Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Procesamiento automático de imágenes y vídeos con técnicas de *Deep Learning*

Alumno: Paula Mielgo Martín

Tutores: Anibal Bregón Bregón
Jorge Silvestre Vilches

Fecha: 26 de julio de 2022

Procesamiento automático de imágenes y vídeos con técnicas de *Deep Learning*

Paula Mielgo Martín

26 de julio de 2022

Resumen

La visión computacional es uno de los campos de la Inteligencia Artificial más presentes en el día a día. Los modelos basados en *deep learning* son los que ofrecen mejores resultados en este ámbito. Sin embargo, muchos de los proyectos planteados en este área no pueden llevarse a cabo debido a la falta de personal cualificado que lo desarrolle. Esto afecta especialmente a las ciudades, que incrementan diariamente su población y, también, el volumen de datos recogidos en las calles.

En esta memoria se revisan las principales arquitecturas utilizadas actualmente para el procesamiento de imágenes y vídeos, y se aportan cuadernos didácticos para fomentar la importancia de la visión computacional en futuras generaciones. Además, se desarrolla una propuesta de segmentador semántico en el contexto de una ciudad inteligente.

Palabras claves: visión computacional, *deep learning*, ciudad inteligente, aprendizaje automático.

Abstract

Computer vision is one of the most present fields of Artificial Intelligence in everyday life. Deep learning based models give the best results in this sphere. However, many of the projects proposed in this area cannot be carried out due to the shortage of qualified personnel to develop them. This especially affects cities, which population increases daily and also the volume of data collected on the streets.

In this report the main architectures currently used for image and video processing are reviewed and didactic notebooks are provided to promote the importance of computer vision in future generations. In addition, a semantic segmentator proposal is developed in the context of a smart city.

Keywords: computer vision, deep learning, smart city, machine learning.

Índice general

Índice de figuras

Índice de tablas

Parte I

Descripción del proyecto

Capítulo 1

Introducción

Los continuos avances tecnológicos que se han introducido en diferentes ámbitos de la sociedad han sido determinantes en la evolución del mundo. Como ejemplo de ello pueden citarse las primeras revoluciones industriales, que supusieron un notable incremento de la productividad y una mejora de la calidad de vida. Continuando con ellas, arranca en torno a 2014, la Cuarta Revolución Industrial [21] [56] en la que la Inteligencia Artificial es una de las tecnologías claves.

Pero el industrial no es el único sector que se ha visto beneficiado por los nuevos avances tecnológicos. A finales del siglo XX surgen las denominadas *smart cities* o ciudades 4.0 [75] [17]. Este concepto nace para satisfacer las necesidades de los ciudadanos, pero también de la administración y de las empresas. La idea es que los gobiernos y ayuntamientos locales puedan utilizar datos relativos a los acontecimientos que se desarrollan en la ciudad para efectuar una mejor gestión en términos medioambientales y económicos entre otros. Los ciudadanos perciben estos cambios en forma de beneficios en ámbitos cotidianos como el transporte público o la circulación por las calles, obteniéndose entornos más seguros, inclusivos y accesibles.

La idea es mantener una conexión constante entre las diferentes áreas de gestión de una ciudad: economía, movilidad, sanidad, seguridad, medio ambiente... Para ello, se introducen diferentes Tecnologías de la Información y la Comunicación (TIC) que se apoyan en el *Big Data* para mejorar la organización y prestación de los diferentes servicios urbanos. Se consigue así un sistema centralizado capaz de recoger y procesar los datos compartidos (*open data*) de los distintos edificios y mobiliario urbano.

Para ello se utilizan, además de sensores y otros sistemas de captación de datos, cámaras de videovigilancia urbana. Este tipo de datos se analiza y procesa para, por ejemplo, conocer los flujos de tráfico a determinadas horas, o la aglomeración de personas en algunas calles. Así pueden tomarse decisiones que alivien o solucionen esos problemas: peatonalización de algunos tramos, añadir o eliminar carriles en la calzada, modificar semáforos, ... En este trabajo nos centraremos en ese procesamiento automático de imágenes y vídeos, también conocido como visión computacional.

Es importante citar también, que este sector no es el único que se ha visto beneficiado por los nuevos avances en visión computacional. En medicina, por ejemplo, permite la detección de tumores o la caracterización de células [27]. Y es incluso capaz de pronosticar, con alta precisión, futuros episodios cardiovasculares mediante imágenes de retinas [43]. En la sección 3.1.2. se mostrarán más campos en los que este tipo de técnicas tienen una gran relevancia.

1.1. Planteamiento del problema

En el contexto actual, en el que la visión artificial está tomando cada vez más peso a la hora de desarrollar nuevas soluciones en proyectos tecnológicos, es importante continuar innovando con nuevas propuestas. Dentro de las arquitecturas más utilizadas en los últimos años, y que reportan mayores beneficios, están aquellas basadas en *Deep Learning*. Es por ello, que surge la necesidad de asentar los fundamentos y bases de estas técnicas para, posteriormente, poder profundizar en campos más concretos. Así se alcanzaría un estado en el que cada vez se generasen más propuestas diferentes, capaces de utilizar los grandes volúmenes de datos almacenados por las empresas para aumentar tanto su rendimiento como sus beneficios. En el ámbito de las ciudades se alcanzaría un estado de armonía entre las diferentes áreas de gestión urbana gracias a la aplicación de acciones eficientes y efectivas decididas en base a los resultados obtenidos por un modelo inteligente. De esta forma, se optimizaría la calidad de vida de las personas en ámbitos como movilidad, economía o medio ambiente. Y esto es solo el principio, ya que cada día se descubren nuevas aplicaciones del *Deep Learning* que mejoran la calidad de vida de la sociedad, como herramientas que sirven de detectores de enfermedades para los profesionales sanitarios [24] o sistemas más seguros para proteger materiales sensibles [5].

Pero para lograr todos los avances citados, es necesario tener personal cualificado para su desarrollo. Una encuesta realizada por la corporación francesa experta en servicios de consultoría tecnológica, el Instituto de Investigación de Capgemini [74],confirma la escasez de personal experto en materias de Inteligencia Artificial. Dicha encuesta, realizada sobre cerca de 1000 organizaciones de 11 países con ingresos anuales estables, afirma que escalar en esta tecnología no es fácil. Tan solo el 13 % de las empresas ha implementado satisfactoriamente sus proyectos piloto. El estudio asocia ese problema con la falta de personal cualificado en niveles medios y senior. Se genera así una brecha entre la oferta y la demanda de personal en disciplinas como el *Machine Learning* o la visión computacional. El estudio concluye, por tanto, que la formación de un mayor volumen de estudiantes en futuras generaciones es imprescindible para obtener nuevos avances en el sector. Esto es muy importante para el desarrollo de las Ciudades 4.0. La tendencia migratoria de la población del campo a la ciudad, presente en los últimos años, ha supuesto una mayor saturación de las mismas. Además, cada día se recogen más datos cuyo análisis permitiría tomar decisiones que mejorasen la calidad de vida de los habitantes. Aunque es cierto que actualmente algunas ciudades como Londres, Nueva York o Barcelona [73] implementan soluciones inteligentes para generar y analizar grandes volúmenes de datos, esto queda lejos de los muchos proyectos innovadores que podrían desarrollarse con la tecnología disponible a día de hoy. Además de que no todas las localidades pueden gozar aún de esos avances.

Un primer paso para lograr el aumento del interés de los jóvenes por estas nuevas tecnologías sería que estuviesen informados de su existencia. Para ello, podría ser oportuno presentarles un tutorial introductorio a la materia, con ejemplos prácticos para que programasen sus primeros algoritmos predictivos. De esta forma, es altamente probable que un mayor porcentaje de los estudiantes se planteara continuar profundizando en este campo. Obtendríamos así a medio o largo plazo, un mayor número de trabajadores cualificados para desarrollar proyectos y sistemas basados en Inteligencia Artificial, de los que la sociedad extraería grandes beneficios. En este Trabajo Fin de Grado se generarán tres *notebooks* de aprendizaje, centrados respectivamente en la clasificación, detección y segmentación de objetos, partiendo desde cero e introduciendo las tecnologías más utilizadas en el ámbito de la visión computacional. Además, como aportación a las *smart cities*, se desarrollará una propuesta de segmentador semántico de objetos utilizan-

do imágenes extraídas de cámaras de videovigilancia urbana para, posteriormente, realizar un proceso de inferencia sobre nuevas imágenes y vídeos.

1.2. Objetivos del trabajo

El proyecto busca resolver el problema expuesto anteriormente, pero se centrará en el ámbito específico de *Deep Learning* aplicado a visión computacional. Los objetivos del proyecto se exponen a continuación:

- **OBJ-1**: Realizar una descripción de las técnicas y arquitecturas de *Deep Learning* utilizadas para el procesamiento de imágenes estáticas y en movimiento.
- **OBJ-2**: Generar material didáctico en forma de *notebooks* introductorios al campo de la visión computacional.
- **OBJ-3**: Utilizar el conocimiento adquirido en un supuesto real de detección de personas y objetos, con datos extraidos de camaras de videovigilancia, como aportación a las citadas ciudades inteligentes.

1.2.1. Restricciones

Por otro lado, las restricciones encontradas para desarrollar el proyecto son:

- **REST-1**: El equipo en el que diseñarán y ejecutarán los diferentes modelos tiene Windows 10 como sistema operativo. Esto puede ser una limitación a la hora de utilizar algunas librerías como Detectron2 o Icevision, que tienen compatibilidad con sistemas Linux o MacOS. Por ello, en esos casos, estaremos obligados a realizar las ejecuciones en el entorno facilitado por el servicio Google Colab, que proporciona una máquina virtual de Linux.
- **REST-2**: Tanto el alcance como la duración del proyecto se verán limitados por la carga de trabajo establecida por la guía docente del Trabajo Fin de Grado (12 créditos).

1.3. Estructura de la memoria

Los capítulos en los que se divide esta memoria son:

- **Capítulo 1**. Introducción. Se introduce el proyecto en el entorno actual y se definen los objetivos y limitaciones.
- **Capítulo 2**. Planificación. En el que se exponen las tareas a desarrollar ordenadas en el espacio temporal y se estiman los costes derivados de su consecución. También se añade un balance analizando si el presupuesto y la planificación prevista se han cumplido correctamente.
- **Capítulo 3**. Contexto de trabajo. Sitúa la necesidad que resolverá el proyecto en la realidad actual y presenta los términos básicos dentro del ámbito de la ciencia de datos. Finalmente expone las propuestas ya existentes para resolver problemas de visión computacional con Deep Learning.

- **Capítulo 4**. Cuadernos de aprendizaje. Introduce los conjuntos de datos y tecnologías utilizadas en los cuadernos de aprendizaje desarrollados.
- **Capítulo 5**. Caso de estudio. Presenta los conjuntos de datos elegidos para el caso práctico final y describe las diferentes transformaciones realizadas sobre ellos. También se detalla cómo se realiza el proceso de carga de los mismos.
- **Capítulo 6**. Experimentación y evaluación. En el que se expone detalladamente el proceso de experimentación paso a paso, se explican las diferentes configuraciones adoptadas y se analizan los resultados obtenidos.
- **Capítulo 7**. Conclusiones y trabajo futuro. Reflexiones finales sobre el proyecto en el ámbito técnico y personal, así como propuestas de mejora.
- **Apéndice A**. Breve manual de instalación de un entorno Anaconda para poder ejecutar los *notebooks* adjuntos en Jupyter Notebooks.
- **Apéndice B**. Contenido adjunto a la memoria.
- **Bibliografía**.

Capítulo 2

Planificación

2.1. Metodología de trabajo

La metodología elegida para el desarrollo del proyecto es UVagile [76][44], que es una propuesta de metodología basada en los principios establecidos en el Manifiesto Ágil[7] aplicados al entorno académico.

UVagile surge como proyecto de innovación docente de la Universidad de Valladolid en el año 2018 con el objetivo de trasladar el éxito cosechado por los marcos de trabajo ágiles en sectores empresariales al ámbito académico, en concreto a la enseñanza universitaria. Para ello propone, entre otras prácticas, delegar una mayor responsabilidad en los alumnos en su proceso de aprendizaje y que este sea incremental e iterativo. De esta forma el proceso se enfoca como un proyecto de aprendizaje que se construye iterativamente mediante incrementos denominados *sprints* de aprendizaje. Además, el proyecto no se define detalladamente al inicio, sino que se profundiza en las diferentes partes del mismo a medida que avanza su desarrollo y teniendo en cuenta el *feedback* que se recibe de manera regular.

La metodología se orienta tanto a la docencia de asignaturas, como al desarrollo de otro tipo de proyectos académicos como pueden ser Trabajos Fin de Grado o Trabajos Fin de Master. Describiremos los roles, eventos y artefactos para esta segunda vertiente.

Comencemos analizando los diferentes roles con los que se trabaja. Algunos de ellos están basados en Scrum [64].

- En primer lugar, diferenciamos el rol del **estudiante**, que es el alumno matriculado en la asignatura Trabajo Fin de Grado. Se basa en el rol de Desarrollador. Es el responsable de generar un producto de calidad, consistente en una memoria escrita en Latex, que documente adecuadamente aspectos del proyecto como pueden ser el contexto en el que se enmarca, el estado del arte, la planificación seguida o el proceso de experimentación (si existiese). Además, en los casos en los que se desarrolle algun tipo de herramienta, estas serán consideradas también parte del producto.

  Una vez construido, el estudiante debe presentar el producto ante el tribunal apoyándose en un conjunto de diapositivas y haciendo una demostración de la herramienta desarrollada en caso de que se haya generado.

  El estudiante debe asistir a todos los eventos definidos por la metodología Uvagile, y debe plantear y planificar las tareas a ejecutar para satisfacer los objetivos de cada *sprint*.

Posteriormente debe completar dichas tareas según lo establecido y mantener actualizado el estado real del proyecto en el tablero Trello y en el espacio de trabajo compartido.

Es importante también comunicar de manera fluida al profesor los avances y dificultades experimentados, bien a través de las reuniones fijadas, o bien mediante los medios establecidos para ello (canal privado de Teams). Finalmente, el estudiante debe completar un cuaderno de aprendizaje en el que indique el tiempo invertido en la realización de las diferentes tareas para medir de forma precisa la dedicación parcial y total en los diferentes objetivos del proyecto.

- A continuación definiremos el rol de **profesor**, que en este caso es desempeñado por el tutor o tutores del Trabajo Fin de Grado. Se basa en los roles de *Product Owner* y *Scrum Master*.

  El profesor, inicialmente, debe plantear un proyecto al estudiante que se adapte a los requerimientos exigidos por el tribunal (los recogidos en la guía docente de la asignatura) y, posteriormente, al inicio de cada *sprint*, elegir qué objetivos del proyecto se desarrollarán en el mismo, detallando los requerimientos en el tablero Trello. Además, debe orientar al estudiante en el desarrollo del proyecto, así como resolver posibles bloqueos o dudas manteniendo una comunicación fluida a través de los medios establecidos y proporcionando *feedback* regular. Todo ello con la intención de maximizar la calidad del producto final.

  También debe asistir a todos los eventos establecidos por Uvagile, y comprobar que se desarrollan de forma adecuada.

- El siguiente rol está formado por los estudiantes que realizan un Trabajo Fin de Grado de temática relacionada (en este caso ciencia de datos) y que siguen una planificación temporal similar (desarrollado en cuatro o cinco *sprints* entre los meses de marzo y julio). Se denomina **comunidad** y participa únicamente en los eventos de comunicación de progresos (aportando *feedback* al estudiante) y retrospectiva (valorando el trabajo realizado y proponiendo acciones de mejora). Además, debe mantener una comunicación fluida con el estudiante, resolviendo dudas o bloqueos, mediante los medios elegidos para ello (canal público de Teams).

- Finalmente, el **tribunal** está formado por el conjunto de profesores designados para evaluar el Trabajo Fin de Grado. Está basado en el rol del cliente. Debe evaluar de forma objetiva el producto de aprendizaje presentado por el estudiante, así como la documentación entregada y la defensa del proyecto, todo ello de acuerdo a lo establecido en el *Reglamento específico relativo a la elaboración y evaluación del Trabajo Fin de Grado*.

A continuación, se diferenciarán los eventos realizados durante la ejecución del proyecto.

- ***Sprint***. División del proyecto que facilita la planificación del trabajo y la priorización de objetivos que proporcionan mayor valor en diferentes momentos. Permite la revisión y corrección periódicas y regulares de los avances realizados en el mismo, así como acotar el ritmo de trabajo para que se ajuste al establecido en la guía docente de la asignatura Trabajo Fin de Grado.

  En este caso, el proyecto abarca 4 *sprints* de 4 semanas, con una carga de unas 75 horas por *sprint*.

- **Reunión de inicio**. Se realiza al comienzo de cada *sprint* y en ella participan estudiante y profesor. Se centra en definir los objetivos del *sprint* en base al *feedback* recibido en el *sprint* anterior, pero también en determinar las tareas necesarias para su consecución y su distribución temporal en el mismo.

  La reunión de sincronización debe tener una duración de 30 minutos y en este proyecto se ha realizado el primer martes de cada *sprint*

- **Reunión de sincronización**. Es una reunión semanal de 15 minutos de duración en la que participan estudiante y profesor. El objetivo de las mismas es mantener una comunicación fluida y periódica entre ambas partes. Para ello se exponen cuáles han sido los avances y bloqueos experimentados desde la última reunión. Además, se planifica también qué tareas se desarrollarán la siguiente semana.

  En este caso, las reuniones de sincronización se han realizado los martes por la tarde.

- **Reunión de comunicación de progresos**. Se realiza al final del *sprint* y en ella participan el estudiante, el profesor y la comunidad. El objetivo de la reunión es que el estudiante presente el incremento generado. El resto de los participantes comentan el progreso y exponen sugerencias de mejora en lo respectivo al contenido del trabajo. La reunión se realiza el último día del *sprint*, en este caso los martes, y debe tener una duración máxima de 1 hora por estudiante.

- **Retrospectiva**. Es la última reunión del *sprint*, de duración inferior a 30 minutos, en la que nuevamente participan el estudiante, el profesor y la comunidad. Tiene como objetivo revisar la dinámica de trabajo utilizada durante el *sprint* y generar valoraciones tanto positivas como negativas sobre el mismo, así como sugerencias de mejora. En este proyecto se realizaba plasmando los comentarios en la herramienta online EasyRetro, que permite mantener una visión compartida y en tiempo real de las notas propuestas por la comunidad.

Finalmente se mostrarán los artefactos.

- En primer lugar el **espacio de trabajo compartido**, que es el lugar en el que deben encontrarse las versiones actualizadas del producto generado por el alumno. En este proyecto es un canal de teams que permite que profesor y estudiante tengan acceso al mismo y almacena una copia de seguridad en la nube.

  El espacio de trabajo compartido se divide en:

  - Un directorio dedicado a la documentación académica relacionada con el Trabajo Fin de Grado, para almacenar archivos como la comunicación de inicio o la solicitud de defensa. Se denomina documentación.
  - Un directorio para depositar los productos de cada uno de los *sprints*. Se denomina incrementos.
  - Un directorio, denominado personal, para que el estudiante almacene los resultados intermedios o temporales que considere.
  - Un último directorio que contenga recursos utilizados en el Trabajo Fin de Grado, como puedan ser referencias o *datasets*.

  Este espacio de trabajo también debe poseer canales de comunicación, en concreto:

- Un canal privado para la comunicación entre estudiante y profesor.
- Un canal público para la comunicación con el resto de la comunidad.

- El **tablero del proyecto**, utilizado para organizar los objetivos y tareas, así como reflejar los progresos realizados (ver Figura 2.1). Es accesible para los roles estudiante y profesor. La herramienta utilizada es Trello, que es un software de administración de proyectos. Dentro del tablero se diferencian las siguientes columnas:

  - *Backlog del proyecto*, contiene una lista priorizada de objetivos del proyecto que no se han incluido en ninguno de los *sprints* pasados ni actuales. Dan una visión muy superficial de los objetivos futuros, por lo que no se describen ni detallan en absoluto.
  - Objetivo del *sprint*. Objetivos seleccionados para el *sprint* actual.
  - Tareas pendientes. Tareas que se deben realizar en el *sprint* actual para alcanzar los objetivos de la columna anterior pero que todavía no se han comenzado.
  - En curso. Tareas que se están desarrollando en el momento actual.
  - Bloqueado. Tareas que se comenzaron pero que no se pueden continuar porque requieren la ayuda del *Product Owner*.
  - Finalizado. Tareas terminadas.

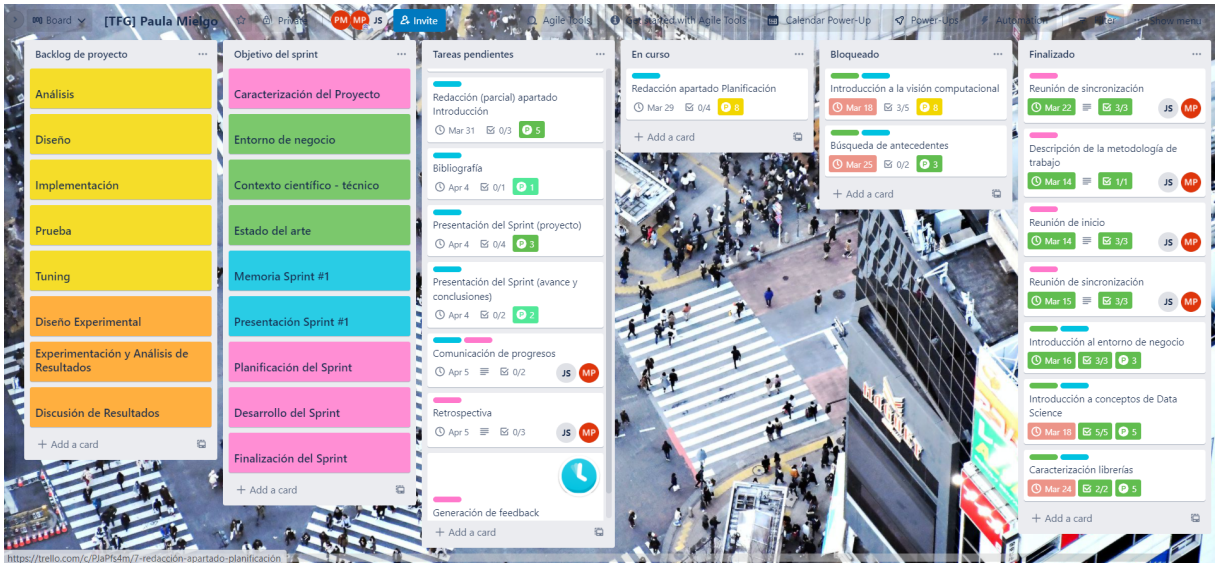


Figura 2.1: Tablero del proyecto

En cada una de esas columnas se localizan las diferentes tareas u objetivos, que tienen los siguientes componentes (ejemplificados en la Figura 2.2):

- Título de la tarea.
- Fecha de vencimiento.
- *Story points*. Es una medida arbitraria y cualitativa utilizada para caracterizar el esfuerzo necesario para completar una tarea, de acuerdo a la experiencia del equipo.

Esta estimación no se corresponde necesariamente con el tiempo requerido para la consecución de dicha tarea. Trello permite asignar una puntuación de 1, 2, 3, 5, 8, 13 o 21.

- Descripción de la tarea.
- *Checklist* o lista de subobjetivos, que permite determinar cuándo la tarea se ha completado.
- Etiquetas, que indican a qué parte del proyecto pertenece la tarea u objetivo. En este caso diferenciamos Proyecto (Rosa), *Background* (Verde), Comunicación (Azul), Desarrollo técnico (Amarillo) y Evaluación (Naranja).

