction2 [10], X3D [11] o MViTv2 [12] son perfectas para esta tarea. Están construidas para detectar una serie de acciones genéricas y extensas. Para poder adaptarlas a las acciones de interés se realiza una técnica de transferencia del conocimiento aprendido por la red denominada *transfer learning*. Significa ajustar los pesos de la red neuronal pre-entrenada. Esto se hace mediante el entrenamiento de la red con los datos propios para adaptarla a la tarea específica que se desea realizar, mejorando así la capacidad de la red para reconocer acciones específicas en los vídeos. Obtienen resultados destacados, no obstante, son redes de gran peso y están concebidas para analizar extensos fragmentos de vídeo, en caso de que los fragmentos sean demasiado cortos, la red no arrojará resultados óptimos.

- Entrenar **clasificadores de imágenes** basados en redes convoluciones. El objetivo será tomar un fotograma del vídeo y clasificarlo entre las diferentes distracciones. Al igual que en el punto anterior, se suele realizar un *transfer learning*, ya que se obtiene mejores resultados que entrenar solo con el *dataset* de distracciones al volante. La inferencia se realiza fotograma por fotograma obteniendo una predicción para cada uno. Cada fotograma se clasificará entre las diferentes acciones contempladas.

  En [13] se hace un estudio y se compara el rendimiento de redes de clasificación de imágenes como AlexNet [14], VGG-19 [15], ResNet[50, 101] [16] o InceptionV3 [17], entre otras, haciendo *transfer learning* con un *dataset* de reconocimiento de distracciones. Además, se emplean combinaciones y modificaciones de las mismas para obtener mejores resultados.

  En algunas referencias se suele preprocesar la imagen ya sea obteniendo la pose o resaltando algunas partes de interés de la misma. En general, son redes mucho más veloces, algunas son capaces de realizar la inferencia de una imagen en menos de 10 ms. Por contra, pierden contexto al hacer inferencia fotograma por fotograma y no aplicar ningún pre-procesamiento sobre la imagen. Por ello el reconocimiento de acciones dinámicas obtiene un rendimiento pobre y se adaptan peor a otros dominios. Actualmente el estado del arte de clasificación de imágenes evaluado en Imagenet [18] se puede resumir en la Figura 2.1.

- Se proponen redes modulares basadas en la **estimación de la pose del conductor** como KNDAR [6]. Se extraen características de las imágenes como la pose y se clasifica entre las diferentes acciones posibles. Si se toman los puntos de la pose se emplean algoritmos de *Machine Learning*, ya que son muy aptos para trabajar con datos tabulados. Si se modifica la imagen resaltando los puntos de interés, se procesan con clasificadores basados en *Deep Learning*. La inferencia de la acción se realiza fotograma a fotograma, ofreciendo buenos resultados, con una buena generalización y rendimiento apto para tiempo real.

Este trabajo tomará como referencia el enfoque modular basado en los estimación de la pose del conductor. A continuación se detalla el estado del arte de los diferentes componentes de

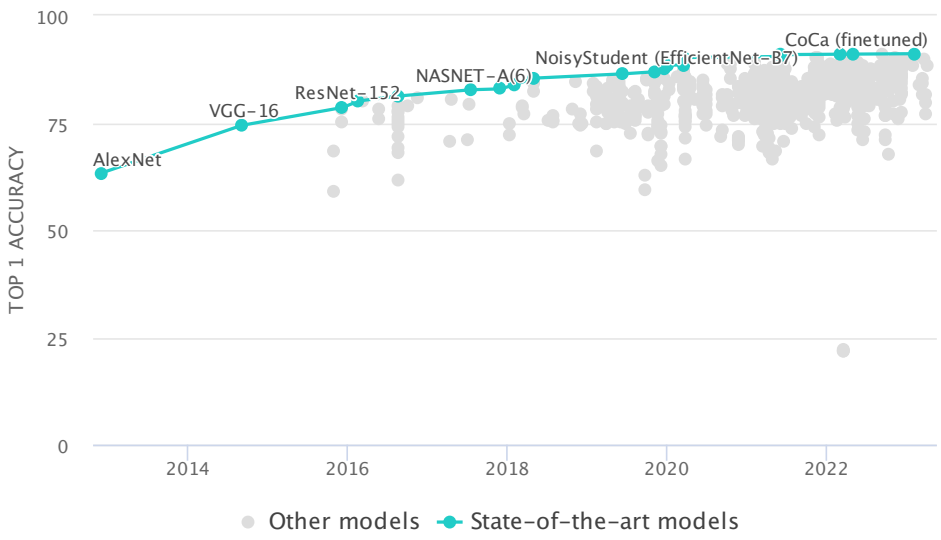


Figura 2.1: Estado del arte en clasificación de imágenes evaluado en Imagenet [19]
Fuente: paperswithcode.com

la arquitectura modular. En primer lugar se expone la detección de objetos, a continuación la estimación de la pose y para concluir un clasificador multiclase basado en *Machine Learning*.

2.3 Detección de objetos

Los detectores multiclase son un componente fundamental de los modelos de aprendizaje profundo más avanzados para tareas de detección de objetos. Estos detectores están diseñados para reconocer y localizar múltiples categorías de objetos simultáneamente en una imagen de entrada. Lo consiguen aprovechando arquitecturas avanzadas de redes neuronales convolucionales y sofisticadas estrategias de entrenamiento que les permiten aprender patrones y relaciones complejas entre los objetos y su entorno. Destacan los siguientes modelos:

2.3.1 EfficientDet

EfficientDet [20] es una familia de modelos de detección de objetos que alcanza un rendimiento de vanguardia en el conjunto de datos COCO [21]. Es un one stage detector, lo que significa que realiza la detección y clasificación conjuntamente.

Como se puede ver en la Figura 2.2 está basada en la red EfficientNet [22], a la cual se le añade una capa BiFPN para la extracción de características. Se añaden dos redes de predicción de clase y *bounding box* en la cabeza (salida de la red). Es una red diseñada para soportar una gran escalabilidad manteniendo una gran eficiencia, permitiéndose la elección de varios tamaño de red.

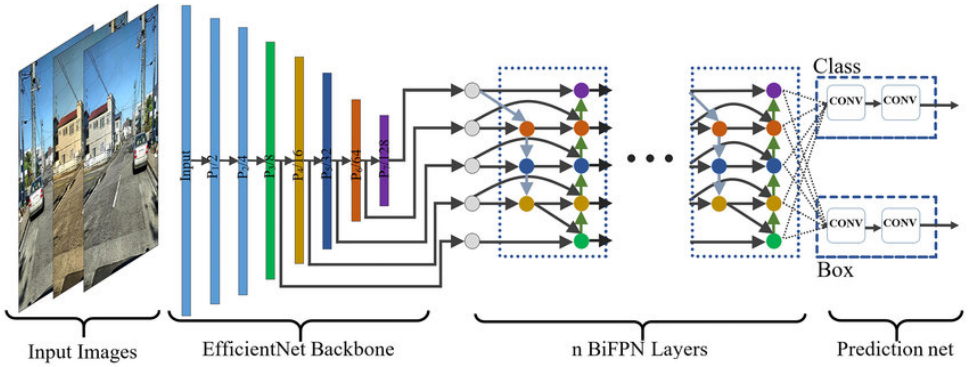


Figura 2.2: Estructura de EfficientDet
Fuente: www.researchgate.net

2.3.2 RCNN

Las RCNN, o Redes Convolucionales Recurrentes, son un tipo de arquitectura de redes neuronales que combina la capacidad de las redes convolucionales (CNN) para extraer características visuales y la capacidad de las redes neuronales recurrentes (RNN) para modelar la dependencia temporal en secuencias de datos. La primera propuesta fue R-CNN [23], la cual utiliza la búsqueda selectiva para extraer 2000 regiones de la imagen. Estas propuestas de regiones candidatas se conforman en un cuadrado y se introducen en una red neuronal convolucional que produce como salida un vector de características de 4096 dimensiones. La CNN actúa como un extractor de características y la capa densa de salida consiste en las características extraídas de la imagen que se introducen en una SVM para clasificar la presencia del objeto dentro de esa región candidata. A diferencia de la red anterior, se utilizan dos etapas, por ello esta red es considerada a *two stage detector*. En la Figura 2.3 podemos observar el flujo de trabajo descrito anteriormente.

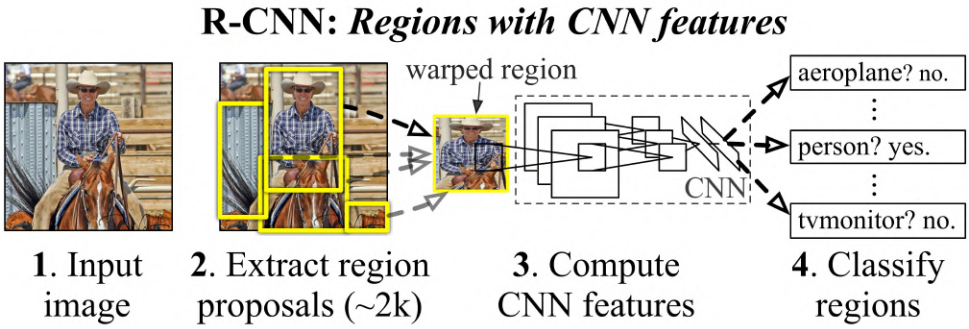


Figura 2.3: Funcionamiento de R-CNN
Fuente: towardsdatascience.com

Posteriormente se propusieron mejoras a esta red, dando lugar a Fast R-CNN [24]. El enfoque es similar al del algoritmo R-CNN. Pero, en lugar de alimentar la CNN con las propuestas de región, alimentamos la CNN con la imagen de entrada para generar un mapa convolucional de características. A partir del mapa de características convolucional, identificamos la región de las propuestas y las conformamos en cuadrados y, mediante el uso de una capa de agrupación RoI, les damos nueva forma en un tamaño fijo para que pueda ser alimentado en una capa totalmente conectada. A partir del vector de características RoI, utilizamos una capa softmax

para predecir la clase de la región propuesta y también los valores de desplazamiento para el cuadro delimitador. En la figura 2.4 observamos el *pipeline* del detector.

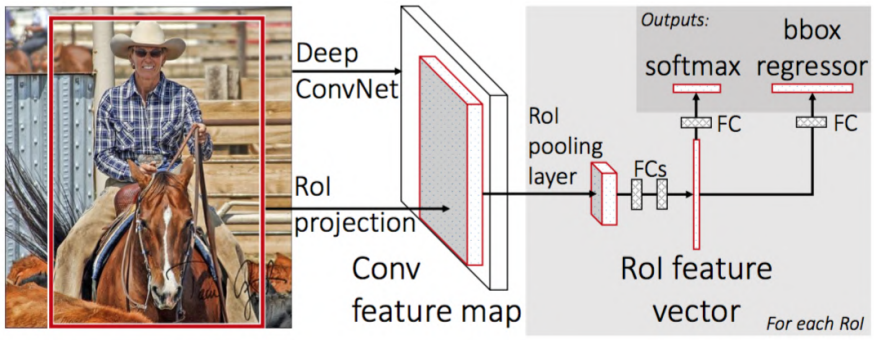


Figura 2.4: Flujo de trabajo de Fast R-CNN
Fuente: www.geeksforgeeks.org

La razón por la que la Fast R-CNN es más rápida que la R-CNN es que no es necesario alimentar cada vez la red neuronal convolucional con 2000 propuestas de región. En su lugar, la operación de convolución se realiza una sola vez por imagen y a partir de ella se genera un mapa de características. Posteriormente se desarrolló Faster R-CNN [25], la cual mejoraba considerablemente la velocidad de inferencia. Como podemos ver en la Figura 2.5 se trata de la opción más rápida de la familia.

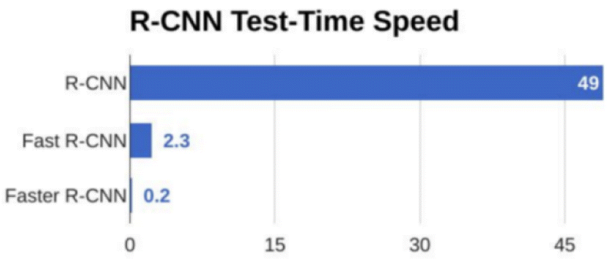


Figura 2.5: Comparación de tiempo de inferencia entre los tres componentes de la familia RCNN
Fuente: www.researchgate.net

2.3.3 DETR

DETR [26] es un modelo *one stage* de detección de objetos basado en *transformers* [27] que utiliza un enfoque sustentado en conjuntos para predecir las ubicaciones y clases de objetos en una imagen. Ha sido desarrollado por Facebook Research y sustituye toda la compleja tubería de detección de objetos hecha a mano por un Transformer, obteniendo el mismo rendimiento que Faster R-CNN con una ResNet-50 pero con la mitad de coste computacional y el mismo numero de parámetros. En la figura 2.6 observamos el flujo de trabajo del proyecto.

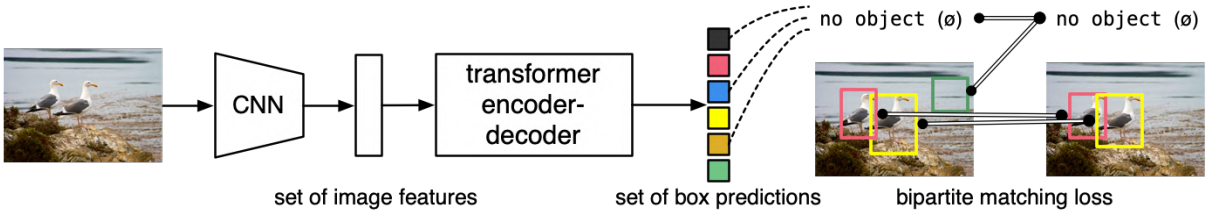


Figura 2.6: Funcionamiento de DETR
Fuente: github.com

2.3.4 YOLO

YOLO [28] (You Only Look Once) es un modelo de detección de objetos ligero y eficiente que alcanza un rendimiento de vanguardia en varios conjuntos de datos de referencia. Es especialmente eficaz en la detección de objetos pequeños y puede manejar un gran número de clases de objetos. Hoy en día es uno de los más utilizados y tiene un rendimiento sobresaliente con un bajo número de parámetros. Además, posee una interfaz sencilla para poder realizar entrenamientos e inferencias. Se enmarca dentro de los detectores *one stage*, como su nombre indica: "Solo Mira una Vez".

YOLO es un algoritmo que utiliza una única red convolucional para la detección de objetos. A diferencia de otros algoritmos de detección de objetos que barren la imagen bit a bit, el algoritmo toma toda la imagen y replantea la detección de objetos como un único problema de regresión, directamente de los píxeles de la imagen a las coordenadas del cuadro delimitador y las probabilidades de clase. Como podemos ver en la Figura 2.7, se toma una imagen de entrada, se divide en una cuadrícula SxS, se pasa por una red neuronal para crear cuadros delimitadores y predicciones de clase para determinar el resultado final de la detección.

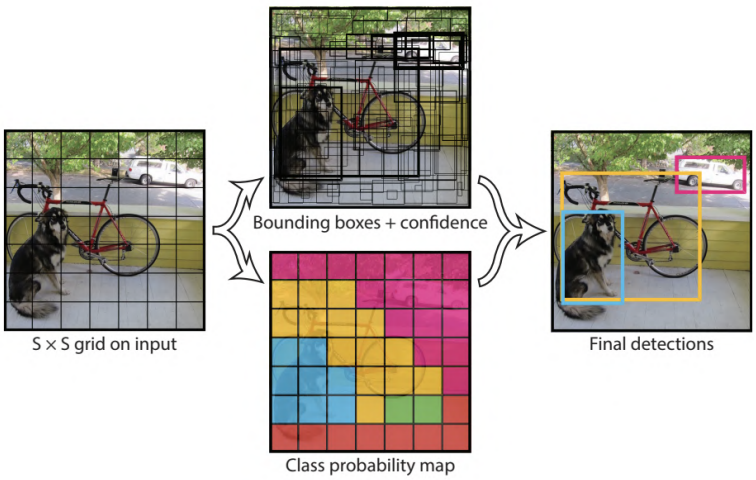


Figura 2.7: Comparación de tiempo de inferencia entre los tres componentes de la familia RCNN
Fuente: towardsdatascience.com

Para calcular las *bounding boxes* finales, YOLO implementa un paso clave de posprocesamiento: NMS (Non-Maximum Suppression). Es una técnica utilizada para suprimir detecciones redundantes. Se encuentra detallado en el apéndice A.1 de este TFG. Consta de tres pasos:

1. Dada una lista de detecciones se ordenan según su *score* o puntación, de mayor a menor.

2. Se selecciona la detección con la puntuación más alta y se eliminan todas las detecciones superpuestas que tengan un IoU (Intersection over Union) mayor a un umbral predefinido.

3. Se repite el paso 2 para las detecciones restantes hasta que no haya más detecciones significativas o se alcance el número máximo de detecciones deseadas.

Como se ve en la figura 2.8, eliminamos detecciones redundantes seleccionando la más relevantes de todas. Esto ayudará a mejorar la precisión con un coste computacional muy pequeño.

Figura 2.8: Ejemplo de aplicación del algoritmo NMS
Fuente: learnopencv.com

Para finalizar destacar que existen varias versiones de este detector, debidas a una evolución y mejora de sus prestaciones. Además de cada una encontramos diferentes tamaños de red. Los más pequeños serán más veloces pero menos precisos y viceversa. Será tarea del ingeniero seleccionar el tamaño que más se ajuste a sus necesidades manteniendo un compromiso de precisión y latencia. Las versiones más representativas son YOLOv3 [29], YOLOv5 [30], YOLOv6 [31], YOLOv7 [32] y YOLOv8 [28]. En cada una de ellas se incluyen diferentes tamaños de red (n, s, m, l y x). En la Figura 2.9 podemos observar una comparación de validación de los diferentes modelos de las tres versiones en el *dataset* COCO [21].

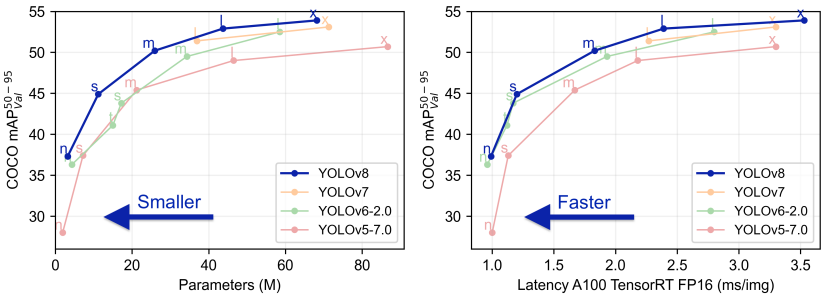


Figura 2.9: Comparación entre las versiones 5, 6, 7 y 8 de YOLO
Fuente: github.com

La versión más actual que existe de YOLO es la NAS[1]. Se trata de una versión todavía en desarrollo, pero que reduce significativamente el tiempo de inferencia, manteniendo unos buenos resultados en validación.

2.3.5 Comparativa de detectores de objetos

En la Tabla 2.1 encontramos una comparativa de todas las redes mencionadas en los apartados anteriores. Se detalla la precisión medida con mAP50:95, la cual se tratará en próximos capítulos, y el número de parámetros. Cuanto más grande sea el valor del mAP mejor será su precisión y cuanto más grande sea el número de parámetros más lenta será la red. Para las diferentes versiones de YOLO se enuncia una horquilla ya que cada modelo (tamaño) se sitúa en ese rango.

Tabla 2.1: Estado del arte de los principales detectores de objetos evaluados en el *dataset* COCO

Método	**mAP50:95**	**Parámetros (M)**
EfficientDet	33.6	66.0
R-CNN	18.1	-
Fast R-CNN	19.3	-
Faster R-CNN (R50)	21.9	41.8
DETR	36.7 - 41.6	41.0 - 60.0
YOLOv5	28.0 - 50.7	1.9 - 86.7
YOLOv6	37.5 - 52.8	4.7 - 59.6
YOLOv7	51.0 - 53.1	38.3 - 70.1
YOLOv8	37.3 - 53.9	3.2 - 68.2
YOLO-NAS	47.5 - 52.1	-

Con lo expuesto se concluye que el detector de objetos más adecuado para este TFG es **YOLO**. Se descarta la versión más reciente (YOLO-NAS) debido al estado de inmadurez del proyecto.

2.4 Estimación de la pose

La estimación de la pose se refiere a la tarea de determinar la orientación espacial y la posición de un objeto o un cuerpo humano en una imagen o un vídeo. A lo largo de la historia se han propuesto diferentes aproximaciones y formas de entender la tarea. En esencia encontramos tres tipos, representados en la Figura 2.10.

a) Cinemáticos (kinematic)

b) Planos (planar)

c) Volumétricos (volumetric)

[1]https://github.com/Deci-AI/super-gradients/blob/master/YOLONAS.md

Los modelos cinemáticos pueden utilizarse para la estimación de la pose tanto en 2D como en 3D. El objetivo es extraer los denominados *keypoints* o puntos clave. Los cuales como su nombre indica son posiciones en la imagen referentes a partes del cuerpo de una persona. Ejemplos son el hombro derecho, la oreja derecha, la rodilla izquierda, etc. Sin embargo, este tipo de modelos tienen algunas limitaciones a la hora de representar información basada en texturas o formas de la persona. A continuación, el modelo plano hace hincapié en la estimación de la pose en 2D. Idealmente, las partes del cuerpo humano se representan mediante rectángulos para proporcionar un contorno corporal aproximado. Este tipo de redes son más precisas en cuanto a la forma y tamaño de las diferentes partes de la persona. Por último, el modelo de estimación de pose volumétrica se centra en la estimación de pose 3D. Se trata de modelos de aprendizaje profundo complejos que son capaces de realizar una inferencia en 3D detallada de la forma corporal de una persona.

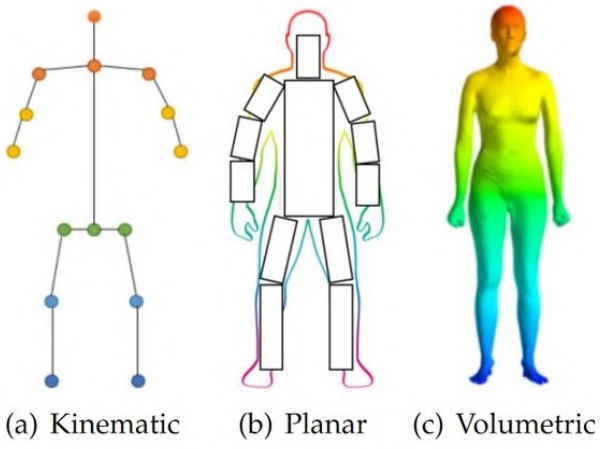
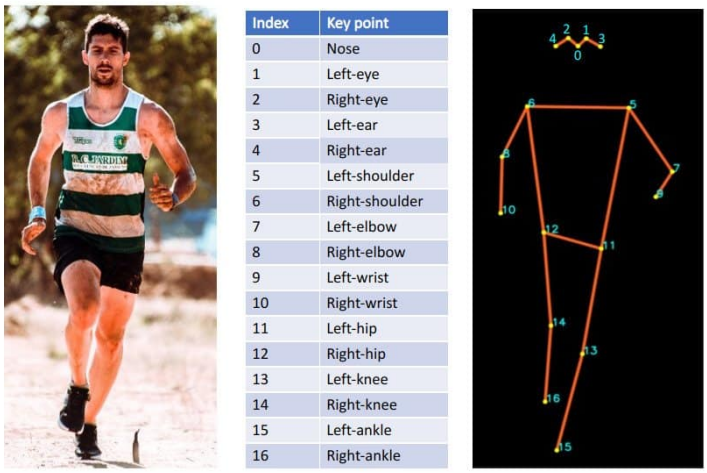


Figura 2.10: Diferentes tipos de modelos de estimación de poses
Fuente: viso.ai

Para la tarea a desarrollar en este TFG lo importante es la distancia entre los diferentes articulaciones y partes del conductor. Además cuantas más características de la imagen extraigamos más costoso será el cómputo de la inferencia. Es por ello que elegimos los puntos clave como la mejor opción. No hay un estándar acerca del número y que puntos usar. Un mayor número de puntos clave implica una mayor precisión en el reconocimiento de la pose pero conlleva un mayor coste computacional. El número es variable y depende directamente del *dataset* con el que han sido entrenados. Para este trabajo se usa el mostrado en la figura mencionada, que corresponde al del *dataset* de COCO con un total de 17 puntos clave. En la Figura 2.11 se observa una representación gráfica de estos puntos.

Para esta tarea existen dos tipos de enfoques. Por un lado, las redes ascendentes estiman las partes del cuerpo humano en la imagen y, a continuación, calculan la pose. Por otro lado, un enfoque descendente localiza a los humanos en la imagen y, a continuación, estima las partes y calcula la pose. En nuestra arquitectura se usa un detector para hacer la doble detección del conductor y los objetos de interés, por eso nos interesa una estimación de pose basada en el segundo enfoque. De esta manera podremos aprovechar la detección para aliviar de trabajo a la red de estimación de pose. Añadir que incrementar el número de clases detectadas no incrementa



Index	Key point
0	Nose
1	Left-eye
2	Right-eye
3	Left-ear
4	Right-ear
5	Left-shoulder
6	Right-shoulder
7	Left-elbow
8	Right-elbow
9	Left-wrist
10	Right-wrist
11	Left-hip
12	Right-hip
13	Left-knee
14	Right-knee
15	Left-ankle
16	Right-ankle

Figura 2.11: Representación gráfica de los puntos clave en un corredor
Fuente: learnopencv.com

el tiempo de inferencia de la detección (o al menos es despreciable). Por tanto se ahorrará tiempo de proceso eligiendo un enfoque descendiente y confiando la tarea de detección de personas a la etapa anterior.

2.4.1 OpenPose

OpenPose[2] [33] se trata de uno de los primeros modelos desarrollados para esta tarea. La imagen se procesa con la ayuda de una red neuronal convolucional (CNN) para generar mapas de características de la entrada específica. Además, el mapa de características se procesa a través de diferentes etapas de la red CNN para obtener el mapa de confianza y el campo de afinidad. Como podemos ver se trata de un enfoque ascendente, ya que no necesita de una detección previa. Está desarrollada en C++ y obtiene resultados del estado del arte. En la Figura 2.12 observamos el flujo de trabajo de la red siguiendo los siguientes pasos:

a) Adquisición de la imagen completa como entrada (imagen o fotograma de vídeo)

b) Las CNN de dos ramas predicen conjuntamente los mapas de confianza para la detección de partes del cuerpo

c) Estimación de los campos de afinidad de partes (PAF) para la asociación de partes

d) Conjunto de correspondencias bipartitas para asociar las partes del cuerpo candidatas

e) Ensamble en poses de cuerpo entero para todas las personas de la imagen

2.4.2 DensePose (R-CNN)

DensePose-RCNN[3] [34] está diseñado para asignar todos los píxeles humanos de una imagen RGB a una representación 3D del cuerpo humano basada en la superficie. Para ello se introduce

[2]https://github.com/CMU-Perceptual-Computing-Lab/openpose
[3]https://github.com/facebookresearch/Detectron

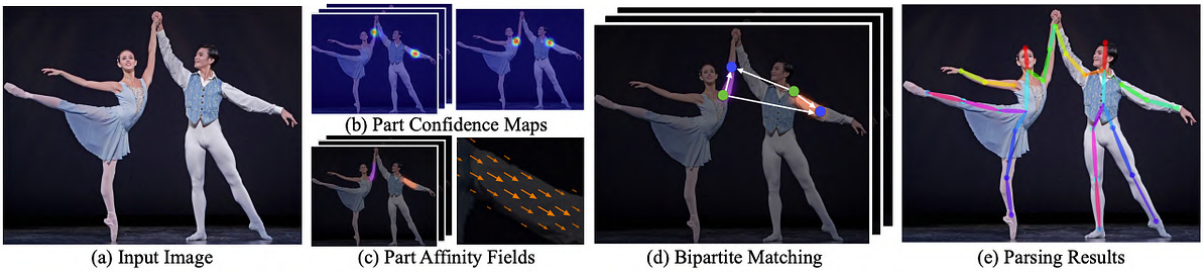


Figura 2.12: Flujo de trabajo de OpenPose
Fuente: medium.com

un conjunto de datos a gran escala, anotados manualmente, y una variante de Mask-RCNN, un marco sencillo y flexible para la segmentación de instancias de objetos. Utiliza como *dataset* de entrenamiento COCO-DensePose Dataset que incluye anotaciones para más de 50k humanos. Como podemos ver en la Figura 2.13, se establece una correspondencia entre la imagen 2d y el plano 3d. Como se enunció anteriormente, corresponde al tipo volumétrico. La red está implementado en Detectron [35]. Es un *framework* de código abierto desarrollado por Facebook AI Research. Aborda tareas como detección y segmentación de objetos, entre otros.

Figura 2.13: Ejemplos de inferencia de la red DensePose
Fuente: analyticsindiamag.com

2.4.3 MoveNet

MoveNet[4] [36] es una red de estimación de tipo cinemático de la pose desarrollada por Google y basada en TensorFlow.js. Se trata de una solución muy eficiente, capaz de detectar 17 puntos clave del cuerpo humano con un tiempo de inferencia de más de 30 fotogramas por segundo en la mayoría de equipos modernos. Se trata de un modelo *bottom-up* o ascendente que utiliza mapas térmicos para localizar con precisión los puntos clave humanos.

Como se detalla en la Figura 2.14, la arquitectura consta de dos componentes: un extractor de características y un conjunto de cabezas de predicción. El extractor de características de MoveNet es MobileNetV2 con una red piramidal de características (FPN) adjunta, que permite obtener un mapa de características de alta resolución y semánticamente rico. Hay cuatro cabezas de predicción adjuntas al extractor de características, responsables de predecir:

- **Mapa de calor del centro de la persona:** predice el centro geométrico de las instancias de persona.

[4]https://github.com/tensorflow/tfjs-models/tree/master/pose-detection/src/movenet

- **Campo de regresión de puntos clave:** predice el conjunto completo de puntos clave de una persona, utilizado para agrupar puntos clave en instancias.
- **Mapa de calor de los puntos clave de la persona:** predice la ubicación de todos los puntos clave, independientemente de las instancias de la persona.
- **Campo de desplazamiento 2D por punto clave:** predice los desplazamientos locales desde cada píxel del mapa de características de salida hasta la ubicación precisa en subpíxeles de cada punto clave.

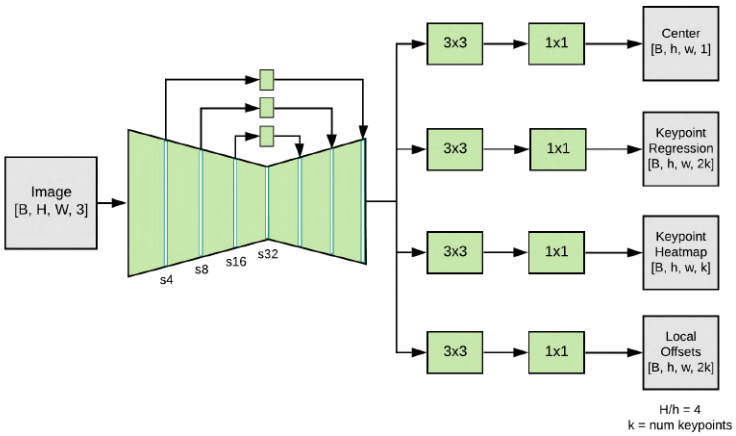


Figura 2.14: Arquitectura de MoveNet
Fuente: blog.tensorflow.org

El modelo tiene dos versiones, la *Lightning*, dirigida a aplicaciones con requisitos de baja latencia y la versión *Thunder*, diseñada para aplicaciones centradas en lograr una mayor precisión. Están orientadas a deportes y *fitness* ya que presentan movimientos rápidos con escenarios sencillos, pocas personas y en primer plano.

2.4.4 AlphaPose

AlphaPose es un popular framework basado en deep learning para la estimación de pose multipersona. Se trata de una red descendente o *top-down* lo que significa que primero detecta las partes individuales del cuerpo antes de estimar la pose global. Utiliza una red neuronal convolucional (CNN) para estimar la pose de los seres humanos en imágenes o vídeos. Analiza la imagen o el vídeo de entrada fotograma a fotograma, detectando partes del cuerpo humano como la cabeza, el torso y las extremidades. A continuación, estima la posición y orientación de cada parte del cuerpo, creando una estimación de la pose de todo el cuerpo humano. AlphaPose también utiliza un enfoque basado en mapas térmicos, lo que significa que estima la probabilidad de que cada píxel pertenezca a una parte específica del cuerpo.

En la Imagen 2.15 se detalla la arquitectura de AlphaPose. El proyecto se divide en cinco módulos, (a) módulo encargado de la carga de datos ya sean imágenes o vídeo, (b) módulo de

detección que proporciona las *bounding boxes* de los humanos, (c) módulo de transformación de datos para procesar los resultados de la detección y recortar a cada persona para los módulos posteriores, (d) módulo de estimación de la pose que genera puntos clave y la posición de la persona, (e) módulo de post-procesamiento, que procesa y guarda los datos de las personas detectadas. La arquitectura es flexible y cada módulo contiene varios componentes que pueden sustituirse y actualizarse fácilmente. El recuadro discontinuo indica los componentes opcionales de cada módulo.

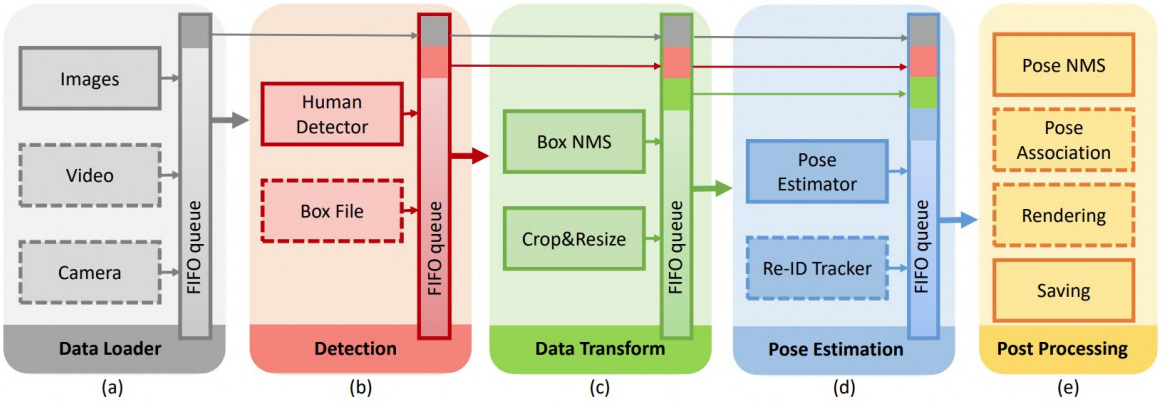


Figura 2.15: Arquitectura de AlphaPose
Fuente: arxiv.org

2.4.5 Comparativa de estimadores de pose

En la Tabla 2.2 podemos observar una comparativa entre el rendimiento de las cuatro redes enunciadas evaluadas en el *dataset* COCO [21]. En este conjunto de datos se incluye anotaciones de pose de las diferentes personas que se encuentran en la imagen. Se ordenan de menor a mayor en relación a los resultados.

Tabla 2.2: Estado del arte de estimación de la pose evaluados en en *dataset* COCO

Método	**AP @0.5:0.95**
OpenPose (CMU-Pose)	61.8
DensePose (R-CNN)	67.0
MoveNet	69.3
AlphaPose	73.3

Con todo lo expuesto se concluye que el estimador de pose más adecuado para este TFG es **AlphaPose**.

2.5 Clasificador XGBoost

XGBoost (Extreme Gradient Boosting) Classifier es un algoritmo de aprendizaje automático de clasificación supervisada que utiliza el método de *boosting* para mejorar la precisión de la

predicción. El algoritmo funciona construyendo una serie de modelos de árboles de decisión débiles y los combina para formar un modelo más fuerte y preciso.

En cada iteración, el modelo evalúa la precisión de la predicción actual y asigna pesos más altos a las instancias que fueron mal clasificadas en el paso anterior. Luego, se construye un nuevo árbol de decisión que se enfoca en corregir las instancias mal clasificadas. Este proceso se repite hasta que se alcanza un cierto criterio de convergencia o se alcanza el número máximo de iteraciones. XGBoost también utiliza técnicas avanzadas de regularización y control de sobre-ajuste para evitar que el modelo se ajuste demasiado a los datos de entrenamiento y generalice mal en los nuevos datos de prueba.

XGBoost es uno de los algoritmos de aprendizaje automático más populares y efectivos en la actualidad debido a su alta precisión y capacidad para manejar grandes conjuntos de datos. Se utiliza ampliamente para el tratamiento de datos tabulares. Es por ello que es una gran elección como etapa final para la clasificación de las distracciones en nuestro proyecto.

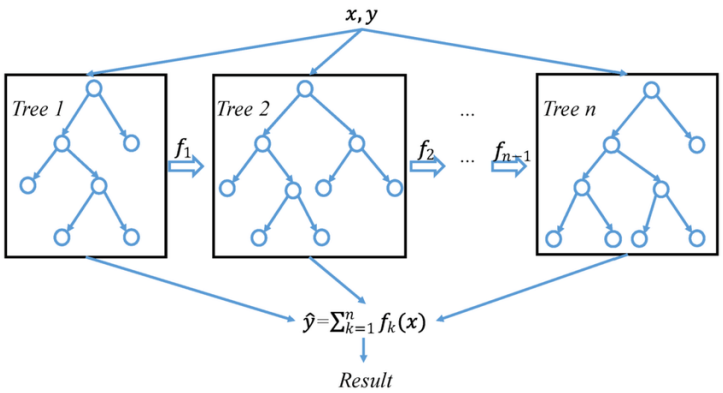


Figura 2.16: Arquitectura general del clasificador XGBoost
Fuente: www.researchgate.net

Capítulo 3

Herramientas software

Las herramientas correctas hacen el trabajo más fácil

Anónimo

En este capítulo se exponen diferentes herramientas software necesarias para desarrollar el proyecto. Primero se hablará de los *datasets* o conjunto de datos, formados por miles de imágenes que nos ayudarán a transformar y adaptar nuestras redes a la temática. A continuación, se explicará brevemente que es *Docker* y su utilidad en el proyecto. Finalmente se enunciarán diferentes librerías de Python utilizadas para desarrollar el código, CUDA y otras herramientas.

3.1 Datasets

Los *datasets* o conjunto de datos son la esencia de todas las redes neuronales. Se trata de una colección de datos, ya sean fotos, vídeos, sonidos o datos tabulares entre otros. Los algoritmos de *Machine Learning* aprenden (entrenamiento) de ellos para ser capaces de replicar el aprendizaje con información nueva. Los datos pueden ser reales, tomados del mundo real, o sintéticos, generados por ordenador en simuladores, y deben adaptarse al problema que se desea resolver. En este proyecto la información proveniente de los sensores (cámaras) son imágenes, por ello los conjuntos de datos estarán formados por imágenes, ya que serán los datos a procesar por las redes neuronales. A continuación vamos a exponer los principales *datasets* utilizados en el proyecto, explicando la información que contienen y como está distribuida. En la Tabla 3.1 se muestra un resumen del número y formato de imágenes. En la Tabla 3.2 aparecen las anotaciones incluidas de los diferentes *datasets*.

3.1.1 COCO

El conjunto de datos COCO [21] (Common Objects in Context) es uno de los más populares utilizados para la detección y el reconocimiento de objetos en imágenes. COCO contiene más de 330,000 imágenes etiquetadas con más de 2.5 millones de anotaciones de objetos en 80 categorías

Tabla 3.1: Información del formato de los *datasets*

			Imágenes				
Sección	**Nombre**		**Tamaño**	**Origen**	**RGB**	**Infrarrojo**	**Profundidad**
3.1.1	COCO (2014)		330000	Real	Variable	-	-
3.1.2	SVIRO		25000	Sintético	512 × 768	512 × 768	512 × 768
3.1.3	TICaM	Real	6700	Real	720 × 1280	512 × 512	512 × 512
		Simulado	3300	Sintético	512 × 512	512 × 512	512 × 512
3.1.4	Track 3 AI CC 23		34h@30fps	Real	-	1080 × 1920	-
3.1.5	AUC	Coche	11700	Real	1080 × 1920	-	-
		Simulador	2800	Real	480 × 640	-	-

Tabla 3.2: Anotaciones incluidas en los *datasets*

			Anotaciones			
Sección	**Nombre**		**Cajas**	**Keypoints**	**Segmentación**	**Actividad**
3.1.1	COCO (2014)		✓	✓	✓	-
3.1.2	SVIRO		✓	✓	✓	-
3.1.3	TICaM	Real	✓	-	✓	✓
		Simulado	✓	✓	✓	-
3.1.4	Track 3 AI CC 23		-	-	-	✓
3.1.5	AUC	Coche	-	-	-	✓
		Simulador	-	-	-	✓

diferentes, como personas, animales, medios de transporte, muebles, alimentos, entre otros. Como se puede ver en la Figura 3.1 se trata de imágenes cotidianas y muy variadas en las que aparecen diferentes clases de objetos con diversos fondos y situaciones. Es un conjunto de datos muy versátil y muy apto para muchos dominios ya que sus imágenes son verdaderamente diversas. Es por ello que comúnmente se utiliza como base para entrenar con otros conjuntos de datos ya que proporciona gran generalización a la red. Además de las anotaciones de objetos, COCO también proporciona información adicional como la detección de objetos, la segmentación semántica, la generación de texto a partir de imágenes y la estimación de la pose en personas, como se observa en la Figura 3.2. Este conjunto de datos se utiliza ampliamente en desafíos de visión por computadora y tareas de aprendizaje automático.

3.1.2 SVIRO

SVIRO [37] es un *dataset* sintético para la detección y clasificación de la ocupación de los asientos traseros del vehículo. Presenta varios escenarios con diferentes ocupantes sentadas (adultos, niños y sillas con bebe) en la parte trasera de un vehículo en diferentes posturas (mirando a los lados, con los brazos extendidos, agachados, etc). Como se observa en la Figura 3.3 provee imágenes RGB, imágenes infrarrojas e imágenes de profundidad. Las anotaciones de las imágenes son *bounding boxes*, máscaras de segmentación y *keypoints* de los ocupantes del vehículo. El *dataset* contiene un total de 25000 escenarios proveniente de interiores de 10 vehículos diferentes con fondos (ventanas) e iluminación variada. El conjunto de datos raíz provee gran variedad de

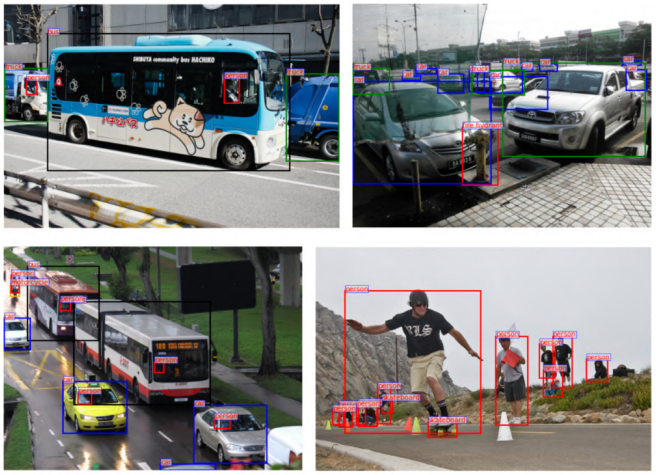


Figura 3.1: Ejemplo de imágenes del *dataset* COCO para la detección de objetos
Fuente: machinelearningspace.com

Figura 3.2: Ejemplo de imágenes del *dataset* COCO para la estimación de la pose
Fuente: blog.roboflow.com

escenarios, iluminación, ocupantes y objetos imprevistos. También podemos encontrar variantes que aportan diferentes características. Están pensadas sobre todo para evaluar y testear la robustez de la red frente a diferentes cambios. Cada una de estas variantes se centra en un elemento variante.

- **SVIRO-Illumination:** 500 escenarios, pero cada uno con 10 iluminaciones diferentes.
- **SVIRO-NoCar:** aproximadamente 5000 escenarios, pero cada uno con 10 fondos diferentes, donde no aparece el interior del coche.
- **SVIRO-Uncertainty:** Aleatoriamente se introducen objetos imprevistos en cada uno de los tres asientos, teniendo que identificar cuales son humanos, están vacíos o contienen un objeto extraño.
- **SVIRO-InterCar:** Al igual que en iluminación, pero esta vez cambiando los vehículos, lo que también modifica el ángulo de la cámara

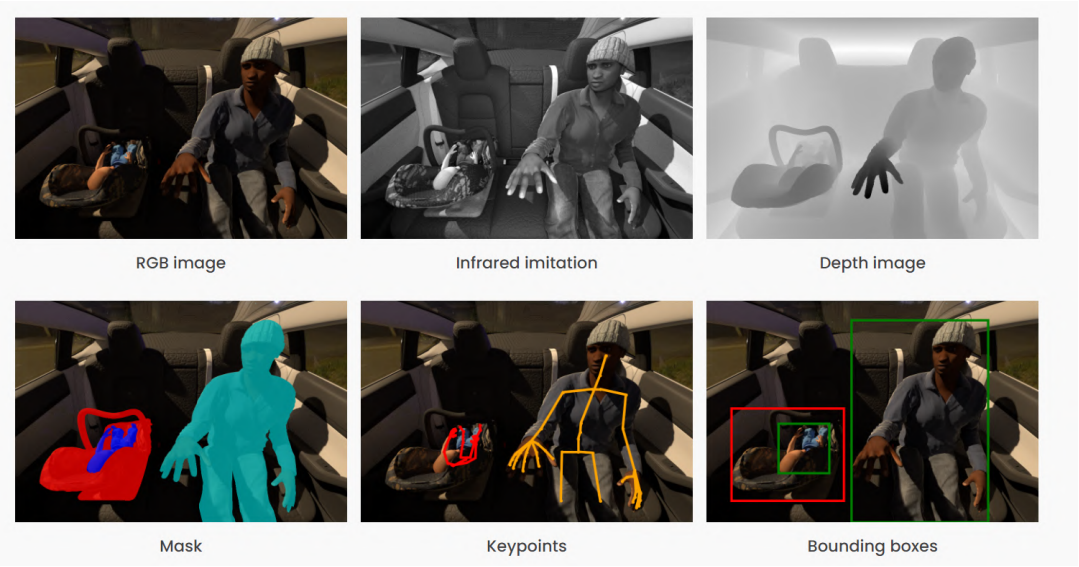


Figura 3.3: Ejemplo de imágenes del *dataset* SVIRO
Fuente: https://sviro.kl.dfki.de/

3.1.3 TICaM

TICaM [38] es un *dataset* real para la monitorización de los ocupantes de la parte delantera del vehículo (conductor y copiloto). Proveen imágenes RGB, infrarrojas y de profundidad tomadas en un simulador de conducción representado en la Figura 3.4. Cubre la mayoría de los escenarios de la vida real, incluyendo adultos, niños, bebes y objetos de la vida cotidiana. Los participantes realizan una conducción normal en el simulador mientras que el acompañante realiza acciones habituales. Además, proporcionan un conjunto de imágenes sintéticas del habitáculo del coche similar al conjunto de datos reales, lo que genera mucha más variedad en las imágenes.