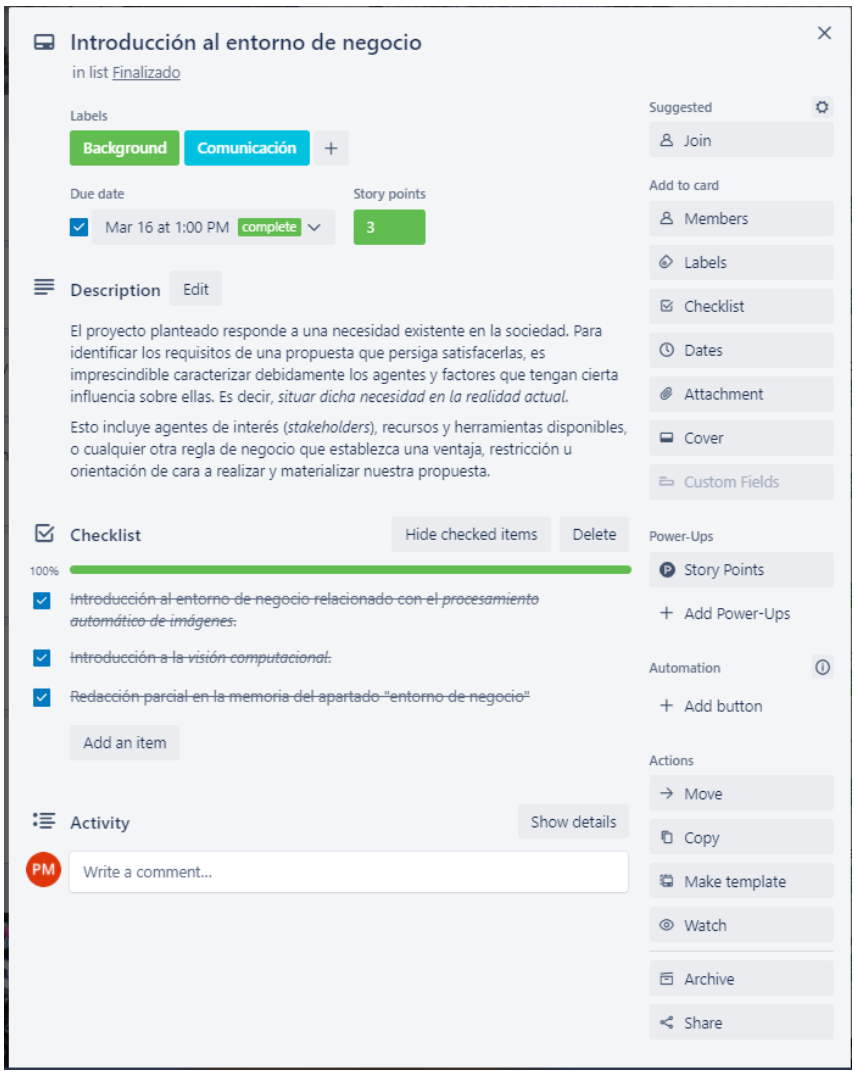


Figura 2.2: Ejemplo de una tarea

- **Incremento**. Es la suma de las tareas completadas durante un *sprint*, junto con las terminadas en los anteriores. Por tanto, es el estado del producto en el momento de finalización

del *sprint*. En ese momento, el incremento debe depositarse en el espacio de trabajo compartido, presentarse en la comunicación de progresos y ser revisado por el profesor para generar *feedback* valioso para el estudiante.

- **Cuaderno de trabajo**. Es un documento en el que el estudiante debe indicar el tiempo invertido en la realización de cada una de las tareas planteadas. Permite medir de forma precisa el total de horas empleado en cada uno de los *sprints*, y por tanto, en el proyecto, y también valorar la dedicación necesaria para satisfacer los objetivos del mismo.
- **Retroalimentación**. Es la valoración del incremento entregado en un *sprint*. Diferenciamos dos tipos:
  - Retroalimentación del profesor. El profesor valorará inicialmente el incremento en la presentación de la comunicación de progresos para, posteriormente, revisar documentación entregada en el canal privado y generar un informe o corrección de la misma. Este *feedback* debe ser valioso para el alumno, claro y completo, y su entrega debe realizarse en la semana posterior a la finalización del *sprint*.
  - Retroalimentación de la comunidad. En este caso, la comunidad aportará los comentarios y sugerencias oportunas únicamente durante la comunicación de progresos.

2.2. Planificación temporal

Dividiremos el proyecto en 4 *sprints*, comenzando con el primero el día 14 de marzo y concluyendo con el último el día 12 de julio. Las reuniones semanales de sincronización se realizarán los martes, y también las de finalización de *sprint*. Veamos cuáles son las historias de proyecto que se abordan en cada *sprint*, así como las tareas planteadas para su consecución.

2.2.1. Sprint #1

- **H1**. Entorno de negocio
  - **T1.1**. Introducción al entorno de negocio
  - **T1.2**. Introducción a la visión computacional
- **H2**. Contexto científico-técnico
  - **T2.1**. Introducción a conceptos de Data Science
  - **T2.2**. Caracterización de librerías
- **H3**. Estado del arte
  - **T3.1**. Búsqueda de antecedentes
  - **T3.2**. Análisis de antecedentes
- **H4**. Memoria
  - **T4.1**. Redacción apartado Planificación
  - **T4.2**. Redacción (parcial) apartado Introducción

  - **T4.3**. Bibliografía
- **H5**. Presentación Sprint
  - **T5.1**. Presentación del Sprint (proyecto)
  - **T5.2**. Presentación del Sprint (avance y conclusiones)

2.2.2. Sprint #2

- **H6**. Análisis de los cuadernos
  - **T6.1**. Caso de estudio cuadernos
  - **T6.2**. Objetivos, ejemplos y tecnologías
- **H7**. Implementación de los cuadernos
  - **T7.1**. Notebook clasificación
  - **T7.2**. Notebook detección
  - **T7.3**. Notebook segmentación
- **H4**. Memoria
  - **T4.4**. Corregir Sprint #1
  - **T4.5**. Redacción del capítulo Cuadernos de Aprendizaje
  - **T4.6**. Actualizar bibliografía
- **H5**. Presentación Sprint
  - **T5.3**. Modificar parte sprint #1
  - **T5.4**. Presentación parte sprint #2

2.2.3. Sprint #3

- **H8**. Completar objetivos pendientes sprint anterior
  - **T8.1**. *Notebook* detección
  - **T8.2**. *Notebook* segmentación
- **H9**. Cuaderno caso final
  - **T9.1**. Notebook caso final
  - **T9.2**. Entrenamiento modelos
- **H4**. Memoria
  - **T4.7**. Completar apartados pendientes sprint #2
  - **T4.8**. Corregir sprint #2
  - **T4.9**. Actualizar bibliografía
- **H5**. Presentación Sprint
  - **T5.5**. Modificar parte sprints #1 y #2
  - **T5.6**. Presentación parte sprint #3

2.2.4. Sprint #4

- **H9**. Cuaderno caso final
  - **T9.3**. Continuar con entrenamiento modelos
- **H4**. Memoria
  - **T4.10**. Corregir sprint #3
  - **T4.11**. Redacción capítulo Caso de estudio
  - **T4.12**. Redacción capítulo Experimentación y evaluación
  - **T4.13**. Actualizar bibliografía
  - **T4.14**. Correcciones finales
- **H5**. Presentación Sprint
  - **T5.7**. Modificar parte sprints #1, #2 y #3
  - **T5.8**. Presentación parte sprint #4

2.2.5. Diagrama de Gantt

A continuación se mostrará el diagrama de Gantt (ver Figura 2.3). En él se ha incluido también los *story points* asignados a cada tarea. Las semanas de trabajo abarcan de lunes a viernes, dejando el fin de semana de descanso. Además, no se tiene en cuenta la Semana Santa (S5) por no ser lectiva.

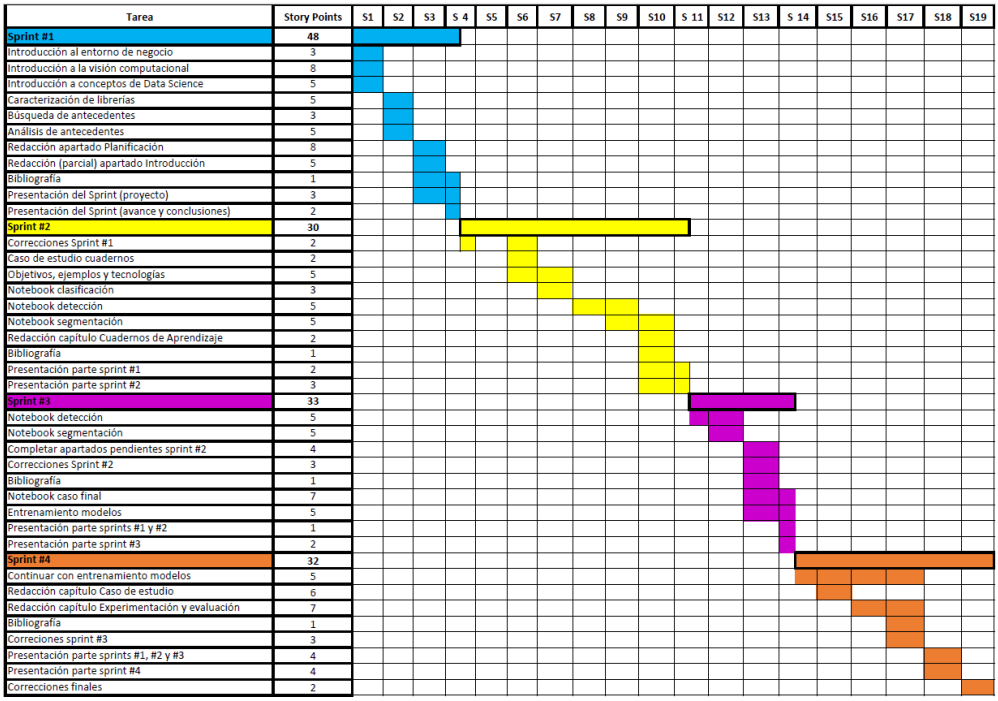


Figura 2.3: Diagrama de Gantt del proyecto

2.3. Presupuestos

Dividiremos el presupuesto en dos apartados. Por un lado, consideraremos los recursos técnicos, que incluirá aquellos materiales y equipos hardware y software que se utilizarán para desarrollar el proyecto. Por otro, tendremos en cuenta cuáles serán los gastos asociados a recursos humanos, es decir, el coste de los trabajadores.

- **Recursos técnicos**

  El proyecto se desarrollará con un ordenador portátil con las siguientes características: procesador Intel Core i7-7700HQ, 16GB RAM, y con dos discos duros, 1TB HDD y 256GB SSD. Este dispositivo, de gama media-alta, ronda los 1100€, y tiene unos 5 años de vida útil. Calcularemos posteriormente el coste de amortización asociado al periodo de trabajo del proyecto, unos 4 meses.

  Por otro lado, será necesario disponer de una buena conexión a Internet para poder realizar las reuniones de inicio y sincronización, para las que se utilizarán las videollamadas de Microsoft Teams. Además, también será necesario para consultar artículos, revistas científicas y libros, así como para poder descargar los *datasets* que se utilizarán posteriormente. Necesitaremos 4 meses de acceso a Internet, la tarifa elegida suministra 300Mb de fibra simétrica por 32€.

  El resto de elementos que se utilizarán se corresponden con herramientas software de licencia gratuita como Overleaf, Trello, Google Colab o Jupyter Notebook. Por otro lado, tanto Microsoft Office como el sistema operativo Windows 10 están incluidos en la compra del ordenador, por lo que supondremos que tienen un coste de 0€. Todo ello aparece resumido en la Tabla 2.1.

	Coste	Porcentaje de uso	Total
Ordenador portátil	1100€	6,7 %	73,7€
Conexión a Internet	32€/mes	400 %	128€
Overleaf	0€		0€
Trello	0€		0€
Google Colab	0€		0€
Jupyter Notebook	0€		0€
Microsoft Office	0€		0€
Windows 10	0€		0€
Total			91,7€

Tabla 2.1: Recursos técnicos.

- **Recursos humanos**

  Consideraremos el salario de los diferentes roles que requeriría nuestro proyecto: analista y desarrollador. Haremos una estimación de su salario suponiendo que el trabajador realiza un total de 40 horas a la semana (160 horas al mes). Considerando que el salario de un

analista ronda los 30000€ de media al año [70] y prorrateándolos entre 12 meses (en lugar de dividirlo en las 14 pagas anuales), obtenemos un salario de 2500€ al mes; unos 15,62€ por hora. Por otro lado, necesitaremos también un *data scientist* o científico de datos, que tiene un salario de unos 35000€ al año [68]; 2920€ al mes; 18,25€ la hora. Finalmente, el salario de un desarrollador es de unos 25000€ al año [69], 2080€ al mes, es decir, unos 13€ por hora.

Además, deben tenerse en cuenta los gastos que se deberán abonar a la Seguridad Social al contratar a los citados trabajadores. Esto supone un 30 % de su salario bruto [80].

Como el Trabajo Fin de Grado está pensado para desarrollarse en unas 300 horas (12 créditos), obtenemos la siguiente tabla (ver Tabla 2.2).

	Salario por hora	Horas totales	Total sin SS	Coste SS	Total
Analista	15,62€	110	1718,2€	515,46€	2233,66€
Científico de datos	18,25€	100	1825€	547,5€	2372,5€
Desarrollador	13€	90	1170€	351€	1521€
Total					6127,16€

Tabla 2.2: Recursos humanos.

Obtenemos así que el presupuesto total es de 6218,86€.

2.4. Balance temporal y económico

En la primera semana de desarrollo del proyecto, algunas tareas se vieron retrasadas debido a problemas de salud. Esto intentó compensarse en el resto del primer sprint, repartiendo dichos retrasos de manera equitativa en cada semana. De esta manera se logró completar todas las tareas a tiempo sin tener que hacer modificaciones mayores como posponer tareas para el segundo sprint.

Sin embargo, en el segundo sprint debido a la alta carga de trabajo académica (externa al Trabajo de Fin de Grado) y en previsión de la futura disponibilidad del alumno para los sprints tercero y cuarto, se pospusieron varias tareas no llegando a completar algunos objetivos. En concreto las tareas T7.2 y T7.3 no se llegaron a completar, quedando pendientes para el sprint #3. Además, finalmente se utilizó la semana 5 (Semana Santa) para adelantar parte del trabajo.

En el sprint #3 se completaron los objetivos pendientes y además se comenzó con el desarrollo del caso final práctico (segmentación con City Scapes), realizándose la implementación del cuaderno y entrenando varios modelos (H9).

Por último, en el *sprint* #4 se cumplió con la planificación inicial y se añadió una nueva tarea para el entranamiento con un nuevo *dataset*.

Todos estos cambios en los tiempos de ejecución de las tareas pueden verse reflejados en la Figura 2.4. Los colores indican lo siguiente:

- Verde. Tareas que se han añadido despues de la planificación inicial.
- Gris. Si la tarea no ha comenzado en el tiempo previsto los recuadros grises indican cuándo debían comenzar según la planificación original.

- Rojo. Si la tarea no ha terminado en el tiempo previsto los recuadros rojos indican cuándo han concluido realmente.

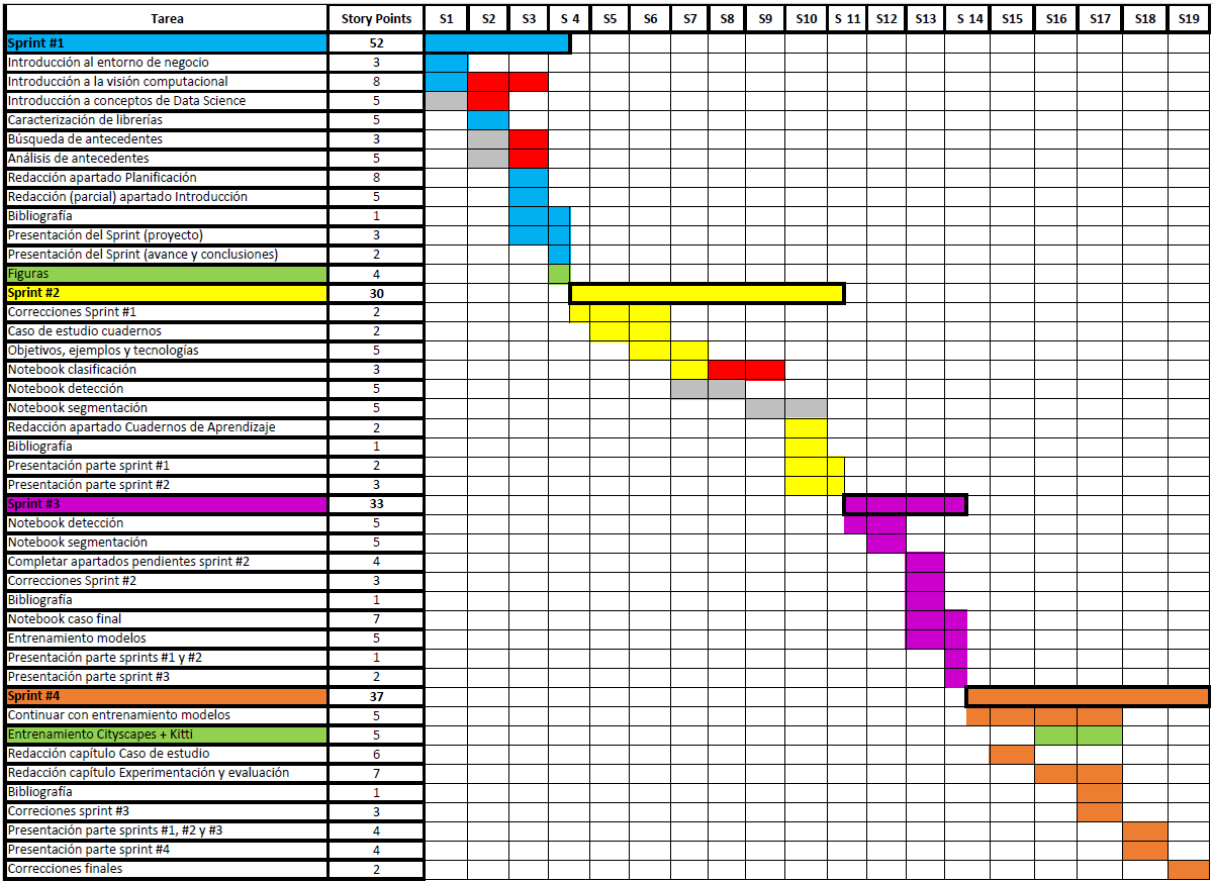


Figura 2.4: Plazos reales de ejecución de las tareas

El primer sprint ha supuesto un total de 92 horas de trabajo y en el segundo se realizaron 43. Por su parte, en el sprint #3 se emplearon 89 horas y media y en el #4 81 y media. La suma total es de 306 horas y, por tanto, se ajusta bastante bien a lo requerido por la guía docente, que establecia la carga de trabajo en 300 horas.

Respecto al ámbito económico, las tareas han podido realizarse con los medios técnicos especificados. Sin embargo, tendremos que sumar el coste de las 6 horas no previstas en el presupuesto, asociadas al rol de científico de datos (ver modificación en la Tabla 2.3).

	Salario por hora	Horas totales	Total sin SS	Coste SS	Total
Científico de datos	18,25€	106	1934,5€	580,35€	2514,85€

Tabla 2.3: Balance económico científico de datos.

El coste económico final es, por tanto, de 6364,21€.

Capítulo 3

Contexto del Trabajo

3.1. Entorno de negocio

En esta sección se presenta una introducción del procesamiento automático de imágenes en la sociedad, en la que se abordarán aspectos como las necesidades existentes en la realidad actual, factores influyentes en esas necesidades o agentes de interés.

3.1.1. Procesamiento automático de imágenes

En la introducción se mencionó la importancia de la Inteligencia Artificial en la sociedad actual. En concreto, se destacó el *Deep Learning* como principal herramienta en el desarrollo de proyectos de visión artificial. Este campo está adquiriendo cada vez más importancia en los últimos años. Esto se debe a la mejora de los elementos de procesamiento que utilizan los modelos para entrenarse, así como a la creación de nuevas arquitecturas y librerías que permiten que los procesos sean más eficientes. Todo ello ha supuesto que la profesión de especialista en Inteligencia Artificial, con habilidades en *Machine Learning*, *Deep Learning* o *Data Science*, se haya postulado como la primera profesión emergente, según un estudio de la plataforma Linkedin[41], con un 76 % de crecimiento anual.

Comencemos introduciendo el concepto de procesamiento automático de imágenes. Este puede definirse como la rama de la Inteligencia Artificial que tiene como objetivo la captación, procesamiento y análisis de imágenes, por parte de un computador, tal y como lo haría un cerebro humano. Es decir, trata de simular la visión humana obteniendo e interpretando la información extraída del campo visual. Podríamos dividir la visión artificial en tres grandes etapas [30].

- **Captura de los datos** mediante el sensor de una cámara digital. Independientemente de si se capturan imágenes o vídeos, los datos se guardarán en un dispositivo de almacenamiento y posteriormente podrán transferirse a otro sistema electrónico.

  Veamos cómo se almacenan estos datos (ver Figura 3.1). Comencemos con el caso de una imagen en blanco y negro. Solo necesitamos un valor para cuantificar la intensidad del color negro en cada *pixel*, de forma que representaremos la imagen como una matriz de tamaño MxN, siendo M el número de *pixels* capturados en cada columna y N el número de columnas. Sin embargo, en el caso en el que capturemos una imagen a color necesitaremos más de un valor para representar esos colores. Se utiliza para ello el sistema RGB (Red, Green, Blue) que representa millones de colores con solo tres valores, que son: la intensidad

del color rojo, la del color verde y la del color azul. Es decir, en este caso necesitamos tres canales, uno por color, de forma que la imagen se almacenará en un array de dimensión 3xMxN.

El caso del vídeo es un poco diferente. Trataremos el caso de un vídeo a color, ya que el caso de capturar un vídeo en blanco y negro se trataría igual. Un vídeo puede verse como una secuencia ordenada de imágenes. Estas imágenes, denominadas frames, podrán representarse al igual que antes con tres canales en formato RGB. Poseerán todas ellas además el mismo número de *pixels* de alto y de ancho, pudiendo representarse entonces en un array de la forma 3xFxMxN, siendo F el número de frames que posee el vídeo.

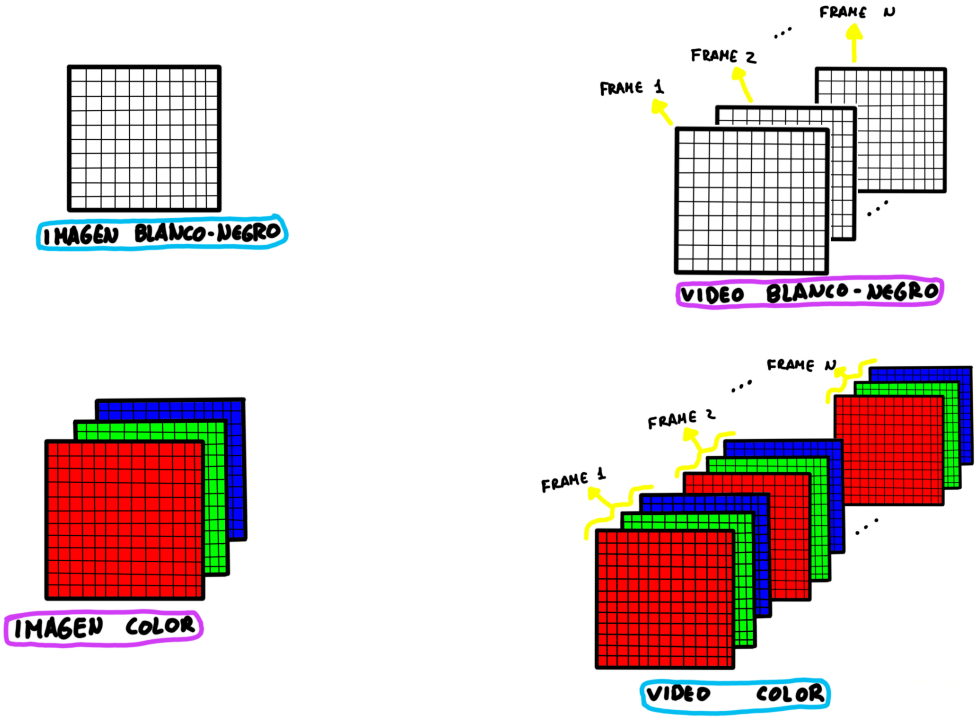


Figura 3.1: Representación de imágenes y vídeos

- **Análisis de los datos**. Una vez capturadas las imágenes podemos realizar un análisis de la información prácticamente en tiempo real gracias a la potencia de los procesadores actuales. Este proceso se conoce como visión computacional e incluye el procesamiento previo de los datos, la obtención de características y la interpretación de la información extraida [30]. En secciones posteriores se profundizará sobre este apartado.

- **Uso de los resultados extraídos de los datos**. Finalmente, tras el análisis realizado y la información obtenida se toma una decisión. Esto puede ser, por ejemplo, mover una articulación de un robot o descartar una pieza defectuosa.

Una vez tenemos las imágenes o vídeos capturados, es decir, hemos completado la primera fase de la visión artificial, podemos realizar una serie de tareas o análisis sobre ellas. Estos procesos

se encuentran enmarcados en la visión computacional. Veamos cuáles son las diferentes acciones que podemos realizar sobre nuestro conjunto de datos (ver Figura 3.2):

- **Clasificación**. Esta tarea tiene como objetivo determinar la categoría de una imagen entre un número finito de categorías posibles. Un ejemplo de clasificación sería diferenciar si en una imagen aparece un perro o un gato.

- **Detección de objetos**. Se ocupa de descubrir la presencia de objetos en una imagen o vídeo. Por ejemplo, detectar que hay una manzana y dos naranjas en una imagen de una mesa y localizar dónde se encuentran.