Figura 3.4: Simulador para la captura de imágenes del *dataset* TICaM
Fuente: vizta-tof.kl.dfki.de

En ambos conjuntos de datos proporcionan anotaciones. En el caso de las imágenes reales se incluye máscara de segmentación y *bounding boxes* de objetos y personas tanto en 2D como en 3D. Además, encontramos otro conjunto de datos más grande en el que se detalla la actividad de los ocupantes (hablar, mirar, ajustar el retrovisor, dormir, etc). En la Figura 3.5 se adjunta ejemplos de imágenes reales.

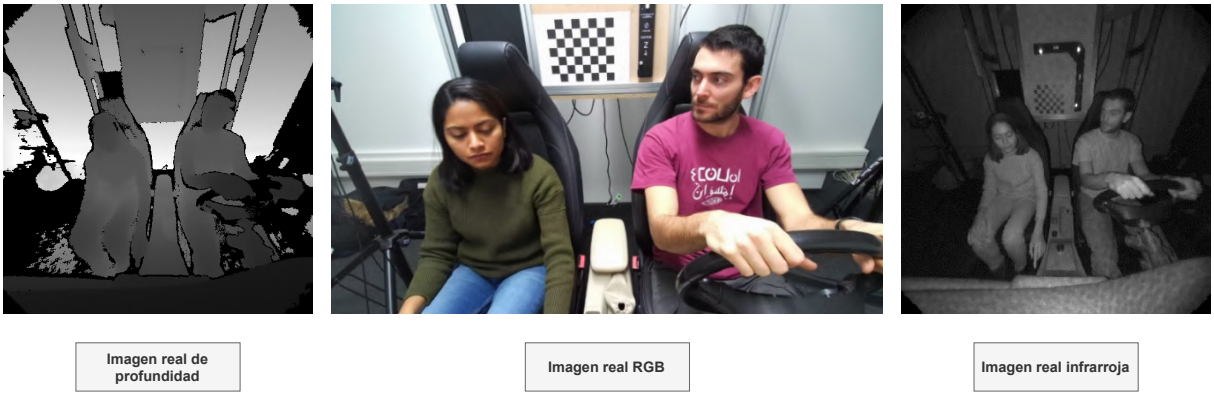


Figura 3.5: Ejemplo de imágenes reales del *dataset* TICaM
Fuente: vizta-tof.kl.dfki.de

Para el caso de las imágenes simuladas se proporcionan además los puntos clave de la pose de los ocupantes, incluidos los bebes. Por contra no se contempla la acción en este conjunto. En la Figura 3.6 se observan tres ejemplos de imágenes sintéticas.

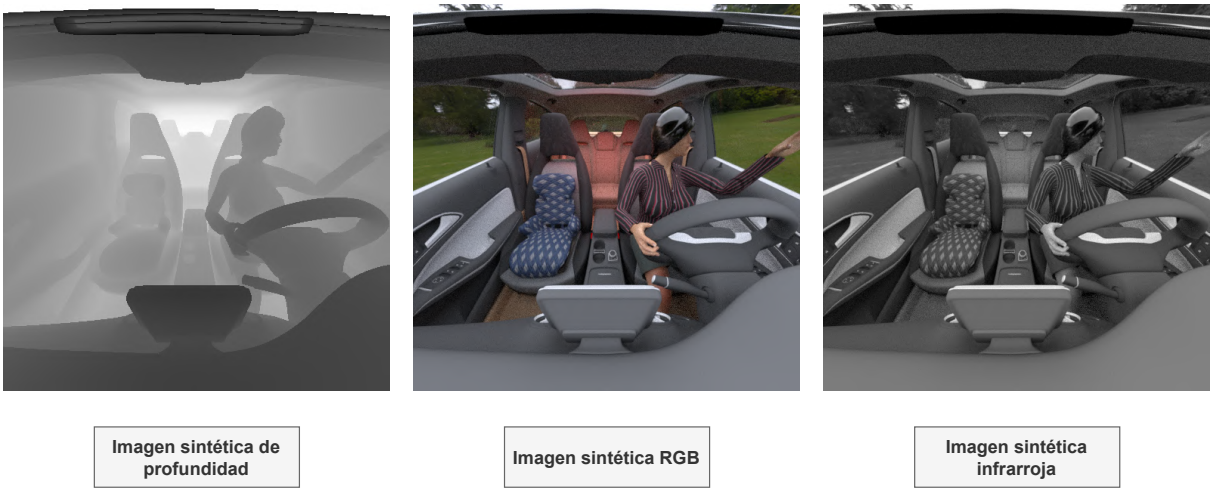


Figura 3.6: Ejemplo de imágenes sintéticas del *dataset* TICaM
Fuente: vizta-tof.kl.dfki.de

3.1.4 Track 3 *dataset* del AI City Challenge 2023

El conjunto de datos Track 3 del AI City Challenge 2023 [39], incluido como *workshop* en el congreso CVPR, proporciona vídeos de conductores que realizan las acciones que se muestran en la Tabla 3.3 mientras simulan la conducción en un vehículo real estacionado. Como se muestra en la Figura 3.7 la acción se graba simultáneamente desde tres ángulos denominados *dashboard* (salpicadero), *rear view* (vista semi-lateral) y *right view* (vista lateral derecha). Las imágenes son infrarrojas para poder abarcar el problema durante el día y con ausencia total de luz. Se nos presentan un total de 25 conductores de diferentes edades, sexos y apariencia para el entrenamiento y otros 5 para la validación (sin etiquetas). Para cada conductor se proporcionan seis vídeos correspondientes a tres puntos de vista y dos secuencias. Uno sin objetos en la cara y la cabeza y otro con objetos para incluir oclusión. Las anotaciones proporcionadas contiene

las diferentes acciones realizadas a lo largo de cada vídeo. Se incluye un *.csv* por vídeo con el tiempo en segundos del inicio y final de cada acción.

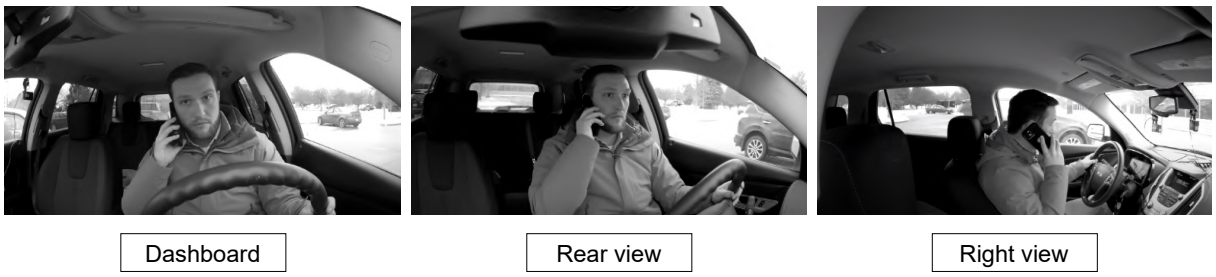


Figura 3.7: Ejemplo de imágenes del *dataset* Track 3 del AI City Challenge 2023
Fuente: https://www.aicitychallenge.org/

Tabla 3.3: Distracciones

ID	English	Español
0	Normal Driving	Conducción normal
1	Drinking	Beber
2	Phone Call (Right)	Llamada telefónica (derecha)
3	Phone Call (Left)	Llamada telefónica (izquierda)
4	Eating	Comer
5	Text (Right)	Mensaje de texto (derecha)
6	Text (Left)	Mensaje de texto (izquierda)
7	Reaching behind	Alcanzar hacia atrás
8	Adjust control panel	Ajustar panel de control
9	Pick up from floor (Driver)	Recoger del suelo (conductor)
10	Pick up from floor (Passenger)	Recoger del suelo (pasajero)
11	Talk to passenger at the right	Hablar con pasajero a la derecha
12	Talk to passenger at backseat	Hablar con pasajero en asiento trasero
13	Yawning	Bostezar
14	Hand on head	Mano en la cabeza
15	Singing or dancing with music	Cantar o bailar con música

3.1.5 Cairo Distracted Driver Dataset

Cairo Distracted Driver Dataset es un conjunto de imágenes que se utiliza comúnmente en la investigación de la seguridad vial. Consta de más de 21,000 imágenes de conductores distraídos y no distraídos en situaciones de conducción cotidianas. Las imágenes se capturaron en las carreteras de El Cairo, Egipto, y se etiquetaron con información sobre la distracción existente en cada momento. Las imágenes están tomadas en RGB durante el día desde la ventanilla derecha. También se incluyen dos versiones, la D2.1, en la que los participantes se encuentran en un coche real, y la D2.2, en la que se encuentran en un simulador, ambos con volante a la izquierda. Añadir que el segundo conjunto presenta algunas imágenes en formato espejo.

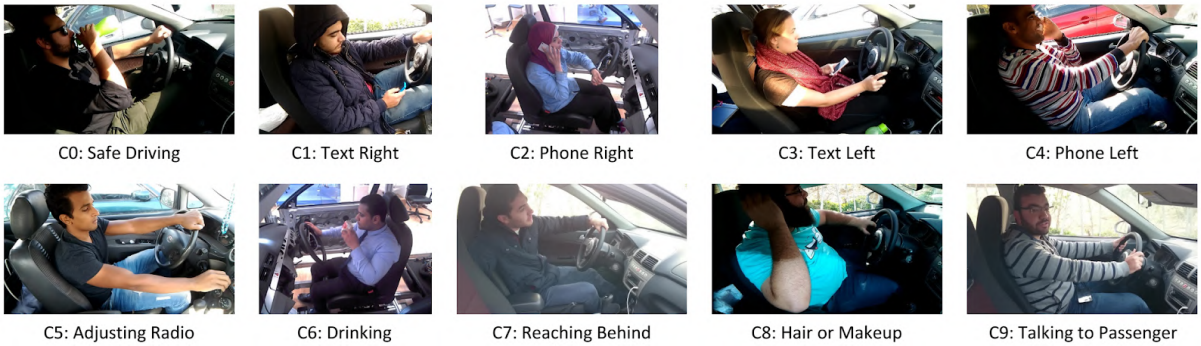


Figura 3.8: Ejemplo de imágenes reales (C0, C1, C3, C4, C5, C7, C8 y C9) y en simulador (C2 y C6) del *dataset* Cairo Distracted Driver
Fuente: heraqi.blogspot.com

3.2 Docker

Docker [40] es una herramienta de virtualización ligera y eficiente que permite la creación de entornos de software aislados y portátiles, conocidos como contenedores. Estos contenedores pueden contener cualquier aplicación, incluyendo el software de entrenamiento e inferencia de redes neuronales.

Permite la creación de entornos de desarrollo estandarizados, lo que facilita la replicación y la colaboración en proyectos de aprendizaje automático. Además, al ejecutar el software en un contenedor, se aísla del resto del sistema, lo que garantiza que las dependencias y configuraciones no entren en conflicto con otras aplicaciones en la máquina anfitriona. El entorno se activa rápidamente y se pueden tener varios a la vez. Pueden acceder a recursos hardware como la GPU, esencial para realizar entrenamientos. Se pueden crear diferentes versiones del mismo contenedor, pudiéndolas recuperar en cualquier momento.

Otra ventaja de usar Docker es la gran comunidad existente. En Docker Hub [41] se encuentran multitud de contenedores creados que facilitan el desarrollo. NVIDIA proporciona una gran colección con todos los drivers necesarios instalados y librerías esenciales de Python para entrenar e inferir redes neuronales en sus GPUs.

En resumen, Docker es una herramienta muy útil para crear entornos aislados para el entrenamiento y uso de redes neuronales, lo que permite una mayor estandarización, portabilidad, seguridad y eficiencia en el desarrollo y ejecución de aplicaciones de aprendizaje automático.

3.3 Librerías de Python

Python, como uno de los lenguajes de programación actuales más populares, cuenta con una amplia variedad de librerías especializadas en IA que ofrecen poderosas herramientas y funcionalidades. Estas librerías proporcionan un conjunto de algoritmos y modelos predefinidos que permiten implementar soluciones de IA de manera eficiente y efectiva. En este apartado se detallan las principales librerías de Python utilizadas en el proyecto que han revolucionado la forma en que se desarrollan aplicaciones y se resuelven problemas complejos en diversos dominios.

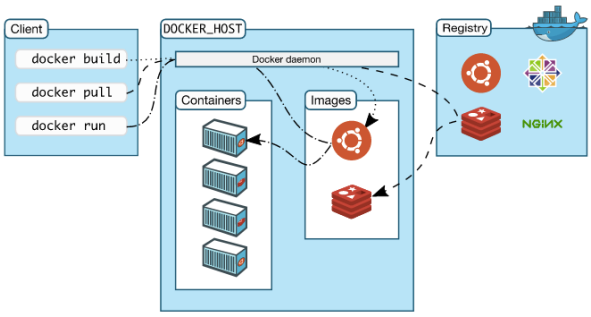


Figura 3.9: Esquema del ciclo de trabajo con Docker
Fuente: ualmtorres.github.io

3.3.1 PyTorch

PyTorch es un *framework* desarrollado para Python y está orientado al aprendizaje profundo basado en la librería Torch. Principalmente ha sido desarrollado por el Laboratorio de Investigación de Inteligencia Artificial de Facebook (FAIR). Se ha convertido en un referente en la comunidad de inteligencia artificial. Presenta un enfoque de alto nivel que permite una fácil aplicación para soluciones de aprendizaje automático, siendo ampliamente utilizada como base para la implementación de numerosas redes neuronales. Algunos de los elementos clave que proporciona esta librería son: tensores, módulos y capas, optimizadores, funciones de activación e integración con GPU.

3.3.2 Scikit-learn

Scikit-learn es una biblioteca de software libre para aprendizaje automático en Python. Ofrece una amplia gama de algoritmos de clasificación, regresión y análisis de grupos, como máquinas de vectores de soporte, bosques aleatorios, Gradient boosting, K-means y DBSCAN. A diferencia de PyTorch, Scikit-learn se enfoca en el aprendizaje automático tradicional y estadístico, y es muy adecuada para problemas con conjuntos de datos de tamaño medio. Por contra PyTorch es especialmente adecuada para el aprendizaje profundo y proyectos de investigación en inteligencia artificial que requieren un mayor nivel de personalización y control.

3.3.3 OpenCV

OpenCV (Open Source Computer Vision) es una biblioteca de código abierto ampliamente utilizada para el procesamiento de imágenes y visión por computador. Proporciona una amplia gama de funciones y algoritmos para la manipulación, análisis y procesamiento de imágenes y vídeos. Además, provee diferentes modelos de inteligencia artificial. Las principales funcionalidades son: lectura y escritura de imágenes y vídeos, procesamiento de imágenes, detección y reconocimiento de objetos, seguimiento de objetos, visión estéreo o aprendizaje automático, entre otros.

3.3.4 NumPy

NumPy es una biblioteca para el lenguaje de programación Python creada para el cálculo matemático. Está especializada en vectores y matrices e implementa gran cantidad de funciones para trabajar con ellas, además de otras funciones matemáticas genéricas. Posee una fuerte integración con PyTorch, en particular en lo que respecta a los tensores. En la actualidad, se destaca como una de las bibliotecas más ampliamente empleadas para el cálculo matemático, y muchas otras bibliotecas se derivan de ella. De hecho, PyTorch aprovecha la funcionalidad de NumPy para llevar a cabo cálculos numéricos.

3.3.5 Matplotlib

Matplotlib es una biblioteca para la generación de gráficos en Python a partir de arrays o listas. Proporciona una forma fácil de crear gráficos de alta calidad, como gráficos de líneas, gráficos de dispersión, gráficos de barras, histogramas, gráficos de tarta y muchos otros tipos de gráficos como los que aparecen en la Figura 3.10. Es posible realizar modificaciones en los colores, tamaños e incluso agregar texto, entre otras opciones disponibles. Esta librería es ampliamente utilizada y cuenta con un sólido respaldo por parte de la comunidad. La gran mayoría de los gráficos presentes en este libro han sido creados utilizando esta biblioteca.

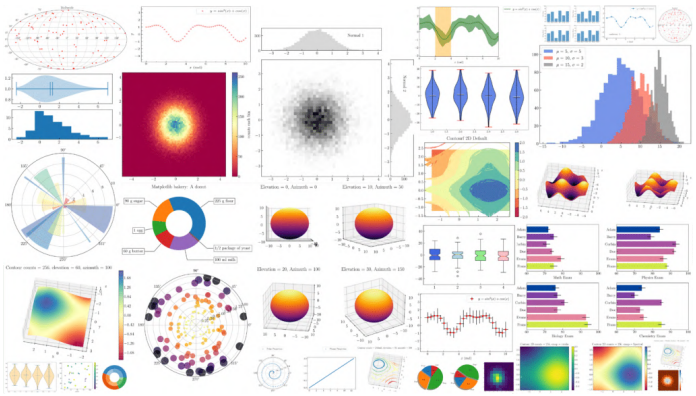


Figura 3.10: Ejemplos de gráficos creados con Matplotlib
Fuente: datascientest.com

3.4 CUDA

CUDA (Compute Unified Device Architecture) es una plataforma de computación paralela desarrollada por NVIDIA. Se utiliza para aprovechar el poder de las tarjetas gráficas (GPUs) en aplicaciones de alto rendimiento, como la inteligencia artificial. CUDA proporciona un modelo de programación y un conjunto de herramientas que permiten a los desarrolladores escribir código que puede ejecutarse en las GPUs de NVIDIA de manera eficiente.

En el campo de la inteligencia artificial, CUDA es ampliamente utilizado para acelerar el entrenamiento y la inferencia de modelos de aprendizaje profundo. Las GPUs son especialmente adecuadas para tareas altamente paralelas, como las operaciones matriciales y los cálculos

intensivos requeridos por las redes neuronales. Al utilizar CUDA, los desarrolladores pueden aprovechar el enorme poder de procesamiento de las GPUs y acelerar significativamente los algoritmos de aprendizaje automático y las aplicaciones de inteligencia artificial, ya sea al entrenar las redes como al realizar inferencias con ellas. En resumen CUDA nos permite sacarle el máximo partido a las tarjetas gráficas de NVIDIA para cualquier proyecto de inteligencia artificial.

3.5 Otras herramientas complementarias

A continuación se detallan otras herramientas software necesarias para desarrollar el proyecto.

- **GitHub**: Sistema de control de versiones en la nube.
- **Visual Studio Code**: Entorno gráfico de programación desarrollado por Microsoft. Permite la instalación de una amplia gama de *plugins* que extienden su funcionalidad. Entre ellos destacar:
  - **Docker**: permite una gestión de imágenes y contenedores de manera gráfica y cómoda. Además implementa la edición de archivos internos al contenedor.
  - **GitHub**: integración del sistema de control de versiones con el IDE (entorno de desarrollo integrado). Facilita la visualización de cambios y versiones anteriores, además del manejo de otras funcionalidades de *git* de manera gráfica.
  - **Jupyter**: cuadernos para *Python* que permite ejecutar código por secciones. Esto permite realizar pruebas ejecutando y modificando un bloque sin tener que ejecutar todo el archivo desde el inicio. Es altamente beneficioso para el aprendizaje de código.
- **Ubuntu**: Es un sistema operativo de código abierto basado en Linux que proporciona una interfaz amigable y una amplia gama de aplicaciones.
- **Compilación de documentos en LaTex**: OverLeaf es una plataforma en línea para la creación y edición colaborativa de documentos LaTeX, que ofrece un entorno de desarrollo integrado y simplifica la gestión de proyectos de escritura académica y científica. Este mismo libro ha sido escrito en esta plataforma.

Capítulo 4

Desarrollo

Nunca permitas que tus errores te definan;
permíteles enseñarte

Anónimo

4.1 Introducción

En esta sección se detalla el desarrollo e implementación del proyecto, desglosando los experimentos y decisiones tomadas para llegar a la propuesta. El proyecto toma KNDAR [6] como punto de partida. Como podemos ver en la Figura 4.1 se trata de una arquitectura modular. El *baseline* fue entrenado y validado en el *dataset* del apartado 3.1.4, el cual presenta tres puntos de vista sincronizados. El procesamiento se realiza de manera secuencial analizando tríos de fotogramas sincronizados. Se realiza de una manera casi independiente, solamente teniendo en cuenta el fotograma anterior. Los diferentes pasos de la arquitectura se detallan a continuación:

Entrada: Tres imágenes

1. **Extracción de las poses de los ocupantes detectados:** Con el estimador de la pose KAPAO [42] se extraen los puntos clave de todos los ocupantes del vehículo presentes en cada imagen.

2. **Generación de distancias y ángulos entre los *keypoints* del conductor:** Entre todas las personas detectadas, se elige el conductor. En cada imagen se calculan una serie de distancias y ángulos representativos entre diferentes partes del cuerpo del conductor. Además se calcula la distancia entre distintos puntos del fotograma análogo anterior y el actual. Todas las distancias se normalizan en función de la altura del conductor para mayor generalización. Las características extraídas se detallan en el Apéndice B.

3. **Unión de características entre las tres imágenes:** Se combinan los tres *arrays* de características en uno solo.

4. **Clasificador:** Se introduce el *array* con las características de los tres puntos de vista y se extrae la acción más probable de la contempladas en la Tabla 3.3.

5. **Post-procesamiento:** Se aplica un filtro temporal para descartar acciones más cortas y reducir parcialmente el ruido.

Salida: Una distracción de la Tabla 3.3.

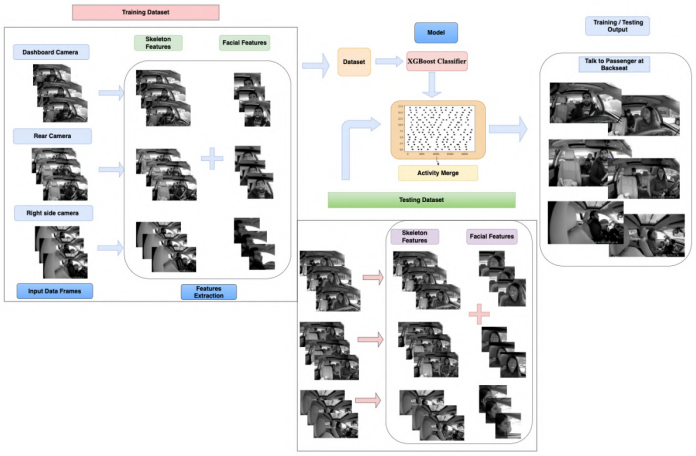


Figura 4.1: Flujo de trabajo de KNDAR
Fuente: openaccess.thecvf.com

4.1.1 Hipótesis e ideas de mejora

Una vez estudiado la arquitectura de referencia se proponen una serie de hipótesis para mejorar el *baseline*. En la Figura 4.2 se presenta la arquitectura propuesta resaltando las mejoras introducidas:

1. Se propone incorporar la detección de objetos de interés de la escena y personas, para fusionarlos con los puntos clave del conductor. Esto añadirá potencialmente contexto espacial y referencia a las detecciones.

2. Además se incrementará el número de características por fotograma, lo que probablemente mejore el rendimiento. Para poder implementar este sistema primero será necesario una detección de esos objetos. Los objetos estáticos, como el volante, se definirán al principio de la secuencia de vídeo. Si el objeto forma parte de la estructura del coche y la cámara está fijada al mismo, éste permanecerá siempre en la misma posición en la imagen. Esto descargará de trabajo a la red y reducirá los posibles fallos de detección del mismo. Por contra, los objetos que se muevan libremente por la imagen tienen que ser detectados por medio de una red neuronal. Una vez detectados habrá que fusionarlos con el esqueleto del conductor. De la misma forma, se calcularán distancias entre los objetos detectados y algunos puntos clave de interés. A continuación el clasificador obtendrá la acción más probable a partir de las características generadas.

3. Para mejorar el déficit de contexto temporal del *baseline*, se investigará un filtro que sin introducir mucho coste computacional, sea capaz de relacionar las acciones pasadas y así descartar cambios bruscos, ruido y falsas detecciones.

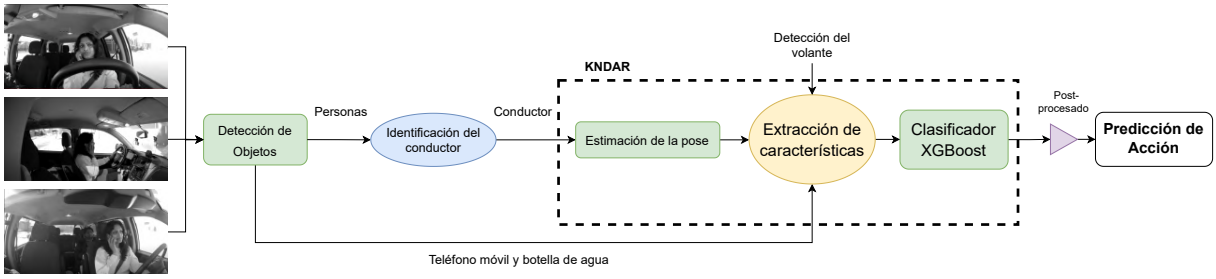


Figura 4.2: Estructura de la propuesta de mejora de KNDAR

4.2 Implementación de la detección de objetos

Como se ha indicado anteriormente se desea detectar objetos de interés relacionados con determinadas acciones. En el capítulo del estado del arte, se han expuesto las principales tecnologías actuales para esta tarea. En ese capítulo se explica además la necesidad de detectar a las personas para que el estimador de pose funcione correctamente. Es por ello que además de detectar personas, detectaremos objetos de interés para las acciones. Para este reto los dos objetos que están relacionados con una acción son el teléfono móvil, que se presenta en las acciones 2, 3, 4 y 5, y la botella de agua en la acción 1, como se observa en la Tabla 3.3. Para la acción de comer no hay ningún alimento visible de interés. Cuando se utilizan pesos entrenados en COCO, tanto las personas como los objetos son clases detectables.

YOLO es una gran elección si queremos un compromiso entre resultados y velocidad. Se trata de un detector *one stage* que logra muy buenos resultados en el estado del arte manteniendo un buen tiempo de inferencia. Para este proyecto el tiempo de inferencia es esencial ya que se trata de una aplicación en tiempo real. Una de las versiones más utilizadas de YOLO es la 5, pero con el lanzamiento de la 8 se propone realizar una comparación de su mejora para la detección de ocupantes de vehículo y objetos. En la sección 4.2.4 se valida la v3-spp, v5 y v8 en SViRO [37] y TiCAM [38], dos conjuntos de datos centrados en la detección de ocupantes de vehículos.

4.2.1 Formato de las anotaciones

Para poder evaluar un modelo en un *dataset* es necesario tener los datos, en este caso imágenes, y sus anotaciones, conocidas como *ground truth*. La verdad de referencia, en español, de un conjunto de datos se refiere a la información objetiva y precisa que se utiliza como base para entrenar o evaluar un modelo. En otras palabras, es la etiqueta o respuesta correcta para cada muestra en el conjunto de datos. De esta manera comparando la predicción de la red y la verdad podemos cuantificar su desempeño en nuestro *dataset*. El objeto detectado se anota con un rectángulo que lo contiene. Hay varias formas de definirlos dentro de una imagen. Para poder realizar una evaluación correcta ambos formatos tiene que coincidir, la predicción y la anotación.

Como observamos en la Figura 4.3, el formato de **YOLO** sitúa el extremo superior izquierdo de la imagen en el punto origen del eje de coordenadas. Además normaliza la imagen para que el tamaño tanto en abscisas y ordenas sea igual a uno. La caja viene definida por las coordenadas de su centro y el ancho y alto de la caja. Por tanto, una tupla de cuatro números flotantes (entre

cero y uno) definen una *bounding box* dentro de una imagen. Por cada imagen hay un documento de texto con el mismo nombre que contiene todos los objetos de la misma. Por cada elemento aparece un número correspondiente a la clase, la coordenada x del centro, la coordenada y del centro, el ancho y la altura.

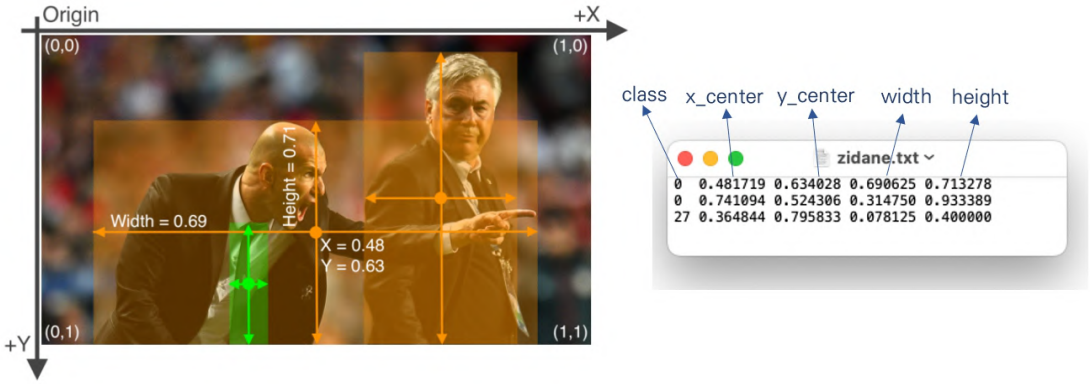


Figura 4.3: Representación del formato de las cajas en YOLO
Fuente: ultralytics.com

Por otro lado, nos encontramos el formato **COCO**. Al igual que el anterior sitúa la imagen en la misma posición, pero en este caso no se normaliza su tamaño. La caja viene representada por las coordenadas del extremo superior izquierdo de la caja y por su anchura y altura. En la Figura 4.4 podemos observar una comparativa entre los dos formatos y su conversión directa entre medidas.

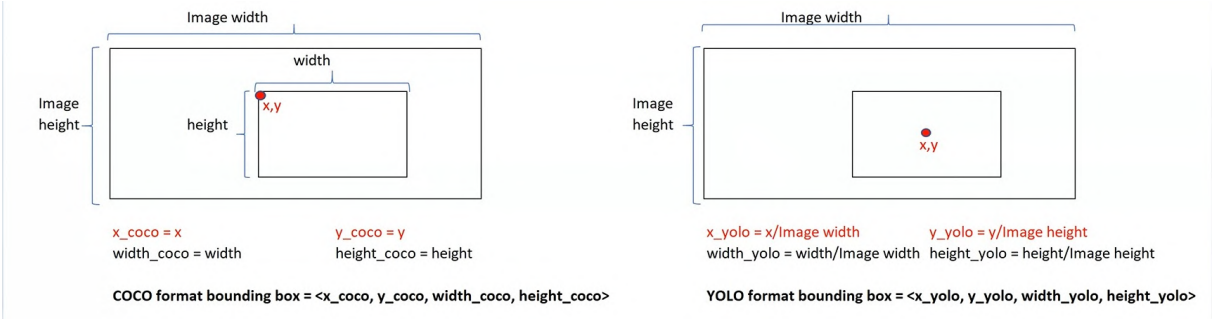


Figura 4.4: Comparación entre el formato YOLO y el formato COCO para detección de objetos
Fuente: prabhjotkaurgosal.com

Como hemos adelantado anteriormente el formato de los *datasets* es del tipo COCO. YOLO, el detector de objetos, tiene su propio formato. Es por ello que habrá que realizar una conversión para poder obtener métricas y realizar el entrenamiento. Se decidió convertir las anotaciones de los conjuntos de datos al formato YOLO. Para ejemplificar esta conversión se utiliza como referencia la Figura 4.5.

Para convertir la *bounding box* del gato del formato COCO al formato YOLO, se deben seguir los siguientes pasos:

1. Obtener las coordenadas en formato COCO de la *bounding box* del gato:

$$[x, y, w, h] = [98, 345, 322, 117].$$

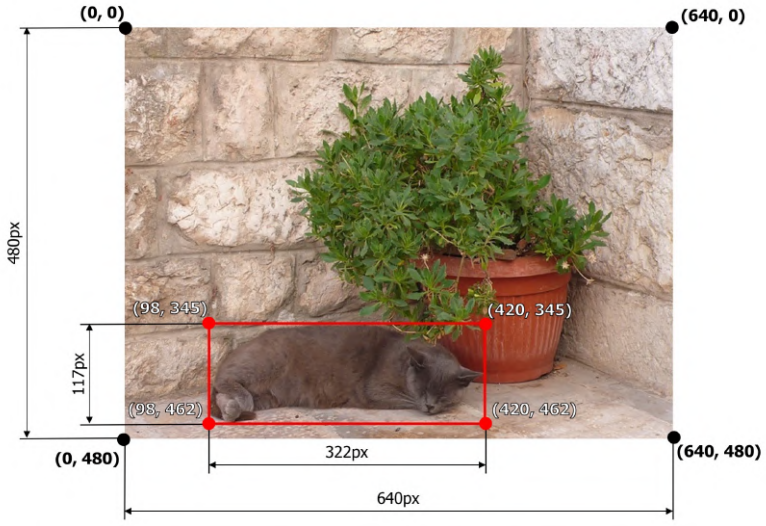


Figura 4.5: Ejemplo de *bounding box* de la detección de un gato
Fuente: albumentations.ai

2. Obtener el tamaño de la imagen: (width,height) = (640,480)

3. Calcular las coordenadas del centro de la *bounding box* y normalizarlas por las dimensiones de la imagen, utilizando las siguientes fórmulas:

$$x_{center} = \frac{x + \frac{w}{2}}{width} = \frac{98 + \frac{322}{2}}{640} = 0{,}4047 \tag{4.1}$$

$$y_{center} = \frac{y + \frac{h}{2}}{width} = \frac{345 + \frac{117}{2}}{480} = 0{,}8406 \tag{4.2}$$

4. Normalizar el ancho y alto de la *bounding box* por las dimensiones de la imagen, utilizando las siguientes fórmulas:

$$w_{norm} = \frac{w}{width} = \frac{322}{640} = 0{,}5031 \tag{4.3}$$

$$h_{norm} = \frac{h}{height} = \frac{117}{480} = 0{,}2438 \tag{4.4}$$

Esto nos dará como resultado:

$$[x_{center}, y_{center}, w_{norm}, h_{norm}] = [0{,}4047, 0{,}8406, 0{,}5031, 0{,}2438].$$

Para implementar la conversión en Python se utiliza la siguiente función:

```
def convert_coco_to_yolo(bbox, img_width, img_height):
    #Extraemos los cuatro valores del array
    x_min, y_min, width, height = bbox
    #Calculamos las coordenadas normalizadas del centro de la bbox
    x_center = (x_min + width / 2) / img_width
    y_center = (y_min + height / 2) / img_height
    #Normalizamos el ancho y alto de la bbox
```

```
    yolo_width = width / img_width
    yolo_height = height / img_height
    #Retornamos los cuatro valores en un array
    return [x_center, y_center, yolo_width, yolo_height]
```

Finalmente se iterará entre las diferentes imágenes modificando cada una de las cajas de cada clase contenida. De esta forma se tendrá el *dataset* preparado para evaluar y entrenar en las distintas versiones del detector.

4.2.2 Métricas

Para poder valorar los resultados se definen una serie de métricas. Entenderlas es de vital importancia para seleccionar la red más adecuada para nuestra arquitectura y necesidades. Quizás la primera pregunta que nos podemos hacer es:

¿Cuándo se considera que un objeto se ha detectado correctamente? Para dar solución a esta pregunta se introduce el concepto de *IOU (Intersection over Union)* o intersección sobre unión en español. Se utiliza para medir cuánto se superponen las predicciones del modelo con las verdades de referencia. Se define como:

$$IoU = \frac{A \cap B}{A \cup B} \tag{4.5}$$

Se toma la intersección entre la predicción del modelo y la verdad de referencia, y se divide por la unión de ambas áreas. En otras palabras, el IOU mide cuánto se superponen las dos áreas. Por ello, habrá que elegir un umbral mínimo a partir del cual se considere como correcta o verdadera la detección. En la Figura 4.6 podemos ver como afecta el umbral mínimo en la validación de una detección de un gato.

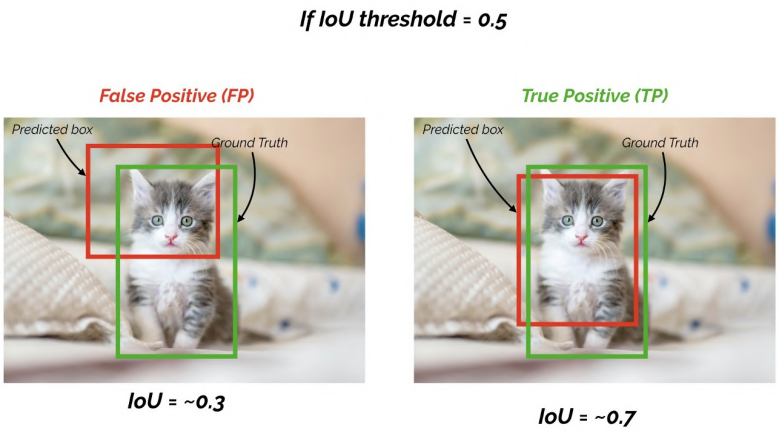


Figura 4.6: Impacto del umbral de IOU para la elección de un verdadero o falso positivo
Fuente: www.v7labs.com

Habiendo definido como se determina cuando una detección es correcta o no, aparecen tres posibilidades:

- **Verdadero positivo (TP)**: La detección y la verdad coinciden. El objeto se ha detectado correctamente.
- **Falso negativo (FN)**: El objeto no se ha detectado. Por tanto hay un objeto en la verdad que no aparece en la detección.
- **Falso positivo (TP)**: Se ha detectado un objeto erróneamente. En la detección aparece un objeto que en la verdad no.

Con estos tres conceptos, se definen dos métricas; *precision* y *recall*.

- La **precisión** mide la proporción de verdaderos positivos (objetos detectados correctamente) entre todos los objetos detectados. Se calcula como:

$$Precision = \frac{TP}{TP + FP} \tag{4.6}$$

  Donde TP es el número de verdaderos positivos y FP es el número de falsos positivos. Un modelo con una alta precisión tiene menos falsos positivos y se considera más preciso.

- El **recall** mide la proporción de verdaderos positivos entre todos los objetos que deberían haber sido detectados. Se calcula como:

$$Recall = \frac{TP}{TP + FN} \tag{4.7}$$

  Donde FN es el número de falsos negativos. Un modelo con un alto recall detecta más objetos positivos y se considera más sensible.

El mAP@.5:.95 es una medida más completa que combina la precisión y el recall a lo largo de varios umbrales de confianza para producir una única métrica que resume el rendimiento del modelo de detección de objetos en todo el rango de umbrales de confianza. En este acaso los umbrales van de 0.5 hasta 0.95 con incrementos de 0.05. El mAP para cada uno de los umbrales viene definido por la siguiente fórmula:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{4.8}$$

El valor de N es variable según la literatura, pero normalmente vale 11. Lo que se consigue con esta formula es hallar el área debajo de la curva Precision-Recall interpolando la curva original con N puntos. En la Figura 4.7 se observa en naranja la curva original y en verde la interpolada. Finalmente se obtiene el área encerrada debajo de la curva verde calculando la integral:

$$\text{Área} = \int_0^1 p(r)dr \tag{4.9}$$

De esta manera se obtendrá el mAP para cada uno de los intervalos y se realizará la media entre ellos.

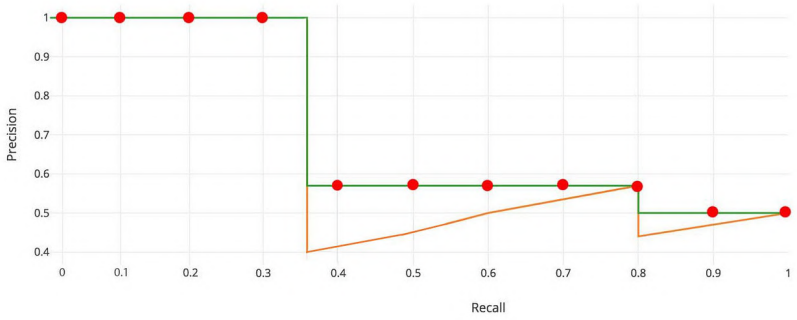


Figura 4.7: Ejemplo de una curva Precision-Recall para cálculo de mAP
Fuente: medium.com

4.2.3 Resultados de la validación

Todos los modelos de cada versión entrenados con COCO se probaron en ambos conjuntos de datos para encontrar el que se adaptaba mejor. Además, la red elegida se reentrenará para obtener mejores resultados y adaptarla al contexto de la aplicación. El conjunto de datos se compone de SVIRO [37] y TiCAM [38] que se centran en el estudio de ocupantes de vehículos. Como podemos observar en la Tabla 4.1, hemos evaluado YOLOV3 en su versión spp, la cual usaremos como *baseline* para compararla con todos los tamaños de YOLOv5 y YOLOv8. Además de las métricas se adjunta el tiempo de inferencia para la gráfica NVIDIA RTX 2080 Ti. Se calcula como la suma de preprocesado, inferencia y NMS (postprocesado).

Tabla 4.1: Tabla comparativa de las versiones 3, 5 y 8 de YOLO evaluados en SVIRO y TiCAM. **P**: Precision; **R**: Recall; **TI**: Tiempo de inferencia (suma de preprocesado + inferencia + NMS (postprocesado))

Modelo	**Pesos**	**P**	**R**	**mAP@.5**	**mAP@.5:.95**	**TI (ms)**
YOLOv3	yolov3-spp.pt	0.955	0.876	0.931	0.837	0.1 + 8.2 + 0.5 = 8.8
YOLOv5	yolov5n.pt	0.901	0.739	0.835	0.515	0.1+ 0.9 + 1.0 = 2.0
YOLOv5	yolov5s.pt	0.898	0.797	0.878	0.624	0.1 + 1.8 + 0.7 = 2.6
YOLOv5	yolov5m.pt	0.916	0.862	0.931	0.765	0.1+ 3.9 + 0.6 = 4.6
YOLOv5	yolov5l.pt	0.956	0.892	0.954	0.838	0.1 + 6.8 + 0.6 = 7.5
YOLOv5	yolov5x.pt	0.961	0.909	0.958	0.861	0.1 + 11.6 + 0.6 = 12.3
YOLOv8	yolov8n.pt	0.925	0.751	0.868	0.7	0.1 + 1.1 + 0.6 = 1.8
YOLOv8	yolov8s.pt	0.938	0.836	**0.93**	**0.809**	0.1 + 2.4 + 0.6 = **3.1**
YOLOv8	yolov8m.pt	0.948	0.865	**0.947**	**0.85**	0.1 + 5.1 + 0.5 = **5.7**
YOLOv8	yolov8l.pt	0.95	0.875	**0.951**	**0.873**	0.1 + 8.6 + 0.5 = **9.2**
YOLOv8	yolov8x.pt	0.958	0.894	0.961	0.887	0.1 + 12.8 + 0.6 = 13.5

Tras comparar todos los modelos observamos que la versión 8 tiene un rendimiento superior. Los tamaños más equilibrados son el s, m y l, ya que obtienen muy buenos resultados con un tiempo de inferencia contenido.

4.2.4 Entrenamiento

Para obtener mejores resultados en nuestro conjunto de datos, se realizará un ***transfer learning***. Es una técnica utilizada que consiste en aprovechar el conocimiento y la experiencia obtenidos al resolver una tarea para ayudar a resolver otra tarea relacionada. En lugar de entrenar un modelo desde cero se toma uno pre-entrenado en una tarea previa y se utiliza como punto de partida. En nuestra situación la red se entrenó con COCO, este *dataset* contiene multitud de fotos y contextos. Esta característica es esencial para poder adaptarse a más variedad de entornos y obtener una mejor generalización. Por otro lado, los *datasets* de entrenamiento solo contemplan la clase "persona". Es por ello que partiendo de COCO se aprovechan conocimientos previos del modelo, lo que repercutirá en mejores resultados, además de incluir clases de objetos que en los *datasets* de entrenamiento no están incluidos. Se entrenarán los tres modelos con mejores características durante 25 épocas (cada una de las iteraciones de entrenamiento). En la Tabla 4.2 podemos observar el resultado de validar las tres redes afinadas en nuestro conjunto de datos. Además en la Figura 4.10, podemos ver una comparativa gráfica entre todos los modelos validados.

Tabla 4.2: Tabla comparativa de los modelos de YOLOv8 entrenados y validados en SVIRO y TiCAM. **P**: Precision; **R**: Recall; **TI**: Tiempo de inferencia (suma de preprocesado + inferencia + NMS (postprocesado))

Modelo	**Pesos**	**P**	**R**	**mAP@.5**	**mAP@.5:.95**	**TI (ms)**
YOLOv3	yolov3-spp.pt	0.955	0.876	0.931	0.837	0.1 + 8.2 + 0.5 = 8.8
YOLOv8	yolov8s.pt (trained)	0.962	0.909	0.961	0.909	0.1 + 2.3 + 0.4 = 2.8
YOLOv8	yolov8m.pt (trained)	0.98	0.931	0.98	0.941	0.1 + 5.1 + 0.4 = 5.6
YOLOv8	yolov8l.pt (trained)	0.972	0.951	0.986	0.941	0.1 + 8.6 + 0.5 = 9.2

Se extraen las siguientes conclusiones:

- Para la **s** se reduce en un **68.18 %** la velocidad y se aumenta en un **8.6 %** el mAP@.5:.95.
- Para la **m** se reduce en un **36.36 %** la velocidad y se aumenta en un **12.43 %** el mAP@.5:.95.
- Para la **l** se reduce en un **-4.55 %** la velocidad y se aumenta en un **12.43 %** el mAP@.5:.95.

Elegiremos la versión que de una manera equilibrada mejore tanto rendimiento como tiempo de inferencia. Por ello la mejor solución para nuestras necesidades es **YOLOv8m afinada**. A continuación se presentan algunos detalles del entrenamiento de la versión elegida. Como hemos adelantado anteriormente se realizó un total de 25 épocas o repeticiones de entrenamiento. Se trata de iteraciones completas a través de todos los datos de entrenamiento. Durante una época, el modelo procesa y actualiza sus parámetros utilizando un conjunto de datos de entrenamiento. Al final de cada época se valida en un conjunto de datos nunca visto, para conocer la eficacia de la red. En la Figura 4.8 se detalla la evolución del entrenamiento a lo largo de las 25 épocas. En la primera fila se encuentra la evolución de la función de pérdida y las métricas para el *set* de entrenamiento (*train*) y en la segunda fila para el de validación (*val*).