- **Segmentación**. Tiene como objetivo delimitar diferentes regiones en una imagen. La principal diferencia con respecto a la detección de objetos es que, mientras que en la detección solo es necesario enmarcar los objetos, por ejemplo, con un cuadrado, en la segmentación se busca diferenciar claramente qué *pixels* pertenecen a cada región. De esta forma todos los *pixels* de la imagen quedan asociados a una categoría concreta. Un ejemplo de esto podría ser una imagen de un árbol en la que se diferencie qué *pixels* pertenecen a la copa, cuáles pertenecen al tronco y, finalmente, cuáles forman parte del fondo. A su vez podemos diferenciar dos tipos de segmentación:

  - **Semántica**. Diferencia las regiones por tipo. Es decir, si poseemos por ejemplo dos casas separadas estas serán asignados a la misma categoría.
  - **De instancias**. Marca como regiones diferentes instancias diferentes de un mismo objeto. En el ejemplo anterior se clasificaría por tanto cada casa como una región diferente de la otra.

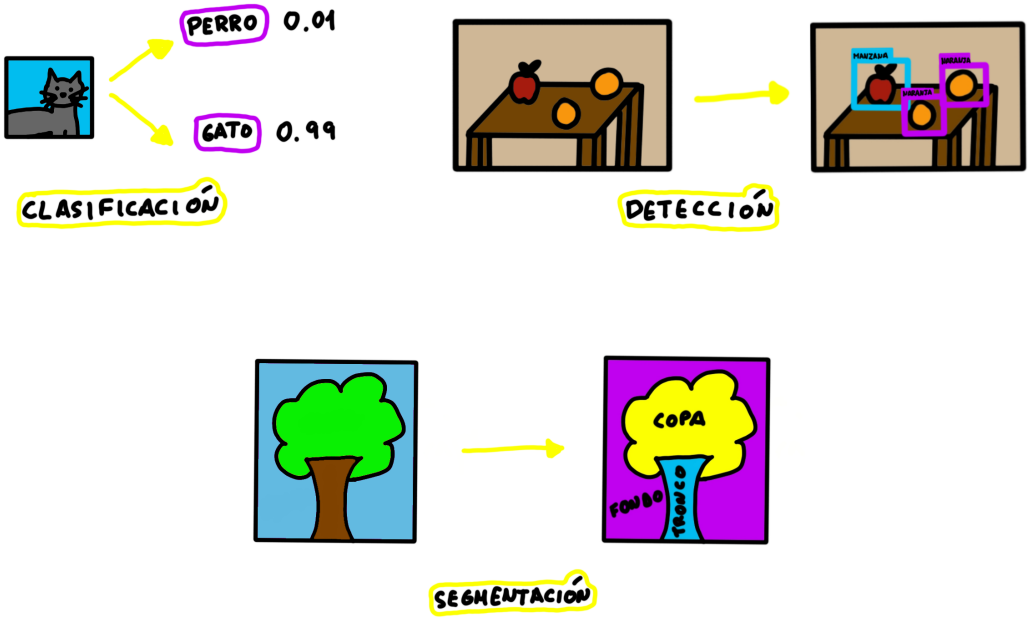


Figura 3.2: Ejemplo de clasificación, detección de objetos y segmentación

- **Generación**. Se centra en la creación de contenido audiovisual desde cero o basándose en otro contenido ya existente. Dentro de los algoritmos generativos encontramos algunas agrupaciones, entre las que destacan los algoritmos de *deep fake* [2] (falsificación de contenido audiovisual), los de *style transfer* [49] (manipulación de un archivo para que adopte el estilo de otro) (ver Figura 3.3) o los algoritmos de *super-resolution* [39] (aumento de la resolución de una imagen o vídeo).

Figura 3.3: Ejemplo de generación utilizando un algoritmo de *style transfer*[49]

La implementación de todas estas tareas se basa en la tecnología *Deep Learning*. Con estas técnicas podemos crear modelos flexibles que trabajan con conjuntos de datos masivos de manera más eficiente que otras técnicas de *Machine Learning*. Podemos obtener así soluciones eficaces a problemas de visión complejos que, con otro tipo de soluciones tradicionales, requerían una enorme cantidad de recursos o que incluso no podían resolverse. Por ejemplo, en el caso en que queramos hacer una lectura de caracteres. Con un sistema de visión tradicional necesitaríamos poseer una colección de patrones que contuviese todas las formas posibles para reconocer cada letra. Sin embargo, además de que esas colecciones son poco manejables, si quisiésemos leer una etiqueta defectuosa (doblada o borrosa) esos algoritmos probablemente no fuesen capaces. Utilizando en cambio un algoritmo de *Deep Learning*, el modelo sería capaz de generalizar la apariencia de la etiqueta y, posiblemente, reconocer correctamente sus letras.

3.1.2. Casos de aplicación

Veremos ahora algunos ejemplos reales de sistemas de visión artificial aplicados en diferentes sectores.

- **Agricultura**. John Deere, en colaboración con Blue River Technology, prepara el lanzamiento de la tecnología de detección *"See&Spray"* [29] que se podrá incorporar a su maquinaria agrícola. El sistema cuenta con cámaras que capturan 20 imágenes por segundo. Con esas imágenes, el modelo es capaz de diferenciar los cultivos de las malas hierbas, aplicando los tratamientos químicos oportunos solo a estas últimas.

Se reduce así el uso de pesticidas, lo cual es beneficioso para la salud de los consumidores y para el medio ambiente. Además, se minimizan los costes en tratamientos, y aumenta la calidad de las cosechas.

- **Medicina**. La termografía médica es una de las técnicas que se usan actualmente para la detección y localización de algunas enfermedades[6]. El sistema está formado por una cámara infrarroja y un modelo de visión computacional que permiten hacer una medición de la temperatura en la zona deseada sin contacto directo. Es por ello una técnica no invasiva que detecta con una alta precisión los cambios que experimenta la temperatura corporal. Esto permite diagnosticar más rápidamente lesiones internas o tumoraciones sin causar molestias o trastornos físicos a los pacientes. En concreto, se ha convertido en una técnica esencial tanto para el diagnóstico como para el seguimiento del cáncer de mama.

- **Conducción automática**. Aunque Tesla no ha alcanzado aún la conducción autónoma total, ha hecho grandes avances en este campo utilizando algoritmos de visión computacional, en concreto de segmentación semántica. Sus sistemas de conducción utilizan técnicas de detección de objetos y personas para tomar decisiones en términos de conducción. Utiliza para ello varias cámaras y sensores que, junto con el citado modelo de Inteligencia Artificial, son capaces de identificar prácticamente todo lo que rodea al vehículo. De esta forma, un coche puede circular correctamente por su carril, sin salirse del mismo, es capaz de frenar si detecta a un peatón, o de tomar una salida o cruce cuando sea necesario. Sin embargo, recientemente se ha cuestionado la capacidad de este tipo de sistemas para detenerse ante la presencia de animales. En concreto en Australia, donde es bastante frecuente la aparición de canguros, se ha especulado seriamente sobre si el vehículo sería capaz de detenerse ante la presencia de uno. Este animal es especialmente problemático porque puede aparecer tanto en el suelo como en medio de un salto. La respuesta a esa cuestión no tardó en aparecer: unas veces lo hace y otras no[22]. Dos usuarios compartieron vídeos en los que se muestran ambos casos. Esto puede suponer un gran problema ya que en zonas del oeste en épocas de sequía es bastante probable encontrar canguros comiendo cerca de carreteras, pudiendo causar graves accidentes.

- **Generación artificial de retratos**. Los algoritmos centrados en generación de contenido audiovisual son capaces de crear rostros artificiales que no existen en la vida real (ver Figura 3.4). Esto es posible con el uso de técnicas como las Redes Neuronales Adversarias, en las que dos modelos de *Deep Learning* compiten por obtener los mejores resultados. Una de las redes, la generadora, crea un determinado contenido, por ejemplo un retrato de una persona que no existe en la realidad, para intentar que la otra red, denominada discriminadora, los perciba como retratos reales. Por otro lado, la red discriminadora se entrena para detectar esos retratos falsos. De esta forma ambas redes se aportan feedback y mejoran colaborativamente.

  Pero la generación de esos retratos también puede realizarse tomando otros archivos como base. En este caso, estamos tratando con algoritmos de *Deep fake* o falsificación de contenido. Se consiguen así piezas audiovisuales en las que un determinado individuo dice o hace cosas que realmente no han ocurrido, bien sea mediante la superposición de su rostro en el de otra persona que sí realiza dicha acción, o bien mediante la generación desde cero del archivo tomando imágenes como base. Como ejemplo reciente de esto último, en el contexto de la invasión de Ucrania por parte de tropas rusas en Febrero de 2022, encontramos un

Figura 3.4: Retratos generados artificialmente

vídeo[25] generado con estas tecnologías, en el que aparece Volodímir Zelenski, presidente ucraniano, anunciando la rendición de sus tropas. Este es el primer caso de *Deep fake* utilizado en un conflicto militar.

- **Seguridad**. En el ámbito de la seguridad destaca el control de acceso mediante reconocimiento facial. En concreto, Apple desarrolló el sistema Face ID[1] para el desbloqueo de sus dispositivos mediante detección de datos biométricos faciales. Esta tecnología se compone de una cámara frontal, una cámara infrarroja, un proyector de infrarrojos y una luz de apoyo. Todo esto, junto con el uso de redes neuronales complejas, consiguen escanear el modelo de la cara en tres dimensiones, es decir, se consigue detectar no solo la distancia entre elementos, sino que también se puede percibir cuál es la profundidad entre los mismos. Se consigue así un sistema robusto de control de acceso al dispositivo, en el que la posibilidad de identificar a un individuo erróneamente como usuario permitido es de una entre un millón.

  Este sistema se hizo tan popular que otros fabricantes han implementado sistemas de desbloqueo similares en sus nuevos dispositivos.

- **Control de calidad**. La detección de defectos en controles de calidad facilita por un lado la percepción de características difícilmente visibles por el ojo humano y disminuye, por otro, los posibles errores de los operadores a la hora de eliminar productos defectuosos causados por cansancio y por el factor repetitivo de la tarea. Expondremos en concreto el caso del sistema SANE desarrollado por la empresa Egitron [62]. Este sistema de análisis, utilizado en cadenas de producción de líquidos almacenados en botellas de vidrio con corcho, permite analizar el nivel de llenado de las botellas, el espacio libre entre el liquido y el corcho o detectar posibles anomalías en el envase. Para ello utiliza un sistema de visión artificial mediante una cámara inteligente que permite inspeccionar unas 130 botellas por minuto.

3.2. Contexto científico-técnico

3.2.1. *Machine Learning*

El *Machine Learning* o Aprendizaje Automático es [12] [42]la rama de la Inteligencia Artificial que tiene por objetivo el diseño y desarrollo de algoritmos que permitan a los ordenadores mejorar su desempeño en la realización de una tarea a partir de la experiencia. Dicho de otra manera, se consigue capturar el conocimiento a partir de la información contenida en los datos aportados como ejemplos para mejorar iterativamente y poder tomar decisiones basadas en esos datos posteriormente.

Pero esta idea choca con la programación tradicional, en la que un software se desarrolla en base a reglas previamente establecidas. En *Machine Learning* dejamos a los ordenadores que "se programen a sí mismos" a través de la información contenida en los datos que se le aportan. Este proceso se conoce como entrenamiento. Una vez concluida esta etapa, el modelo que se obtiene será capaz de obtener respuestas para conjuntos de datos nuevos.

Pero debemos tener claro que el *Machine Learning* no es siempre una buena solución. En ocasiones un programa convencional, sin aprendizaje basado en datos, podrá resolver el problema de forma eficiente. Por eso hay que tener claro en qué casos es útil desarrollar un programa basado en Aprendizaje Automático y en qué ocasiones es mejor utilizar programación convencional. Por ejemplo, en las situaciones en las que no se pueden codificar las reglas que determinan la solución buscada o aquellas en las que se presentan problemas de complejidad, son muy eficaces las soluciones basadas en algoritmos de *Machine Learning*. Por el contrario, si debemos determinar un valor y es posible hallarlo mediante sencillos cálculos o reglas lo ideal sería utilizar algoritmos tradicionales.

Podemos diferenciar principalmente dos categorías para clasificar los algoritmos de *Machine Learning*:

- **Aprendizaje supervisado**. Los algoritmos se entrenan con datos de ejemplo etiquetados, esto quiere decir que conocemos los resultados para esos ejemplos. Es decir, conocemos los datos de entrada, y la salida correcta que debería devolver el algoritmo para esos datos. El objetivo de este tipo de algoritmos es predecir la respuesta correcta para nuevos datos desconocidos. Algunas de las técnicas de aprendizaje supervisado utilizadas actualmente son las regresiones lineales, los árboles de decisión, las máquinas de vectores de soporte o las redes neuronales.

- **Aprendizaje no supervisado**. En este caso los datos de ejemplo con los que se entrena el algoritmo no están etiquetados, es decir, no sabemos cuál es su resultado. El objetivo es entonces encontrar asociaciones o patrones entre los datos según sus similitudes. Como ejemplo de técnicas de aprendizaje supervisado podemos diferenciar dos grupos: las orientadas al *clustering* (búsqueda de agrupaciones en los datos), que engloban algoritmos como k-medias, k-medianas o mapas autoorganizados, y aquellas orientadas a la reducción de la dimensionalidad, como el análisis de componentes principales.

3.2.2. Redes Neuronales

Los seres humanos somos capaces de resolver ciertos problemas de detección o clasificación de manera casi instantánea. Si por ejemplo vemos un nuevo modelo de coche, aún siendo la primera

vez que lo observamos seremos capaces de afirmar que efectivamente se trata de un coche. Esta capacidad, la de poder extraer y reconocer patrones en conjuntos de datos para, posteriormente, generalizar y realizar predicciones, sería tremendamente útil para los ordenadores a la hora de resolver ciertos problemas. Es por ello que surgen las Redes Neuronales como un modelo computacional que trata de imitar esos procesos biológicos.

Una Red Neuronal Artificial se define como [32][10] un modelo matemático que trata de imitar el comportamiento de las neuronas y la estructura del cerebro. Esta rama de la Inteligencia Artificial se enmarca dentro de los métodos de resolución del *Machine Learning*.

Comencemos describiendo las unidades básicas del sistema, las neuronas artificiales, que como es de esperar, tratan de imitar formalmente a las neuronas biológicas (ver Figura 3.5). En un cerebro humano las neuronas se encuentran interconectadas entre sí en redes muy complejas. Destacan las siguientes partes:

- **Dendritas:** ramificaciones que reciben la información.
- **Cuerpo celular:** que contiene el núcleo y procesa la información.
- **Axón:** prolongación del cuerpo celular que transporta la información procesada a otras neuronas.

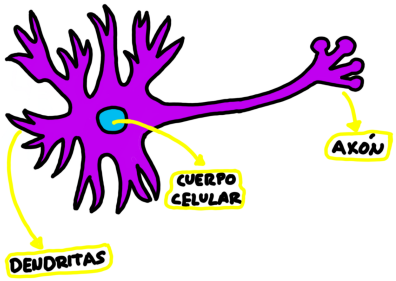


Figura 3.5: Estructura de una neurona biológica

Uno de los primeros modelos desarrollados, el Perceptrón, utilizaba un tipo especial de neuronas artificiales. Veamos de qué partes se componen sus neuronas (ver Figura 3.6):

- **Un conjunto de entradas** $x_1, x_2, \ldots, x_n$. Reciben información de otra neurona o del exterior. Se corresponderían con las dendritas en una neurona biológica.
- **Un conjunto de pesos** $w_1, w_2, \ldots, w_n$. Ponderan las entradas una a una, es decir, w_1 ponderará el valor de x_1, w_2 el de x_2 y así sucesivamente.
- **Un sesgo o *bias*** w_0. Controla lo predispuesta que está la neurona a devolver un valor u otro. Esto quiere decir que si tenemos un sesgo alto requeriremos que las entradas y los pesos sean altos para contrarrestarlo.
- **Regla de propagación** $\sum x_i w_i$ - w_0. Imita el cuerpo de la neurona. Calcula la diferencia entre la suma de los valores de entrada ponderados por el peso de cada dendrita y el sesgo.
- **Función de activación escalón**. Simula el axón y la conexión sináptica con la siguiente neurona. Limita la amplitud de la salida de una neurona devolviendo 0 o 1 según si el resultado de la función sumatorio es negativo o positivo.

- **Salida** y. Es el valor devuelto por la función de activación.

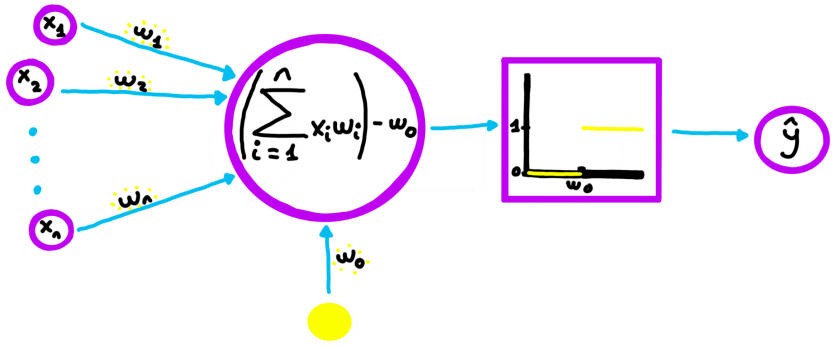


Figura 3.6: Estructura del Perceptrón

Si quisiéramos generalizar las partes del Perceptrón a las de una neurona artificial general solo hay que hacer dos cambios:

- La regla de propagación puede ser otra distinta de la función sumatorio, aunque no es lo común.

- La función de activación puede ser distinta de la escalón. En la actualidad se utilizan muchas otras (ver Figura 3.7) como la sigmoidea, que tiene la ventaja de ser diferenciable. También es común usar tangentes hiperbólicas o la función lineal rectificada conocida como ReLU.

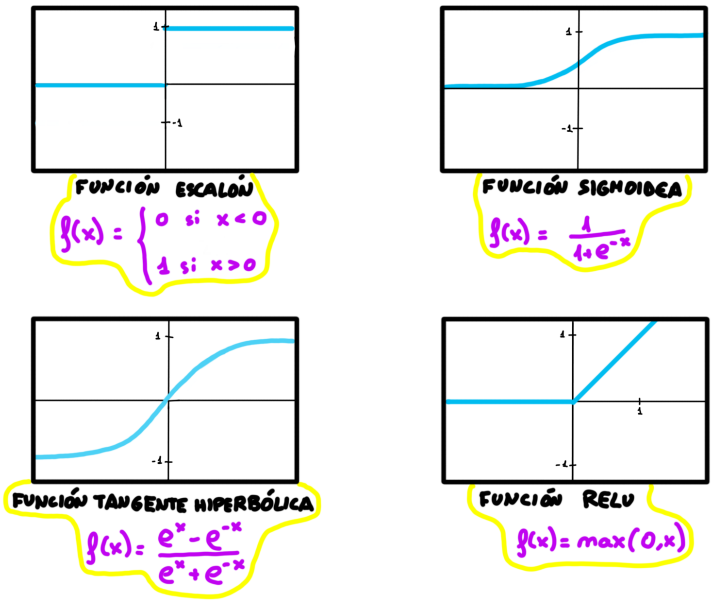


Figura 3.7: Representación de las principales funciones de activación

Una vez definida la unidad básica de estos sistemas, definiremos una red neuronal artificial como un conjunto de neuronas artificiales que se distribuyen en diferentes capas interconectadas entre sí. De esta forma podemos representar más información y más compleja.

Diferenciamos diferentes tipos de capas (ver Figura 3.8):

- **Capa de entrada**. Es la primera de las capas. Es la única que no recibe sus entradas de otra capa, sino que lo hace desde el exterior del sistema.
- **Capa de salida**. Es la última capa. Envía sus salidas al exterior del sistema.
- **Capas ocultas**. Son las capas intermedias. Reciben sus entradas de la capa inmediatamente anterior y envían sus salidas a la siguiente.

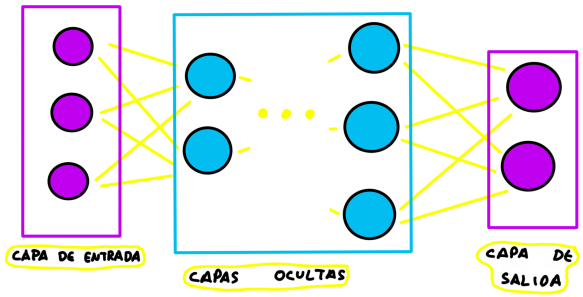


Figura 3.8: Capas de una Red Neuronal

Una red neuronal artificial representa entonces una función f($\mathbf{x}$)=$\mathbf{y}$ que transforma las entradas $\mathbf{x}$ en salidas $\mathbf{y}$. El resultado que devuelve la función depende entonces de los pesos asociados a las conexiones, los w_i, y del sesgo w_0. Esos valores se determinan a través de un proceso de aprendizaje. A esto se le conoce como entrenamiento de la red neuronal.

El entrenamiento de la red consiste, por tanto, en encontrar con qué pesos w_1, w_2, ..., w_n y sesgo w_0 de cada una de las neuronas que forman la red conseguimos que las salidas obtenidas coincidan con aquellas que se esperan, ya que se utilizan datos etiquetados.

Para ello se inicializan esas variables y se prueba a clasificar los datos de ejemplo. Se comparan los resultados obtenidos con los esperados (el valor de las etiquetas) y se calcula el error cometido mediante una función de costo determinada. Es habitual utilizar, por ejemplo, el error cuadrático medio.

Una vez calculado el error se utilizan algoritmos optimizadores como el Descenso del gradiente [60] (que necesita funciones diferenciables) para actualizar los parámetros w_0, w_1, ..., w_n de forma que se minimice la función de costo.

Este proceso se repetiría iterativamente hasta alcanzar una condición de parada previamente establecida. Este proceso se conoce como algoritmo de retropropagación y cada una de las iteraciones del mismo se denomina época. Es importante destacar que no es necesario que la red clasifique correctamente todos los ejemplos de entrenamiento para detener las iteraciones. Es más, esto no suele ser lo deseable ya que entonces suele producirse sobreajuste (*overfitting*), es decir, el modelo se centra demasiado en los datos de ejemplo y no generaliza correctamente sobre los nuevos datos que son los que interesa clasificar verdaderamente.

3.2.3. *Deep Learning*

El *Deep Learning* o Aprendizaje Profundo es [11][4] un subcampo del *Machine Learning* especializado en el aprendizaje de patrones complejos en conjuntos de datos mediante el uso de redes neuronales profundas como pueden ser las Redes Neuronales Convolucionales. De esta forma, con tiempos de computación lo suficientemente grandes, se pueden crear modelos que extraen eficientemente información útil en grandes volúmenes de datos.

En este tipo de algoritmos existe una jerarquía de capas que extraen el conocimiento de los datos de forma progresiva: se aprende algo sencillo en la primera capa y se envía esa información a la siguiente capa, donde se combina con más información y se obtiene algo un poco más complejo y abstracto. Esto se envía a la siguiente capa y así sucesivamente.

En la actualidad el *Deep Learning* está teniendo un alto impacto en diversos campos como en visión computacional, reconocimiento del habla o en procesamiento del lenguaje natural. Destaca también en el sector de la seguridad, en medicina o en controles de calidad.

Para generar todas esas soluciones se recurre al uso de Redes Neuronales Profundas (*Deep Neural Networks*). En el ámbito de la visión computacional destacan dos, las Redes Neuronales Convolucionales y las Redes Neuronales Recurrentes.

Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN) [53] son un tipo de redes neuronales artificiales que tratan de imitar las neuronas de la corteza visual primaria en un cerebro humano. Es una variación del modelo de perceptrón multicapa y en lugar de trabajar sobre vectores lo hace sobre volúmenes. Mientras que las primeras capas son capaces de detectar curvas o líneas, las capas más profundas pueden ser capaces de detectar siluetas o rostros. De esta forma se convierten en un recurso muy efectivo a la hora de resolver problemas de visión artificial como pueden ser problemas de clasificación o segmentación.

Trabajan de manera jerárquica, de tal manera que las primeras capas identifican elementos básicos que se van combinando para formar objetos en las capas más profundas.

Veamos cómo funciona una red neuronal convolucional. Existen principalmente cuatro tipos de capas (ver Figura 3.9): capa de Convolución, capa de *Pooling*, capa *Fully Connected* y capa *Softmax*. Pero existe una etapa previa de preprocesamiento en la que se normalizan los valores de la matriz. Para ello se divide cada valor de pixel por 255, de forma que se obtienen ahora valores entre 0 y 1.