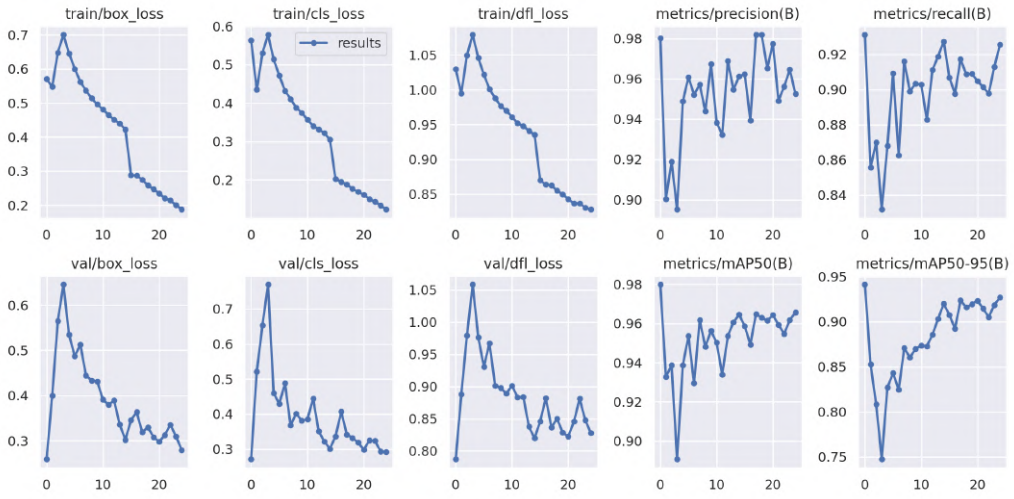


Figura 4.8: Curvas del re-entrenamiento de YOLOv8

El objeto más importante detectado es el conductor. Un falso negativo invalidaría el resto de la red. En la Figura 4.9 se presenta una matriz de confusión donde se muestra el gran desempeño de la red afinada en esta clase.

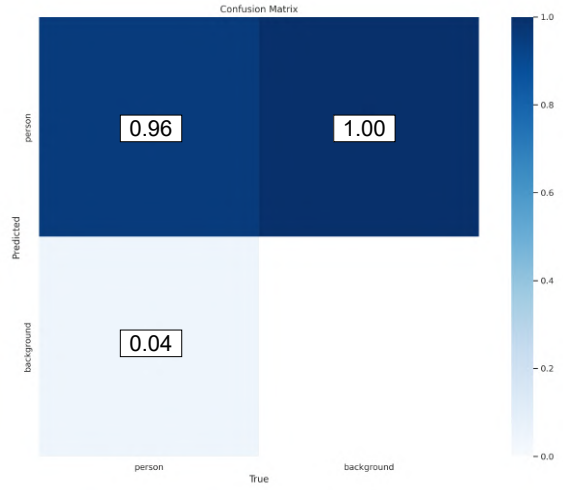


Figura 4.9: Matriz de confusión para la clase *person* en el modelo YOLOv8-m afinado.

Como conclusión, se ha obtenido un detector de objetos adaptado a nuestra temática. Centrado en la detección de conductores, botellas de agua y teléfonos móviles. Además obtiene un tiempo de inferencia muy apto para aplicaciones en tiempo real.

4.3 Estimador de poses

Como se aprecia en la Figura 4.2, una vez detectada la persona se enviará su caja al estimador de poses. Como se ha explicado en el capítulo 2.4 las redes con un *top-down approach* necesitan de esta detección previa para estimar los diferentes puntos clave del humano. Estudiando el estado del arte de las principales redes se ha elegido AlphaPose por tres razones principales:

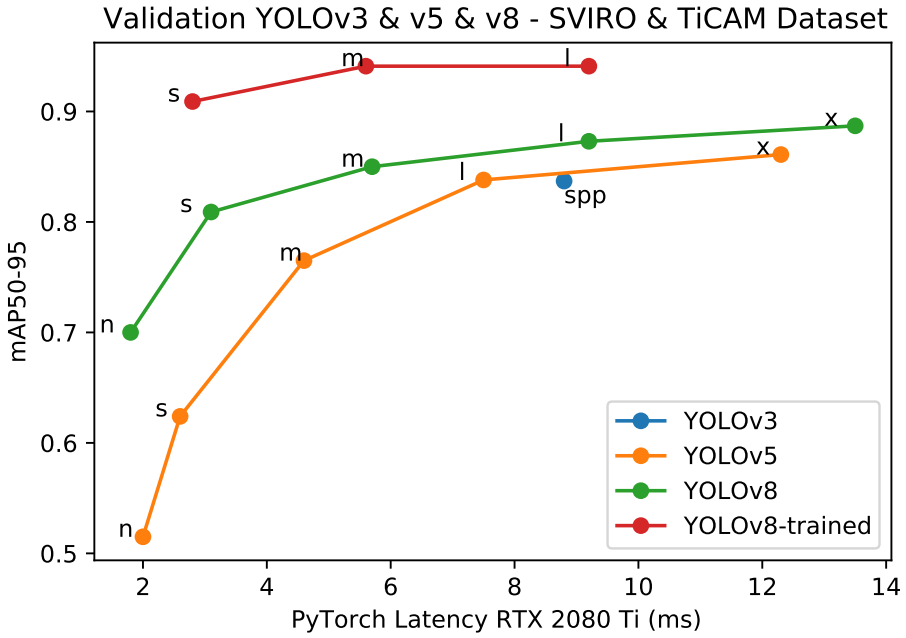


Figura 4.10: Relación mAP@.5:.95 y tiempo de inferencia para cada una de los modelos validados y entrenados

- **Buenos resultados**: AlphaPose se encuentra entre las principales redes estudiadas en el estado del arte, demostrando un rendimiento alto.
- **Lenguaje de programación**: La red ha sido implementada en Python, lo cual facilitará su modificación e integración en nuestra arquitectura.
- **Gran comunidad y documentación**: Se trata de una red muy utilizada y ampliamente soportada por la comunidad. Además hay multitud de documentación y tutoriales disponibles que facilitan su uso.

4.3.1 Integración de YOLO con AlphaPose

Se procede a la integración del detector con el estimador de la pose. AlphaPose trabaja con YOLOv3 [29] y EfficientDet [43] como detectores integrados. El código de la versión 8 de YOLO es sustancialmente diferente al de la versión 3, por tanto no hay compatibilidad. Es importante conocer el formato de datos de entrada para la red de estimación de la pose. Por un lado hay que introducir la imagen a inferir. Por otro, la *bounding box* de las personas de las que queremos obtener su pose. La caja de entrada de las personas tiene el siguiente formato:

$$[array([x_{min}, y_{min}, x_{max}, y_{max}], dtype = float32)] \tag{4.10}$$

Se trata de una lista de *NumPy arrays* con cuatro elementos cada uno. El formato es ligeramente diferente al formato COCO mencionado anteriormente. Se trata de *pascal voc*, el cual suma a x_{min} el ancho para obtener x_{max} y a y_{min} suma el alto para obtener y_{max}. Como se ha comentado anteriormente YOLO usa su propio formato y habrá que convertirlo a este. El array se devuelve

en forma de tensor, con la función *xyxy* de torch convierte al formato *pascal voc* directamente y con la función *cpu* y *numpy* convierte el tensor en un array de *NumPy* guardado en CPU. Añadir que YOLO devuelve todos los objetos detectados de las clases con las que ha sido entrenada, por ello no recibiremos solamente personas. Habrá que filtrar por *id*, y guardar en un array diferente los objetos detectados de cada clase.

Añadir, que YOLO permite realizar la inferencia por *batch*. Esto significa que es capaz de procesar varias imágenes sin apenas aumentar su tiempo de inferencia total. Esto dotará de mayor velocidad a la red. El tamaño del *batch* dependerá directamente de la memoria que posea la GPU. En nuestro caso la tarjeta gráfica usada permite introducir un *batch* de tres imágenes, ideal para procesar un fotograma. Para obtener la *bounding box* de los objetos de un array de imágenes, se hará de la siguiente forma en Python:

```
#Cargar el modelo
nombre_modelo = "yolov8m-trained.pt"
self.modelo = YOLO(nombre_modelo)
#Realizar inferencia de un array de imágenes (en batch)
img = [dashboard, rear, right]
results = self.modelo.predict(img,verbose=False)
```

Será entonces la variable *results* la que habrá que iterar separando los objetos por clase y convirtiendo su formato como se ha explicado anteriormente. Las clases que no son personas se usarán más tarde para realizar la fusión. Serán las detecciones de la clase 0 (persons) las que se transmitirán al estimador de la pose.

Primero se filtrarán los humanos detectados para obtener solo la caja del conductor. Ya que así ahorramos tiempo de cómputo al eliminar los posibles pasajeros o incluso personas en el exterior del vehículo. Suponemos que el coche se conduce por el lado izquierdo. Como podemos ver en la Figura 4.11, de todas las personas detectadas, elegiremos la que esté situada en esa zona y cuya caja sea la mayor para asegurarnos de que es la que está sentada en primera fila, ya que por perspectiva será mayor.

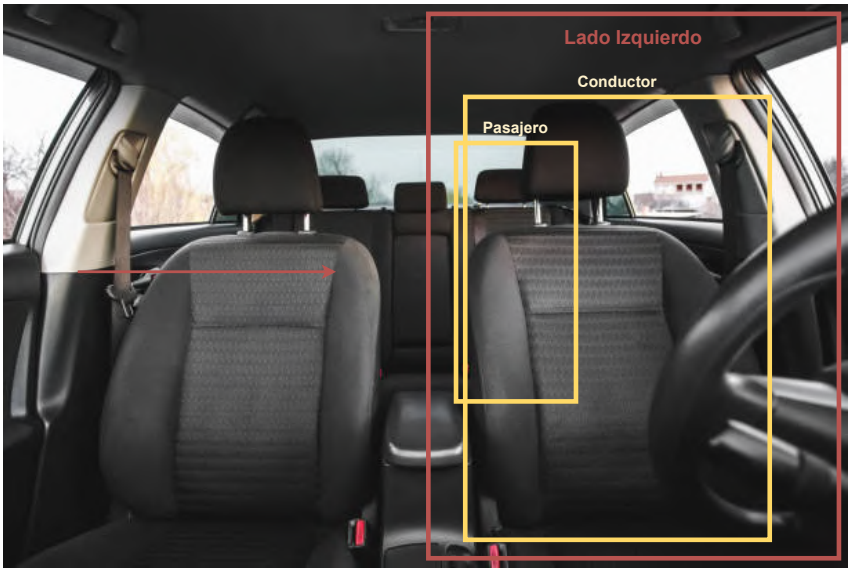


Figura 4.11: Posible localización de conductor y pasajero en un coche

4.3.2 Creación de script combinado

Una vez realizada la integración entre el detector de objetos y el estimador de la pose humana, se desarrollará un *script* que permita una fácil implementación. El objetivo es crear una clase, que cuando cree un objeto cargue ambas redes. El objeto creado tiene un método (*boxesandposes*) que acepta como argumento un array de imágenes y devuelve un array por cada cada clase de objeto detectado y la pose del conductor. Se usa de la siguiente manera:

```
#Crear objeto = Cargar ambos modelos
estimator = YoloAlphapose()
#Realizar inferencia de un array de imágenes (en batch)
poses,bboxes,phone,bottle = estimator.boxesandposes(img)
```

De este modo se tiene de forma unificada el módulo de detección de objetos y estimación de la pose. En la Figura 4.12 se detalla el funcionamiento de este *script*.

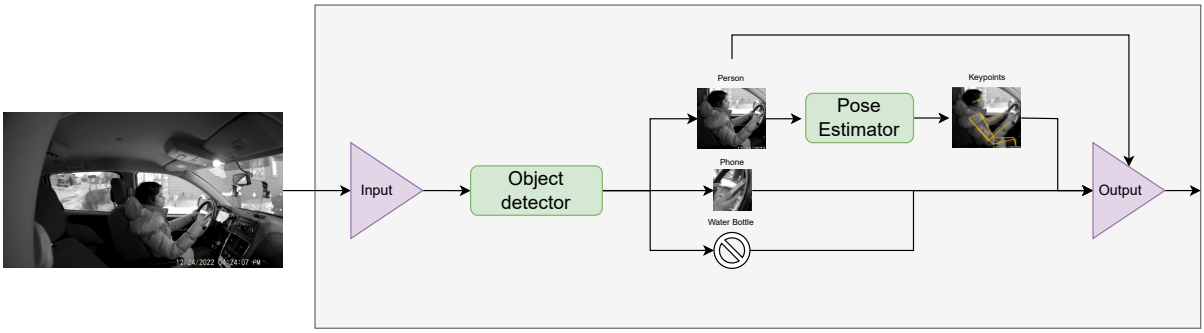


Figura 4.12: Esquema gráfico del flujo de datos dentro del script

4.4 Integración de la estimación de la pose con KNDAR

Como se ha comentado en varias ocasiones partimos de KNDAR [6] como *baseline*. Este proyecto utiliza KAPAO [42] como estimador de la pose. En el repositorio [1] encontramos varios ficheros de código Python. Estos se estructuran de la siguiente manera:

- **Extracción de características**: Primeramente se extraen los puntos clave de cada fotograma del *dataset*, además se generan las distancias y ángulos entre ellos. Se encargan principalmente los ficheros *extract.py* y *features.py* de llevarlo a cabo. Además, si se trata del *dataset* de entrenamiento se añade el *ground truth* a cada frame, junto con las características.

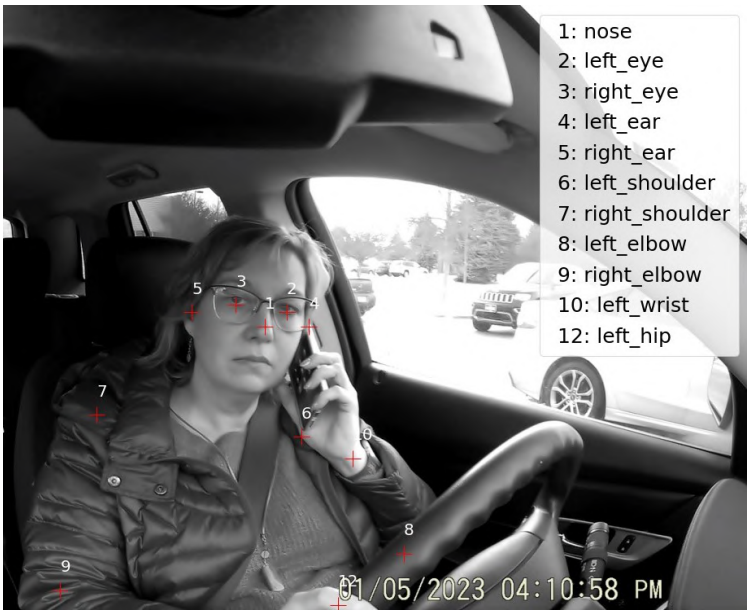
- **Entrenamiento del clasificador**: Una vez convertido el *dataset* de entrenamiento y validación a características, se entrenará el clasificador con el archivo *train.py*. Teniendo la posibilidad de probar con varios parámetros para obtener el mejor resultado. Internamente divide el *dataset* de entrenamiento en dos partes, uno para entrenar y otro para validar el entrenamiento.

[1] https://github.com/davidanastasiu/kndar

- **Estimación del *dataset* de validación**: Con los pesos del clasificador se inferirá el *dataset* de validación obteniendo como salida un archivo de texto con un formato determinado. Cada fila se dedica a una acción inferida y se añade el inicio y final dentro del vídeo expresado en segundos. El fichero *test.py* se encarga de realizar esta función.

Encontramos más archivos que dependen directamente de los mencionados, que se encargan de gestionar el *dataset* o realizar cálculos, entre otros. Observando la estructura general del repositorio, modificando la sección de inferencia de *keypoints*, se puede realizar una integración sin problema.

Como hemos realizado anteriormente, es necesario identificar y entender el formado de datos de entrada y de salida de ese bloque. Los datos de entrada son imágenes en formato *NumPy array*. Si el *dataset* proporciona imágenes infrarrojas es importante ampliar a tres dimensiones, para que sea procesada por AlphaPose. Al igual que las imágenes en blanco y negro solo tienen una dimensión. El formato de salida de kapao es el mismo que AlphaPose, por tanto, no hay que realizar ninguna conversión de formato. Ambos utilizan el formato de COCO que incluye un total de 17 puntos clave. Estos se encuentran en *NumPy array*. Este posee una fila por cada punto clave, en la cual podemos encontrar tres columnas. Las dos primeras correspondientes al eje de coordenadas del punto clave en la imagen y el tercero a la confianza de la predicción de ese punto. En la Figura 4.13 encontramos un ejemplo del formato mencionado. Recalcar que las confianzas menores de un *threshold* se ponen a cero y estos puntos no se muestran en la imagen.



Punto	x	y	Conf.
1	964.42	505.91	0.86138
2	1005.1	460.02	0.85724
3	904.13	453.87	0.88792
4	1059.8	467.17	0.74065
5	837.6	479.84	0.82952
6	1200.3	643.41	0.75895
7	695	695.72	0.84567
8	1407.9	886.02	0.88745
9	492.13	943.7	0.74587
10	1481.4	904.41	0.91548
11	544.72	1006.4	0
12	1207.1	1084.8	0.83457
13	874.6	1108.4	0
14	1346	1069.5	0
15	759.47	1085.6	0
16	1245	1071.6	0
17	886.76	1066.8	0

Figura 4.13: Representación gráfica y valores de los puntos clave en formato COCO.

4.5 Generación de características

A continuación, se generan las características que alimentan el algoritmo de clasificación. Partimos de las propuestas en el proyecto KNDAR [6] que aparecen recogidas y numeradas en el Anexo B. Se calculan los ángulos (0-15) y distancias (16-33) entre puntos clave del conductor

que interactúan o están relacionados entre sí. Además, se añade el desplazamiento de posición (38-48) y de ángulo (49-55) de posición con respecto al fotograma anterior de algunos puntos. Todas ellas son normalizadas con respecto al tamaño de la persona (34-37). Sumamos a las mencionadas una serie de distancias por cada objeto de interés detectado. Si el objeto no aparece esa distancia se establece como *null*. Encontramos las siguientes:

- Para la detección del **teléfono móvil** (+5) se ha añadido una categoría binaria que indica su presencia en la escena, además de la distancia entre la muñeca izquierda, la muñeca derecha, la oreja izquierda y la oreja derecha al centro de la caja delimitadora del teléfono móvil. Nótese que si el módulo de detección detecta más de un teléfono móvil o botella, se queda con el que está más cerca del conductor que es más probable que interactúe con él.
- Lo mismo ocurre con la **botella de agua** (+4), que también tiene una clase que informa de su existencia y de su distancia a las muñecas del conductor y a su nariz.
- Se incluye el **volante** (+4) anotado manualmente. Para este caso introducimos dos variables categóricas que indican si cada una de las manos del conductor se encuentra dentro del cuadro delimitador del volante. Además, se calcula la distancia de éste a cada muñeca del conductor. Esto puede ayudar a reducir las detecciones de falsos positivos y mejorar el rendimiento global.

Podemos ver en la Figura 4.14 las distancias dibujadas en la imagen tanto para el teléfono móvil como para el volante y en la Figura 4.15 las distancias para la botella de agua. A las 56 distancias originales se les suman las 13 mencionadas. En total se obtienen 207 características (valores) por fotograma, lo que corresponde a 69 por punto de vista.

Figura 4.14: Distancias del centro del volante y del teléfono móvil a muñecas y orejas (sólo para el teléfono móvil).

4.6 Entrenamiento del clasificador

XGBoost es un algoritmo de aprendizaje automático basado en árboles de decisión y *gradient boosting* que se utiliza comúnmente para problemas de clasificación y regresión. Al entrenar XGBoost Classifier, hay varios hiper-parámetros que se pueden ajustar para mejorar el rendimiento del modelo.

Figura 4.15: Distancias del centro de la botella de agua a las muñecas del conductor.

- **n_estimators**: Es el número de árboles que se utilizan en el modelo. Un valor más alto generalmente mejora el rendimiento del modelo, pero también aumenta el tiempo de entrenamiento.
- **max_depth**: Define la profundidad máxima de cada árbol en el modelo. Controla la complejidad del modelo y evita el sobre-ajuste. Un valor más alto permite que los árboles sean más profundos y más complejos.
- **learning_rate**: Es la tasa de aprendizaje del modelo, que determina cuánto contribuye cada árbol al modelo general. Un valor más bajo requiere más árboles para alcanzar el mismo rendimiento, pero puede mejorar la generalización del modelo.
- **subsample**: Es la proporción de muestras utilizadas para entrenar cada árbol. Un valor menor reduce la varianza pero puede aumentar el sesgo del modelo.
- **colsample_bytree**: Es la proporción de características (columnas) utilizadas para entrenar cada árbol. Un valor menor reduce la complejidad y el sobre-ajuste del modelo.
- **reg_alpha y reg_lambda**: Son los parámetros de regularización L1 y L2 respectivamente, que ayudan a controlar el sobre-ajuste penalizando los pesos de las características.

La selección adecuada de los parámetros es crucial para obtener un buen rendimiento. Cada problema requiere una configuración y una decisión de los parámetros. Para lograr la combinación óptima se realizarán diferentes entrenamientos probando diversas combinaciones y se elegirá la que mejores resultados de validación obtenga. Ciertos parámetros serán fijos y toman un valor por defecto adecuado para la mayoría de entrenamientos. Será el caso del ***learning_rate*** que se establece a **0.1**, el ***subsample*** a **0.8**, ***colsample_bytree*** a **0.8**, ***reg_alpha*** a **0.01** y ***reg_lambda*** a **0**. Se probará con diferentes valores de ***n_estimators*** y ***max_depth*** ya que son los más dependientes del tipo y cantidad de datos y modifican más drásticamente el modelo. En la Tabla 4.3 y Figura 4.16 se observan las diferentes combinaciones probadas.

El alto rendimiento en los resultados se debe a que el conjuntos de datos de validación procede del mismo conjunto de conductores que *training*. Eso implica que se valida en el mismo conductor y con las mismas acciones que el entrenamiento. Al validar en el conjunto de datos de test y encontrarse conductores diferentes los resultados descienden. Observando los resultados

Tabla 4.3: Evaluación del modelo para diferentes parámetros del clasificador

n_estimators	**max_depth**	**test_f1**
400	6	0.9386
	8	0.9447
	10	0.9459
	12	0.9454
500	6	0.9519
	8	0.9557
	10	0.9566
	12	0.9559
600	6	0.9637
	8	0.9668
	10	**0.9674**
	12	0.9663
800	6	0.9576
	8	0.9580
	10	0.9579
	12	0.9575
1000	6	0.9494
	8	0.9496
	10	0.9492
	12	0.9453

de las diferentes combinaciones validadas encontramos que los parámetros más óptimos son **600** para ***n_estimators*** y **10** para ***max_depth***.