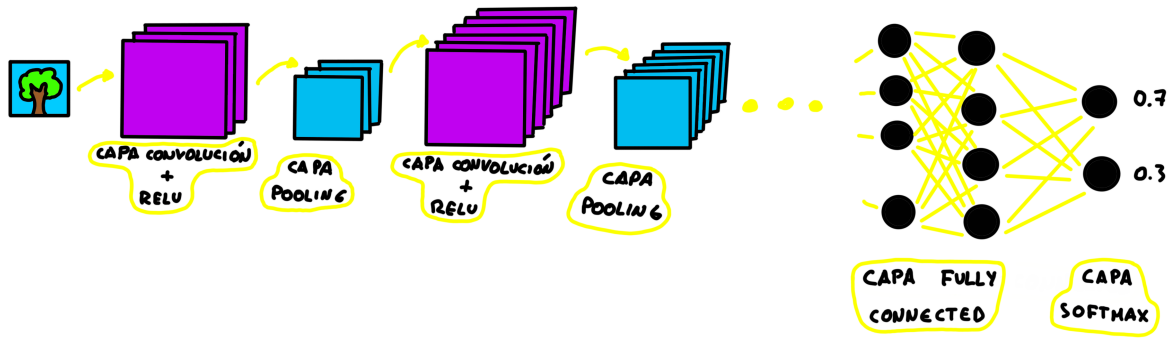


Figura 3.9: Estructura de una CNN

Partiremos de una matriz de pixels MxN ya que, en el caso de tener una imagen a color, de la forma 3xMxN, se tratará cada uno de los canales por separado, es decir, tendremos tres matrices MxN.

- **Capa de Convolución**. Esta capa se encarga de extraer las características de la imagen. Para ello se le aplica un filtro o *kernel*, que es otra matriz que pondera los valores de la matriz de la imagen. Se obtiene así otra matriz denominada Mapa de características. Pero, en general, no se aplica un solo filtro, sino que suelen aplicarse varios de forma paralela para obtener varios mapas diferentes (ver Figura 3.10).

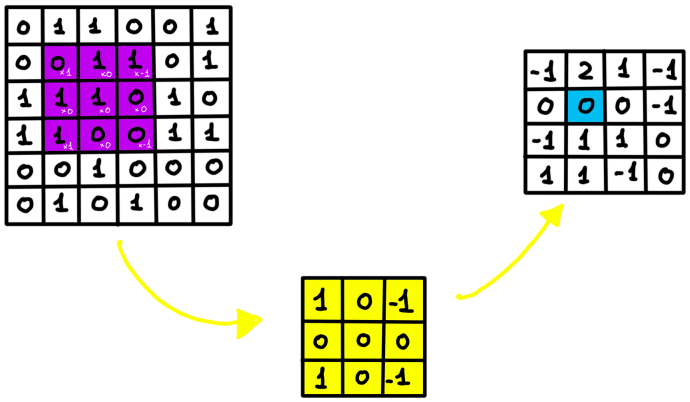


Figura 3.10: Ejemplo aplicación capa de Convolución con filtro 3x3

Tanto el número de filtros que se aplican, como el tamaño del filtro o el número de *pixels* que se avanzan al mover el filtro pueden configurarse a la hora de aplicar la convolución.

Después de aplicar una convolución es necesario aplicar una función de activación ReLU. Esta es una función de calculo simple ya que se define como $f(x) = max\{0, x\}$. Es decir, los valores negativos se transforman en 0 y los positivos se mantienen igual (ver Figura 3.11).

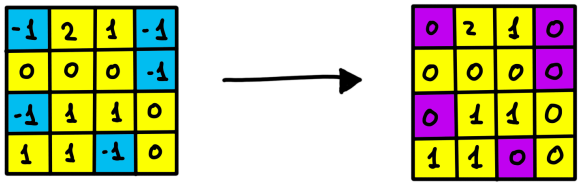


Figura 3.11: Ejemplo aplicación función de activación ReLU

- **Capa de *Pooling***. Reduce el tamaño de la matriz preservando las características más importantes que detectaron los filtros. Esto se hace para que la red entrene más rápido. Consiste en aplicar un filtro de *pooling* sobre la salida de la capa anterior. Existen varios tipos de filtros, como el *Average Pooling*, que selecciona el valor medio, o el *Max Pooling*, que es el más usado, y que selecciona el valor máximo (ver Figura 3.12).

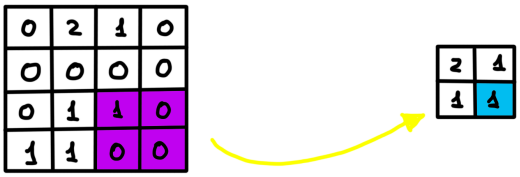


Figura 3.12: Ejemplo aplicación *pooling* con filtro *Max Pooling*

- **Capa *Fully Connected***. Una vez repetidas las capas 1, 2 y 3 iterativamente, el proceso concluye con la capa *Fully Connected*. La finalidad de esta última es transformar la matriz tridimensional de salida del citado proceso en un vector unidimensional combinando las características extraídas por todos los filtros.

- **Capa *Softmax***. Finalmente se aplica una capa *Softmax* que trasforma el vector obtenido en la capa *Fully Connected* en otro que contiene la probabilidad de pertenencia de la imagen a cada una de las categorías posibles.

Redes Neuronales Recurrentes

En el caso del procesamiento automático de vídeos suelen utilizarse, además de redes convolucionales, redes neuronales recurrentes (RNN) [20] combinadas con las anteriores. Esto es necesario porque una red convolucional permite procesar una sola imagen a la vez y en el caso de un vídeo tendría que procesar una secuencia de ellas. Por el contrario, una red recurrente tiene la capacidad de procesar todo tipo de secuencias como audios o vídeos que mantienen una cierta correlación entre cada uno de sus elementos. Por ejemplo, si escuchamos una frase y queremos comprender su contenido no es suficiente con leer cada palabra de forma individual, debemos situarlas en el contexto de la frase para entender su significado completo. Lo mismo ocurre en el caso del vídeo. Además, no podemos saber con antelación la duración de las secuencias. Las redes recurrentes resuelven ambos problemas.

Para ello se basan en el concepto de recurrencia: para generar la salida se tienen en cuenta tanto la entrada actual, como la salida de la iteración anterior. De esta forma se establecen conexiones dentro de la misma capa o incluso dentro de la misma neurona, es decir, la salida de una neurona puede ser la entrada de esa misma neurona en la siguiente iteración (ver Figura 3.13).

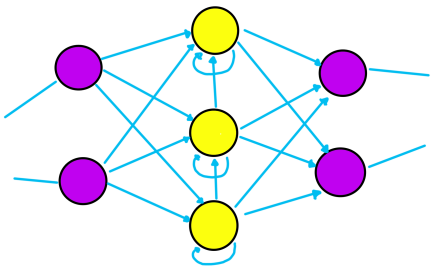


Figura 3.13: Ejemplo de Red Neuronal Recurrente

Así se consigue que la red pueda tener una especie de memoria que mantiene información sobre las iteraciones anteriores para resolver problemas de secuencias de datos. Además, las entradas y salidas pueden combinarse en diferentes formas según cuál sea la finalidad de su uso como puede verse en la Figura 3.14.

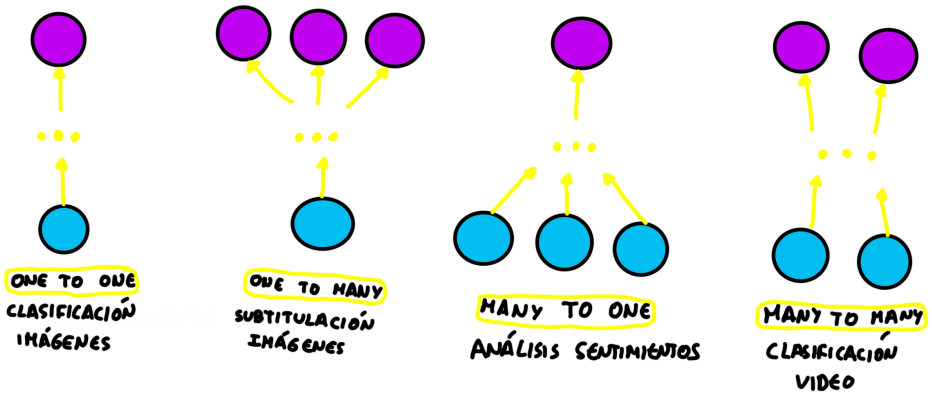


Figura 3.14: Combinaciones de entradas y salidas en una Red Neuronal Recurrente

Para entrenar redes recurrentes lo más común es utilizar el algoritmo *Backpropagation Through Time* (BPTT), que es una variante extendida del algoritmo de retropropagación visto en redes convolucionales. La idea en la que se basa dicho algoritmo es que una Red Recurrente puede visualizarse también como una sucesión de Redes Neuronales no recurrentes que representan a la red original en diferentes instantes de tiempo [57]. Este diagrama conocido como forma "desenrollada"de la red (ver Figura 3.15), facilita la comprensión de su funcionamiento.

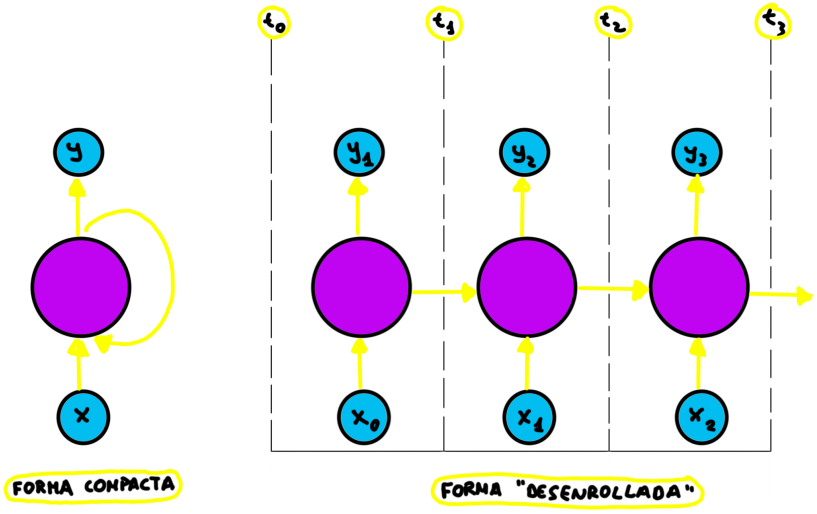


Figura 3.15: Diagrama de una Red Recurrente en su forma compacta y "desenrollada"

Generalización del procesamiento de imágenes a vídeo

Veamos ahora cómo podemos generalizar el problema de procesamiento de imágenes con redes convolucionales al procesamiento de vídeo utilizando *Deep Learning* [78]. Cuando se aplica una convolución sobre canales de dos dimensiones, es decir, imágenes tanto en blanco y negro como a color, se dice que se está aplicando una convolución 2D. En estos casos como ya vimos, los filtros tienen también dos dimensiones.

En el caso en que consideremos datos en forma de vídeo poseeremos una dimensión más: el tiempo. Como ya vimos, los archivos se almacenarán en arrays de la forma 3xFxMxN. Poseemos, por tanto, arrays en cuatro dimensiones, de forma que los filtros que se le apliquen tendrán tres dimensiones. Esta convolución se denomina convolución 3D y los filtros se mueven de manera análoga a la vista en el apartado de Redes Convolucionales. Se consideran por tanto las características espaciales de cada fotograma pero también las temporales entre fotogramas consecutivos.

Esta no es la única forma de procesar los datos en forma de vídeo, pero si es la primera forma en que se hizo utilizando *Deep Learning* y se ve fácilmente cómo se generaliza el tratamiento de los datos para el caso de vídeos.

Otra diferencia en cuanto al procesamiento de vídeo es la posibilidad de no disponer del archivo completo al inicio, es decir, que procesemos un vídeo en *streaming* [23] [67]. La idea es entonces procesar los datos de manera continua usando flujos de datos infinitos, lo que se conoce como *stream processing*. Además, en estos casos no hay picos de carga (grandes cantidades de datos a tratar de forma simultánea), ya que los diferentes *frames* se recibirán de forma continua (a diferencia de otros flujos de datos de diferente naturaleza).

En el *stream processing* se utilizan una serie de técnicas especiales que permiten acelerar la velocidad a la que se extrae valor de los datos. Una de ellas es el enriquecimiento del *dataset*, que consiste en añadir los nuevos datos al conjunto ya almacenado en tiempo real.

3.2.4. Librerías

Veremos ahora cuales son las principales librerías utilizadas en proyectos de visión computacional en *Python*. Destacaremos las tres más populares.

- **Icevision**. Es una librería creada por Airctic [38] especializada en técnicas de visión computacional, en concreto de detección de objetos, aunque también de segmentación y clasificación. Facilita tanto la preparación de los datos como la implementación del modelo, de forma que los usuarios solo deben centrarse en la resolución del problema mediante detección de objetos. Posee modelos preentrenados que utilizan TorchVision, YOLOv5 o PyTorch y permite el acceso a diferentes modelos de redes neuronales. Además, ejecuta entrenamientos multitarea para mejorar la eficiencia del proceso.

  Está disponible para su uso en sistemas Mac OS y Linux.

- **Detectron2**. Es una librería [26] desarrollada por Facebook AI que surge como una segunda versión de Detectron e implementando avanzados algoritmos de detección de objetos. Está construido con PyTorch y es compatible con Linux y con Mac OS. Incluye además nuevas funciones como segmentación panóptica, que consiste en una combinación entre segmentación por instancias y segmentación semántica. Se asocia así a cada pixel con dos

valores: una clase a la que pertenece (segmentación semántica) y un número de instancia (segmentación por instancias) (ver Figura 3.16).

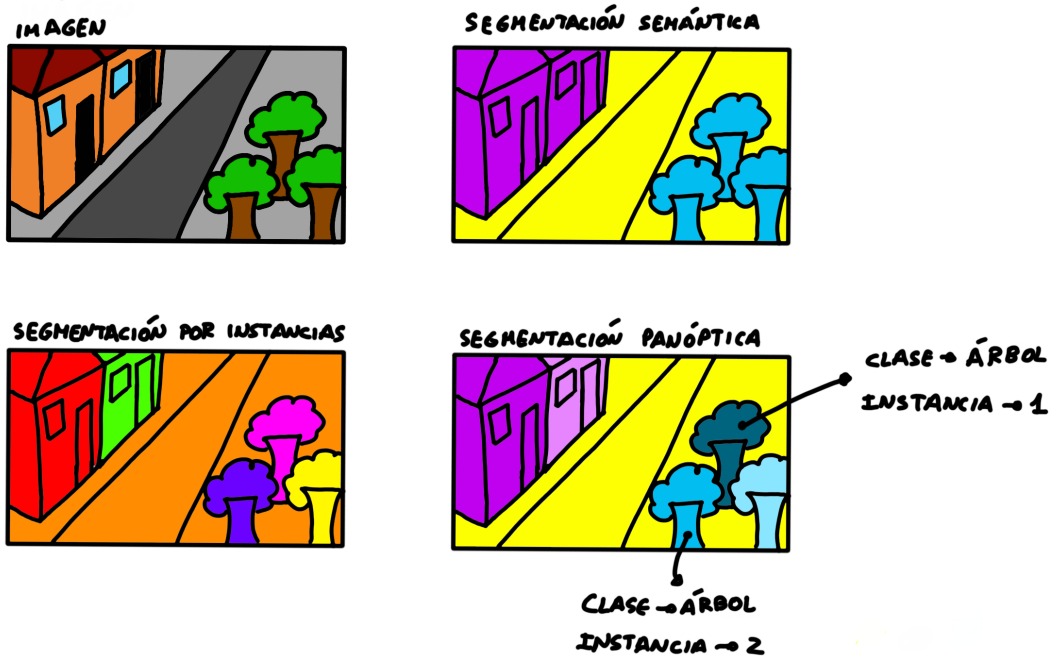


Figura 3.16: Segmentación panóptica

- **OpenCV**. Es una biblioteca [9] libre y multiplataforma lanzada en 1999 por Intel, orientada a *Machine Learning* y, en concreto, en visión artificial. Es muy popular en el desarrollo de algoritmos relacionados con el sector de la seguridad y control de calidad ya que está especialmente orientada a la clasificación y detección de rostros y objetos. Posee más de 500 funciones enfocadas a esas tareas.

3.3. Estado del arte

Se estudiarán a continuación algunas de las soluciones propuestas para la resolución de problemas en el ámbito del procesamiento de imágenes y vídeos.

3.3.1. Descripción de trabajos relacionados

Los primeros métodos utilizados en el procesamiento automático de imágenes antes de la aparición de arquitecturas *Deep learning* solo eran capaces de detectar características muy superficiales. Requerían además la definición de esas características por parte de un experto previamente a la ejecución del algoritmo. De esta forma no se podían alcanzar los detalles más profundos y se perdía gran parte de la comprensión de la fotografía.

Es por ello que las arquitecturas basadas en *Deep Learning* supusieron una revolución en el campo de la visión computacional, porque permitían un aprendizaje autónomo sin necesidad de predefinir cómo debía realizarse la extracción de características.

Se analizarán a continuación las principales arquitecturas de *Deep Learning* utilizadas en la actualidad para cada tipo de problema concreto.

Clasificación

Comencemos con las propuestas orientadas a la clasificación [46].

- **ResNet**.

  Esta arquitectura, denominada red neuronal residual (ResNet) fue presentada por Microsoft en el año 2015 [35]. Surge con la idea de solventar el problema de optimización de redes muy profundas. El problema existente consistía en que, al construir redes muy profundas, el error obtenido aumentaba en lugar de disminuir. Esto era contraproducente, ya que la idea de aumentar el número de capas era, precisamente, obtener una mayor precisión. La arquitectura ResNet trata de evitar esta degradación del error introduciendo el concepto de bloque residual como componente básico del sistema (ver Figura 3.17). Esto consiste en combinar la salida de la red con la entrada. De esta forma el error de entrenamiento al aumentar el número de capas disminuye en lugar de aumentar.

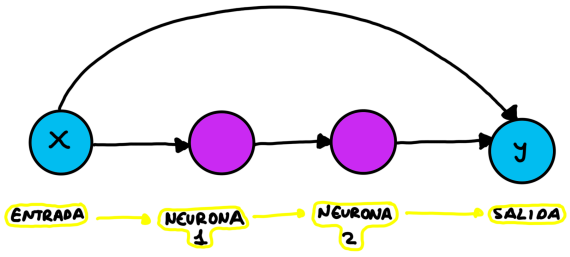


Figura 3.17: Uso de bloque residual en una arquitectura ResNet

  Se pueden encontrar versiones de esta arquitectura con un número variable de capas, teniendo la más reducida 34 y la más extensa 152.

- **SeNet**.

  Esta arquitectura fue presentada por la universidad de Oxford en 2017 [37]. Al igual que ResNet, se basa en la introducción de bloques, en este caso para mejorar la interdependencia entre capas. De esta forma, se consigue escalar los canales asignándoles más o menos importancia. Dichos bloques pueden ser de tipo residual (los mismos que ResNet) o de tipo *Inception*. Estos últimos permiten realizar múltiples convoluciones y agrupaciones en paralelo, de forma que se combinen posteriormente en capas más profundas.

Detección de objetos

Continuaremos ahora con las arquitecturas orientadas a la detección de objetos [48].

- **R-CNN**.

  Surge en el año 2014 [34] como un sistema de detección de objetos o personas. Es una arquitectura basada en regiones. Esto quiere decir que en un primer paso selecciona zonas

candidatas a poseer aquello que se quiere detectar y, posteriormente analiza únicamente esas regiones en lugar de la imagen completa. Este tipo de soluciones tratan de mejorar la precisión de la red neuronal. Sin embargo, el coste computacional de entrenamiento aumenta notablemente al realizarse este proceso en dos etapas.

Para tratar de mitigar este problema, se desarrollaron dos versiones posteriormente: Fast R-CNN [33] y Faster R-CNN [59]. La primera introduce el uso de una capa de agrupación de regiones de interés, denominada RoI Pooling. Por otro lado, Faster R-CNN, presenta como novedad una red denominada Region Proposal Network (RPN) encargada de generar un mapa de características y encontrar cuáles son las regiones de interés de la imagen en la primera etapa mencionada.

- **You Only Look Once (YOLO)** [63] [58]

  Es una arquitectura de *Deep Learning* orientada a la detección de objetos en tiempo real. La primera versión fue lanzada en el año 2015 pero posteriormente se presentaron YOLOv2, YOLOv3, YOLOv4 y YOLOv5.

  Consiste en una red convolucional que es capaz de predecir, recorriendo una sola vez la imagen, cuadros delimitadores de posibles objetos detectados y la probabilidad de pertenencia a cada una de las clases especificadas. La red convolucional trabaja con una imagen dividida en celdas. Utiliza las características extraídas de la imagen al completo para delimitar en qué casilla o casillas se encuentran los objetos detectados. En el caso de que el objeto se extienda por varias casillas, la red ejecutará una serie de algoritmos matemáticos para elegir en qué casilla es más probable que se encuentre el centro geométrico del objeto. Es en esa casilla en la que se asignará la probabilidad de pertenencia del objeto detectado a una u otra clase. Esto se indica en forma de vector con los siguientes elementos (ver Figura 3.18):

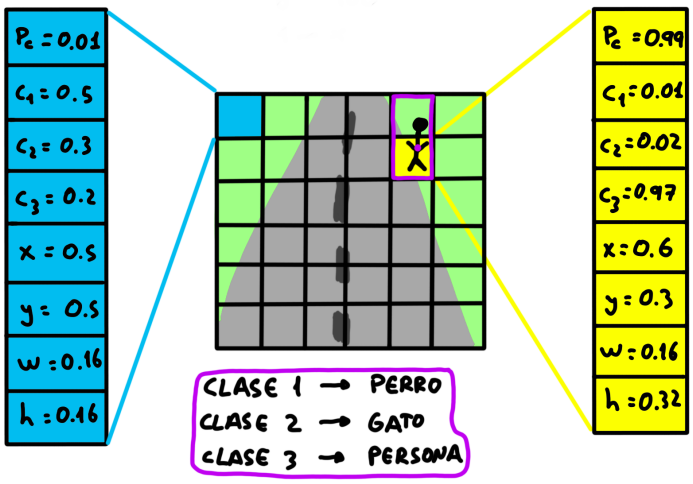


Figura 3.18: Ejemplo de detección con la arquitectura YOLO

- Probabilidad de que la celda posea un objeto, P_c.
- Probabilidad de pertenencia a cada una de las clases predefinidas, c_1, c_2, c_3, ...

- Coordenadas relativas del centro del objeto con respecto a la celda, x e y.
- Altura y anchura en celdas del objeto detectado w y h. Se expresan en porcentaje con respecto al tamaño total de la imagen.

- **End to End Object Detection with Transformers (DETR)**

  Lanzada en 2020 por Facebook AI [15], es la más novedosa de las arquitecturas presentadas en este documento. Se basa en el uso de *transformers* [14], que son arquitecturas capaces de obtener relaciones y dependencias en las entradas, bien sean imágenes o vídeos, mediante *encoders* y *decoders* y sin utilizar recurrencia.

  DETR se define como arquitectura híbrida ya que utiliza una CNN y posteriormente *transformers*. En primer lugar se realiza una extracción de características de bajo nivel y, posteriormente, se utiliza el mapa de características obtenido como entrada para los *transformers*, que se encargarán de establecer las dependencias y relaciones de alto nivel.

- **Single Shot Multibox Detector (SSD)** [65]

  Se presenta en 2016 con el objetivo de solventar el problema de detección de pequeños objetos. Se basa en el uso de la red Region Proposal Network (RPN) ya mencionada en la arquitectura Faster R-CNN. Sin embargo, en SSD, la RPN calcula las coordenadas del objeto detectado y la probabilidad de pertenencia a una u otra clase de forma simultánea. Además, este modelo utiliza una red VGG16 adaptada, en la que se sustituyen capas *Fully Connected* por capas convolucionales que permiten la extracción de mapas de características en diferentes escalas, facilitando así la detección de objetos de menor tamaño.

- **RetinaNet** [48]

  Esta arquitectura fue lanzada en 2017 con la finalidad de detectar también pequeños objetos que pudieran aparecer en la imagen.

  Como principal novedad añade el concepto de *Feature Pyramid Network* (FPN) o Red Piramidal de Características que permite detectar objetos a diferentes escalas. En lugar de generar la misma imagen a diferentes escalas y obtener de cada una de ellas su mapa de características, lo cual es muy costoso en términos computacionales, crea una pirámide de características. Sobre esa pirámide es sobre la que se trabaja para detectar los objetos.

  Por otro lado, la asignación de probabilidad de pertenencia a una u otra clase se produce de manera simultánea a la detección del objeto, por lo que no se trata de una arquitectura basada en regiones.

Segmentación

Finalmente veremos las principales arquitecturas utilizadas en el ámbito de la segmentación de imágenes o vídeos.

- **U-Net** [18]

  Es una red de segmentación semántica presentada en el año 2015. Recibió ese nombre por la forma de U que adquiere su estructura, ya que posee una parte descendente y posteriormente otra ascendente.