Para el entrenamiento se define la función de pérdidas, que es una medida que cuantifica el error entre las predicciones del modelo y los valores reales en el problema de clasificación. Esta función guía al algoritmo durante el proceso de entrenamiento para minimizar el error y mejorar la precisión del modelo. Como se trata de una clasificación multi-clase se elige como función de pérdidas la *multiclass logloss*:

$$L_{\log}(y, p) = -(y \log(p) + (1 - y) \log(1 - p)) \tag{4.11}$$

Por otro lado, también se define el error como la tasa de error de clasificación multi-clase. Este se refiere a la discrepancia o diferencia entre el valor predicho por un modelo y el valor real o verdadero en un conjunto de datos. Por ello se calcula como:

$$\text{multiclass classification error rate} = \frac{\text{wrong cases}}{\text{all cases}} \tag{4.12}$$

El objetivo principal en el aprendizaje automático es minimizar el error del modelo para que las predicciones se aproximen lo más posible a los valores reales. En la Figura 4.17 se observa la evolución de la función de pérdidas y del error a lo largo del entrenamiento. Como podemos ver

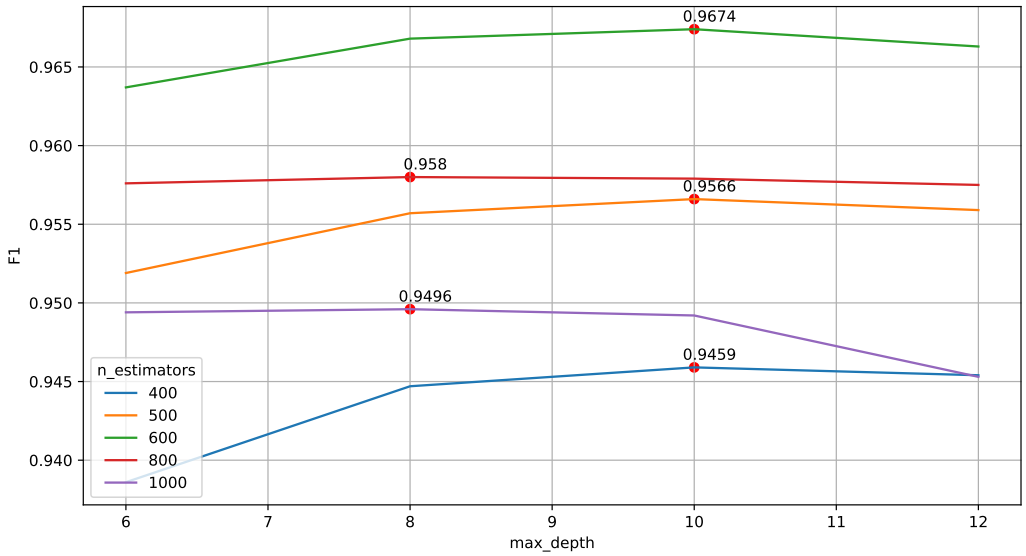


Figura 4.16: Resultados de F1 en testing para diferentes valores de *n_estimators* y *max_depth*

ambos se minimizan con el paso de las épocas (iteraciones de entrenamiento), hasta alcanzar una asíntota.

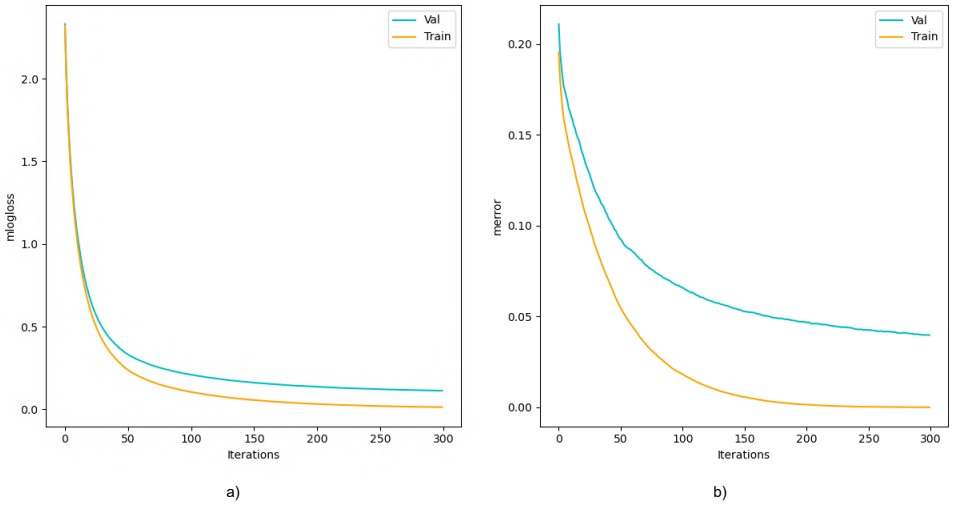


Figura 4.17: Gráficas de a) la función de pérdidas y b) el error del modelo a lo largo del entrenamiento

4.7 Mejora postprocesado

Para la mejora del rendimiento de las inferencias se propone un postprocesado doble. El primero añadirá contexto temporal, eliminando el ruido presente. Y el segundo, eliminará acciones muy cortas en el tiempo en función de la naturaleza de la misma. Las probabilidades de salida del clasificador XGBoost representan la probabilidad de pertenencia de una instancia a una

clase específica. La función de enlace softmax utilizada en XGBoost asegura que las probabilidades de todas las clases sumen uno. Es por ello que cualquier modificación que se realice a las probabilidades de salida habrá que normalizarlas para que sumen uno nuevamente.

4.7.1 Filtro Paso Bajo

Uno de los grandes problemas de esta arquitectura, es la carencia de correlación temporal entre fotogramas. Al inferir una acción fotograma a fotograma se pierde el contexto temporal de la actividad del conductor y se produce un resultado ruidoso. El *baseline* propone un filtro que toma la acción más probable de cada fotograma y aplica un algoritmo personalizado. Al hacer esto se pierde mucha información acerca de la confianza de la predicción. Es por ello que nuestra arquitectura propone un filtro que procesa directamente la confianza en una ventana temporal, que no es más que la seguridad de la red acerca de una predicción. Para cada una de las 16 distracciones y una correspondiente a la "no distracción", tendremos un flujo de probabilidades de confianza. A cada una se le aplicará un filtro paso bajo de forma independiente. Así se eliminarán los cambios bruscos de probabilidad. A diferencia de los filtros basados en promedios, éste elimina totalmente las acciones consideradas como ruido sin modificar la probabilidad de las que no lo son. De este modo, la curva se suavizará significativamente y se eliminarán los picos correspondientes a falsos positivos en fotogramas aislados. A continuación, entre las 17 probabilidades de un fotograma, se tomará la más alta de las 16 distracciones, siempre que superen el umbral mínimo. Si ninguna de ellas lo supera, se establecerá la "no distracción". Para llevar a cabo este proceso se implementa un filtro de tipo Butterworth a través de la librería scikit-learn. Tras varias pruebas se concluyó que los parámetros que obtienen mejores resultados para una generación continua a 15 Hz de probabilidades con frecuencia de corte a 2 Hz y orden cuatro para el filtro. En resumen, el filtrado nos proporcionará contexto al relacionar temporalmente las predicciones de una acción. Además, todo esto ocurre fotograma a fotograma sin necesidad de analizar un fragmento de vídeo completo. En el apéndice A.2 encontramos una representación esquemática de los expuesto.

A modo de ejemplificación en la Figura 4.18 y 4.19 podemos ver el FPB y el algoritmo en acción. En la primera se muestra el flujo de confianza de la acción 2 para 350 fotogramas (unos 23 segundos). En la segunda podemos ver el flujo de confianza para el resto de acciones en el mismo periodo de tiempo. En negro se muestra el flujo de confianza a la salida de la red. Por cada fotograma tendremos un valor entre 0.0 y 1.0. En azul se muestra la señal a la salida del filtro paso bajo. Como podemos ver se suaviza la curva. En primer lugar se reduce notoriamente el pico inicial. Luego en el segundo bloque se eliminan grandes caídas momentáneas de la confianza. Finalmente se aplica el algoritmo obteniendo la señal en verde. Como vemos, el pico inicial es atenuado por el filtro y eliminado por el algoritmo. El bloque grande comienza y termina cuando la señal del filtro corta con la línea del *threshold*. Remarcar que durante los 350 fotogramas representados encontramos otros 16 flujos pero en todo momento su confianza después del FPB se encuentra por debajo del valor mínimo. En conclusión, podemos ver cómo a partir de una señal muy ruidosa e inestable obtenemos una salida con un error despreciable.

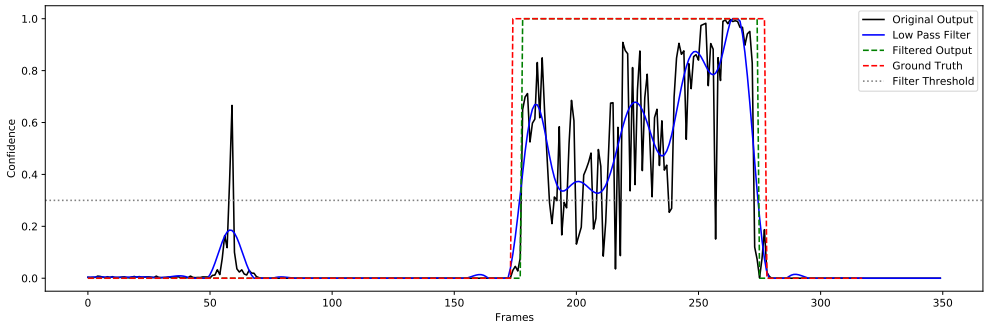


Figura 4.18: Salida de la señal de la acción 2 antes y después del filtro de post-procesado.

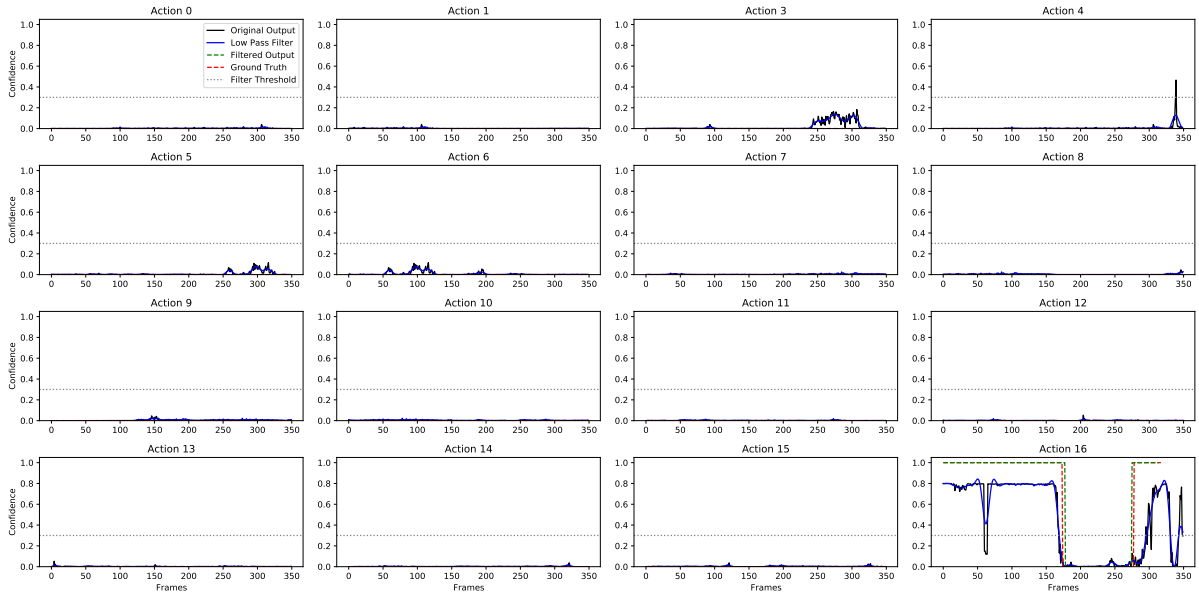


Figura 4.19: Salida de la señal de todas las acciones excepto la 2 antes y después del filtro de post-procesado.

4.7.2 Duración mínima de la acción

En la segunda fase del post-procesado, se realiza la exclusión de las acciones inferidas de corta duración. Se ha realizado un estudio de la duración de cada acción del set de entrenamiento del *dataset*. Como se puede ver en la Tabla 4.4, se ha extraído de cada acción contemplada la duración mínima, media y la desviación estándar. Por ejemplo, se ha observado que ciertas actividades, como hablar por teléfono o chatear, suelen tener una duración considerablemente mayor en comparación con otras, como recoger objetos del suelo. Es importante tener en cuenta que la duración de una acción está directamente relacionada con su naturaleza y características inherentes.

Por consiguiente, resultaría inapropiado aplicar un umbral de duración mínima idéntico a todas las clases de acciones. En cambio, se establece un umbral específico para cada clase, basado en su propia dinámica temporal. Este enfoque permite una mayor precisión en la clasificación y considera las particularidades de cada tipo de acción. Por ello se establecerá el umbral como el valor mínimo de la duración de cada acción.

Tabla 4.4: Tabla comparativa de las longitudes de cada acción en el conjunto de datos de prueba

ID	count	mean (s)	std (s)	min (s)
0	49	17	7.020802	1
1	55	8.127273	10.815035	1
2	52	15.153846	5.788374	2
3	46	14.891304	6.763393	3
4	50	14.18	8.923004	1
5	50	15.76	5.133165	7
6	50	17.94	10.52579	7
7	50	6.04	2.649297	2
8	51	12.098039	7.611189	1
9	47	5.574468	3.090973	2
10	49	7.040816	8.607069	1
11	45	18.533333	6.74065	7
12	51	17.431373	5.177856	5
13	56	9.622147	2	4
14	52	7.370475	2	12
15	51	18.686275	4.061971	6

4.8 Adaptación código a tiempo real

Todas las secciones de la arquitectura han sido tratadas e integradas para procesar un vídeo grabado. Primero se extraen las características de la secuencia completa para a continuación inferir las acciones, esto imposibilita su uso en tiempo real. Es por ello que se desarrollará un nuevo código que realiza el procedimiento por fotograma logrando un resultado con una baja latencia. Para ello se hará uso del *script* encargado de inferir las *bounding boxes* y la pose del conductor. Se crearán funciones encargadas entre otras de generar las distancias y ángulos, cargar el modelo del clasificador o controlar los datos de entrada. El código permitirá analizar un vídeo proveniente de una *webcam* en tiempo real. Además, se puede sustituir por un vídeo grabado o por un streaming de una manera totalmente trasparente para el programa. En la Figura 4.21 aparece un diagrama de flujo del código mencionado. Encontramos un resumen esquemático del código en el apéndice C. Es importante resaltar que existe la posibilidad de reemplazar las cámaras con fuentes de vídeo en determinados sistemas o aplicaciones. Aunque el funcionamiento es similar, hay una diferencia fundamental: la lectura de los fotogramas es secuencial. Por lo tanto, para lograr un rendimiento equiparable a la realidad, es necesario forzar la omisión de algunos fotogramas. Esta alternativa resulta útil en situaciones en las que no se disponga de cámaras físicas o cuando se desee simular una fuente de vídeo en lugar de utilizar una entrada en tiempo real. Además, el código permite habilitar una función que guarda de manera gráfica tanto las detecciones como los puntos clave inferidos para dotar al sistema de mayor aplicabilidad. En la Figura 4.20 se observa un fotograma de un fragmento de vídeo analizado.

Figura 4.20: Fotograma de la demo de la implementación del TFG en tiempo real

4.9 Cumplimiento de la normativa europea

Para finalizar el desarrollo de este proyecto se propone una implementación conceptual de nuestro sistema siguiendo la normativa europea adjunta en el Anexo D. Esta implementación no se llegó a realizar pero servirá como referencia para líneas futuras de trabajo.

En la Figura 4.22 se describe el diagrama de estados del sistema de aviso de distracciones. En función de la velocidad del vehículo y la autonomía de la conducción se establecen diferentes estados para el sistema. Al arrancar el vehículo el sistema realizará una auto-comprobación. Se testeará si todas las cámaras funcionan o si se detecta correctamente al conductor entre otros. En la normativa europea se establecen diferentes aspectos a comprobar. Si algo falla el sistema entrará en "Fallo permanente". Si por el contrario no se detecta ninguna anomalía se establece como "Apagado". El sistema estará encendido pero no se realiza ninguna detección. Será cuando el vehículo supere los 20km/h en conducción manual cuando pase a estado "Activo". En este estado comienza a realizar detecciones de las distracciones. Se establece una duración mínima para alertar al conductor en función de la velocidad del vehículo. Si este se encuentra entre 20 y 50 km/h, el umbral se establece a 6 segundos. Por el contrario si supera los 50km/h deberá detectarse la distracción durante 3,5 segundos. Además, si durante el estado activo se detecta un fallo, como la no detección del conductor o estimación de la pose del conductor nula, se pasa a "Fallo temporal" hasta que esta situación finalice. Se volverá a estado "Apagado" si se reduce la velocidad por debajo de 20 km/h o se pasa a conducción autónoma. Si durante este tipo de conducción se desea realizar la transición a manual, el sistema pasará a estado "Monitorizar". En este estado se informa de manera continua del estado de la atención del conductor al sistema encargado de realizar la transición. Después de completar la transición, el estado cambiará a

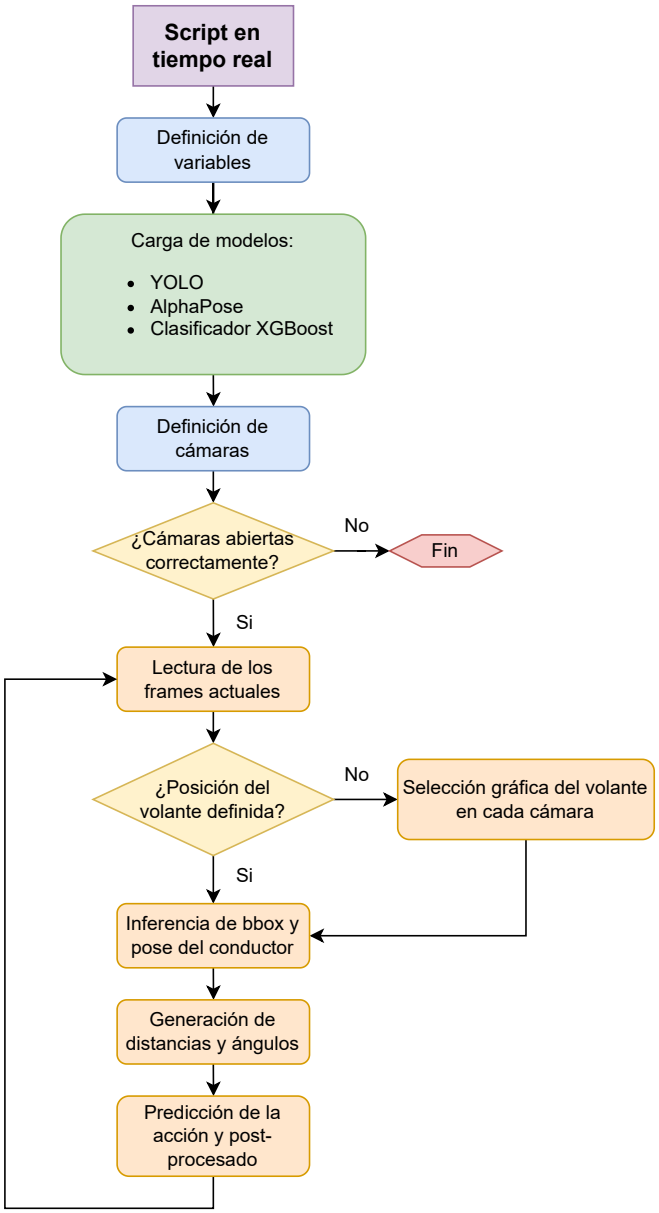


Figura 4.21: Diagrama de flujo de la implementación del proyecto en tiempo real.

"Apagado". Dependiendo de la velocidad actual, se cambiará a "Activo" o se mantendrá en "Apagado".

Además, como se observa en la Figura 4.23, se propone un esquema del sistema tri-cámara infrarrojo con alertas sonoras y luminosas a través del HMI (interfaz hombre-máquina). Se observan tres cámaras situadas en la parte frontal, semi-lateral y lateral del vehículo, para maximizar la información sobre el conductor. Se incluyen dos altavoces para las alertas sonoras y finalmente un sistema HMI para las alertas visuales e información sobre fallos y distracciones.

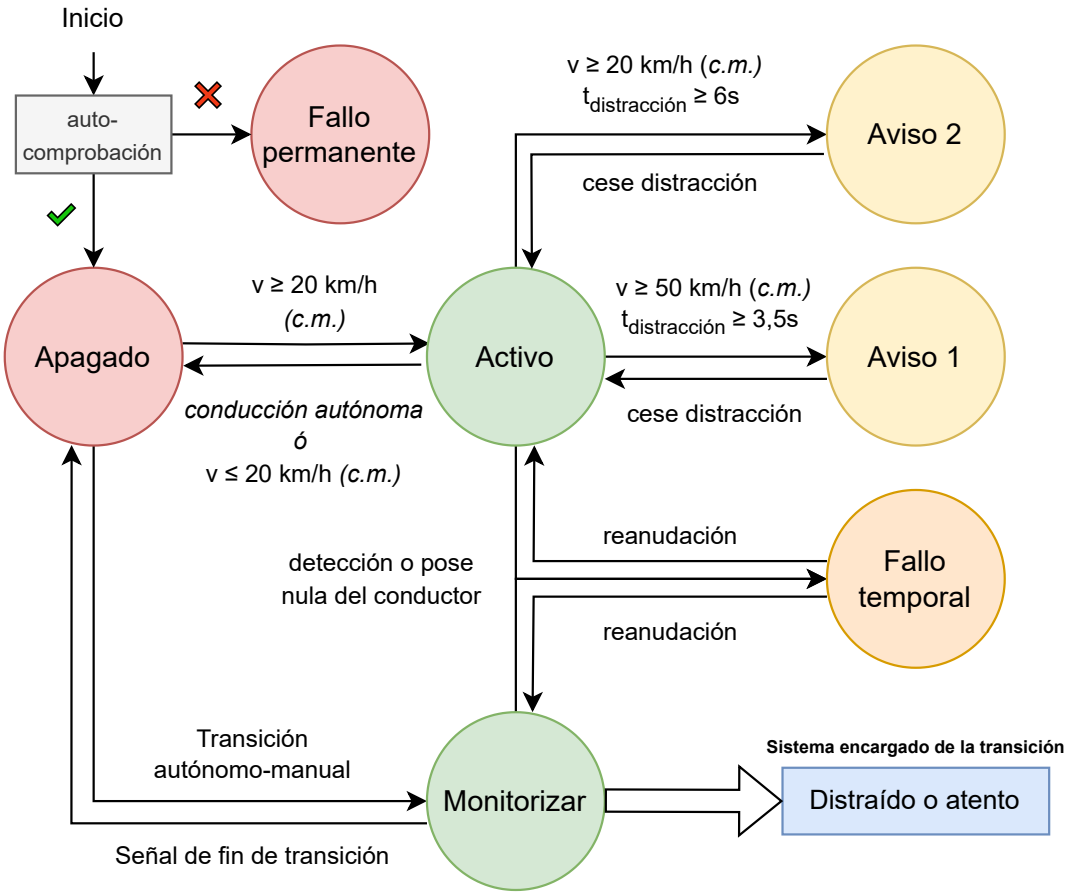


Figura 4.22: Diagrama de estados del sistema ADDW propuesto
c.m. = conducción manual

Figura 4.23: Esquema de la implementación del sistema ADDW propuesto

Capítulo 5

Resultados

Los resultados son la evidencia del esfuerzo invertido

Albert Einstein

Tras haber explicado el diseño e implementación del sistema se procederá a evaluarlo y obtener su rendimiento en dos *datasets* públicos. Se evaluarán tanto los resultados como el tiempo de inferencia de la red. Para una aplicación en tiempo real es casi tan importante sus resultados como su latencia, por ello encontrar un compromiso entre los dos es fundamental. Primero se enuncian las métricas utilizadas para evaluar nuestro sistema en los diferentes *datasets*. Seguidamente se evalúa el rendimiento en el *Dataset I* (3.1.4), comparándolo con el *baseline* para poder observar las mejoras aportadas. Además, se valida en el *Dataset II* (3.1.5) para poder comparar el proyecto con el estado del arte. Finalmente, se realiza un estudio de ablación para observar la aportación de cada cámara en el sistema y un estudio comparativo conservando el post-procesado original para medir la aportación de la fusión de una manera exclusiva.