La primera parte, denominada codificador, está formada por una secuencia de capas de convolución y *pooling* que tratan de contraer la información de la imagen disminuyendo iterativamente su dimensión. Posteriormente, en la zona ascendente, denominada decodificador, se hace una expansión del mapa de características obtenido. Para ello se aplican las convoluciones de manera inversa de forma que se incrementa la dimensión del citado mapa.

La arquitectura posee cuatro bloques de codificador y cuatro bloques de decodificador, de forma que es simétrica.

- **Seg-Net** [61]

  Se presenta en el año 2016 con la finalidad de implementarse en sistemas de conducción autónoma, aunque actualmente se utiliza en todo tipo de tareas de segmentación semántica. Se especializa en la segmentación semántica y, en concreto, en la detección de bordes y contornos.

  Al igual que la red U-Net se divide en dos partes, el codificador y el decodificador. Además, al final presenta una capa que clasifica cada uno de los pixels de la imagen en las diferentes categorías posibles. La particularidad que presenta es que el codificador se corresponde a una red convolucional VGG16 que elimina las capas *Fully Connected*. Los índices resultantes generados por esta red se pasan al decodificador reduciéndose así el número de parámetros que se deben entrenar.

- **Mask R-CNN**

  Es una arquitectura especializada en segmentación de instancias lanzada en el año 2018. Se basa en la arquitectura Faster R-CNN mencionada en el apartado de detección de objetos.

  Mantiene el procesamiento en dos etapas que presenta la familia R-CNN, pero reemplaza la capa de agrupación RoI Pooling por un módulo de alineación, denominado RoI Align. De esta forma, además de localizar y delimitar los objetos deseados, se predicen paralelamente las máscaras de los objetos, obteniendose así la segmentación de la imagen.

3.3.2. Dimensiones comparativas

En primer lugar, definiremos las dimensiones comparativas que se utilizarán posteriormente para comparar las propuestas mencionadas. Diferenciaremos las métricas como *High Best* o *Low Best*. En el primer caso, cuanto más alto sea el valor asociado a la métrica mejor funcionará el sistema, mientras que si tenemos una dimensión comparativa de tipo *Low Best* interesará que el valor obtenido sea lo más pequeño posible. Veamos las métricas elegidas para cada tipo de problema ya que, para cada uno de ellos se hará una comparación independiente porque no tendría mucho sentido, por ejemplo, comparar la precisión de un algoritmo de clasificación con otro de segmentación.

- **Clasificación**.

  - Frames analizados por segundo (FPS). Cantidad de imágenes que el modelo puede procesar por segundo. Determina la capacidad de la arquitectura para procesar los datos en tiempo real. Es una métrica *High Best*.

- Exactitud o *accuracy*. Consiste en el porcentaje total de elementos clasificados de forma correcta. Si analizamos una matriz de confusión (ver Figura 3.19), que muestra el desempeño de un determinado algoritmo en un problema de aprendizaje supervisado, nos encontraremos con los siguientes valores:

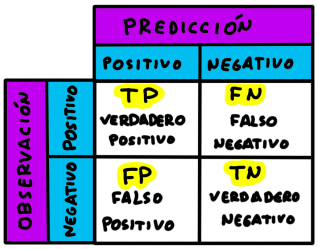


Figura 3.19: Matriz de confusión

  Definimos $accuracy = \frac{TP+TN}{TP+TN+FP+FN}$. Es *High Best*.

- Número de parámetros a entrenar. Determinarán en cierta medida el coste computacional que suponen a la hora de entrenar las redes neuronales. Es un parámetro *Low Best*.

- **Detección**

  - *Mean Average Precision* (mAP). Mide el rendimiento de los algoritmos de detección. Es la métrica más usual en este tipo de problemas. Se calcula a partir de la precisión media (AP), que es la media de la precisión de todos los elementos de una categoría. Si denominamos AP_i a la precisión media de detección en la categoría i, se define mAP$= \frac{1}{n} \cdot \sum_{i=1}^{n} AP_i$. Es un parámetro *High Best*.
  - *Mean Average Recall* (mAR). De manera análoga a mAP, definimos mAR. Veamos, en primer lugar, que es el *recall*. El *recall* mide la cantidad de datos que el modelo es capaz de identificar y se define como $recall = \frac{TP}{TP+FN}$.

    Por tanto, el *recall* medio (AR), se define como la media de todos los *recalls* de una categoría. Denominando AR_i al AR de la catogoría i, definimos $mAR = \frac{1}{n} \cdot \sum_{i=1}^{n} AR_i$

    Este parámetro es también *High Best*.

- **Segmentación**.

  - Número de parámetros a entrenar.
  - Exactitud o *accuracy*.
  - *F1-score*, que combina la precisión con el *recall*. Conocido cómo se define el *recall*, veamos ahora cómo se define la precisión, que mide la calidad del modelo:

    $precision = \frac{TP}{TP+FP}$

    Podemos definir ya el *F1-score*: $F1 = 2 \cdot \frac{precision \cdot recall}{precision+recall}$

    Este parámetro es *High Best*.

3.3.3. Discusión

Veamos a continuación cuáles son las mejores propuestas en cada uno de los diferentes tipos de soluciones según los parámetros analizados. Se mostrará una tabla resumen en la que se añade el año de presentación del algoritmo para analizar si existe una cierta evolución con el paso del tiempo.

- Problema de clasificación (ver Tabla 3.1). Los datos de precisión y número de parámetros aportados se han extraído de un estudio realizado sobre el *dataset ImageNet* [40]. Por otro lado, el número de frames por segundo pertenece a otro *benchmark* realizado sobre las principales arquitecturas de clasificación [8].

	Año	FPS	Accuracy (%)	Nº parámetros
ResNet50	2015	1000	80,4	25M
SeNet152	2017	1000	82,2	66,6M

Tabla 3.1: Discusión de la tarea de clasificación.

  En este último caso destaca la arquitectura ResNet, ya que posee valores similares a la red SeNet en términos de frames por segundo y de *accuracy*, pero requiere el entrenamiento de menos de la mitad de parámetros.

- Problema de detección (ver Tabla 3.2). Compararemos tres de las arquitecturas presentadas: YOLOv4, Faster R-CNN y DETR. Para medir la precisión de estos algoritmos nos basaremos en un estudio [77] planteado por la Tianjin University, que compara las arquitecturas en el ámbito de una ciudad inteligente, detectando vehículos, bicicletas, personas, ropa y accesorios mediante cámaras de vigilancia.

	Año	mAP	mAR
Faster R-CNN	2016	29,9	39,4
YOLOv4	2020	25,3	40,6
DETR	2020	0,294	24,1

Tabla 3.2: Discusión de la tarea de detección.

  No se aprecian diferencias significativas entre los algoritmos YOLOv4 y Faster R-CNN, pero sí se observa una clara diferencia con la arquitectura DETR, que presenta peores valores en ambas métricas, pero sobre todo en mAP.

- Problema de segmentación (ver Tabla 3.3). En este caso, compararemos las redes U-net y Seg-Net, y los datos de cada algoritmo se han extraído de un Trabajo Fin de Grado desarrollado [18] en la Universidad Politécnica de Madrid.

  En el caso de la segmentación se aprecia una clara mejora en el ámbito del rendimiento y la velocidad de entrenamiento de la red neuronal, ya que el número de parámetros a entrenar en el caso del Seg-Net se reduce casi 41 veces con respecto a la red U-Net. Además, esto no supone una disminución significativa en el resto de parámetros, que se mantienen parecidos en ambos casos.

	Año	Nº parámetros	Accuracy (%)	F1-score(%)
U-Net	2015	2,4M	91,66	67,7
Seg-Net	2016	59K	90,41	64,36

Tabla 3.3: Discusión de la tarea de segmentación.

Concluye así el estudio del estado del arte en el contexto de *Deep Learning* aplicado a visión computacional. Las arquitecturas presentadas son buenas candidatas a la hora de resolver los diferentes problemas que se planteen, y deberán tenerse en cuenta los requisitos concretos de cada caso para decantarse por una u otra. En cualquier caso, todas ellas ofrecen la ventaja de que son arquitecturas ya definidas y preentrenadas, por lo que con un pequeño entrenamiento y un *dataset* no demasiado grande pueden obtenerse resultados muy competitivos.

Parte II

Desarrollo de la propuesta y resultados

Capítulo 4

Cuadernos de aprendizaje

Se presentará en esta sección un pequeño análisis del caso de estudio trabajado en los cuadernos de aprendizaje, así como los objetivos concretos de cada uno de ellos. Pero antes introduciremos el concepto de *notebook*. Para ello debemos remontarnos al año 2014, año en el que se presentó el Proyecto Jupyter *jupyter* como un *software open-source*. Dentro de este se enmarcan los Jupyter Notebooks, que son aplicaciones web que nos permiten combinar texto con código en diferentes lenguajes. Pero no solo eso, ese código puede ejecutarse dentro de la propia aplicación de forma que también pueden visualizarse los resultados de sus ejecuciones.

Los *notebooks* tuvieron tanto éxito que otros equipos, como Google Research, desarrollaron servicios similares basados en Jupyter dando lugar, por ejemplo, a Google Colab *colab*, que es la herramienta que se utilizará para los cuadernos de aprendizaje. La ventaja que aporta Google es que proporciona una máquina virtual de Linux con un entorno en el que vienen preinstalados ciertos paquetes y librerías y, además, permite la ejecución de procesos en GPU para aligerar los entrenamientos.

4.1. Caso de estudio elegido

Como ya se mencionó en la introducción, el nacimiento de las primeras ciudades inteligentes supuso una mejora en la gestión de las mismas, y un aumento de la calidad de vida de sus habitantes. Una de las tecnologías claves en este proyecto es la visión computacional, ya que la principal fuente de información de los acontecimientos ocurridos en las calles de una determinada localidad son las cámaras de videovigilancia. Estos dispositivos permiten capturar en un mismo archivo diversos datos de interés. Por ejemplo, analizando un vídeo de una calle podríamos contar cuantas personas pasean cada instante de tiempo; o cuántos coches circulan por la calzada contigua a la acera; o si los peatones suelen ir acompañados de animales; o incluso si es habitual el uso de bicicletas por esas zonas. Este es el principal interés de los archivos multimedia, que permiten resolver varios problemas sin necesidad de acudir a más fuentes de información, a diferencia de otro tipo de sensores que suelen especializarse en algún dato o conjunto de datos en concreto. De esta forma, se obtiene también una disminución del coste de los dispositivos destinados a la captura de datos, ya que en lugar de colocar varios sensores para capturar distintos tipos de información, es suficiente con utilizar una única cámara, que cumplirá las diversas funciones requeridas y que ya no se centra únicamente en el ámbito de la seguridad.

Dado el importante desarrollo que están experimentando las ciudades inteligentes, los casos de

estudio presentados en esta memoria se centrarán en la aplicación de técnicas de visión computacional en *smart cities*. En concreto, se presentarán tres cuadernos de aprendizaje centrados respectivamente en problemas de clasificación, detección de objetos y segmentación.

4.2. *Datasets* seleccionados

Mostraremos las principales características de los conjuntos de datos utilizados en los *notebooks* didácticos.

4.2.1. The German Traffic Sign Recognition Benchmark

En el primer ejemplo trataremos de clasificar imágenes de señales de tráfico. El *dataset* que utilizaremos, The German Traffic Sign Recognition Benchmark (GTSRSB) [36] [31] [66], se creó en 2010 para utilizarse en un desafío propuesto en el International Joint Conference on Neural Networks (IJCNN).

Las imágenes fueron extraídas de un vídeo de unas 10 horas grabado desde un coche en diferentes carreteras alemanas. Del archivo original se separaron los diferentes *frames*, 25 por cada segundo de vídeo, y se llevó a cabo un proceso de selección. Esto fue necesario ya que en los tramos con señales de stop o ceda el paso, la disminución de la velocidad del coche que capturaba los datos suponía que se encontrasen un gran número de imágenes similares, mientras que en aquellas zonas en las que se permitía una velocidad mayor (100 o 120 km/h) se capturaban muy pocas imágenes. De esta forma se intentó balancear la cantidad de datos presentes en cada una de las clases. Esto supuso una reducción en el número de imágenes de unas 145.000 a aproximadamente 52.000.

Posteriormente, en cada imagen se seleccionaron solamente las zonas en las que aparecían señales de tráfico para simplificar el proceso de clasificación de las mismas (ver Figura 4.1). Sin embargo, las anotaciones que se aportan junto con el conjunto de datos especifican, además de la categoría de la imagen, el tamaño y localización de las señales dentro de la imagen original. Dichas anotaciones se aportan en un archivo CSV. En el caso de las imágenes de entrenamiento encontramos también las imágenes ya clasificadas en carpetas numeradas.

Figura 4.1: Ejemplo imagen original capturada frente a la zona con señal extraida

El dataset se compone, por tanto, de unas 52.000 imágenes de señales de tráfico en formato PNG divididas en dos carpetas: una de *train* y otra de *test*. Dichas imágenes se clasifican en 43 categorías según el tipo de señal que presente (ver Figura 4.2).

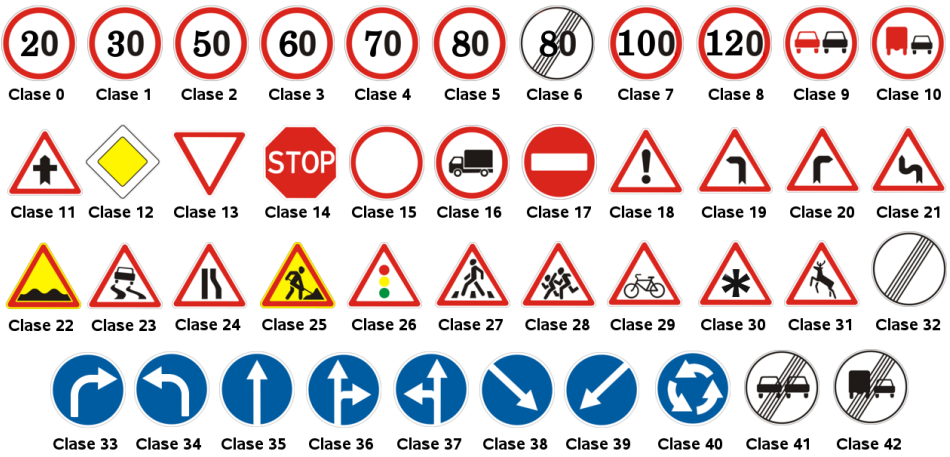


Figura 4.2: Clases existentes en GTSRSB

4.2.2. Penn-Fudan Database for Pedestrian Detection and Segmentation

El *dataset* elegido para el segundo ejemplo, detección de objetos, se compone de imágenes capturadas en los alrededores de las universidades de Pensilvania y Fudan [54]. Se creó en 2007 para llevar a cabo un estudio sobre detección de personas combinando reconocimiento y segmentación [50].

El objetivo es, por tanto, detectar a todas las personas que aparecen en cada imagen. Para ello se aportan un total de 170 imágenes en formato PNG, etiquetadas para detección de objetos (ver ejemplo en Figura 4.3) y también para segmentación de instancias. En este caso solo trataremos la tarea de detección de personas, por lo que solo comentaremos cómo vienen etiquetados los datos para dicho fin. Las etiquetas se aportan desde la página oficial en formato Pascal 1.00. Sin embargo, en el *notebook* se utilizarán en formato Pascal VOC (las etiquetas se aportan en este formato junto con el archivo IPYNB).

Figura 4.3: Ejemplo de imagen etiquetada para detección

Veamos brevemente qué datos aporta Pascal VOC. Mostremos una etiqueta del conjunto de datos. Escribimos los comentarios oportunos con # (aunque como es un xml debería hacerse con <! >).

```
<annotation>
    <folder>images</folder>  #Carpeta en la que se encuentra la etiqueta
    <filename>FudanPed00011.png</filename>  #Nombre del archivo
    <path>../images/FudanPed00011.png</path>  #Ruta del archivo
    <source>
        <database>The Penn-Fudan-Pedestrian Database</database>  #Dataset
    </source>
    <size>
        <width>459</width> #Ancho en pixels
        <height>420</height> #Alto en pixels
        <depth>3</depth> #Número de canales
    </size>
    <segmented>0</segmented> #No está segmentada
    <object>  #Propiedades del objeto
        <name>Pedestrian</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox> #Coordenadas de las esquinas que delimitan la bounding box
            <xmin>278</xmin>
            <ymin>112</ymin>
            <xmax>438</xmax>
            <ymax>394</ymax>
        </bndbox>
    </object>
</annotation>
```

4.2.3. Cambridge-driving Labeled Video Tiny Dataset

En el último cuaderno se abordará la tarea de detección semántica. Para ello se utilizará el *dataset* Camvid en una versión reducida de 100 imágenes (Camvid Tiny). Esta variante del conjunto de datos original se incluye en la librería Icevision, que es la que usaremos para la resolución del problema.

El *dataset* original, creado en 2008 [13] [47], se compone de unas 700 imágenes en formato PNG y con resolución 960 × 720 *pixels*, que fueron extraidas de 4 vídeos grabados desde un coche en las calles de Cambridge. Posteriormente se etiquetaron manualmente para la tarea de segmentación semántica diferenciando 32 categorías (ver Figura 4.4).

Dichas etiquetas se aportan como las máscaras de cada una de las imágenes (también en formato PNG). Es decir, se proporcionan imágenes en las que se asigna a cada pixel el valor asociado a la categoría a la que pertenece (en formato RGB). Por ejemplo, si asociamos el valor (255,0,0), que se corresponde con el color rojo, a la categoría coche, en las máscaras de las imágenes, los pixeles correspondientes a los coches serán de color rojo. Se obtienen así las máscaras que etiquetan los datos (ver Figura 4.5).

Void	Building	Wall	Tree	VegetationMisc	Fence
Sidewalk	ParkingBlock	Column_Pole	TrafficCone	Bridge	SignSymbol
Misc_Text	TrafficLight	Sky	Tunnel	Archway	Road
RoadShoulder	LaneMkgsDriv	LaneMkgsNonDriv	Animal	Pedestrian	Child
CartLuggagePram	Bicyclist	MotorcycleScooter	Car	SUVPickupTruck	Truck_Bus
Train	OtherMoving				

Figura 4.4: Asociación entre valores RGB y categorías

Figura 4.5: Ejemplo de una imagen y su máscara

4.3. Objetivos de aprendizaje

La idea principal de los *notebooks* es introducir al usuario al campo de la visión computacional, de manera que pueda ver y ejecutar sus primeros ejemplos con datos reales de manera guiada. Es por ello que los objetivos de los tres cuadernos siguen una misma línea, presentando todos ellos apartados y secciones similares. Se engloban, por tanto, los objetivos de aprendizaje de los diferentes *notebooks* en una misma lista que se presenta a continuación:

- Cuáles son las principales librerías utilizadas en visión computacional y cómo se cargan.
- Cómo cargar un dataset de imágenes etiquetadas desde el almacenamiento local o desde una librería.
- Cómo aplicar técnicas de preprocesamiento (*data augmentation*, redimensionamiento de los datos, normalización...) sobre las imágenes cargadas.
- Cómo visualizar las imágenes cargadas con y sin etiqueta.
- Cómo cargar un modelo preentrenado desde una librería.
- Cómo entrenar con nuestro conjunto de datos el modelo cargado.
- Cómo exportar el modelo con los pesos obtenidos tras el entrenamiento y cómo volver a cargarlo después.
- Cómo realizar predicciones sobre nuevas imágenes (inferencia).

4.4. Tecnologías utilizadas

Las librerías utilizadas como base son:

- **Fastai.**

  Es una librería de código abierto basada en Pytorch y diseñada para ser utilizada en problemas de *deep learning*. Incorpora diversas funciones que simplifican las tareas de carga de datos, entrenamiento y evaluación. Además, proporciona algunos modelos preentrenados de torchvision como alexnet, densenet o resnet, que permiten obtener modelos con buenos resultados empleando menos datos y tiempo.

- **Icevision.**

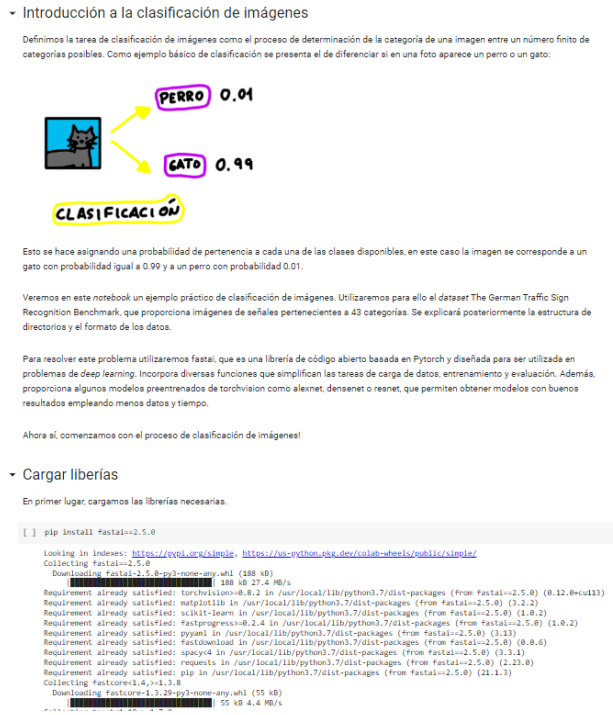
  Como se comentó en la sección 3.2.4., Icevision es una librería basada en Pytorch y especializada en las tareas de detección de objetos, segmentación semántica y segmentación de instancias. Simplifica las tareas de carga de datos, de entrenamiento del modelo y de su posterior uso para inferir información de nuevos datos. Además, permite parsear las anotaciones en varios formatos como COCO o Pascal VOC. Proporciona modelos preentrenados y también algunos *datasets* y permite trabajar con fastai y pytorch-lightning.

Además, también se utilizan las siguientes librerías:

- **sklearn**, también denominada scikit-learn, utilizada para el preprocesamiento de datos, aunque también es útil en la creación de modelos.
- **shutil**, que facilita tareas como copiar, eliminar o descomprimir archivos.
- **os**, que actúa como interfaz con el sistema operativo.
- **pandas**, utilizado para el análisis de datos.
- **matplotlib**, especializada en la visualización de gráficos e imágenes.

4.5. Muestra de los cuadernos

Mostraremos un par de capturas extraidas del *notebook* de clasificación. En estas puede apreciarse el carácter didáctico de los mismos, combinando introducciones teóricas, junto con comentarios y código ejecutable. De esta forma se pretende construir cuadernos autocontenidos. Por otro lado, en el Apéndice A, se aporta un breve manual de instalación de un entorno de Anaconda por si el usuario quisiera ejecutarlos en local.



Introducción a la clasificación de imágenes

Definimos la tarea de clasificación de imágenes como el proceso de determinación de la categoría de una imagen entre un número finito de categorías posibles. Como ejemplo básico de clasificación se presenta el de diferenciar si en una foto aparece un perro o un gato:

Esto se hace asignando una probabilidad de pertenencia a cada una de las clases disponibles, en este caso la imagen se corresponde a un gato con probabilidad igual a 0.99 y a un perro con probabilidad 0.01.

Veremos en este *notebook* un ejemplo práctico de clasificación de imágenes. Utilizaremos para ello el *dataset* The German Traffic Sign Recognition Benchmark, que proporciona imágenes de señales pertenecientes a 43 categorías. Se explicará posteriormente la estructura de directorios y el formato de los datos.

Para resolver este problema utilizaremos fastai, que es una librería de código abierto basada en Pytorch y diseñada para ser utilizada en problemas de *deep learning*. Incorpora diversas funciones que simplifican las tareas de carga de datos, entrenamiento y evaluación. Además, proporciona algunos modelos preentrenados de torchvision como alexnet, densenet o resnet, que permiten obtener modelos con buenos resultados empleando menos datos y tiempo.

Ahora sí, comenzamos con el proceso de clasificación de imágenes!