5.1 Métricas

En la edición 2023 del *AI City Challenge Track 3* [39], incluido como *workshop* en el CVPR 2023, se ha propuesto como métrica principal la puntuación media de solapamiento de actividades. Dada una verdad de base con segundo de inicio (gs) y segundo final (ge), se elige la predicción que mejor se ajuste a una ventana temporal de ±10s. Siendo g la verdad (*ground truth*) y p la predicción, se define la siguiente fórmula:

$$os(p,g) = \frac{max(min(ge,pe) - max(gs,ps), 0)}{max(ge,pe) - min(gs,ps)} \tag{5.1}$$

La puntuación final es la media del solapamiento entre todas las actividades emparejadas y no emparejadas. Esta métrica nos da una idea de la precisión de las predicciones y penaliza los falsos negativos sin tener en cuenta los falsos positivos. Además, también hemos considerado utilizar la puntuación F1 (eq. 5.2), ya que fue la utilizada en la edición del 2022. Es la media

armónica de la precisión y el *recall*, y oscila entre 0 y 1, indicando los valores más altos un mejor rendimiento. Anteriormente se han introducido estos dos conceptos para la detección de objetos. En este apartado se evaluará fotograma por fotograma. Se considerará como correcta si la predicción de acción coincide con la verdad. De esta forma se construyen las formulas de precisión y *recall*, que juntas forman:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2tp}{2tp + fp + fn} \tag{5.2}$$

5.2 Resultados globales

Se muestra la mejora de nuestra red por clase respecto a la propuesta de KNDAR (nuestra línea base). Podemos ver en la Tabla 5.1 y Figura 5.1 una clara mejora en todas las acciones del *Dataset I* (3.1.4). Sobre todo, las clases más beneficiadas son las estáticas, es decir, aquellas que no necesitan fotogramas previos para determinar la acción. Éstas alcanzan un rendimiento en el estado del arte superior al 60 % en muchas de ellas. Se observa un peor rendimiento en las clases que sí necesitan contexto. Es el caso de cantar o hablar con los pasajeros. Se trata de acciones que incluso para un humano serían prácticamente imposibles de inferir observando un solo fotograma. Además, son difíciles de generalizar, ya que son características de cada persona. Todas las clases se han beneficiado directa o indirectamente de la fusión de distancia entre manos y volante, mejorando el rendimiento global. Esta característica ayuda a localizar al conductor en el espacio. Si sólo se utilizan los puntos clave del conductor no podemos saber dónde se encuentra exactamente dentro del vehículo. Por un lado, muchas clases utilizan estas características para inferir la acción, especialmente aquellas en las que la posición de las mano en el volante está íntimamente relacionada con ella. Por otro lado, ayuda a reducir los falsos positivos o a descartar algunas acciones.

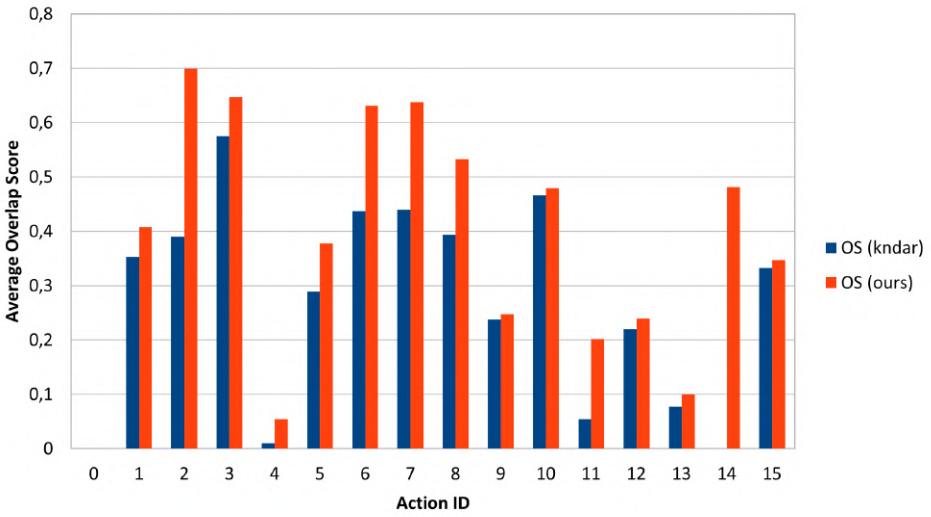


Figura 5.1: Histograma comparativo de la puntuación OS en las diferentes acciones entre la línea de base y nuestro método.

Tabla 5.1: Puntuación OS para acciones relevantes entre la línea de base y nuestra propuesta. Las pruebas se realizaron dividiendo el conjunto de datos de entrenamiento en entrenamiento y prueba, ya que no se dispone de las etiquetas del conjunto de datos de prueba.

ID	Action	KNDAR	Propuesta	Mejora (%)
0	Normal Driving	0	0	-
1	Drinking	0.353	0.408	15.60
2	Phone Call (Right)	0.390	0.699	79.23
3	Phone Call (Left)	0.575	0.647	12.52
4	Eating	0.009	0.054	500.00
5	Text (Right)	0.289	0.377	30.42
6	Text (Left)	0.437	0.631	44.43
7	Reaching behind	0.439	0.638	45.20
8	Adjust control panel	0.394	0.533	35.24
9	Pick up from floor (Driver)	0.237	0.247	4.22
10	Pick up from floor (Passenger)	0.466	0.479	2.79
11	Talk to passenger at the right	0.054	0.201	272.22
12	Talk to passenger at backseat	0.219	0.239	9.13
13	Yawning	0.077	0.100	29.87
14	Hand on head	0	0.481	-
15	Singing or dancing with music	0.333	0.347	4.20
	Mean	**0.267**	**0.380**	**42.34**

El rendimiento global de la red en el *Dataset I*, siguiendo la métrica oficial, para la clasificación pública fue de **0.3605**, alcanzando la 18ª posición de 27 participantes (25 de marzo de 2023). Como hemos mencionado antes, nuestro trabajo no se centró en mejorar todas las clases, sino sólo algunas de ellas. Si tomamos aquellas que se ven directamente beneficiadas por la fusión de objetos, conseguimos un **0.5515**, situándonos entre los 10 primeros (puesto 8) de la tabla de clasificación pública. También nuestra línea de base del año pasado (2022) logró un **0.2558** de *F1 score* [44] mientras que el nuestro alcanza un **0.3766**. En la Tabla 5.2 podemos observar la puntuación de los 15 mejores del año 2022 y 2023. Remarcar que el año anterior su utilizó *F1 score* como métrica y este año se ha usado *OS Score*.

En la Tabla 5.3, podemos ver una comparativa del rendimiento y tiempo de inferencia de los métodos del estado del arte evaluados en el *Dataset II* (3.1.5) y extraídos de [13]. Nuestro método se sitúa entre las mejores propuestas del estado del arte al conseguir un bajo tiempo de inferencia y un alto rendimiento. Además, en comparación con otros métodos, nuestra red es mucho más adaptable a otros tipos de datos, como imágenes de profundidad o infrarrojas. La adaptación al dominio de nuestra red es mayor, ya que el estimador de pose y el detector de objetos se entrenan con una variedad de personas mucho más amplia que el conjunto de datos mencionado. Además, se pueden añadir o eliminar cámaras de forma sencilla y añadir nuevos objetos para la detección de nuevas clases.

Hemos realizado nuestros experimentos en un ordenador con Ubuntu Linux versión 20.04 y equipado con CPU Intel(R) Core(TM) i7-9700K de 8 núcleos a 3,60GHz y GPU NVIDIA RTX 2080Ti. El sistema con las tres cámaras funciona a 15-17 Hz, incluyendo la detección de objetos, puntos clave y la posterior inferencia de acciones y post-procesamiento.

Tabla 5.2: Resultado de los 15 primeros en el *AI City Challenge Track3* en el año 2022 y 2023.

	2022		**2023**	
Rank	**Team Name**	**F1 score**	**Team Name**	**OS Score**
1	VTCC-UTVM	0.3492	Meituan-IoTCV	0.7416
2	Stargazer	0.3295	JNU_boat	0.7041
3	CybercoreAI	0.3248	ctc-AI	0.6723
4	OPPilot	0.3154	RW	0.6245
5	SIS Lab	0.2921	Purdue Digital Twin Lab	0.5921
6	BUPT-MCPRL2	0.2905	BUPT-MCPRL	0.5907
7	Winter is Coming	0.2902	DiveDeeper	0.5881
8	HSNB	0.2849	INTELLI_LAB	0.5426
9	VCA	0.271	AILAB	0.5424
10	Tahakom	0.2706	AIMIZ	0.5409
11	**SCU Anastasiu**	**0.2558**	hybrid lab	0.5382
12	union	0.2301	Tahakom	0.5350
13	Starwar	0.216	HST	0.5054
14	Aespa wInter	0.144	VAS_VTS	0.5026
15	SEEE-HUST	0.1348	AAA_Stars	0.4904
			propuesta (estáticos) (8)	**0.5515**
	propuesta (1)	**0.3766**	**propuesta (18)**	**0.3605**

Tabla 5.3: Diferentes métodos de clasificación de distracciones validados en el *Dataset* II. AA: Average Accuracy; AF: Average F1 Score; IT: Inference Time;

Model	**AA**	**AF**	**IT(ms)**
AlexNet	0.738	0.741	**2.61**
GWE-Resnet50	0.8169	NA	NA
VGG-19	0.833	0.835	**20.46**
HRNN	0.8485	NA	71
ResNet50	0.877	0.882	**14.26**
InceptionV3	0.884	0.890	22.85
InceptionV3-RNN	0.884	0.899	23.42
ResNet152	0.8852	NA	62
Densenet-201	0.890	0.895	46.05
InceptionV3-LSTM	0.8982	NA	23.24
GWE-InceptionV3	0.9006	NA	NA
InceptionV3-GRU	0.903	0.909	23.18
InceptionV3-BiLSTM	**0.917**	**0.931**	23.30
InceptionV3-BiGRU	**0.917**	**0.922**	23.24
ResNet+HRNN+Inception	**0.9236**	NA	114
ResNet101 + ResNet50	**0.9428**	**0.9427**	668.20
Nuestra propuesta	**0.9075**	**0.9025**	**29.60**

Como podemos ver en la Tabla 5.4, hemos evaluado el método con mejores resultados y un tiempo de inferencia similar al nuestro en otros conjuntos de datos públicos. Entrenamos ambos modelos en D2.1 y los evaluamos en D2.2 y D1. Se observa que nuestra arquitectura generaliza mejor al ser independiente del tipo de persona y coche donde se encuentre, obteniendo mejores

resultados en ambos *datasets*. La red está menos influenciada por la perspectiva y la posición en la que aparece el conductor. Además, es capaz de aislar y excluir a los pasajeros de la detección. Por último, la otra red solo entrena con las personas que aparecen en el *dataset*, mientras que nuestra propuesta se basa en el estimador de la pose que es entrenado en COCO, el cual contiene gran número y variedad de personas. Esto sirve como prueba fehaciente de la eficacia de nuestro enfoque.

Tabla 5.4: Comparación de la generalización de modelos

Model	**AA - D2.2**	**AA - D1**
InceptionV3	0.2198	0.2835
propuesta	**0.3701**	**0.5138**

5.3 Estudio de ablación

Una vez obtenido el rendimiento global de la arquitectura propuesta, se realiza un estudio de ablación encargado de estudiar la contribución concreta de dos aspectos del sistema. Por un lado estudiaremos la aportación del número de cámaras y por otro lado se aislará la mejora incorporada por la fusión de puntos clave y detección de objetos.

5.3.1 Cámaras

Nuestra arquitectura admite un número variable de cámaras. Cada una de ellas proporciona información valiosa para cada acción. Por ello, hemos realizado un estudio de ablación eliminando cada cámara para cuantificar la cantidad de información que proporciona cada una. La Tabla 5.5 muestra la puntuación OS con cada pareja de cámaras evaluada en el *Dataset I*. Como podemos observar, la que más información aporta es la cámara del salpicadero, seguida de la cámara trasera y por último la cámara derecha.

Tabla 5.5: OS score para el *set* de *testing* del Dataset I.

Dashboard	**Rear**	**Right**	**Score**
✓	✓		0.317
✓		✓	0.305
	✓	✓	0.293
✓	✓	✓	0.3605

5.3.2 Fusión

En esta sección se detalla empíricamente cómo mejoran las acciones relacionadas con la fusión de puntos clave y la detección de objetos. Como podemos ver en la Tabla 5.6, hay un incremento general para las cinco acciones. Esta tabla compara la mejora con las nuevas características mientras se utiliza el filtro original. Pero podemos ver que esta mejora es mucho mayor para

las acciones en las que se detecta mejor el objeto. Para “Llamada telefónica”, como el teléfono está a la altura de la cabeza del conductor, no hay ninguna dificultad para detectarlo, ya que no está oculto. Para “Texto”, como el teléfono está más bajo, queda oculto para algunas cámaras y puede estar oculto por otro pasajero o un objeto del vehículo. Por lo tanto, la inferencia de estas acciones está íntimamente ligada a la detección de objetos. Como conclusión vemos que la mejora es sustancial, pero un correcto posicionamiento de la cámara o pesos YOLO especializados llevarán a una mejor detección del objeto y por tanto a una mejor inferencia de la acción.

Tabla 5.6: Comparación del rendimiento de las acciones que intervienen en la fusión de la detección de objetos y puntos clave con el filtro original

ID	**Acción**	**KNDAR**	**Propuesta**	**Mejora (%)**
1	Drinking	0.353	0.383	8.49
2	Phone Call (Right)	0.390	0.631	61.79
3	Phone Call (Left)	0.575	0.610	5.65
5	Text (Right)	0.289	0.32	10.73
6	Text (Left)	0.437	0.59	34.97

Muchas otras clases se han beneficiado directa o indirectamente de la característica de distancia entre manos y volante, mejorando el rendimiento general. Esta característica ayuda a localizar al conductor en el espacio. Si sólo se utilizan los puntos clave del conductor, no podemos saber dónde se encuentra exactamente dentro del vehículo. Por un lado, muchas clases utilizarán estas características para inferir la acción, especialmente aquellas en las que la posición de la mano en el volante está íntimamente relacionada con ella. Por otro lado, ayudará a reducir los falsos positivos o a descartar algunas acciones.

Capítulo 6

Conclusiones y líneas futuras

Celebra tus logros, pero nunca olvides que el siguiente desafío te está esperando

Nelson Mandela

Después de haber desarrollado la arquitectura y obtenido resultados, se extraen conclusiones de las hipótesis e ideas de mejora. Se destacarán las mejoras logradas y se plantearán líneas futuras de trabajo para mejorar el rendimiento del sistema.

6.1 Conclusiones

Se presenta una arquitectura modular y ligera para el reconocimiento de distracciones del conductor mediante la fusión de la detección de objetos y los puntos clave del conductor basado en el método KNDAR. Nuestro sistema consta de tres etapas; en primer lugar, inferimos las personas que se encuentran en el interior del vehículo (incluido el conductor) y otros objetos de interés. En segundo lugar, inferimos los puntos clave del conductor. En tercer lugar, extraemos características de todos los datos y hacemos una predicción de la acción mediante un método de clasificación basado en *Machine Learning*. Finalmente, procedemos a aplicar filtros con el objetivo de proporcionar un contexto temporal y eliminar aquellas acciones cuya duración es más breve de lo habitual. Entrenamos y validamos el proyecto en el conjunto de datos Track 3 del AI City Challenge 2023 y Cairo Distracted Driver Dataset.

Las principales aportaciones respecto al *baseline* (KNDAR) son las siguientes:

- La fusión apenas aumenta el coste computacional al reutilizar el detector para la generación de puntos clave. Se añaden unas pocas características a las ya existentes, pero tienen un gran impacto en el rendimiento, mejorándolo significativamente. Las acciones relacionadas con el teléfono móvil son las más beneficiadas, duplicando en algunos casos el rendimiento. El volante añade contexto espacial a las predicciones y mejora en la mayoría de acciones. Además, por la forma como se ha construido el sistema añadir nuevos objetos solamente implica modificar los pesos del detector y añadir las nuevas distancias y/o ángulos. Esto

será totalmente transparente tanto para el clasificador, el cual habrá que entrenar con las nuevas *features*, como para el estimador de la pose.

- El filtro propuesto se aplica en tiempo real y proporciona contexto temporal a la predicción, ya que tiene en cuenta la confianza de fotogramas anteriores. Esto ayuda a paliar el principal defecto del *baseline*. Usar las confianzas de todas las acciones añade todavía más contexto que solo tomar la más probable. El filtro es sencillo y eficaz, se puede implementar en hardware y se adapta fácilmente a cualquier ratio de fotogramas, modificando la frecuencia de corte.

- Nuestro enfoque se adapta mucho mejor a otros entornos y tipos de datos. Tanto el detector como el estimador de la pose están entrenados en bases de datos muy amplias y muy variadas. Además encontramos pesos destinados a la inferencia en otro tipo de imágenes como profundidad o infrarrojas. Esto nos permite adaptarnos a este formato sin tener que entrenar el clasificador o modificar la red. Simplemente cambiando los pesos al iniciar la arquitectura será suficiente para obtener un rendimiento similar.

- La arquitectura tiene un buen rendimiento en tiempo real y una baja latencia de predicción. Las modificaciones apenas añaden tiempo de inferencia, manteniendo un tiempo suficiente para la aplicación. En el interior de los vehículos los cambios son más lentos y más controlados, es por ello que el rendimiento obtenido es suficiente para esta aplicación.

6.2 Líneas futuras

Nuestra propuesta obtiene una gran mejora en ciertas acciones, logrando rendimientos competitivos en el estado del arte. Pero todavía hay otras donde su rendimiento es pobre a causa de su gran dependencia con el contexto. Son acciones que se entienden con su evolución y no con un fotograma aislado. Como mejora futura se planea:

- Añadir seguimiento a los puntos clave y dar contexto a cada inferencia. De esta manera al realizar el *tracking* de los puntos clave se podrán entender acciones complejas como bailar, hablar con el pasajero, entre otras.

- Añadir puntos clave en la boca ayudaría también en las acciones como cantar, hablar o bostezar. Actualmente las redes entrenadas añaden puntos muy detallados en la cara que aumentan mucho el coste total y solo benefician un par de acciones concretas. Por ello habrá que encontrar una forma de solo implementar ciertos puntos estratégicos que conservando un tiempo de inferencia adecuado mejoren las acciones mencionadas.

- Una mejor detección de objetos podría mejorar las acciones relacionadas con ellos. La implementación de más variedad de los mismos, como más tipos de botellas de agua, cantimploras, latas de bebida, etc, podría mejorar el rendimiento del sistema.

- Acelerar las redes para obtener mayor velocidad y obtener un menor uso de recursos tanto de potencia de cómputo como energéticos.

- Implementar y probar el TFG en CARLA, un simulador de conducción autónoma muy extendido que permite recrear situaciones de conducción de una manera muy flexible.

Con estas ideas de mejora se podrá lograr un sistema con un rendimiento equilibrado en todas las acciones y manteniendo un tiempo de inferencia adecuado. Además, como se ha visto en la normativa europea, como trabajo futuro se añadiría el estudio de la mirada para tener un sistema más completo aún y preciso.

Bibliografía

[1] *La Moncloa. 07/01/2022. Los accidentes de tráfico se cobraron la vida de 1.004 personas el pasado año [Prensa/Actualidad/Interior]*, (Accessed on 04/01/2023).

[2] *Preliminary 2021 EU Road Safety Statistics*, (Accessed on 04/04/2023).

[3] I. of Advanced Motorists, http://www.iam.org.uk/, (Accessed on 05/17/2023).

[4] *Road safety – advanced driver distraction warning systems*, https://ec.europa.eu/info/law/better-regulation/have-your-say/initiatives/13740-Road-safety-advanced-driver-distraction-warning-systems_en, (Accessed on 05/18/2023).

[5] Society of Automotive Engineers, *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, Standard J3016, 2016. dirección: https://www.sae.org/standards/content/j3016_201609/.

[6] A. Vats y D. C. Anastasiu, "Key Point-Based Driver Activity Recognition", en *2022 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, ép. CVPRW'22, vol. 1, New Orleans, LA, USA, jul. de 2022.

[7] M. T. Tran, M. Q. Vu, N. D. Hoang y K.-H. N. Bui, "An effective temporal localization method with multi-view 3D action recognition for untrimmed naturalistic driving videos", en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, págs. 3168-3173.

[8] M. Alyahya, S. Alghannam y T. Alhussan, "Temporal Driver Action Localization using Action Classification Methods", en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, págs. 3319-3326.

[9] H. Fan, Y. Li, B. Xiong, W.-Y. Lo y C. Feichtenhofer, *PySlowFast*, https://github.com/facebookresearch/slowfast, 2020.

[10] M. Contributors, *OpenMMLab's Next Generation Video Understanding Toolbox and Benchmark*, https://github.com/open-mmlab/mmaction2, 2020.

[11] C. Feichtenhofer, *X3D: Expanding Architectures for Efficient Video Recognition*, 2020. arXiv: 2004.04730 [cs.CV].

[12] Y. Li, C.-Y. Wu, H. Fan et al., *MViTv2: Improved Multiscale Vision Transformers for Classification and Detection*, 2022. arXiv: 2112.01526 [cs.CV].

[13] H. V. Koay, J. H. Chuah, C.-O. Chow, Y.-L. Chang y B. Rudrusamy, "Optimally-Weighted Image-Pose Approach (OWIPA) for Distracted Driver Detection and Classification", *Sensors*, vol. 21, n.º 14, 2021, ISSN: 1424-8220. DOI: 10.3390/s21144837. dirección: https://www.mdpi.com/1424-8220/21/14/4837.

[14] A. Krizhevsky, I. Sutskever y G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", en *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou y K. Weinberger, eds., vol. 25, Curran Associates, Inc., 2012. dirección: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[15] K. Simonyan y A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[16] K. He, X. Zhang, S. Ren y J. Sun, *Deep Residual Learning for Image Recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens y Z. Wojna, *Rethinking the Inception Architecture for Computer Vision*, 2015. arXiv: 1512.00567 [cs.CV].

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li y L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", en *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, págs. 248-255.

[19] *ImageNet Benchmark (Image Classification) | Papers With Code*, https://paperswithcode.com/sota/image-classification-on-imagenet, (Accessed on 05/10/2023).

[20] M. Tan, R. Pang y Q. V. Le, "Efficientdet: Scalable and efficient object detection", en *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, págs. 10 781-10 790.