Cargar liberías

En primer lugar, cargamos las librerías necesarias.

```
[ ]  pip install fastai==2.5.0

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Collecting fastai==2.5.0
       Downloading fastai-2.5.0-py3-none-any.whl (188 kB)
          ▌                                    ▐ 188 kB 27.4 MB/s
     Requirement already satisfied: torchvision>=0.8.2 in /usr/local/lib/python3.7/dist-packages (from fastai==2.5.0) (0.12.0+cu113)
     Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from fastai==2.5.0) (3.2.2)
     Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages (from fastai==2.5.0) (1.0.2)
     Requirement already satisfied: fastprogress>=0.2.4 in /usr/local/lib/python3.7/dist-packages (from fastai==2.5.0) (1.0.2)
     Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (from fastai==2.5.0) (3.13)
     Requirement already satisfied: fastdownload in /usr/local/lib/python3.7/dist-packages (from fastai==2.5.0) (0.0.6)
     Requirement already satisfied: spacy<4 in /usr/local/lib/python3.7/dist-packages (from fastai==2.5.0) (3.3.1)
     Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from fastai==2.5.0) (2.23.0)
     Requirement already satisfied: pip in /usr/local/lib/python3.7/dist-packages (from fastai==2.5.0) (21.1.3)
     Collecting fastcore<1.4,>=1.3.8
       Downloading fastcore-1.3.29-py3-none-any.whl (55 kB)
          ▌                                    ▐ 55 kB 4.4 MB/s
```

Y ahora modificamos el nombre de los directorios.

```
[ ]  for i in range(len(señales)):
         originalname = trainpath + str(i)
         newname = trainpath + señales[i]
         os.rename(originalname,newname)
```

Preparar datos de *test*

Por otro lado, las imágenes de test vienen etiquetadas en un archivo CSV, a diferencia de las de train que aparecen separadas en carpetas, el segundo paso que realizaremos para homogeneizar el proceso, es preclasificarlas según su etiqueta. Es importante tener claro que esto no es estrictamente necesario, ya que podríamos aportar las etiquetas al modelo seleccionando la columna apropiada de la tabla aportada (Test.csv).

En primer lugar cargamos el archivo CSV y mostramos las primeras filas.

```
[ ]  df = pd.read_csv("gtsrsb/Test.csv")
     df.head()
```

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
0	53	54	6	5	48	49	16	Test/00000.png
1	42	45	5	5	36	40	1	Test/00001.png
2	48	52	6	6	43	47	38	Test/00002.png
3	27	29	5	5	22	24	33	Test/00003.png
4	60	57	5	5	55	52	11	Test/00004.png

Nos interesan tanto el nombre como la etiqueta de cada imagen, por lo que seleccionamos la columna *ClassId* y *Path* y guardamos ambas como lista.

```
[ ]  testlabels = df['ClassId'].tolist()
     testimages = df['Path'].tolist()
```

Creamos las carpetas de cada clase.

```
[ ]  for i in range(len(señales)):
         newfolder = testpath + señales[i]
         os.mkdir(newfolder)
```

A continuación realizamos el proceso de clasificación de las imágenes y almacenamos una copia de las primeras 1000.

```
[ ]  for i in range(len(testimages)):
         originallocation= testpath + testimages[i][5:] #seleccionamos el nombre de la imágen
         newlocation= testpath + señales[testlabels[i]] + "/" + testimages[i][5:] #clasificamos la imagen en la carpeta que le corresponda
         os.rename(originallocation, newlocation)
```

(a) *Muestra 1* (b) *Muestra 2*

Figura 4.6: Capturas del cuaderno de clasificación

Capítulo 5

Caso de estudio

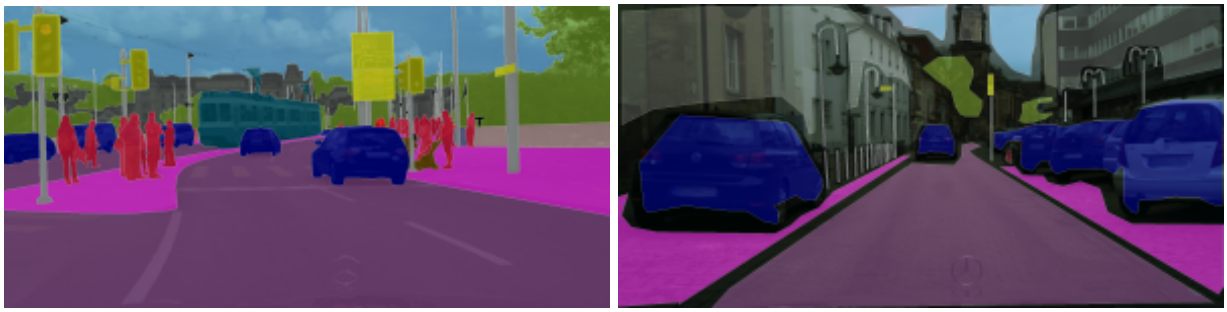
5.1. Descripción del conjunto de datos

Describiremos en esta sección los *datasets* que se utilizarán para entrenar el segmentador de imágenes.

- En primer lugar, se usará *dataset* **Cityscapes** [19], que fue creado en 2015 por un equipo alemán, y presenta como objetivo la segmentación semántica de objetos en 30 clases diferentes en un entorno urbano.

  Las imágenes fueron extraídas de diferentes vídeos grabados desde un vehículo en movimiento en 50 ciudades diferentes (mayoritariamente alemanas). Este proceso se extendió durante varios meses de forma que se obtuviese variedad climatológica en las capturas, aunque no se incluye ningún *frame* en el que se presenten condiciones extremas como fuerte lluvia o nieve. Posteriormente, se seleccionaron 5000 imágenes correspondientes a 27 de esas ciudades, en concreto aquellas que presentaban una mayor diversidad de categorías, para etiquetar de manera más cuidadosa (anotaciones finas)(ver Figura 5.1). Las imágenes de las 23 ciudades restantes se etiquetaron de manera menos precisa (anotaciones gruesas)(ver Figura 5.2), añadiendo otras 20000 imágenes al conjunto.

  Todos estos conjuntos están disponibles para uso gratuito en su página web oficial [16]. Para este proyecto se han utilizado únicamente aquellas imágenes que poseen anotaciones



(a) Ejemplo anotación fina (b) Ejemplo anotación gruesa

Figura 5.1: Anotaciones Cityscapes [16]

finas, que se corresponden con las carpetas gtFine (anotaciones) y leftImg8bit (imágenes). En cada una de estas se presentan carpetas dividiendo los datos en tres conjuntos: *train*, *validation* y *test*, y dentro de cada uno de ellos, se subdividen en ciudades. En este caso, se utilizan solo los conjuntos de *train* (2975 imágenes) y de *validation* (500 imágenes).

Las imágenes se aportan en formato PNG con resolución 2048 x 1024. En el caso de las etiquetas, para cada una de las imágenes se proporciona:

- Una máscara a color para segmentación semántica en formato PNG con resolución 2048 x 1024.
- Una máscara en blanco y negro para segmentación semántica en formato PNG con resolución 2048 x 1024.
- Una máscara en blanco y negro para segmentación de instancias en formato PNG con resolución 2048 x 1024.
- Un archivo JSON con las parametrizaciones de los polígonos que delimitan los diferentes objetos.

- El otro conjunto de datos utilizado es **Kitti**, que fue creado en 2015 por un equipo [45] compuesto por miembros del Instituto Tecnológico de Karlsruhe (KIT) y el Instituto Tecnológico Toyota de Chicago, y presenta un objetivo similar a Cityscapes de manera que, las imágenes aportadas en este conjunto de datos son compatibles con el primero.

  En este caso se aportan un total de 200 imágenes con anotaciones finas en forma máscaras, similares a las mencionadas en Cityscapes. Dichas imágenes fueron extraidas de un vídeo capturado desde un vehículo en circulación en zonas rurales y carreteras de Karlsruhe.

En ambos casos, el código de colores utilizado para hacer el mapeo entre clase y color es el ilustrado en la Figura 5.2.

Figura 5.2: Mapeo entre clase y color

5.2. Carga y preparación de los conjuntos de datos

La idea original de experimentación era cargar una red preentrenada de Unet, disponible en la librería Icevision, y reentrenarla con los datos del *dataset* Cityscapes. Sin embargo, cuando concluyó esta tarea surgió la idea de añadir también imágenes del *dataset* Kitti y comparar si el resultado podía ser mejorado. Por ello, en este apartado de carga y preparación de los datos se explicará cómo se realizaron dichos procesos para ambos conjuntos de datos.

La carga de datos se realiza en este caso, por comodidad y simplicidad, desde Google Drive. Para ello, es necesario que, previamente, se carguen en esta herramienta los *datasets* comprimidos en formato ZIP. Ahora ya podemos comenzar a procesar esos datos en un *notebook* en Google Colab.

En primer lugar, se descargará e instalará la librería de Icevision desde la plataforma GitHub, así como el resto de librerías y funciones que utilizaremos posteriormente y que son, nuevamente, las descritas en la sección 4.4. A continuación se importan los datos almacenados en la cuenta de Google Drive. Estos se cargarán en el almacenamiento local de la máquina que Colab nos proporciona. Descomprimimos ambos archivos, el correspondiente a Cityscapes y el de Kitti, y definimos el mapeo entre clases y *píxels* de las máscaras, es decir, asociamos el número 26 con coche o el número 28 con autobús, porque así es como vienen definidos en las máscaras correspondientes. El siguiente paso es integrar los datos en un único directorio para facilitar la posterior carga de datos. Para ello renombramos el fichero de "Cityscapes" (por ser el de mayor tamaño) a "datasets". Y añadimos en las carpetas correspondientes, *train* y *val*, las imágenes del *dataset* Kitti. La división entre ambos conjuntos se realiza de forma manual, separando 40 imágenes (20 %) para validación y manteniendo el resto para entrenamiento. Además, es necesario que a la hora de mover dichos archivos al nuevo directorio, se renombren para que posean el mismo formato de cadena que los del *dataset* Cityscapes, esto es (las partes entrecomilladas son literales): NombreCiudad + "_" + 13 caracteres + "_gtFine_color.png" en el caso de las máscaras, y para las imágenes: NombreCiudad + "_" + 13 caracteres + "_leftImg8bit.png". La estructura de los directorios puede visualizarse en la Figura 5.3.

```
datasets
|- gtFine
   |- test
      |- berlin
      ...
   |- train
      |- aachen
      ...
      |- kitti
      ...
   |- val
      |- frankfurt
      |- kitti
      ...
|- leftImg8bit
   |- test
      |- berlin
      ...
   |- train
      |- aachen
      ...
      |- kitti
      ...
   |- val
      |- frankfurt
      |- kitti
      ...
```

Figura 5.3: Estructura de directorios en Colab

El siguiente paso es almacenar las imágenes junto con las máscaras en formato Record Collection, que es la estructura base con la que trabaja Icevision. Además, definimos el ColorMap que utilizaremos, esto es, el mapeo entre clase y color que se utilizará para la representación de las máscaras. En este caso, lo definimos manualmente para mantener el código de colores original (Figura 5.2). Es importante mencionar que, junto con el *notebook* utilizado para esta tarea de segmentación, se adjunta un fichero de funciones, *functions.py*, en el que se modifican los métodos de visualización de imágenes que aporta Icevision ya que, por defecto, no permite personalizar el ColorMap.

A continuación se define el preprocesamiento de los datos, en este caso, las imágenes de entrenamiento se redimensionarán a 384 x 288 y, además, se añadirán algunas de ellas recortadas y otras desplazadas y rotadas. También se normalizarán, junto con las de validación.

Finalmente se aplica el preprocesamiento a los datos y se almacenan en formato Dataset.

Capítulo 6

Experimentación y evaluación

Describiremos en esta sección cómo se ha realizado el proceso de experimentación y analizaremos los resultados obtenidos. Comenzaremos introduciendo los hiperparámetros que han ido variando a lo largo del proceso, así como las métricas elegidas para su posterior valoración.

6.1. Hiperparámetros

En este apartado se definen los hiperparámetros en los que se han fijado diferentes valores en los entrenamientos.

- Tamaño de *batch*. Establece el número de imágenes que el algoritmo procesará dentro de cada subciclo en los que se divide una época. En la experimentación se ejecutó el modelo con valores de *batch* iguales a 8, 16, 32 y 64, sin embargo, los dos últimos concluyeron con el aborto del procesamiento por la limitación de memoria RAM que presenta Google Colab.
- *Learning rate*. Es el valor del paso en el que se actualizan los pesos en cada iteración para intentar minimizar la función de pérdida. En este caso se utiliza la función *lr_find()* de la librería fastai para encontrar el punto óptimo para nuestro modelo y datos concretos.
- Épocas. Una época es un ciclo completo de procesamiento de todos los datos agrupados en lotes o *batches*. Por tanto, el número de épocas establecidas para un entrenamiento equivale al número de veces que se procesa cada una de las imágenes. Dentro del número de épocas debemos diferenciar dos tipos de hiperparámetro asociados a las mismas:
  - Época congelada. Consiste en procesar una época completa manteniendo fijos los pesos de todas las capas a excepción de la última. Esto se realiza porque, al cargar un modelo preentrenado, se presupone que dicho modelo ya funciona bien con *datasets* similares. La idea entonces es concentrar el aumento del error generado al utilizar un conjunto de datos diferente en una sola capa, de forma que disminuya más rápidamente. En el entrenamiento se fijaron valores de épocas congeladas igual a 1, 5, 10 y 20.
  - Época libre. Es la ejecución de una época completa sin fijar los pesos de ninguna de las capas, de forma que todos ellos puedan modificarse para intentar disminuir la función de pérdida. En este caso se utilizaron como número de épocas libres 5, 10, 20 y 40.

6.2. Métricas

Se describirán en esta sección las métricas elegidas para medir la mejora del modelo en la segmentación de las imágenes. Se han seleccionado dos, ambas definidas en la librería fastai.

- *Dice Multi.* Denominado también Macro F1 [51], consiste en realizar una media aritmética de la métrica *F1-score*, definida en las dimensiones comparativas del estado del arte, para cada una de las clases o categorías a las que pueden pertenecer los *pixels*.

  Es decir, $dice\_multi = \frac{1}{n}\sum_x \text{F1}_x$, perteneciendo x al conjunto de las diferentes clases definidas.

  Este valor variará entre 0 y 1, siendo este último el valor óptimo.

- *Foreground accuracy.* Mide el *accuracy* de los *pixels* para todas las categorías eliminando el fondo. Es decir, compara las categorías predichas para cada *pixel* con las reales, que son las que aparecen en las máscaras de los datasets, asignando valor 1 si coinciden y 0 si son distintas. Posteriormente se calcula una media aritmética de todos esos valores. Esto puede expresarse matemáticamente con la función delta de Kronecker, que se define de la siguiente manera:

  $$\delta_{ij} \equiv \begin{cases} 0 & \text{si } i \neq j \\ 1 & \text{si } i = j \end{cases}$$

  Entonces, si denominamos P_i al pixel i de la máscara predicha y R_i al pixel i de la máscara real y suponemos que hay $n+1$ categorías y que la categoría número 0 se corresponde con el fondo:

  $$foreground\_accuracy = \frac{1}{n}\sum_{i=1}^{n} \delta_{P_i R_i}$$

  Este valor variará también entre 0 y 1 y, nuevamente, el valor 1 es el máximo.

6.3. Proceso de entrenamiento

Una vez almacenados los datos en formato Dataset, es necesario cargar el modelo preentrenado con el que iteraremos para intentar mejorar los resultados. Esto se hace desde dos fuentes diferentes:

- Desde Icevision: se carga un modelo preentrenado de red Unet que proporciona la propia librería. Esta es la fuente escogida cuando comenzamos con un entrenamiento desde cero.

- Desde un modelo local: cargamos un modelo que ya ha sido entrenado con ciertas épocas con los *datasets* elegidos para este problema (bien solo Cityscapes, bien Cityscapes y Kitti). Esto es necesario debido a las limitaciones que presenta Google Colab en sus licencias gratuitas, que no permite ejecuciones de más de 7 u 8 horas. Como parecía interesante probar a aumentar el número de épocas con el que se entrenaba el modelo, la solución fue realizar ciclos de ejecuciones de 40 épocas en los que el modelo exportado en el ciclo anterior se utilizaba como modelo a cargar en el siguiente. Así se pudieron obtener modelos con 120 épocas de entrenamiento.

El siguiente paso es definir el tamaño de *batch* y, a continuación, conectar el *notebook* con la herramienta Weights & Biases, que permite comparar diferentes modelos entrenados, así como diferentes combinaciones de hiperparámetros mediante la representación gráfica de varias métricas.

Tras esto se define el *learner*, que es una estructura creada por la librería fastai, que agrupa el modelo con el que se realizará el entrenamiento con los *batches* definidos y las métricas escogidas.

En este punto, tenemos la sesión preparada para comenzar con el entrenamiento, pero antes se utiliza la función *lr_find()*, que proporciona fastai, para sugerir un buen valor de *learning rate*. Ahora sí se empieza a entrenar. Para ello se utiliza la función *fine_ tune()*, también de fastai, que permite combinar el procesamiento de épocas congeladas y épocas libres.

El último paso para finalizar cada entrenamiento es exportar el modelo procesado. Icevision proporciona una función para este fin, pero es necesario modificarla puesto que solo viene definida para redes YOLOv5. Basta con permitirle aceptar redes Unet para solucionar el problema. Una vez exportado el modelo se descarga y almacena en la cuenta de Google Drive para utilizarla en el siguiente entrenamiento si fuese necesario.

6.4. Experimentación y resultados

Describiremos, a continuación, las diferentes pruebas y ajustes de hiperparámetros realizados y el porqué de dichas decisiones. Durante todo el proceso se ha utilizado una red Unet preentrenada que proporciona *Icevision*.

La fase experimental comenzó utilizando únicamente el *dataset* Cityscapes. Debido a las limitaciones técnicas relacionadas con el uso de Google Colab, el primer paso era determinar con qué tamaño de *batch* se completaban más rápido los procesos de entrenamiento. Este fue el objetivo de las cuatro primeras ejecuciones, cuyos resultados se muestran a continuación (ver Tabla 6.1.)

	Épocas congeladas	Épocas sin congelar	*Learning rate*	*Batch*	Tiempo ejecución
E1	2	10	10^{-4}	8	126 min
E2	2	10	10^{-4}	16	147 min
E3	2	10	10^{-4}	32	Abortado
E4	2	10	10^{-4}	64	Abortado

Tabla 6.1: Ejecuciones E1, E2, E3 y E4.

Nótese que el *learning rate* se fijó en ese valor por ser el recomendado para este problema concreto como ya se mencionó en la sección 6.1.

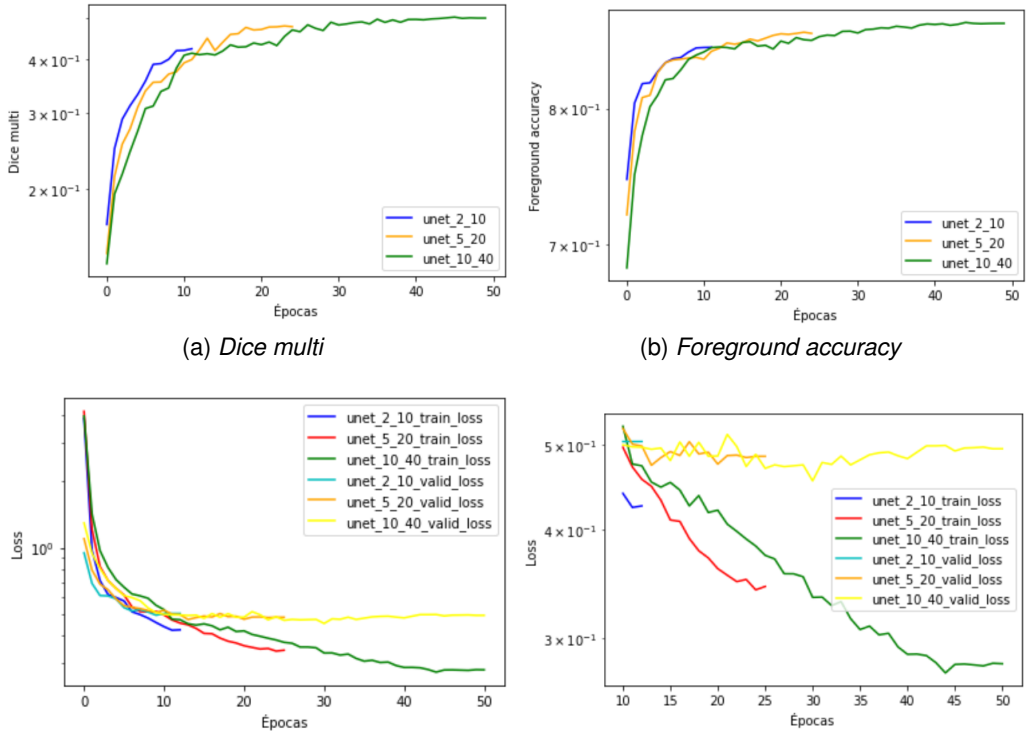
Analizando los resultados, y descartando la posibilidad de utilizar valores superiores a 16 (por limitaciones de memoria RAM), se decidió que el resto de entrenamientos se realizarían fijando el tamaño de *batch* a 8.

El siguiente paso era modificar el número de épocas y comprobar si la mejora era significativa o si, por el contrario, se producía sobreajuste. Los resultados se muestran en la siguiente tabla omitiendo los valores ya fijados (ver Tabla 6.2).

Tras esto, se revisó la curva de pérdida en entrenamiento frente a pérdida en validación, para comprobar si se estaba produciendo sobreajuste (ver Figura 6.1).

	Épocas congeladas	Épocas sin congelar	*Foreground accuracy*	*Dice multi*
E1	2	10	0,8497	0,4233
E5	5	20	0,8614	0,4765
E6	10	40	0,87	0,499
E7	20	80	Abortado	Abortado

Tabla 6.2: Ejecuciones E1, E5, E6 y E7.



(a) *Dice multi* (b) *Foreground accuracy*

(c) *Perdida entrenamiento/validación* (d) *Pérdida entrenamiento/validación con zoom*

Figura 6.1: Figuras métricas ejecuciones E1, E5 y E6

Parecía interesante continuar aumentando el número de épocas. Sin embargo, Colab restringía las ejecuciones de más de 7 u 8 horas. La solución fue realizar varias ejecuciones independientes de la siguiente forma:

- Un primer entrenamiento cargando el modelo preentrenado desde Icevision con 20 épocas congeladas.
- Un segundo entrenamiento cargando el modelo exportado en el paso anterior y ejecutando 40 épocas. En este caso y en los posteriores ya no se mantienen fijos los pesos de ninguna capa.
- En una tercera etapa se carga el modelo generado en la etapa dos para entrenar otras 40 épocas.

- Finalmente, cargamos el modelo exportado en el paso tres para ejecutar las últimas 40 épocas.

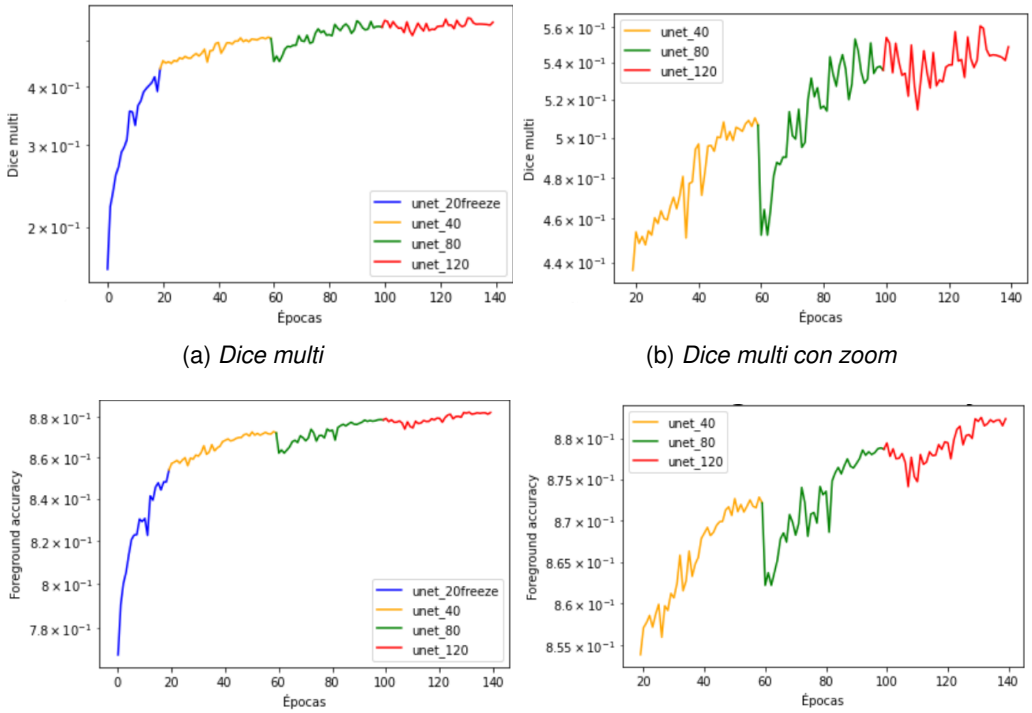
Así podemos extraer resultados para un modelo ejecutado con 40, 80 y 120 épocas sin congelar. Es importante mencionar que, para que las ejecuciones independientes fueran lo más similar posible a lo que habría sido una única ejecución completa, la división de los conjuntos en entrenamiento y validación no se realizaron de forma aleatoria, para que en cada uno de los entrenamientos se mantuviese fijo.

Los resultados obtenidos se muestran en la siguiente tabla.

	Total épocas congeladas	Total épocas sin congelar	*Foreground accuracy*	*Dice multi*
E8	20	40	0,8721	0,5064
E9	20	80	0,8787	0,5357
E10	20	120	0,8824	0,5487

Tabla 6.3: Ejecuciones E8, E9 y E10.

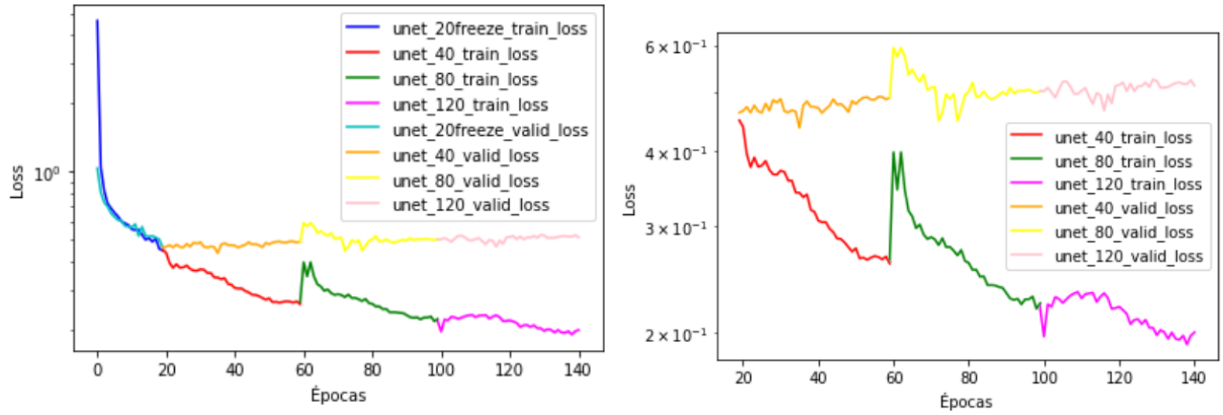
Las curvas de cada métrica así como la pérdida de validación frente a la de entrenamiento se muestran el las Figuras 6.2 y 6.3. En ellas se han representado las curvas completas. Es decir, cuando se representa, por ejemplo, unet_80, representamos las 100 épocas procesadas (20 congeladas, 80 sin congelar). Es por ello que las curvas aparecen solapadas.



(a) *Dice multi* (b) *Dice multi con zoom*

(c) *Foreground accuracy* (d) *Foreground accuracy con zoom*

Figura 6.2: Figuras métricas ejecuciones E8, E9 y E10



(a) *Perdida entrenamiento/validación* (b) *Pérdida entrenamiento/validación con zoom*

Figura 6.3: Pérdida ejecuciones E8, E9 y E10

Como se puede observar, los modelos continuan mejorando al aumentar el número de épocas, aunque es cierto que el modelo unet120 presenta diferencias mínimas con respecto a unet80. Sin embargo, si que se aprecia una mayor separación en las curvas de pérdida en este caso, ya que la tendencia de las mismas es, aparentemente, separarse más y más. Idealmente, ambas curvas deberían mantenerse en valores más próximos, por lo que la decisión tomada fue no continuar con el aumento del número de ciclos.

En este punto, se barajó la posibilidad de variar la arquitectura utilizada pero, finalmente, se descartó por los buenos resultados ya obtenidos con la red Unet y por la falta de compatibilidad con algunos paquetes de la máquina proporcionada por Colab. Sin embargo, se decidió continuar con la experimentación para contrastar estos resultados con otras variaciones. Surgió entonces la idea de introducir un segundo *dataset* y repetir las últimas pruebas utilizándolo junto con Cityscapes. Para ello, se añadieron nuevos datos procedentes del *dataset* Kitti.

Los resultados obtenidos se muestran a continuación (ver Tabla 6.4).

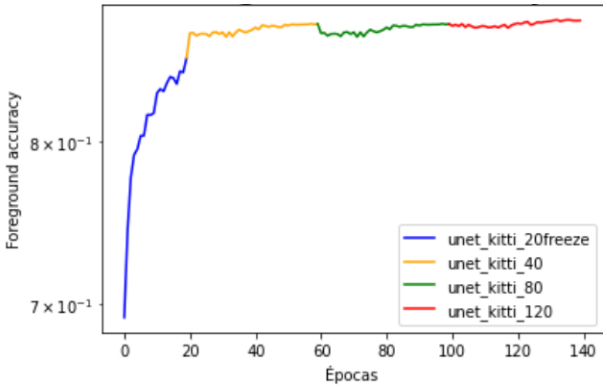
	Total épocas congeladas	Total épocas sin congelar	*Foreground accuracy*	*Dice multi*
E11	20	40	0,8752	0,5345
E12	20	80	0,8812	0,5538
E13	20	120	0,8837	0,5589

Tabla 6.4: Ejecuciones E11, E12 y E13.

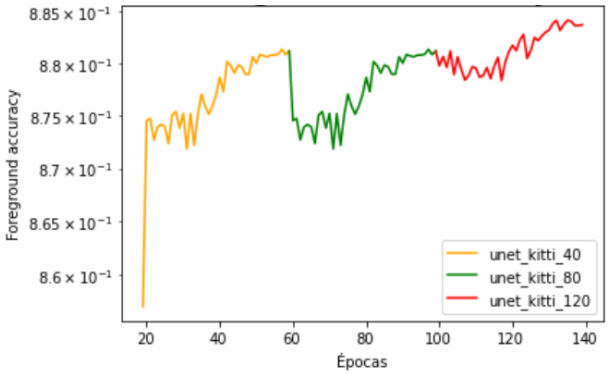
Las curvas obtenidas pueden observarse en la Figura 6.4.

En este caso, la tendencia de las curvas asociadas al *foreground accuracy* y *dice multi* también presentan un buen crecimiento y, nuevamente, las diferencias entre las 80 y las 120 épocas no son demasiado significativas. De hecho, en el caso de *dice_ multi*, la primera (unet_kitti_80) es superior a la que posee un mayor número de épocas (unet_kitti_120).

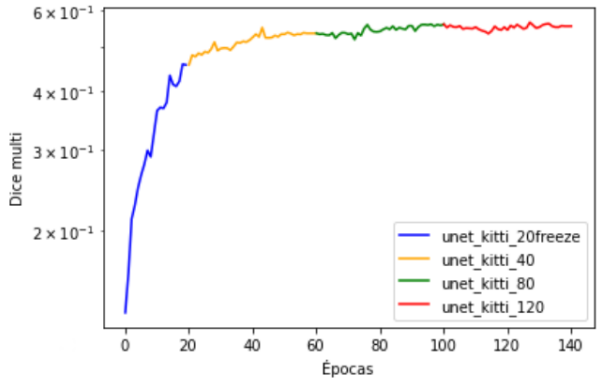
Por otro lado, en las curvas de pérdida de entrenamiento frente a validación se obtienen valores más próximos entre sí. Aún así, en este caso tampoco continuaremos aumentando el número de épocas porque la tendencia vuelve a ser la separación de las mismas y llegaríamos, finalmente, a un caso análogo al experimentado en el paso anterior.



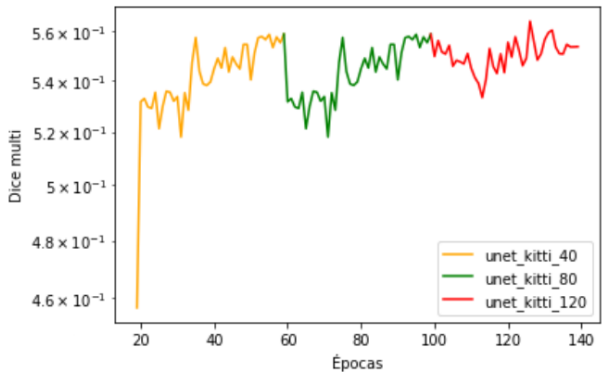
(a) *Foreground accuracy*



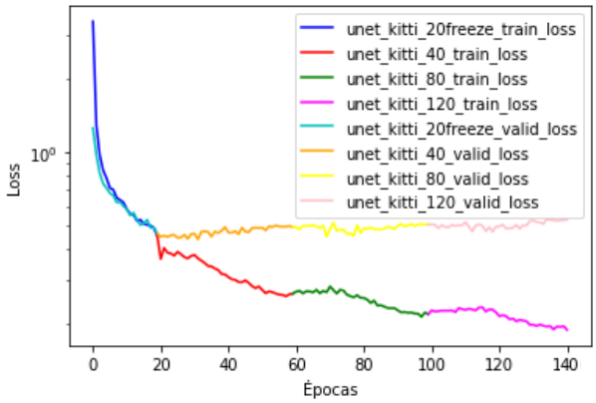
(b) *Foreground accuracy con zoom*



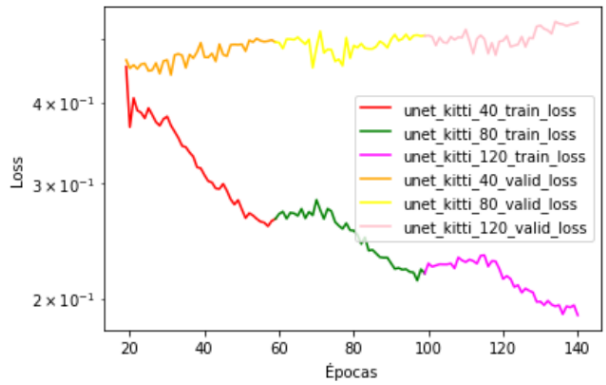
(c) *Dice multi*



(d) *Dice multi con zoom*



(e) *Perdida entrenamiento/validación*



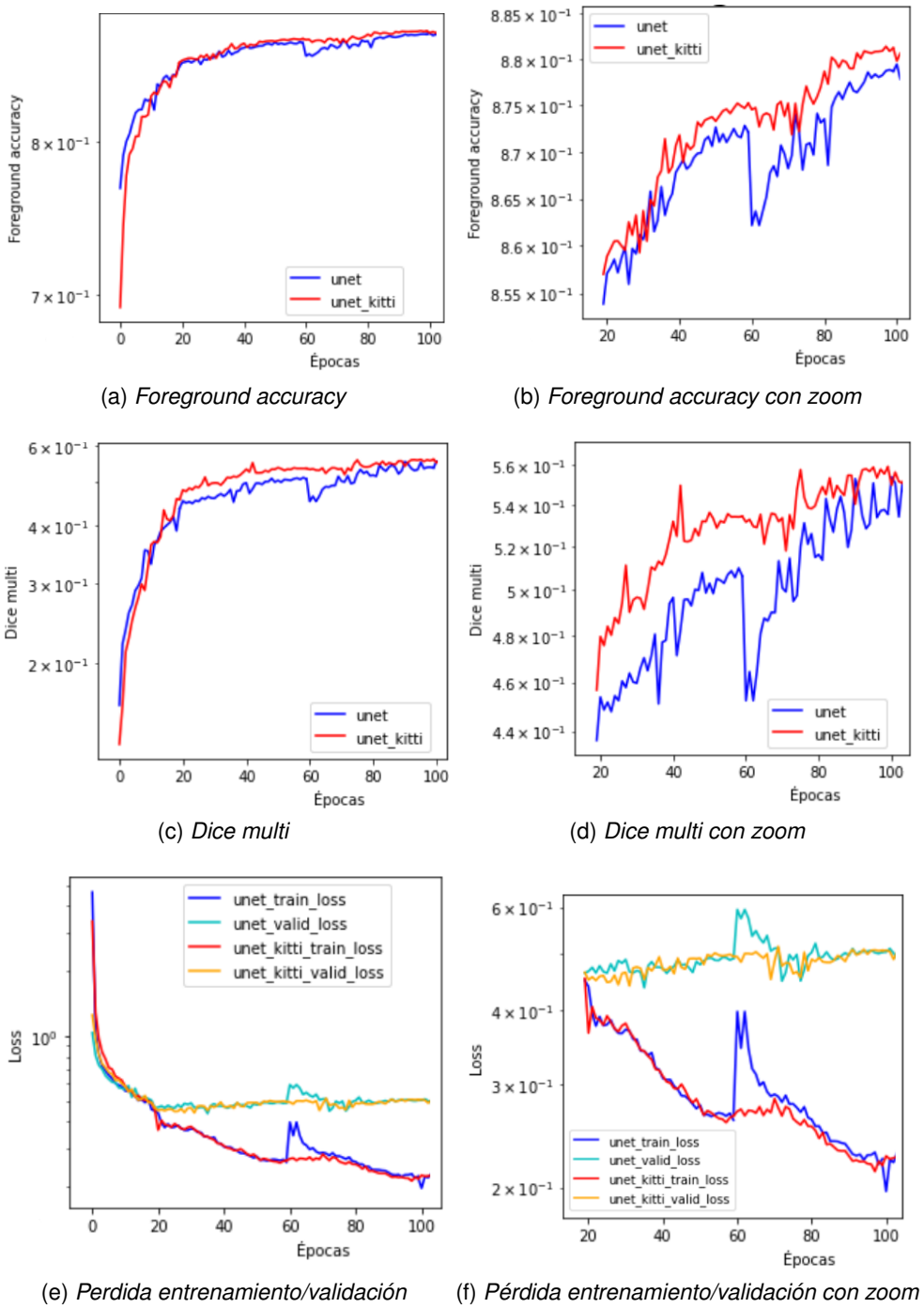
(f) *Pérdida con zoom*

Figura 6.4: Figuras métricas ejecuciones E11, E12 y E13

Por todo lo mencionado, se decidió, en un último paso, hacer una comparación entre los dos modelos que parecían presentar mejores resultados:

- Unet con 20 épocas congeladas y 80 sin congelar entrenado con Cityscapes.
- Unet con 20 épocas congeladas y 80 sin congelar entrenado con Cityscapes y Kitti.

Veamos las curvas obtenidas en la Figura 6.5.



(a) *Foreground accuracy* (b) *Foreground accuracy con zoom*

(c) *Dice multi* (d) *Dice multi con zoom*

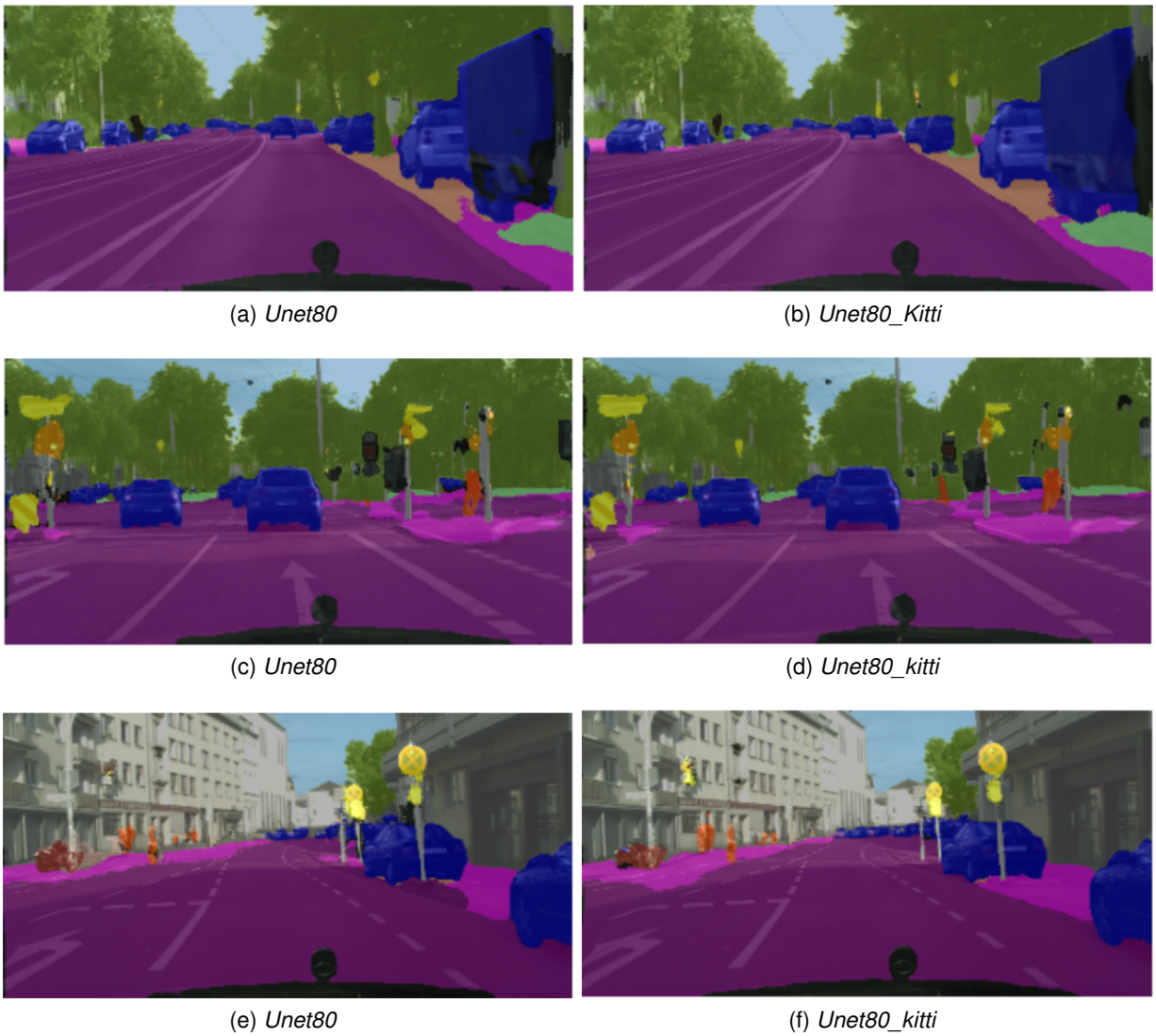
(e) *Perdida entrenamiento/validación* (f) *Pérdida entrenamiento/validación con zoom*

Figura 6.5: Figuras métricas ejecuciones E9 y E12

Los valores obtenidos por el modelo entrenado con Cityscapes y Kitti son ligeramente superiores a los obtenidos por unet80. Veamos ahora qué ocurre al realizar el proceso de inferencia.

6.5. Inferencia

Ambos conjuntos de datos poseen imágenes de *test* para realizar el proceso de inferencia. Veamos los resultados obtenidos con las redes Unet entrenadas con 80 épocas en la Figura 6.6. Aunque ambos modelos funcionan bastante bien, se aprecia una gran mejora a la hora de predecir, en la primera imagen, el camión situado a la derecha, y también puede notarse una mejor definición en las líneas que delimitan los diferentes objetos en el caso del modelo entrenado con Cityscapes y Kitti.



(a) *Unet80* (b) *Unet80_Kitti*

(c) *Unet80* (d) *Unet80_kitti*

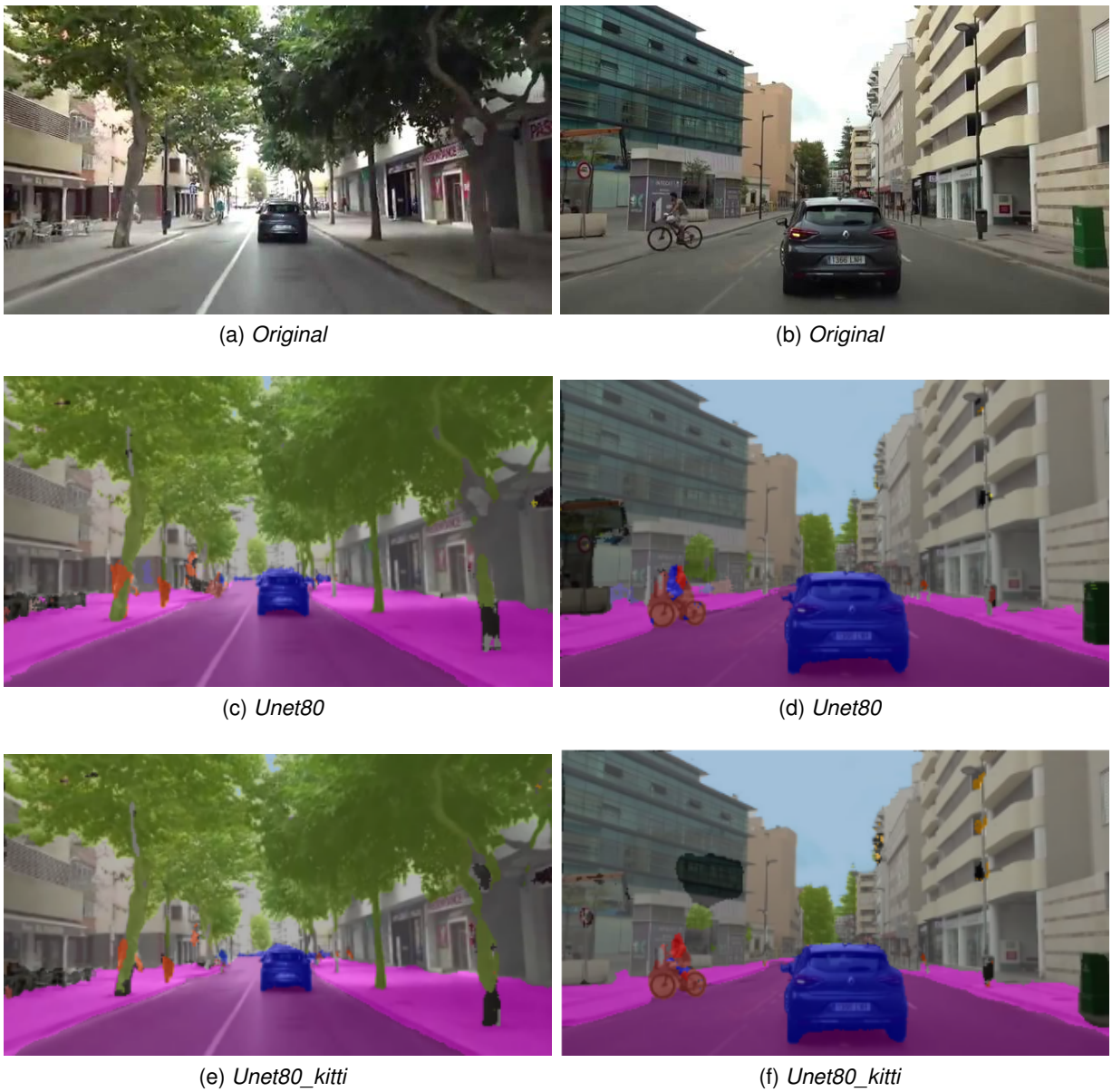
(e) *Unet80* (f) *Unet80_kitti*

Figura 6.6: Predicción de imágenes de *test*

El proceso de inferencia podría haber concluido aquí, pero se ha ido un paso más allá y se ha realizado la inferencia sobre vídeos, construyendo una función para ello. Como se vio en secciones anteriores, un vídeo no es más que una secuencia ordenada de imágenes, denominados *frames*. La idea es entonces dividir, en un primer paso, el vídeo en *frames* para, posteriormente, realizar

la inferencia en esas imágenes y, finalmente, unir las máscaras predichas de manera ordenada y obtener así un vídeo de máscaras, que será la salida de la función.

Para probar los resultados obtenidos con ambas redes, se ha elegido un vídeo extraído de Youtube, que graba las calles de Ibiza desde un coche en circulación. Veamos algunas capturas del mismo para comparar los modelos (ver Figura 6.7).



(a) *Original* (b) *Original*

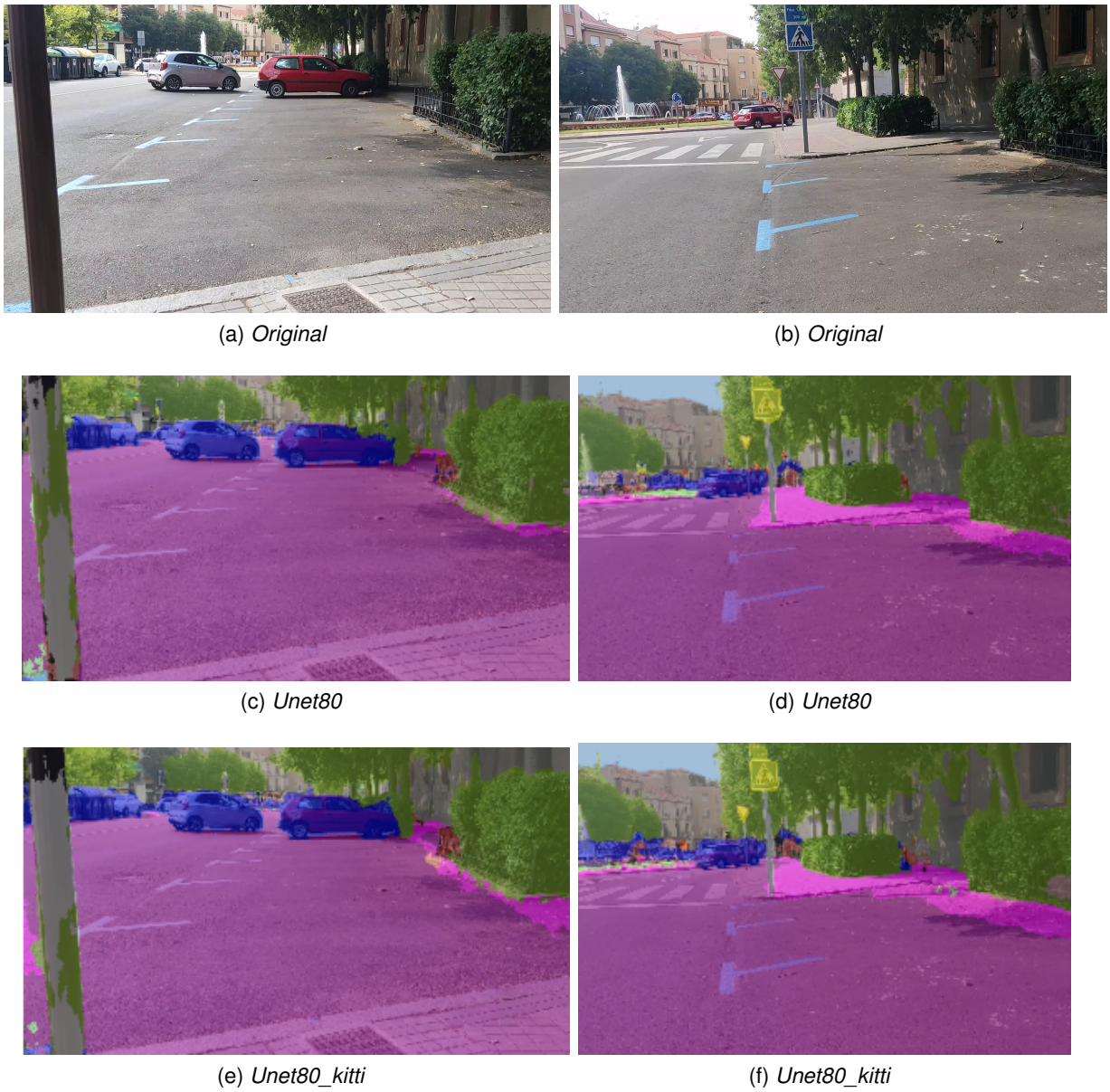
(c) *Unet80* (d) *Unet80*

(e) *Unet80_kitti* (f) *Unet80_kitti*

Figura 6.7: Imágenes extraidas de la máscara del vídeo de Ibiza

En este caso, como puede observarse, la versión entrenada con Kitti y Cityscapes presenta algunas imprecisiones en la imagen derecha a la hora de segmentar el edificio izquierdo. Sin embargo, localiza mejor al *rider* que el modelo entrenado únicamente con Cityscapes.

Finalmente, como última prueba, se ha grabado un video en los alrededores de la facultad y se ha procesado con ambos modelos. Los resultados se muestran en la Figura 6.8.



(a) *Original* (b) *Original*

(c) *Unet80* (d) *Unet80*

(e) *Unet80_kitti* (f) *Unet80_kitti*

Figura 6.8: Imágenes extraidas de la máscara del vídeo de la facultad

Ambos modelos segmentan bien coches, señales y vegetación pero confunden, en algunos casos, la acera con la carretera y viceversa. Sin embargo, podemos conluir que los resultados son, en general, satisfactorios.

Capítulo 7

Conclusiones y trabajo futuro

En este último capítulo se presentan las conclusiones extraídas tras la finalización del proyecto. Asimismo, se exponen algunas líneas de trabajo futuro.

7.1. Conclusiones

Analizaremos, en primer lugar, las conclusiones relativas al proyecto. En concreto, se valorará si se han cumplido o no los objetivos planteados.

- En el Capítulo 3, en la revisión del estado del arte, se realizó un estudio de las principales arquitecturas usadas en el procesamiento de imágenes y vídeos (OBJ-1). Además, pudo comprobarse el notable crecimiento que está experimentando el campo de la visión computacional en estos últimos años, que se ve reflejado en el volumen y calidad de las diferentes propuestas analizadas.
- A continuación, en el Capitulo 4, se documentan las diferentes tecnologías y conjuntos de datos utilizados en el desarrollo de los cuadernos de aprendizaje (OBJ-2). En ellos se presentan diferentes ejemplos de procesamiento de imágenes desde una perspectiva didáctica, combinando explicaciones teóricas con código ejecutable de forma que son autocontenidos.
- Finalmente, en los Capítulos 5 y 6, se muestra paso a paso, el proceso de experimentación realizado, que se centra en la segmentación semántica de imágenes en una ciudad inteligente (OBJ-3). Este fue quizá, el proceso más difícil de todos. El uso de tecnologías aún en desarrollo, como es Icevision, derivó en varias ocasiones en problemas de compatibilidad con el entorno, además del hecho de que algunos módulos y funciones no estaban completamente desarrollados aún. A pesar de ello, resultó tremendamente útil ya que facilitó el proceso de visualización de las máscaras y, lo más importante, permitió el uso de modelos preentrenados que mejoraron notablemente los resultados obtenidos.

En definitiva, se han completado satisfactoriamente todos los objetivos y, algunos de los productos generados en su consecución, como los *notebooks* de aprendizaje del OBJ-2, podrían ser de utilidad en el futuro para otros alumnos que se sientan atraídos por el tema y quieran aprender sobre ello de la misma manera que lo he hecho yo.

Por otro lado, la metodología escogida ha sido un gran acierto. La dinámica establecida por Uvagile ha favorecido el trabajo constante y regular, que es lo que ha permitido terminar el

proyecto en tiempo y forma. Por otro lado, la comunicación fluida con los profesores y comunidad a través de Teams ha facilitado teminar con bloqueos y resolver las dudas que han ido surgiendo de manera rápida y eficaz.

En el ámbito personal, el desarrollo de este Trabajo Fin de Grado me ha permitido descubrir un nuevo campo que me ha resultado tremendamente interesante. Es cierto que la visión computacional es un ámbito complejo, pero también es un ámbito aún en desarrollo que presenta, y presentará, numerosas oportunidades tanto a nivel académico como laboral. Con este proyecto he aprendido a ser constante con mi ritmo de trabajo y a persistir día tras día en cumplir los objetivos planteados. Ha sido una también una oportunidad para aprender a valorar el trabajo realizado por los profesores en su tarea de enseñanza, porque no es algo sencillo. En definitiva, esta experiencia me ha permitido desarrollar algunas habilidades tanto académicas como personales que me acompañarán por siempre.

7.2. Trabajo futuro

Las limitaciones temporales que plantea la asignatura Trabajo Fin de Grado acotan el alcance del proyecto. Mostraremos, a continuación, algunas líneas de trabajo que no han podido abarcarse pero que supondrían una mejora del modelo implementado.

- En primer lugar, sería interesante aumentar aún más el *dataset* y comprobar si los resultados obtenidos son mejores. En concreto, parece adecuado añadir las imágenes del conjunto de datos WildDash 2, también compatible con Cityscapes.
- Otra posibilidad, sería realizar nuevos entrenamientos con arquitecturas diferentes y más avanzadas como puede ser Unet++ [79].
- Por otro lado, las ciudades demandan la capacidad de procesar los datos en tiempo real, por tanto, añadir esta propiedad a nuestro sistema haría posible su implementación en una *smart city*.

Parte III

Apéndices

Apéndice A

Manual de Instalación

En esta sección, se mostrarán los pasos a seguir para crear un entorno local de Anaconda y poder ejecutar los *notebooks* presentados en la memoria en Jupyter Notebooks. Este proceso se realizará para una máquina con sistema operativo Linux ya que es el que presenta el entorno proporcionado por Colab y mantendríamos así la compatibilidad de los cuadernos.

En caso de que la máquina utilizada no utilice un sistema Linux, el primer paso es crear una máquina virtual para cumplir con este requisito. Para ello, puede utilizarse el software Virtual Box [52] e instalar una máquina con la última versión de Ubuntu con soporte a largo plazo (*Long Term Support*), que en este caso es la 22.04 [28]. No se aportan detalles de este proceso, pero puede consultarse en la siguiente referencia [81].

Una vez instalada la versión correspondiente de Linux, el siguiente paso es instalar una distribución de Python y un entorno de desarrollo. En este caso, Anaconda es el software elegido, por lo que descargamos una distribución desde su página oficial [3] y la instalamos.

Para ello, actualizamos primero los repositorios con el comando

```
$ sudo apt update
```

A continuación, instalaremos el paquete *curl*, que permitirá posteriormente descargar el script de Anaconda, para ello ejecutamos

```
$sudo apt install curl -y
```

Descargaremos el archivo en el directorio de archivos temporales (/tmp), para ello ejecutamos

```
$ cd / tmp
$ curl --output anaconda.sh https://repo.anaconda.com/archive/Anaconda3-5.3.1-
Linux-x86_64.sh
```

Podemos verificar que la descarga se ha completado correctamente comprobando el *checksum* del archivo, que debe ser:

```
d4c4256a8f46173b675dd6a62d12f566ed3487f932bab6bb7058f06c124bcc27
```

Utilizamos para ello el comando *sha256sum*:

```
$ sha256sum anaconda.sh
```

El siguiente paso es ejecutar el script para instalar el entorno:

```
$ bash anaconda.sh
```

Durante el proceso aceptamos los términos del Acuerdo de Licencia y especificamos la ruta en la que deseamos realizar la instalación.

Una vez completado el paso anterior, solo es necesario activar el entorno para finalizar el proceso. Para ello:

```
$ source ~/.bashrc
```

El proceso ha finalizado. Para abrir la aplicación en el navegador solo hay que ejecutar el comando

```
$ jupyter notebook
```

Ahora solo faltaría descargar e instalar los paquetes y librerías que se utilizarán en los cuadernos. Utilizaremos el comando *pip*, por lo que primero lo actualizamos:

```
$pip install --upgrade pip
```

E instalamos los paquetes necesarios en las versiones adecuadas. En primer lugar instalamos la librería Fastai. Para ello utilizaremos el siguiente comando que, además, instalará sus dependencias.

```
$pip install fastai==2.7.6
```

Continuamos instalando la librería Icevision en su última versión. Con el siguiente comando añadimos también las dependencias de la misma.

```
$pip install icevision[all]
```

Y continuamos con el resto:

```
$pip install sklearn==0.0
$pip install pytest-shutil==1.7.0
$pip install pandas
$pip install matplotlib==3.2.2
```

Apéndice B

Contenido adjunto

Se detallará en este último apéndice el contenido que se adjunta con la memoria.

- ***Notebooks* de aprendizaje.** Tanto los archivos .ipynb, como los modelos entrenados y, en algunos casos, los *datasets*.
  - *Notebook* clasificación.
    - Cuaderno: Notebook_clasificacion.ipynb
    - Modelo: clasificadorgtsrsb.pth
    - *Dataset*: gtsrsb.zip
  - *Notebook* detección.
    - Cuaderno: Notebook_deteccion.ipynb
    - Modelo: detectorpennfudan.pth
    - *Dataset*: pennfudanped.zip
  - *Notebook* segmentación.
    - Cuaderno: Notebook_segmentacion.ipynb
    - Modelo: segmentadorcamvid.pth
    - *Dataset*: En este caso no se aporta porque se descarga en el propio cuaderno directamente de Icevision.
- ***Notebook* caso final.** En este caso se aporta el cuaderno con el que se han entrenado los modelos, el cuaderno con el que se ha realizado la inferencia en video, los dos modelos seleccionados finalmente (unet con 80 épocas en la versión entrenada solo con Cityscapes y en la versión entrenada con Cityscapes y Kitti), un archivo de funciones para poder visualizar las máscaras con los colores originales y el vídeo de la facultad de segovia sobre el que se ha realizado el proceso de inferencia. Los *datasets* no se aportan puesto que ocupan más de 12GB. Todas las modificaciones sobre los datos se realiza dentro del propio cuaderno por lo que pueden descargarse de sus páginas oficiales y utilizarlos con el *notebook* aportado.
  - Cuaderno: Segmentacion_cityscapes_kitti.ipynb
  - Cuaderno: inferencia_funcion.ipynb
  - Modelo: unet80.pth

- Modelo: unetkitti80.pth
- Funciones: functions.py
- Vídeo: inferencia.mp4

Bibliografía

[1] *Acerca de la tecnología avanzada Face ID*. Accedido el 09/04/2022. 2022. URL: https://support.apple.com/es-es/HT208108.

[2] *AI and Machine Learning Exploit, Deepfakes, Now Harder to Detect | PCMag*. Accedido el 18/03/2022. URL: https://www.pcmag.com/news/ai-and-machine-learning-exploit-deepfakes-now-harder-to-detect.

[3] *Anaconda | Anaconda Distribution*. Accedido el 19/07/2022. URL: https://www.anaconda.com/products/distribution.

[4] Ludovic Arnold y col. «An Introduction to Deep Learning». En: *HAL open science* (2016).

[5] Ana Ayerbe. «La ciberseguridad y su relación con la inteligencia artificial». En: (2021). Accedido el 10/05/2022. URL: http://blogs.tecnalia.com/inspiring-blog/2020/03/12/los-datos-nunca-.

[6] Nazaret Roda Barrios. *Visión Artificial en el Sector Médico*. Accedido el 11/05/2022. 2021. URL: https://www.vidaip.es/vision-artificial-en-el-sector-medico/.

[7] Kent Beck y col. *Manifesto for Agile Software Development*. 2001. URL: http://www.agilemanifesto.org/.

[8] «Benchmark Analysis of Representative Deep Neural Network Architectures». En: (2018). Accedido el 03/043/2022. URL: https://github.com/CeLuigi/models-comparison.pytorch.

[9] Gary Bradski. *OpenCV*. 2010.

[10] Anibal Bregón. «Tema 10 - Aprendizaje - RRNN». En: (2021).

[11] Anibal Bregón. «Tema 11 - Aprendizaje - DL». En: (2021).

[12] Anibal Bregón. «Tema 7 - Aprendizaje - Introduccion». En: (2021).

[13] Gabriel J. Brostow, Julien Fauqueur y Roberto Cipolla. «Semantic Object Classes in Video: A High-Definition Ground Truth Database». En: *Pattern Recognition Letters* (2008).

[14] Rodrigo Cabello. *Vision-Transformers: ¿Fin de las redes neuronales convolucionales?*

[15] Nicolas Carion y col. *End-to-End Object Detection with Transformers*. 2020.

[16] «Cityscapes Dataset – Semantic Understanding of Urban Street Scenes». En: (2015). Accedido el 26/06/2022. URL: https://www.cityscapes-dataset.com/.

[17] *Ciudades Inteligentes: Su Relación con IoT y Machine Learning*. Accedido el 06/05/2022. URL: https://www.encora.com/es/blog/ciudades-inteligentes-su-relacion-con-iot-y-machine-learning.

[18] Eduardo Temprano Coleto. *Métodos de aprendizaje profundo para la segmentación semántica de personas.* 2020.

[19] Marius Cordts y col. «The Cityscapes Dataset for Semantic Urban Scene Understanding». En: (2015). Accedido el 14/06/2022. URL: www.cityscapes-dataset.net.

[20] Isis Bonet Cruz y col. *Redes neuronales recurrentes para el análisis de secuencias.* 2007.

[21] Alberto Maisueche Cuadrado. *Utilización del Machine Learning en la industria 4.0.*

[22] Iván Cárdenas. *¿Puede el piloto automático Tesla detectar un canguro que salta?* Accedido el 11/05/2022. 2021. URL: https://electronia.es/puede-el-piloto-automatico-tesla-detectar-un-canguro-que-salta-video/.

[23] *Deep Learning-based Real-time Video Processing - KDnuggets.* Accedido el 27/03/2022. URL: https://www.kdnuggets.com/2021/02/deep-learning-based-real-time-video-processing.html.

[24] *Deep Learning en sanidad, aliado del machine learning sanitario.* Accedido el 08/05/2022. URL: https://www.cogesasl.com/2018/11/29/deep-learning-en-sanidad/.

[25] *Deepfake video of Volodymyr Zelensky surrendering surfaces on social media.* Accedido el 20/03/2022. 2022. URL: https://www.youtube.com/watch?v=X17yrEV5sl4&ab_channel=TheTelegraph.

[26] *Detectron2's documentation.* Accedido el 30/03/2022. URL: https://detectron2.readthedocs.io/en/latest/.

[27] Jairo Eduardo Márquez Diáz. «Visión artificial profunda aplicada a la identificación temprana de cáncer no melanoma y queratosis actínica». En: *Computación y Sistemas* 24 (2 2021), págs. 751-766. ISSN: 1405-5546. DOI: 10.13053/CYS-24-2-2901.

[28] *Download Ubuntu Desktop | Download | Ubuntu.* Accedido el 19/07/2022. URL: https://ubuntu.com/download/desktop.

[29] *El futuro de la agricultura: Inteligencia Artificial.* Accedido el 13/05/2022. 2021. URL: https://www.deere.es/es/agricultura/el-futuro-de-la-agricultura/.

[30] *Etapas de un sistema de visión - Visión artificial - Solución ingenieril.* Accedido el 29/03/2022. URL: http://solucioningenieril.com/vision_artificial/etapas_de_un_sistema_de_vision.

[31] *German Traffic Sign Benchmarks.* Accedido el 5/05/2022. URL: https://benchmark.ini.rub.de/gtsrb_news.html.

[32] Claudio Javier Tablada Germán y Ariel Torres. *Redes Neuronales Artificiales.*

[33] Ross Girshick. *Fast R-CNN.* Accedido el 20/05/2022. URL: https://github.com/rbgirshick/.

[34] Ross Girshick y col. «Rich feature hierarchies for accurate object detection and semantic segmentation». En: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (nov. de 2013), págs. 580-587. ISSN: 10636919. DOI: 10.48550/arxiv.1311.2524.

[35] Kaiming He y col. *Deep Residual Learning for Image Recognition.* Accedido el 22/05/2022. URL: http://image-net.org/challenges/LSVRC/2015/.

[36] Sebastian Houben y col. «Detection of traffic signs in real-world images: The German traffic sign detection benchmark». En: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. 2013, págs. 1-8.

[37] Jie Hu. *Squeeze-and-Excitation Networks*.

[38] *Icevision*. Accedido el 30/03/2022. URL: https://pypi.org/project/icevision/.

[39] *Image Super Resolution | Deep Learning for Image Super Resolution*. Accedido el 18/03/2022. URL: https://www.analyticsvidhya.com/blog/2021/05/deep-learning-for-image-super-resolution/.

[40] *ImageNet Benchmark (Image Classification) | Papers With Code*. Accedido el 02/04/2022. 2022. URL: https://paperswithcode.com/sota/image-classification-on-imagenet.

[41] *Informe Empleos Emergentes 2020*. Linkedin, 2020.

[42] «Introduction to machine learning, neural networks, and deep learning». En: *Translational Vision Science and Technology* (2020).

[43] *La visión artificial pone su ojo sobre las enfermedades cardiovasculares*. Accedido el 08/05/2022. URL: https://blogthinkbig.com/vision-artificial-enfermedades-cardiovasculares.

[44] Miguel A. Martínez-Prieto y col. «Agilizando el aprendizaje de bases de datos». En: *Actas de las JENUI* 6 (2021), págs. 83-90.

[45] Moritz Menze y Andreas Geiger. «Object Scene Flow for Autonomous Vehicles». En: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[46] Lara Moreno Díaz-Alejo. *Análisis comparativo de arquitecturas de redes neuronales para la clasificación de imágenes*. 2020.

[47] *Motion-based Segmentation and Recognition Dataset*. Accedido el 23/05/2022. URL: http://mi.eng.cam.ac.uk/research/projects/VideoRec/CamVid.

[48] Trabajo Fin De Máster, Luis Baumela Molina y Aitor Gutierrez Basauri. *Deep Learning object detection architectures in ADAS*. 2019.

[49] *Neural Style Transfer: Everything You Need to Know [Guide]*. Accedido el 18/03/2022. URL: https://www.v7labs.com/blog/neural-style-transfer.

[50] «Object Detection Combining Recognition and Segmentation». En: (2007).

[51] Juri Opitz y Sebastian Burst. «Macro F1 and Macro F1 a note». En: (2019).

[52] *Oracle VM VirtualBox*. Accedido el 19/07/2022. URL: https://www.virtualbox.org/.

[53] Keiron O'shea y Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015.

[54] *Pedestrian Detection Database*. Accedido el 21/05/2022. URL: https://www.cis.upenn.edu/~jshi/ped_html/.

[55] *Project Jupyter | About Us*. Accedido el 23/05/2022. URL: https://jupyter.org/about.

[56] *Qué es la Cuarta Revolución Industrial y cuáles sus tecnologías - Iberdrola*. Accedido el 02/053/2022. URL: https://www.iberdrola.com/innovacion/cuarta-revolucion-industrial.

[57] *Redes Neuronales Recurrentes*. Accedido el 19/03/2022. URL: http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/RNN_LTSM/introduccion_rnn.html.

[58] Joseph Redmon y Ali Farhadi. *YOLOv3: An Incremental Improvement*. Accedido el 05/04/2022. URL: https://pjreddie.com/yolo/..

[59] Shaoqing Ren y col. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. Accedido el 21/05/2022. URL: http://image-net.org/challenges/LSVRC/2015/results.

[60] Sebastian Ruder. *An overview of gradient descent optimization algorithms **. Accedido el 29/03/2022. 2017. URL: http://caffe.berkeleyvision.org/tutorial/solver.html.

[61] Pedro Ignacio Orellana Rueda. *Segmentación semántica y reconocimiento de lugares usando características CNN pre-entrenadas*. 2019.

[62] *SANE*. Accedido el 05/04/2022. 2016. URL: https://www.egitron.pt/en/quality-control-corkbeverages/control-the-filling-level/artificial-vision-systems/sane/.

[63] Saúl y Rozada Raneros. *Estudio de la arquitectura YOLO para la detección de objetos mediante deep learning*. 2021.

[64] Ken Schwaber y Jeff Sutherland. *The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game*. 2020.

[65] «SSD: Single shot multibox detector». En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2016).

[66] J. Stallkamp y col. «Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition». En: *Neural Networks* 32 (ago. de 2012), págs. 323-332. ISSN: 08936080. DOI: 10.1016/J.NEUNET.2012.02.016.

[67] *Stream processing - Tecnologías y comparativa - Aprender BIG DATA*. Accedido el 27/03/2022. 2022. URL: https://aprenderbigdata.com/stream-processing/.

[68] *Sueldo: Data Scientist (Junio, 2022) | Glassdoor*. Accedido el 14/05/2022. URL: https://www.glassdoor.es/Sueldos/data-scientist-sueldo-SRCH_KO0,14.htm?clickSource=searchBtn.

[69] *Sueldo: Programador (Abril, 2022) | Glassdoor*. Accedido el 15/04/2022. URL: https://www.glassdoor.es/Sueldos/programador-sueldo-SRCH_KO0,11.htm?clickSource=searchBtn.

[70] *Sueldos de analista | Glassdoor*. Accedido el 15/04/2022. URL: https://www.glassdoor.es/Sueldos/analista-sueldo-SRCH_KO0,8.htm?clickSource=searchBtn.

[71] Lu Tan y col. «Comparison of RetinaNet, SSD, and YOLO v3 for real-time pill identification». En: *BMC Medical Informatics and Decision Making* (2021).

[72] *Te damos la bienvenida a Colaboratory - Colaboratory*. Accedido el 23/05/2022. URL: https://colab.research.google.com/.

[73] *The 7 Top Smart Cities Around the World*. Accedido el 24/05/2022. URL: https://blog.bismart.com/en/top-smart-cities-around-world.

[74] *The AI-powered enterprise*. Capgemini Research Institute. 2020.

[75] Zaib Ullah y col. «Applications of Artificial Intelligence and Machine learning in smart cities». En: *Computer Communications* 154 (mar. de 2020), págs. 313-323. ISSN: 1873703X. DOI: 10.1016/J.COMCOM.2020.02.069.

[76] Metodología Uvagile Uvagile y col. *¿Puede ser Agile la Docencia Universitaria?* 2019.

[77] Yaowei Wang y col. «Peng Cheng Object Detection Benchmark for Smart City». En: (mar. de 2022). DOI: 10.48550/arxiv.2203.05949.

[78] Cameron Wolfe. *Deep Learning on Video (Part One): The Early Days...* Accedido el 27/03/2022. 2021. URL: https://towardsdatascience.com/deep-learning-on-video-part-one-the-early-days-8a3632ed47d4.

[79] Zongwei Zhou y col. *UNet++: A Nested U-Net Architecture for Medical Image Segmentation.*

[80] *¿Cuál es el coste de la empresa al contratar a un trabajador?* Accedido el 16/04/2022. URL: https://blog.kenjo.io/es/cual-es-el-coste-de-la-empresa-al-contratar-a-un-trabajador.

[81] *Cómo instalar Ubuntu 22.04 Jammy Jellyfish en VirtualBox[2022].* Accedido el 19/07/2022. URL: https://comoinstalar.me/como-instalar-ubuntu-22-04-lts-jammy-jellyfish-en-virtualbox/.