[21] T.-Y. Lin, M. Maire, S. Belongie et al., "Microsoft coco: Common objects in context", en *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, págs. 740-755.

[22] M. Tan y Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks", *CoRR*, vol. abs/1905.11946, 2019. arXiv: 1905.11946. dirección: http://arxiv.org/abs/1905.11946.

[23] R. Girshick, J. Donahue, T. Darrell y J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014. arXiv: 1311.2524 [cs.CV].

[24] R. Girshick, *Fast R-CNN*, 2015. arXiv: 1504.08083 [cs.CV].

[25] S. Ren, K. He, R. Girshick y J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, 2016. arXiv: 1506.01497 [cs.CV].

[26] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov y S. Zagoruyko, "End-to-end object detection with transformers", en *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, Springer, 2020, págs. 213-229.

[27] A. Vaswani, N. Shazeer, N. Parmar et al., *Attention Is All You Need*, 2017. arXiv: 1706.03762 [cs.CL].

[28] G. Jocher, A. Chaurasia y J. Qiu, *YOLO by Ultralytics*, ver. 8.0.0, ene. de 2023. dirección: https://github.com/ultralytics/ultralytics.

[29] J. Redmon y A. Farhadi, "YOLOv3: An Incremental Improvement", *CoRR*, vol. abs/1804.02767, 2018. arXiv: 1804.02767. dirección: http://arxiv.org/abs/1804.02767.

[30] G. Jocher, *YOLOv5 by Ultralytics*, ver. 7.0, mayo de 2020. DOI: 10.5281/zenodo.3908559. dirección: https://github.com/ultralytics/yolov5.

[31] C. Li, L. Li, H. Jiang et al., *YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications*, 2022. arXiv: 2209.02976 [cs.CV].

[32] C.-Y. Wang, A. Bochkovskiy y H.-Y. M. Liao, *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, 2022. arXiv: 2207.02696 [cs.CV].

[33] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei e Y. A. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[34] R. A. Güler, N. Neverova e I. Kokkinos, *DensePose: Dense Human Pose Estimation In The Wild*, 2018. arXiv: 1802.00434 [cs.CV].

[35] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár y K. He, *Detectron*, https://github.com/facebookresearch/detectron, 2018.

[36] *MoveNet: Ultra fast and accurate pose detection model. TensorFlow Hub*, https://www.tensorflow.org/hub/tutorials/movenet, (Accessed on 06/06/2023).

[37] S. D. D. Cruz, O. Wasenmuller, H.-P. Beise, T. Stifter y D. Stricker, "Sviro: Synthetic vehicle interior rear seat occupancy dataset and benchmark", en *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, págs. 973-982.

[38] J. S. Katrolia, B. Mirbach, A. El-Sherif, H. Feld, J. Rambach y D. Stricker, "Ticam: A time-of-flight in-car cabin monitoring dataset", *arXiv preprint arXiv:2103.11719*, 2021.

[39] M. Naphade, S. Wang, D. C. Anastasiu et al., *The 7th AI City Challenge*, 2023. arXiv: 2304.07500 [cs.CV].

[40] *Docker: Accelerated, Containerized Application Development*, https://www.docker.com/, (Accessed on 05/11/2023).

[41] *Docker Hub Container Image Library | App Containerization*, https://hub.docker.com/, (Accessed on 05/11/2023).

[42] W. McNally, K. Vats, A. Wong y J. McPhee, "Rethinking Keypoint Representations: Modeling Keypoints and Poses as Objects for Multi-Person Human Pose Estimation", *arXiv preprint arXiv:2111.08557*, 2021.

[43] M. Tan, R. Pang y Q. V. Le, "EfficientDet: Scalable and Efficient Object Detection", *CoRR*, vol. abs/1911.09070, 2019. arXiv: 1911.09070. dirección: http://arxiv.org/abs/1911.09070.

[44] M. Naphade, S. Wang, D. C. Anastasiu et al., "The 6th AI City Challenge", en *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, IEEE Computer Society, jun. de 2022, págs. 3346-3355. DOI: 10.1109/CVPRW56347.2022.00378.

Apéndice A

Algoritmos

A.1 Non Max Suppression

Data: D = lista de detecciones ordenadas por score descendente
Result: OUT = detecciones filtradas
begin
    OUT = lista vacía;
    for *detección in D* **do**
        Añadir detección a la lista OUT;
        Borrar detección de la lista D;
        for *elemento in D* **do**
            Calcular el IoU de detección con elemento;
            if $IoU > threshold$ **then**
                Borrar elemento de D;

Algoritmo A.1: Algoritmo Non Max Suppression

A.2 Post-procesado de las confianzas de las predicciones de distracción

Entrada: Flujo de probabilidades de confianza para 16 distracciones y "no distracción"
Salida : Predicción de acción con contexto temporal
foreach *distracción* **do**
  Aplicar filtro de paso bajo a la probabilidad de confianza de cada distracción;
end
Seleccionar la probabilidad más alta entre las 16 distracciones;
if *la probabilidad seleccionada supera el umbral mínimo* **then**
  Establecer la acción correspondiente a la distracción con la probabilidad más alta;
end
else
  Establecer la acción como "no distracción";
end

Algoritmo A.2: Filtro de contexto para predicciones de acción

Apéndice B

Características extraidas de los puntos clave del conductor

FID	Característica
	Ángulos
0	nariz, esquina inferior izquierda, esquina superior izquierda de la imagen
1	muñeca izquierda, esquina inferior izquierda, esquina superior izquierda de la imagen
2	muñeca derecha, esquina inferior izquierda, esquina superior izquierda de la imagen
3	codo izquierdo (hombro izquierdo, codo izquierdo, muñeca izquierda)
4	hombro izquierdo (codo izquierdo, hombro izquierdo, cadera izquierda)
5	codo derecho (hombro derecho, codo derecho, muñeca derecha)
6	hombro derecho (codo derecho, hombro derecho, cadera derecha)
7	ojo izquierdo, nariz, ojo derecho
8	nariz, hombro izquierdo, hombro derecho
9	nariz, hombro derecho, hombro izquierdo
10	nariz, oreja izquierda, ojo izquierdo
11	nariz, oreja derecha, ojo derecho
12	oreja izquierda, cadera derecha, hombro derecho
13	oreja derecha, cadera izquierda, hombro izquierdo
14	hombro izquierdo, cadera derecha, hombro derecho
15	hombro derecho, cadera izquierda, hombro izquierdo
	Distancias
16	distancia de la nariz a la esquina inferior izquierda
17	distancia de la muñeca izquierda a la esquina inferior izquierda
18	distancia de la muñeca derecha a la esquina inferior izquierda
19	distancia de la nariz a la esquina superior izquierda
20	distancia de la muñeca izquierda a la esquina superior izquierda
21	distancia de la muñeca derecha a la esquina superior izquierda
22	distancia de la nariz al hombro izquierdo

23	distancia de la nariz al hombro derecho
24	distancia de la nariz a la oreja izquierda
25	distancia de la nariz a la oreja derecha
26	distancia del ojo izquierdo al ojo derecho
27	distancia de la oreja izquierda a la oreja derecha
28	distancia de la oreja izquierda a la muñeca izquierda
29	distancia de la oreja derecha a la muñeca derecha
30	distancia del codo izquierdo a la cadera izquierda
31	distancia del codo derecho a la cadera derecha
32	distancia de la muñeca izquierda a la cadera izquierda
33	distancia de la muñeca derecha a la cadera derecha
	Posiciones
34	ancho relativo del cuadro delimitador
35	altura relativa del cuadro delimitador
36	posición relativa del centro del cuadro delimitador en x
37	posición relativa del centro del cuadro delimitador en y
	Desplazamientos de posición
38	nariz
39	ojo izquierdo
40	ojo derecho
41	oreja izquierda
42	oreja derecha
43	hombro izquierdo
44	hombro derecho
45	codo izquierdo
46	codo derecho
47	muñeca izquierda
48	muñeca derecha
	Desplazamientos de ángulo
49	nariz, esquina inferior izquierda, esquina superior izquierda de la imagen
50	muñeca izquierda, esquina inferior izquierda, esquina superior izquierda de la imagen
51	muñeca derecha, esquina inferior izquierda, esquina superior izquierda de la imagen
52	codo izquierdo (hombro izquierdo, codo izquierdo, muñeca izquierda)
53	hombro izquierdo (codo izquierdo, hombro izquierdo, cadera izquierda)
54	codo derecho (hombro derecho, codo derecho, muñeca derecha)
55	hombro derecho (codo derecho, hombro derecho, cadera derecha)

Apéndice C

Código resumido del sistema en tiempo real

```
#Variables
ejecucion_total = []
steeringwheel = None
last_bboxes = [None,None,None]
last_poses = [None,None,None]
last_i = [None,None,None]
frame = 0
preds_total = []

# Cargamos los modelos
estimator = AlphaposeYOLO()
with open('modelo.pkl','rb') as pickled:
    data = pickle.load(pickled)
    model = data['model']

#Dashboard
cam1 = cv2.VideoCapture(0)
#Rear
cam2 = cv2.VideoCapture(1)
#Right_side
cam3 = cv2.VideoCapture(2)

#Nos aseguramos que se han abierto las cámaras correctamente
if not cam1.isOpened() or not cam2.isOpened() or  not cam3.isOpened():
    #Finalizar programa

try:
    while(True):
        #Leemos un frame de cada camara
        ret1,frame1 = cam1.read()
        ret2,frame2 = cam2.read()
        ret3,frame3 = cam3.read()

        #Si hay algun fallo en alguna camara se termina el programa
```

```
35        if not ret1 or not ret2 or  not ret3:
36            #Terminar programa
37
38        #Batch de imagenes
39        img = [frame1,frame2,frame3]
40
41        #Para el primer frame leido se indica la posición del volante
42        if steeringwheel is None:
43            steeringwheel = []
44            steeringwheel.append(select_steering_wheel(frame1))
45            steeringwheel.append(select_steering_wheel(frame2))
46            steeringwheel.append(select_steering_wheel(frame3))
47            cv2.destroyWindow('image')
48
49        #height - alto de la imagen
50        h = int(cam1.get(cv2.CAP_PROP_FRAME_HEIGHT))
51        #width - ancho de la imagen
52        w = int(cam1.get(cv2.CAP_PROP_FRAME_WIDTH))
53        st = time.time()
54        #Realizar inferencia de un array de imágenes (en batch)
55        poses,bboxes,phone,bottle = estimator.boxesandposes(img)
56        #Obtenemos las distancias y angulos
57        features = get_features(poses,phone,bottle,steeringwheel)
58        #Obtenemos la prediccion a partir de las 'features' y aplicamos el
    post_procesado
59        prediction = get_prediction(features)
60        et = time.time()
61        #Añadimos la prediccion a la lista
62        prediction.append([pr1,int(frame/30)])
63        #Añadimos el tiempo de procesado
64        ejecucion_total.append(et - st)
65
66        print('* Tiempo de ejecucion:', '%.3f' % ((et - st)*1000), 'ms -','%.3f' %
    (1/(et - st)) ,'Hz')
67        print("Accion inferida: " + str(pr1) + ' --> '  + activity_type[pr1] + ' |
    Second: ' + str(int(frame/30)))
68
69
70    print("**Proceso finalizado**")
71    cam1.release()
72    cam2.release()
73    cam3.release()
74    with open('prediccion.txt', 'w') as f:
75        writer = csv.writer(f)
76        writer.writerows(prediction)
77    print("==> Se ha guardado las predicciones en 'prediccion.txt'.")
78    print("==> Se ha finalizado correctamente el analisis de",len(ejecucion_total),"
    imagenes.")
79    print("==> Tiempo medio de ejecucion: ", '%.3f' % ((array.mean())*1000), 'ms -',
    '%.3f' % (1/(array.mean())) ,'Hz')
```

Apéndice D

Requisitos técnicos específicos para un sistema ADDW

D.1 Control del sistema ADDW

1. El sistema ADDW se activará automáticamente por encima de la velocidad de 20 km/h, a menos que se especifique lo contrario en los requisitos establecidos en los puntos 1.2. a 1.6. El fabricante del vehículo podrá optar por establecer la activación automática del sistema ADDW a una velocidad inferior. Se permite un periodo acumulado de hasta 1 minuto de conducción a velocidades mayor o igual a 20 km/h para que el sistema comience a medir el estado del conductor y se calibre.

2. El conductor deberá poder desactivar manualmente el aviso ADDW o el sistema ADDW, en función de cuál de las dos posibilidades (o ambas) haya activado el fabricante del vehículo.

3. El sistema ADDW puede desactivarse automáticamente en las situaciones predefinidas por el fabricante, concretamente en situaciones:

   (a) cuando otro sistema asume toda la tarea de conducción dinámica de forma sostenida y cuenta con el apoyo de un sistema adecuado de supervisión del conductor.

   (b) cuando un sistema de vehículo accionado por el conductor, que ayuda a un conductor humano a controlar el movimiento longitudinal y lateral de forma sostenida, está activo y contiene un sistema adecuado de supervisión del conductor.

   El sistema ADDW se reactivará automáticamente en cuanto dejen de darse las condiciones que provocaron su desactivación automática.

   En este contexto, la tarea de conducción dinámica incluirá todas las funciones operativas en tiempo real y las funciones tácticas necesarias para manejar el vehículo, excluyendo las funciones estratégicas como la programación de viajes y la selección de destinos y puntos de paso, e incluyendo las siguientes subtareas:

   (a) control del movimiento lateral del vehículo mediante la dirección (operativo)

(b) control del movimiento longitudinal del vehículo mediante aceleración y deceleración (operativo)

(c) vigilancia del entorno de conducción mediante la detección, el reconocimiento, la clasificación y la preparación de respuestas (operativas y tácticas) de objetos y eventos

(d) ejecución de respuestas a objetos y eventos (operativas y tácticas)

(e) planificación de maniobras (táctica)

(f) mejora de la visibilidad mediante la iluminación, el sonido de la bocina, la señalización o los gestos (tácticos)

4. El sistema ADDW no se desactivará automáticamente en las condiciones establecidas en el punto 5, pero los avisos de distracción del sistema ADDW podrán desactivarse automáticamente. La emisión de avisos de distracción deberá reactivarse automáticamente en cuanto dejen de darse las condiciones que motivaron su desactivación.

5. La emisión de avisos de distracción por parte del sistema ADDW puede desactivarse automáticamente en condiciones en las que otro sistema de asistencia a la conducción esté avisando de un peligro inminente o de una situación crítica, pero no es una condición para la desactivación automática del sistema ADDW. La emisión del aviso de distracción debe reactivarse automáticamente en cuanto dejen de darse las condiciones que provocaron su desactivación.

6. El sistema ADDW, incluidas las advertencias de interfaz hombre-máquina, se restablecerá automáticamente al modo de funcionamiento normal cada vez que se active el interruptor de control principal del vehículo. El fabricante del vehículo podrá introducir y añadir otras condiciones de restablecimiento automático.

D.2 Condiciones medioambientales

1. El sistema ADDW funcionará eficazmente tanto de día como de noche.

D.3 Control de las distracciones del conductor

Nota: *En este apartado se determina el análisis de las distracciones basándose en el seguimiento de la mirada. Esta sección se omite ya que se realiza una re-interpretación en este trabajo.*

Activadores de alerta y situaciones cubiertas por el sistema ADDW

1. Se enviará una advertencia al conductor en cuanto se den las dos condiciones siguientes:

   (a) velocidad del vehículo a 50 km/h o superior;

   (b) la distracción dure un tiempo máximo de [3,5] segundos en la situación nominal. Las situaciones no nominales establecidas en el punto 1.3 de la parte 3 podrán ampliar el límite de tiempo máximo de la situación nominal en 1,5 segundos adicionales.

Siempre que se prueben las condiciones enumeradas en la letra b), deberán aplicar un tiempo de amortiguación adicional para compensar las incertidumbres técnicas de medición.

2. Se enviará una advertencia al conductor en cuanto se verifiquen las dos condiciones siguientes:

   (a) velocidad del vehículo a 20 km/h o superior;

   (b) la distracción dure un tiempo máximo de 6 segundos en la situación nominal. Las situaciones no nominales establecidas en el punto 1.3 de la parte 3 pueden ampliar el límite de tiempo de la situación nominal máxima en 1,5 segundos adicionales.

   Siempre que se prueben las condiciones enumeradas en la letra b), deberán aplicar un tiempo de amortiguación adicional para compensar las incertidumbres técnicas de medición.

3. Se contabilizará el tiempo durante el cual el conductor está distraído y cuando el sistema ADDW esté activado. Este tiempo se medirá independientemente de la velocidad del vehículo.

4. La tolerancia de tiempo permitida para los acontecimientos descritos será definida por el fabricante del vehículo, con una tolerancia de tiempo mínima de 50 milisegundos (sacadas oculares).

5. El fabricante del vehículo podrá optar por establecer un requisito de velocidad mínima inferior en las situaciones mencionadas en los puntos 3.3.2.1. y 3.3.2.2.

6. El fabricante del vehículo puede aplicar estrategias de alerta adicionales, basadas en datos adicionales que ayuden al sistema a comprender el comportamiento del conductor, su distracción cognitiva o su entorno inmediato dentro del vehículo.

D.4 Requisitos de la interfaz hombre-máquina

Siempre que se prueben las condiciones enumeradas en la letra b), deberán aplicar un tiempo de amortiguación adicional para compensar las incertidumbres técnicas de medición. Siempre que se prueben las condiciones enumeradas en la letra b), deberán aplicar un tiempo de amortiguación adicional para compensar las incertidumbres técnicas de medición.

D.4.1 Advertencia visual

1. La advertencia visual deberá estar situada de forma que sea fácilmente visible y reconocible por el conductor tanto de día como de noche y distinguible de otras descripciones.

2. La advertencia visual consistirá en una indicación fija o intermitente (por ejemplo, un testigo, un mensaje emergente, etc.).

D.4.2 Aviso acústico

1. El aviso acústico deberá ser fácilmente reconocible por el conductor.

2. La mayor parte de la advertencia acústica deberá estar comprendida en el espectro de frecuencias de 200- 8000 Hz y en la gama de amplitud de 50-90 dB. El fabricante del vehículo podrá ajustar la amplitud en función del nivel de ruido ambiental.

3. Si se utilizan alertas vocales, el vocabulario utilizado deberá ser coherente con cualquier texto utilizado como parte de la alerta visual.

4. La parte audible de la alerta durará al menos el tiempo que permita al conductor comprenderla.

D.4.3 Advertencia háptica

1. La advertencia háptica será perceptible para el conductor y se proporcionará directa o indirectamente a través de cualquier interfaz que se espere atraiga la atención del conductor de vuelta a la tarea de conducción.

D.5 Aviso de fallo del sistema ADDW

D.5.1 Fallos permanentes

1. Cuando se detecte un fallo permanente en el sistema ADDW, se proporcionará una señal visual constante de aviso de fallo.

2. No se habrá completado al menos una auto-comprobación inicial del ADDW antes de que el ADDW esté operativo. Posteriormente se indicará al conductor una señal de advertencia de fallo en caso de fallo detectable eléctricamente.

3. El sistema reconocerá un caso de oscurecimiento no temporal del sensor y emitirá la señal de advertencia de fallo establecida en el punto 3.5.1.1, que deberá visualizarse. Un caso de oscurecimiento del sensor se produce, como mínimo, cuando el sensor no mide ninguna luz al activarse el sistema ADDW.

4. Los fallos que activan la señal de advertencia, pero que no se detectan cuando se desactiva el ADDW, se retendrán al detectarse y seguirán mostrándose desde el arranque del vehículo después de cada activación del interruptor de control principal del vehículo, mientras persista el fallo o defecto.

D.5.2 Fallos temporales

1. Cuando se detecte una situación de fallo temporal no eléctrico, podrá mostrarse la señal de aviso de fallo establecida en el punto 3.5.1.

2. Se facilitará al conductor información sobre la limitación actual del sistema ADDW y/o las limitaciones típicas del sistema ADDW. Las limitaciones en cuestión son las que hacen que el sistema ADDW funcione temporalmente de forma inadecuada porque no se detectan suficientes rasgos faciales del conductor debido a un exceso de elementos relacionados con el conductor, el vehículo, el entorno u otros elementos que afectan al funcionamiento del sistema ADDW y que no pueden tratarse como una situación no nominal. El fabricante del vehículo puede utilizar un enfoque activo mediante una advertencia visual adicional y/o un enfoque pasivo mediante información escrita.

Apéndice E

Presupuesto

Se detallan a continuación los costes económicos derivados del proyecto expuesto. Se dividen en dos partes, por un lado los materiales y por otro las horas de trabajo dedicadas.

E.1 Recursos Hardware y Software

El principal elemento de trabajo es el ordenador. En la Tabla E.1 se desglosa sus componentes y su precio de mercado. El componente principal para este proyecto es la tarjeta gráfica ya que es la encargada de realizar los entrenamientos e inferencias. Estimamos una vida útil de 5 años para los componentes del ordenador. A pesar de que la duración puede ser mayor, para un uso intensivo y unas necesidades altas de cómputo, esa es la vida adecuada. El ordenador se ha usado durante 10 meses, lo que corresponde un 17 % (0,17) de la vida útil del mismo.

Tabla E.1: Componentes y precio del equipo de trabajo.

Tipo	**Modelo**	**Precio**	**Coef**	**Subtotal**
CPU	Intel Core i7-9700K CPU @ 3.60GHz	349,87€	0,17	59,48€
ROM	SSD Controller SM981/PM981 500 GB	78,00€	0,17	13,26€
RAM	2 * HyperX DDR4 2400 MHz 16GB	297,00€	0,17	50,49€
GPU	GeForce RTX 2080 Ti	521,00€	0,17	88,57€
Fuente	Corsair 650W 80 Plus Gold (TX650M)	71,00€	0,17	12,07€
Periféricos	Teclado, Ratón y Pantallas	120,00€	0,17	20,40€
Total		**1.436,87€**		**244,27€**

Los recursos software son gratuitos en su totalidad. Desde el sistema operativo, los conjuntos de datos y todas las tecnologías mencionadas en el trabajo.

E.2 Recursos Humanos

El trabajo ha sido desarrollado en su totalidad por Pablo Pardo Decimavilla con la tutoría de Luis M. Bergasa Pascual. En la Tabla E.2 encontramos el desglose del coste personal. Se aplica un salario de 15€ netos la hora con una jornada laboral de 4 horas diarias.

Tabla E.2: Desglose de precio y dedicación a las principales tareas del proyecto.

Tarea	**Días de trabajo**	**Coste**
Estudio del estado del arte y de herramientas software	40	2400,00€
Puesta en marcha de KNDAR	7	420,00€
Unión YOLOv8 con Alphapose	30	1800,00€
Mejora post-procesado	10	600,00€
Entrenamiento sistema completo	20	1200,00€
Resultados y experimentos	15	900,00€
Escritura de la memoria	45	2.700,00€
Total	**167**	**10.020,00€**

Al total del precio de trabajo hay que añadir el 21 %, lo que hace que el total ascienda a 12.124,20€. En la Tabla E.3 podemos observar el coste total del proyecto sumando los tres apartados mencionados y los impuestos aplicables.

Tabla E.3: Coste económico total del proyecto

Concepto	**Subtotal**
Costes de Hardware	244,27€
Costes de Software	0,00€
Costes de personal	10.020,00€
IVA (21 %)	2.104,20€
Precio Total	**12.368,47€**

El coste del proyecto asciende a **doce mil trescientos sesenta y ocho con cuarenta y siete euros**.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá