ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Procesado de Imagen con *deep learning*

AUTOR: Raimon Blanes Saumell

TITULACIÓN: Sonido e Imagen

TUTOR: Nicolás Sáenz Lechón

DEPARTAMENTO: Ingeniería Audiovisual y Comunicaciones

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Hugo Alexer Parada

TUTOR: Nicolás Sáenz Lechón

SECRETARIO: Rubén Fraile Muñoz

Fecha de lectura: 28/10/2020

Calificación:

El Secretario,

Resumen

Este proyecto es un diseño de una aplicación que es capaz de identificar objetos con un tiempo de respuesta inmediato o prácticamente inmediato, a la vez que pretende ser una ventana que permita al lector conocer una parte de las últimas tecnologías desarrolladas en el campo de las redes neuronales convolucionales, más concretamente aquellas enfocadas en identificación mediante imágenes con tiempos de procesado bajo.

Actualmente la investigación en este tipo de redes está en alza tanto en el sector público como el privado. Cada vez son más las carreras tecnológicas que agregan este campo de conocimiento a su lista. Esto se debe principalmente a la gran eficiencia y versatilidad que el aprendizaje profundo puede brindar.

A día de hoy son varias las tecnologías que permiten alcanzar el objetivo de este proyecto, sin embargo, se ha optado por la opción que aportaba más libertad al proyecto. Tensorflow, escrito en Python y de código abierto, aporta una gran gama de posibilidades ya que dispone de muchas librerías con las que se puede complementar. Además, el autor posee mucho interés en desarrollar sus conocimientos en este entorno de trabajo.

A la hora de trabajar en el proyecto se han utilizado varias herramientas de desarrollo como Anaconda o Google Colaboratory, que han aportado gran utilidad y han reducido la carga de trabajo, sobre todo esta última ya que se usó esta plataforma como vía para entrenar la red neuronal. Proceso que requiere de un hardware de rendimiento aceptable y de mucho tiempo de computación (días).

El aprendizaje profundo busca patrones para identificar los objetos que se desea mediante imágenes que se introducen como entrenamiento, pero en ocasiones no se obtiene el resultado esperado. En estos casos es necesario volver a realizar el entrenamiento con una base de datos con un mayor número de muestras. En este proyecto aparte de contar con una gran base de datos de imágenes se ha utilizado la técnica de aprendizaje por transferencia, que reduce significativamente el tiempo de entrenamiento.

Para acercarse a las tecnologías más potentes del panorama se ha decidido usar el sistema de detección de objetos Yolov4, que fue publicado mientras se planteaban las líneas de desarrollo de este proyecto. Sin duda este sistema ha brindado una gran eficiencia al proyecto y ha aportado unos resultados muy satisfactorios.

Para hacer más accesible esta tecnología se ha desarrollado una interfaz gráfica con la que un usuario sin grandes conocimientos técnicos puede utilizar este sistema de detección. Siendo más funcional que estética, esta aplicación se ha desarrollado con la librería *tkinter* ya integrada por defecto en el lenguaje Python.

Todo lo anteriormente mencionado ha ayudado al desarrollo de una aplicación que permite la identificación de un determinado número de objetos (en concreto 10) sobre imágenes y videos. Además, los bajos tiempos de procesado han permitido que esta detección se pueda aplicar sobre el flujo de imágenes continuo que proporciona una cámara web y obtener unos resultados inmediatos y precisos.

Abstract

This project is a design of an application able to identify objects with an immediate or almost immediate response time, while it aims to be a window that allows the reader to know a part of the latest technologies developed in the field of convolutional neural networks, more specifically those focused on identification through images with low processing times.

Currently research in this type of networks is on the rise in the public and private sectors. There are more and more technological degrees that add this field of knowledge to their list. This is mainly due to the great efficiency and versatility that deep learning can provide.

Nowadays, there are several technologies that allow to reach the objective of this project, however, the option that has been chosen is the one that brings more freedom to the project. Tensorflow, written in Python and open source, provides a wide range of possibilities since it has many libraries with which it can be complemented. In addition, the author is very interested in developing his knowledge in this work environment.

When working on the project, several development tools have been used such as Anaconda or Google Collaboratory, which have been very useful and have reduced the workload, especially the latter since this platform was used as a way to train the neural network. This process requires an acceptable performance hardware and a lot of computing time (days).

Deep learning looks for patterns to identify the desired objects through images that are introduced as training, but sometimes the expected result is not obtained. In these cases, it is necessary to retrain with a database with a larger number of samples. In this project apart from having a large database of images it has used the technique of transfer learning, which significantly reduces training time.

In order to approach the most powerful technologies of the panorama, it has been decided to use the Yolov4 object detection system, which was published while the development lines of this project were being considered. Undoubtedly this system has provided great efficiency to the project and has provided very satisfactory results.

To make this technology more accessible, a graphic interface has been developed with which a user without great technical knowledge can use this detection system. Being more functional than aesthetic, this application has been developed with the tkinter library already integrated by default in the Python language.

All the above has helped the development of an application that allows the identification of a certain number of objects (specifically 10) on images and videos. In addition, the low processing times have allowed that this detection can be applied on the continuous flow of images provided by a webcam and obtain immediate and accurate results.

Índice de contenidos

Lista de acrónimos

API – Interfaz de programación de aplicaciones / *Application programming interfaces*

CPU – Unidad central de procesamiento / *Central precessing unit*

CUDA – Arquitectura unificada de dispositivos de cómputo / *Compute unified device architecture*

cuDNN - Librería de redes neuronales profundas de CUDA / *CUDA deep neural network library*

FN – Falso negativo

FP – Falso positivo

FPS – Fotogramas por segundo / *Frames per second*

GPU – Unidad de procesamiento gráfico / *Graphics processing unit*

IA – Inteligencia artificial

IsU, Iou – Intersección sobre union / *Itersection over union*

RNA – Red neuronal artificial

RNC – Red neuronal convolucional

SDK – Kit de desarrollo software / *Sofware development kit*

TPU – Unidad de procesamiento tensorial / *Tensor processing unit*

VP – Verdadero positivo

YOLO – *You only look once*

Lista de ilustraciones

Lista de tablas

Capítulo 1. Introducción y objetivos

El grupo de investigación GAMMA (Grupo de Aplicaciones Multimedia y Acústica) de la UPM se especializa en procesado de audio y entre sus líneas de investigación figuran la detección, localización y clasificación de eventos acústicos, así como las aplicaciones que se puedan derivar de ello. Entre estas aplicaciones figura por ejemplo la generación de alarmas ante sonidos determinados o imprevistos, que se implementarían por medio de sistemas autónomos con uno o varios micrófonos que captarían los sonidos ambientes, los procesarían en tiempo real y tomarían decisiones según su contenido.

Para complementar este tipo de sistemas, podría ser útil disponer además de señales visuales (imágenes captadas por una cámara, por ejemplo) que permitieran obtener más información del entorno físico en el que se produce la alarma. Este proyecto pretende ser un paso inicial en el estudio de esa posibilidad, desarrollando un sistema capaz de detectar automáticamente una serie de objetos comunes en una escena visual.

El tratamiento de imágenes tiene un largo recorrido a lo largo de la historia, los usos al igual que las técnicas han ido evolucionando de manera progresiva hasta llegar a un punto que era difícil de imaginar hace pocas décadas. Copan el mercado aplicaciones como los videojuegos de realidad virtual, los efectos especiales de grandes producciones o simples herramientas de creación para usuarios como Instagram o TikTok.

Siguiendo una senda parecida pero más exponencial se encuentra la inteligencia artificial, que deseaba brindar con la capacidad humana de adaptación y razonamiento a las máquinas. Cada vez son más los campos que tratan de incluir este tipo de tecnología ya que se encuentra en un gran estado de desarrollo y con un gran futuro por delante.

Uno de los grandes logros de la inteligencia artificial, y más concretamente del aprendizaje profundo han sido las redes neuronales, que permiten el desarrollo de funcionalidades más complejas. Dentro de este tipo de redes existen unas llamadas convolucionales que por su estructura están especialmente diseñadas para trabajar con imágenes, y es precisamente en esta vía en la que se desarrolla este proyecto.

Como guía para entender los conceptos, técnicas y tecnologías utilizados se redacta esta memoria que a lo largo de los capítulos hace una inmersión en el marco tecnológico que rodea al tratamiento de imagen y la inteligencia artificial, haciendo hincapié en los campos que engloban y en las técnicas que utilizan. Además, se acota el alcance de este proyecto y se definen las líneas de desarrollo que se han seguido.

Una vez que el lector haya obtenido el conocimiento suficiente sobre las tecnologías y los métodos que se han utilizado, se narra el proceso que se ha seguido para la elaboración del sistema que

permite identificar los objetos deseados de manera técnica y precisa para que pueda ser reproducida en un futuro.

Finalmente, se relata cómo se ha realizado la fusión de la funcionalidad de la red con una interfaz gráfica de usuario y se presentan los resultados obtenidos por el sistema de identificación de objetos, precisando en los casos particulares de cada objeto.

Para este fin se plantean una serie de objetivos: el sistema debe poder funcionar con una cámara web sin prestaciones especiales, el número de fotogramas por segundo a la salida debe permitir una visualización fluida, el sistema debe ser adaptable a dispositivos de bajo rendimiento, también debe ser capaz de aplicar la detección sobre imágenes y vídeos seleccionados por un usuario. Como objetivo secundario se propone que este proyecto sea escalable, de manera que pueda ser tomado como punto de partida para seguir desarrollando este sistema de manera rápida e intuitiva.

Capítulo 2. Marco tecnológico

2.1. Contexto histórico del reconocimiento de objetos

El reconocimiento de objetos es una de las principales aplicaciones del procesado de imágenes. Para ello se han creado numerosos algoritmos que obtienen diferentes características de imágenes dadas con las que pueden formar patrones de interpretación. Cada método tiene requisitos y restricciones distintas, y pueden ser usados para uno o varios campos de aplicación.

Se pueden encontrar ejemplos del uso del reconocimiento de objetos en diferentes ámbitos, en el industrial, por ejemplo, se usan medidores de posicionamiento que permiten situar pequeños componentes con mucha precisión, como es el caso de la fabricación de los circuitos integrados en placas de circuito impreso. En el ámbito de la seguridad, sistemas de reconocimiento facial, identificación de matrículas, control de parking o peajes… En el ámbito médico, clasificación de muestras, procesado y tratamiento de radiografías y ecografías… En el ámbito comercial, sistemas de asistencia para el aparcamiento, sistemas de navegación, sistemas de video vigilancia doméstica…

Sin embargo, a parte del ámbito de aplicación, los diferentes modelos se pueden distinguir por sus limitaciones y sus condiciones iniciales, ya sea el tiempo necesario para procesar la información de entrada, la precisión que ofrecen, la fiabilidad a la hora de clasificar objetos o la capacidad que poseen de dar resultados correctos si las condiciones de la toma de información de entrada cambian. Como es lógico, cada aplicación selecciona el método más acorde a sus necesidades; sin embargo, en ocasiones estos métodos no tienen capacidades suficientes.

Los procedimientos tradicionales para identificar objetos se basan en establecer reglas (formas, proporciones, colores...) en torno a un patrón modelo con el fin de detectar los objetos deseados en todas sus formas posibles y en diferentes circunstancias y escenarios. El principal problema que presentan este tipo de algoritmos radica en que existe una gran necesidad de particularización para cada aplicación. No existe un método lo suficientemente potente como para abarcar todos, o casi todos los usos que se le da a la identificación de objetos. Es por ello que estos métodos, a pesar de seguir usándose, han pasado a un segundo plano, mientras que la llamada “inteligencia artificial” se ha puesto a la vanguardia.

2.2. Inteligencia Artificial

En primer lugar, hay que saber de qué se está hablando cuando se habla de inteligencia artificial, ya que su significado ha ido variando a lo largo de su desarrollo. Cosas que hace 50 años se clasificaban en este campo ahora son cosas triviales que cualquier computadora de usuario estándar es capaz de realizar.

Se puede decir que se intenta plasmar el pensamiento humano en un software. Y éste no precisa de órdenes para obtener un resultado, si no que mediante la introducción de datos de entrada debe ser capaz de interpretar, procesar y finalmente obtener resultados.

Las inteligencias artificiales tienen dos características principales: La autonomía y la adaptabilidad.

La **autonomía** es la capacidad de realizar tareas en escenarios complejos sin la guía constante de un usuario.

La **adaptabilidad** es la habilidad de mejorar el desempeño de su función mediante el aprendizaje a través de la experiencia.

Estos dos pilares fundamentan lo que hoy en día se conoce como inteligencia artificial.

La IA es un campo de las ciencias de la computación, aunque también ciertas partes pueden estar igual o mejor englobadas dentro de la estadística, que se ha ido desarrollando con los años y ha dado lugar a diferentes campos muy relacionados, como se puede observar en la figura 1, que son fácilmente confundibles: Aprendizaje automático, Aprendizaje profundo, Ciencias de datos... A continuación, se van a explicar los más relevantes para este proyecto.

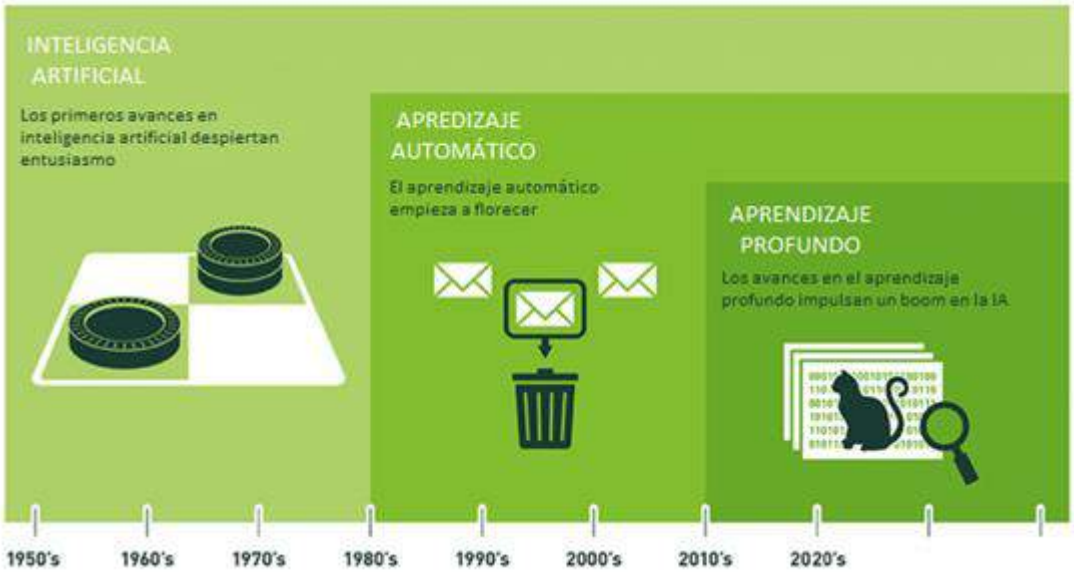

Figura 1.Cronología del desarrollo de la IA y sus campos

2.3. Aprendizaje automático o *Machine Learning*

El **aprendizaje automático** permite que los sistemas de inteligencia artificial sean adaptativos. Se podría definir como: Sistemas que mejoran su rendimiento en una tarea determinada cuanta más experiencia o datos dispongan.

El aprendizaje en estos sistemas hace referencia a la habilidad que tienen de deducir complejos patrones determinados por multitud de parámetros.

A medida que el flujo de datos llega a la interfaz de entrada, es un algoritmo propio de su programación el encargado de modificarse de manera continua. A cada dato de entrada se incrementa la efectividad y la complejidad del sistema. Podría decirse que, en cierta medida, las máquinas se programan a sí mismas.

Dentro del *machine learning* es posible distinguir 3 tipos de áreas dependiendo del tipo de problema que se vaya a atender: Aprendizaje supervisado, no supervisado y por refuerzo. Éstos siguen diferentes técnicas y tienen propósitos distintos. El primer tipo se basa en predecir la etiqueta o salida correcta dada una entrada, por ejemplo, distinguir señales de tráfico. El segundo está orientado a la detección de patrones y características similares entre los datos de entrada para formar categorías. Por último, el tercer tipo consiste en que la IA tome decisiones libremente hasta cierto punto donde se le da un *feedback* positivo o negativo mediante el cual optimiza el árbol de decisiones.

2.4. Redes neuronales artificiales y aprendizaje profundo o Deep learning

Las **RNA** tratan de imitar a su homólogo biológico, siendo un conjunto de unidades llamadas *neuronas artificiales*, interconectadas entre sí mediante enlaces para transmitir señales. Estas neuronas están organizadas en capas y toman como entrada la salida de la capa de neuronas previa, la cual es multiplicada por unos pesos y, tras una serie de operaciones, se obtiene una nueva salida.

Estas conexiones entre capas están ponderadas: a mayor valor, mayor es la influencia que tiene una capa sobre la siguiente. La red aprende progresivamente a medida que la información va atravesando la estructura neuronal, este aprendizaje se traduce en la variación del valor de los pesos.

La red neuronal más simple es la que dispone de una única capa (monocapa), pero su uso es más bien reducido. Las redes suelen estar compuestas por varias capas (multicapa), en las que, aparte de existir una capa de entrada y otra de salida, disponen de varias capas intermedias (capas ocultas) que poseen diferentes funciones.

Existen varios tipos de RNA: Redes neuronales convolucionales, redes neuronales recurrentes, redes de base radial… En este proyecto se va a utilizar una red neuronal convolucional (RNC), que son las más adecuadas cuando la entrada está compuesta por imágenes.

En este tipo de redes se restringe que solo se puedan introducir imágenes como entrada. La característica principal de las RNCs es que la disposición de las neuronas de sus capas es en tres dimensiones, que coincide precisamente con las matrices de valores que representan a las imágenes: anchura, altura y color. De esta manera se consigue un proceso más eficiente y se reduce significativamente el número de parámetros.

Estas redes se componen principalmente de un tipo de capas del mismo nombre, convolucionales. A diferencia de lo que ocurría en las capas de las RNAs estándar, las neuronas que las conforman están conectadas únicamente en una zona específica con la capa anterior, y no completamente conectadas, como se puede observar en las figuras 2 y 3[1].

Gracias a esto es posible que, dada una imagen a la entrada, se vea reducida a la salida a un vector de valores que pertenezcan a una clase específica

Este tipo de capas filtran la información de tal manera que solo se quedan con la relevante para el mapa de características, que es donde se almacena la información sobre la ubicación y la fuerza de las características relevantes detectadas en la entrada.

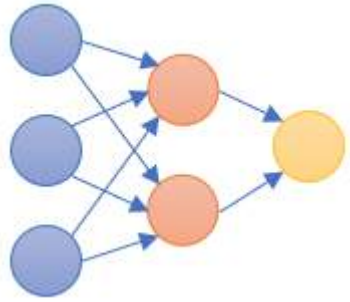

Figura 3. Estructura de capas de una RNA estándar

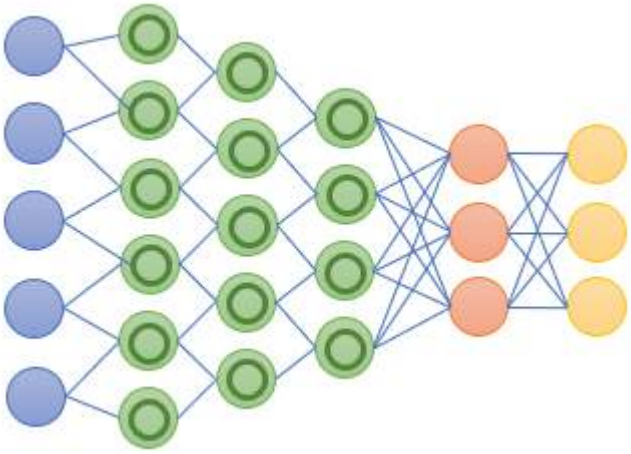

Figura 2. Estructura de capas de una RNC

Cuando las redes están compuestas por una gran cantidad de capas entra en escena el denominado **aprendizaje profundo** o *Deep learning* que es un subcampo del aprendizaje automático. Es un conjunto de técnicas y algoritmos que se caracterizan por tener una arquitectura de capas donde cada capa aprende patrones más complejos a medida que aumenta la profundidad.

Esta capacidad de poder abstraer y detectar patrones cada vez más complejos ha hecho que el uso de sistemas basados en aprendizaje profundo se haya expandido a muchos campos tales como traductores inteligentes, reconocimiento de voz, interpretación semántica, reconocimiento facial, etc.

[1] https://www.interactivechaos.com/manual/tutorial-de-machine-learning/arquitectura-de-redes-neuronales

Para que un sistema basado en aprendizaje profundo adquiera conocimiento es necesario una combinación entre un aprendizaje supervisado y otro no supervisado. En la figura 4[2] se puede apreciar cuáles serían las funciones de las distintas capas para procesar la información e interpretarla en el caso de que el sistema estuviera programado para identificar si existe algún perro en una imagen. Mediante el primer aprendizaje es el usuario quien debe hacer corresponder la información de entrada con una etiqueta adecuada, en este caso, una imagen con un perro en ella con la etiqueta de que la imagen sí contiene un perro. Y mediante el segundo, es el propio sistema el encargado de identificar sus propios patrones y analizar las relaciones entre los datos proporcionados.

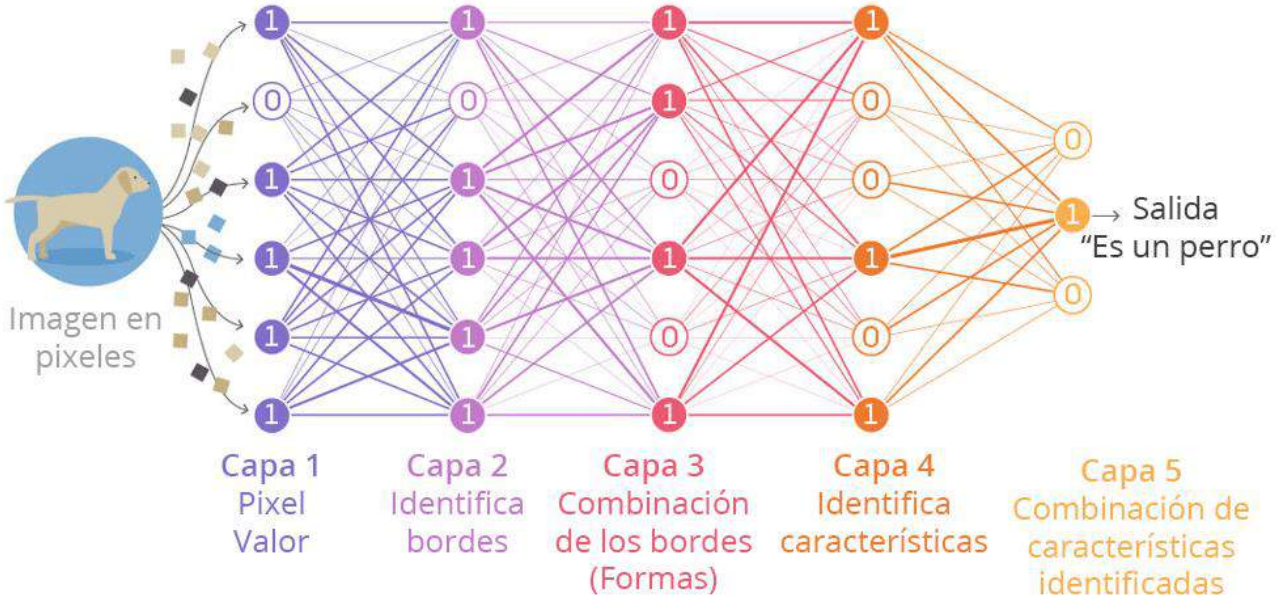

Figura 4. Esquema funcional de las capas de una red neuronal para la identificación de un perro

2.5. Comparación entre RNCs

Ya se ha hablado de la estructura de estas redes y cómo funcionan, se ha mencionado que son redes optimizadas para el tratamiento de imágenes, pero no todas las redes de detección de objetos iguales.

Para la identificación de objetos se buscan tiempos de inferencia bajos y la mayor precisión posible. Para ello los distintos desarrolladores crean redes con distinto número de capas, y cambian los parámetros de éstas, como el número de filtros, el tamaño o las dimensiones de la salida.

Para tener una red con un rendimiento óptimo también es necesario someterla a una fase de entrenamiento adecuada. Comúnmente esta fase se basa en proporcionar una gran cantidad de imágenes divididas en tres subgrupos: un conjunto de datos de *entrenamiento*, un conjunto de

[2] https://www.quantamagazine.org/

validación y un conjunto de *prueba*. El primero se utiliza junto a etiquetas, que indican la salida correcta, para que la red desarrolle sus algoritmos de identificación. Por otro lado, el conjunto de validación compara los resultados obtenidos por los distintos algoritmos desarrollados y decide cuál es el mejor. Por último, el conjunto de datos de prueba se utiliza para evaluar los resultados obtenidos por el algoritmo elegido. No obstante, este último es menos frecuente que los dos anteriores.

A continuación, se presentan algunos parámetros para medir la efectividad de estas redes:

El término **precisión** en este ámbito hace referencia a cómo de acertadas son las predicciones de la red, es decir, el porcentaje de predicciones que son correctas. Por otro lado, la **sensibilidad** o *recall* se refiere al porcentaje de aciertos de entre todos los casos verdaderos.

$$Precisión = \frac{VP}{VP + FP} \tag{1}$$

$$Sensibilidad = \frac{VP}{VP + FN} \tag{2}$$

VP (Verdadero Positivo), es la identificación de un objeto con salida correcta. FP (Falso positivo), es la identificación de un objeto con salida incorrecta. FN (Falso Negativo), es la omisión de identificación de un objeto que debería haber sido identificado.

Para una mejor comprensión de estos términos se pone como ejemplo un caso de prueba de test de COVID-19. Se realiza un ensayo clínico con 100 pacientes, de los cuales 38 presentan síntomas claros propios de este virus. Al realizar el test de prueba se detectan 40 casos positivos, de los cuales 6 son asintomáticos y no presentan anticuerpos ni en el momento del test ni en semanas posteriores, por lo que se determina que la prueba emitió 6 falsos positivos y, por otro lado, 4 personas con síntomas claros del virus dan negativo en la prueba, sin embargo, más adelante se comprueba que desarrollaron anticuerpos por lo que sí estaban infectados por el virus, por lo que la prueba generó 4 falsos negativos.

Por lo tanto, la precisión de este test sería:

$$Precisión = \frac{34}{34 + 6} \cdot 100 = 85\% \tag{3}$$

Y la sensibilidad:

$$Sensibilidad = \frac{34}{34 + 4} \cdot 100 = 89.5\% \quad (4)$$

El siguiente parámetro importante es **IsU** o *IoU* (*Intersection over Union* o Intersección sobre Unión), que mide la relación entre dos áreas, el área de solapamiento y el área de unión, que pueden verse reflejadas en la figura 5. Si la predicción de una RNC es un área donde debería encontrase el objeto, el parámetro IsU mide cuánto de la predicción coincide con el área que verdaderamente corresponde al objeto predicho. En algunas bases de datos de imágenes se definen unos umbrales de IsU para establecer si una predicción es un verdadero positivo o un falso positivo.

Como último parámetro está **AP** (*Average Precision* o precisión promedio), que es un indicador que ha ido cambiando la forma en la que se calcula dependiendo de la competición o desafío donde se ha utilizado (PASCAL VOC 2008, VOC2010-2012, COCO), pero siempre mediante los mismos parámetros, descritos previamente.

La forma de calcular que se sigue actualmente es la definida por COCO Consortium [1], creador del COCO dataset, que actualmente es un referente para la evaluación de distintas capacidades de las redes neuronales.

De forma general se conoce la precisión promedio como la tasa de acierto media que tiene un sistema con un porcentaje de Intersección sobre unión determinado. Por otro lado, existe el concepto precisión media promedio (mAP, mean average precisión) que es la media de los promedios anteriores que se obtienen al variar el porcentaje de IsU. COCO, sin embargo, considera directamente AP como la media de las tasas de acierto con IsU desde 50% al 95% en pasos de 0,05 sobre las 80 categorías de objetos que tiene el conjunto de datos de COCO.

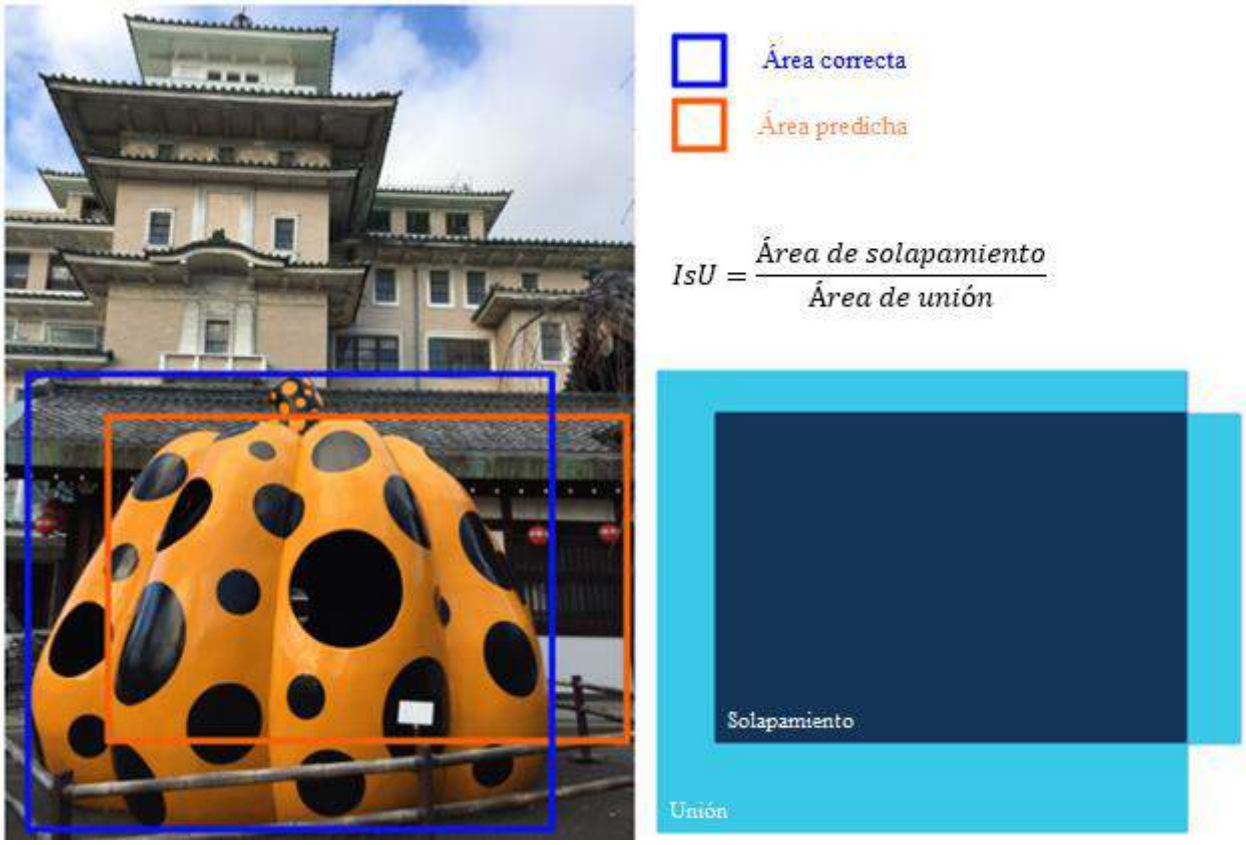

Figura 5. Representación de las áreas de solapamiento y unión y ecuación de cálculo de IsU

A la hora de presentar resultados, AP puede ir acompañado de ciertas especificaciones:

AP – Precisión media calculada según COCO

AP^{50} – Precisión media con un 50% de IsU

AP^{75} – Precisión media con un 75% de IsU

AP^{S} – Precisión media para objetos pequeños

AP^{M} – Precisión media para objetos medianos

AP^{L} – Precisión media para objetos grandes

2.6. Aprendizaje por transferencia o *transfer learning*

Es un método de *machine learning* para reutilizar un modelo diseñado para realizar una tarea como punto de partida para otro modelo con una tarea distinta, o la misma, pero con unos datos de entrenamiento diferentes, también denominado *dominio de aplicación*.

Este proceso es muy utilizado porque reduce significativamente el tiempo y la carga computacional que supone crear un modelo desde cero.

Dependiendo de si la tarea es distinta a la original y de si el dominio de aplicación varía existen diferentes estrategias y técnicas para abordar cada situación. En la figura 6 se puede ver un esquema de las diferentes estrategias que se utilizan en el *transfer learning*.

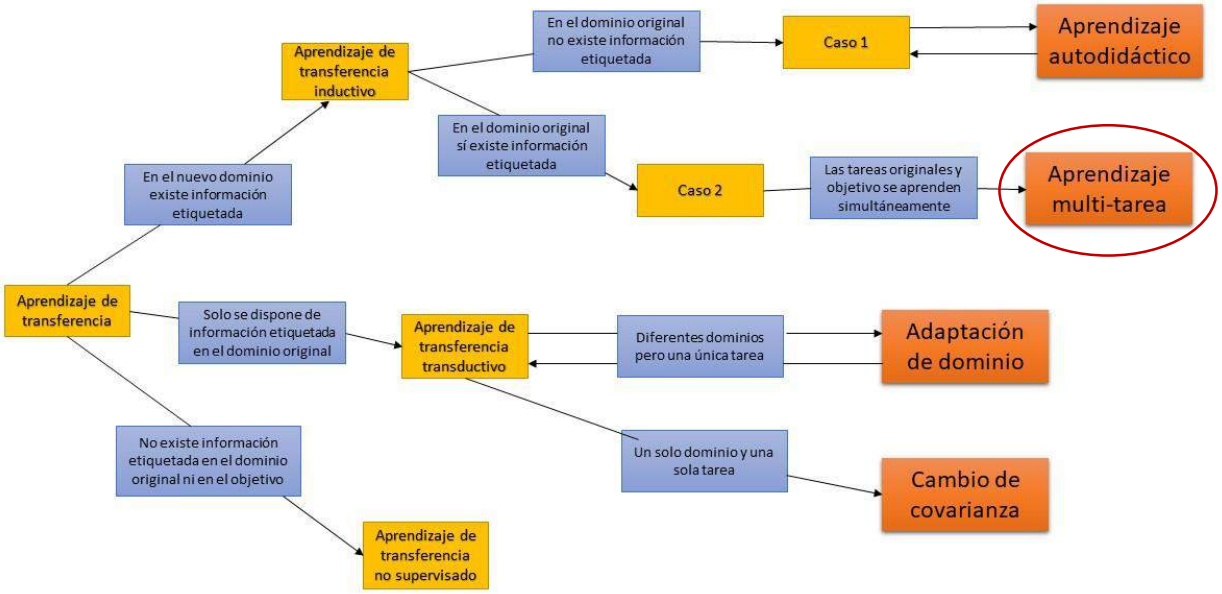

Figura 6.Estrategias de aprendizaje de transferencia

En este proyecto se va a utilizar el *aprendizaje multi-tarea* (indicado en la figura), donde la red aprende a realizar nuevas tareas encontrando diferencias y similitudes con las que tenía anteriormente y con las que aprendiendo en ese momento para optimizar el rendimiento.

Capítulo 3. Especificaciones y restricciones de diseño

A continuación, se presentan las especificaciones y restricciones del sistema de detección de objetos planteado en este proyecto.

Especificaciones de diseño

E1: La aplicación tendrá como entrada la señal de vídeo de una cámara web

E2: La aplicación podrá reconocer hasta 10 objetos distintos

E3: En escenas que contengan simultáneamente más de uno de los objetos seleccionados, éstos serán identificados de forma individual

E4: Se informará al usuario de la identificación mediante un recuadro que delimite el objeto y el nombre de éste

E5: La aplicación en su función de detección con cámara web generará una salida con unos fotogramas por segundo variables que permitan una detección fluida o semi-fluida

E6: El sistema proporcionará el índice de confianza que tienen sus predicciones y en caso de no superar un umbral, podrá no dar un resultado

Restricciones de diseño

R1: La aplicación se ejecutará en un ordenador personal estándar, sin unas prestaciones especiales, con un sistema operativo Windows.

R2: Para que la aplicación pueda realizar identificaciones correctas y fiables, son necesarias unas condiciones mínimas de iluminación y que el objeto no se encuentre muy alejado en la escena.

R3: La aplicación puede no reconocer objetos de forma clara si hay una gran cantidad de éstos o están demasiado juntos

Capítulo 4. Descripción de la solución propuesta

En este capítulo se van a presentar todas las tareas desarrolladas para alcanzar los objetivos del proyecto. En primer lugar, la sección 4.1 muestra las distintas posibilidades que se plantearon a la hora de la implementación y justifica la elección de la solución adoptada definitivamente. La sección 4.2 describe brevemente todas las herramientas seleccionadas para el desarrollo del proyecto y su interacción entre sí. En la sección 4.3 se da cuenta de la creación y selección del conjunto de imágenes (*dataset*) necesario para entrenar el sistema de clasificación desarrollado. La sección 4.4 muestra cómo se ha realizado el entrenamiento del sistema, combinando las tecnologías presentadas en la sección 4.2 y el banco de imágenes de la 4.3. Por último, la sección 4.5 se dedica a la aplicación final con la que se puede realizar un reconocimiento de objetos en un entorno desconocido a partir de la red entrenada y una cámara web, así como otras funciones añadidas.

4.1. Líneas de desarrollo estudiadas

En este apartado se consideran dos opciones tecnológicas para llevar a cabo el proyecto, se presentan sus ventajas y sus inconvenientes y, finalmente, se selecciona una de ellas.

4.1.1. Matlab Deep Learning toolbox

En primer lugar, se estudia la posibilidad de utilizar el conjunto de herramientas que proporciona Matlab para trabajar con aprendizaje profundo, dado que el autor posee experiencia en el uso de este software como herramienta de ingeniería durante sus estudios universitarios.

Matlab proporciona un *framework* para el diseño e implementación de redes neuronales profundas a través de algoritmos, aplicaciones y modelos entrenados previamente [2]. Posee la capacidad de utilizar distintos tipos de redes como las convolucionales y de memoria de corto-largo plazo, o diferentes regresiones. También es posible crear, editar y analizar arquitecturas de red avanzadas. Destaca por sus funciones de tratamiento de imágenes, los módulos de redes neuronales instalados en las últimas versiones y el módulo de computación paralela que permite el uso de GPU para acelerar los cálculos.

En esta línea de desarrollo, el objetivo es utilizar un modelo de red neuronal convolucional pre-entrenado para reutilizarlo como base para el detector de objetos mediante *transfer learning*. El sistema se compone de dos dispositivos: una cámara web destinada a captar imágenes que compongan la entrada de vídeo y un ordenador encargado de procesar la información y ejecutar la RNC para obtener los resultados.

Para la captura de vídeo a través de la cámara web se utilizaría la herramienta de Mathworks® *Image Acquisition Toolbox* [3] para conectar el hardware necesario con Matlab.

Como puntos fuertes de esta opción destacan; su capacidad computacional, la facilidad de depuración de código, y la disponibilidad de herramientas dedicadas a tareas específicas. Por otro lado, presenta ciertas desventajas: no es de código abierto, lo que limita la compatibilidad entre otras plataformas, y es un lenguaje interpretado, lo que afecta negativamente al rendimiento.

Cabe destacar que actualmente sus desarrolladores se encuentran actualizando periódicamente sus productos para un mayor rendimiento y para que sean compatibles con librerías de código abierto.

4.1.2. Tensorflow + Darknet

Por otro lado, se contempla la opción de utilizar Tensorflow junto a Darknet como motor principal del proyecto. La red neuronal se entrenaría mediante Darknet y se ejecutaría mediante Tensorflow.

En primer lugar, **Darknet** [4] es un *framework* especializado en redes neuronales de código abierto escrito en C y CUDA, presenta una gran rapidez y alta precisión en detección de objetos en tiempo real. Estas cualidades se deben al lenguaje en el que está escrito [5].

Fue creado por el mismo desarrollador que el sistema de detección en tiempo real *You Only Look Once* (YOLO) del que se hablará más adelante. Darknet está optimizado para trabajar con este sistema, por ello no hay mejor opción para llevar a cabo el entrenamiento.

Por otro lado, **Tensorflow** [6] fue creado por el equipo de *Google Brain* [7], equipo de investigación de inteligencia artificial de aprendizaje profundo, y es una librería de código abierto para computación numérica y aprendizaje automático a gran escala. Reúne una gran gama de modelos y algoritmos de aprendizaje automático y aprendizaje profundo para ponerlos al servicio del usuario mediante una API de Python, lenguaje con el que el autor posee experiencia.

También permite crear, entrenar y desarrollar redes neuronales con distintas funcionalidades. Gracias a su arquitectura flexible es posible configurar una o varias unidades de CPU/GPU para la computación, bien sea en equipos locales o servidores externos.

Al estar desarrollado en Python, Tensorflow aporta muchas funcionalidades, la más importante para este proyecto es la captura de vídeo, que se realiza mediante OpenCV [8], una biblioteca de código abierto muy utilizada en todo lo que se refiere a tratamiento y gestión de imágenes o vídeo.

Una gran ventaja de esta línea de desarrollo es que es posible entrenar la red neuronal mediante una plataforma en la nube (Google Colaboratory), de tal manera que no es necesario aplicar una gran carga computacional durante un tiempo prolongado en el equipo, ni una computadora con grandes prestaciones, ya que es la plataforma la que provee el hardware y el software necesarios.

4.1.3. Selección de la línea de desarrollo

La implementación de cualquiera de las dos líneas de desarrollo anteriores aporta una serie de características y prestaciones muy interesantes para el proyecto. Por ello, de cara a elegir una de las opciones se ha priorizado la facilidad de encontrar documentación y contenido didáctico para una rápida y mejor compresión de la tecnología y así poder crear un proyecto sólido y con las cualidades necesarias. Además, se ha tenido en cuenta la versatilidad y la escalabilidad que se quiere brindar a este proyecto.

Ambas opciones disponen de mecanismos para la resolución de dudas o problemas. Matlab pone a disposición del usuario un equipo de soporte que puede dar soluciones precisas y técnicas. Por otro lado, Tensorflow dispone de una gran comunidad donde se puede encontrar mucha información práctica y de interés que puede ayudar a solventar todo tipo problemas. Darknet no posee mucha documentación, pero al ser un *framework* muy especializado aporta la información suficiente para hacerlo funcionar sin problemas.

En cuanto al presupuesto, Matlab dispone de una licencia dedicada a los estudiantes y al personal de investigación que pertenezcan a universidades suscritas, permitiendo su uso gratuito siempre y cuando se utilice para un ámbito no comercial. En cambio, Tensorflow y Darknet son totalmente gratuitos y de uso libre.

Como se ha descrito anteriormente, las dos herramientas tienen mucho potencial y tienen las capacidades suficientes para llevar a cabo el objetivo del proyecto. Al no existir desventajas técnicas claras que influyan negativamente en el desarrollo o implementación proyecto ni ninguna otra carencia en los criterios comentados anteriormente en ninguna de las dos opciones, finalmente se ha decidido utilizar Tensorflow y Darknet al ser software libre y por elección personal del autor.

4.2. Entorno de desarrollo

El proyecto se va a desarrollar en un sistema *Windows 10 home* con un procesador *Intel Core i7-7700HQ* y sobre una tarjeta gráfica *NVIDIA GeForce GTX 1050*.

A parte de Tensorflow, se utilizan otras herramientas, algunas de ellas no son imprescindibles, pero son de gran utilidad para el desarrollo del proyecto. En primer lugar, Anaconda, que aporta Python como lenguaje de programación y la posibilidad de generar entornos virtuales de trabajo, *CUDA Toolkit*, que habilita el modo GPU de tensorflow, Yolov4, sistema de detección de alto rendimiento, y Google Colaboratory, una plataforma de servicio en nube.

4.2.1. Anaconda

Anaconda es una distribución libre y de código abierto de Python dedicada a las ciencias de la computación (Aprendizaje automático, ciencia de datos, análisis predictivo, etc), que pretende simplificar la gestión de paquetes y su implementación [9].

Contiene 1500 paquetes seleccionados del repositorio de Python, cuyas versiones son gestionadas por el sistema de gestión de paquetes *conda*. Este sistema es muy práctico debido a que a diferencia del gestor de paquetes estándar de Python *pip*, al instalar un nuevo paquete y sus dependencias, comprueba previamente si existe algún conflicto.

A la distribución le acompaña una interfaz gráfica de usuario llamada *Anaconda Navigator* (figura 7), que permite ejecutar aplicaciones y gestionar paquetes, entornos virtuales y canales sin usar comandos de línea.

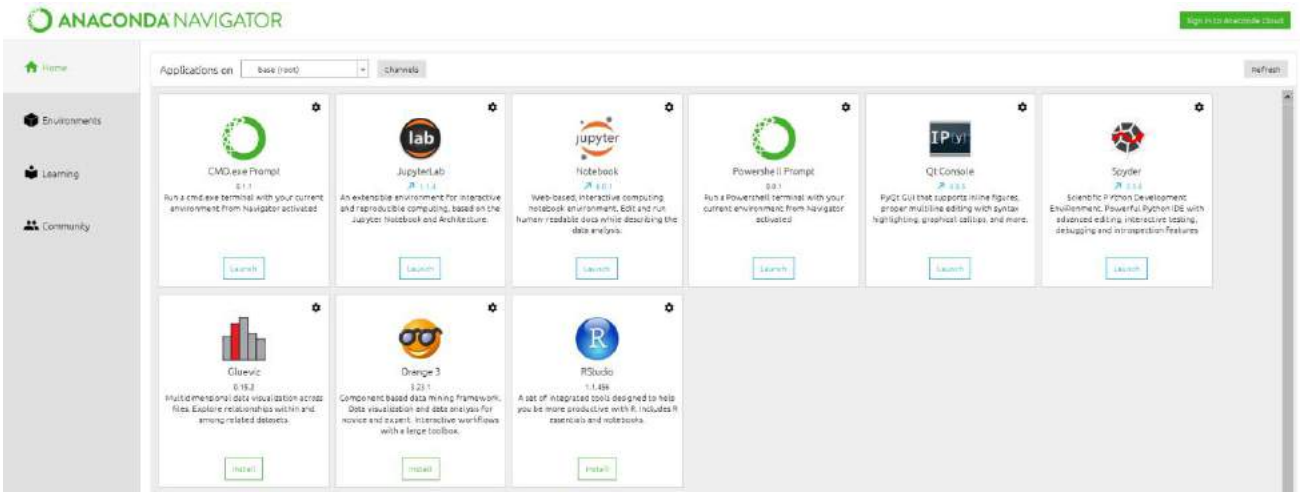

Figura 7. Interfaz gráfica Anaconda Navigator

4.2.2. CUDA Toolkit y Tensorflow en modo GPU

CUDA (*Compute Unified Device Architecture*) o Arquitectura unificada de dispositivos de cómputo es una plataforma de computación en paralelo y un modelo de API creado por Nvidia® que permite utilizar las unidades de procesado gráfico como unidades de procesado general [10].

La diferencia principal frente a las CPUs es que el procesado es en paralelo y no secuencial, lo que incrementa de forma muy significativa la velocidad en ciertos procesos.

Las versiones de CUDA se han ido sucediendo desde su lanzamiento en 2007. Para implementarlas hay que consultar previamente la versión correspondiente a la tarjeta gráfica (de la misma compañía) que se esté utilizando [11]. Con cada versión se está tratando de abarcar el mayor número de dispositivos posible. La tarjeta gráfica empleada en este proyecto pertenece a la micro-arquitectura Pascal 6.1, que es soportada por las versiones 10.0 a10.2 de CUDA SDK, se va a utilizar la versión 10.1 por tener una mejor compatibilidad con la versión de Tensorflow utilizada.

Por otro lado, dentro de la API de CUDA, se encuentra cuDNN [12], que es una librería de aceleración de GPU para redes neuronales profundas, que mejora el rendimiento de los *frameworks* de aprendizaje profundo.

Estos dos componentes son imprescindibles para habilitar el modo GPU de Tensorflow. De lo contrario se usaría el modo por defecto (CPU) ralentizando la velocidad del sistema de detección.

4.2.3. Yolov4

YOLO (*You Only Look Once*) es un sistema de detección de objetos en tiempo real desarrollado por Joseph Redmon, Santosh Divvala, Ross Girshick y Ali Farhardi, cuya primera versión fue publicada en 2016 [13].

El objetivo de estos investigadores era desarrollar uno de los sistemas de detección de objetos más competitivos del panorama tecnológico. Para ello introdujeron técnicas que en aquel momento no eran estándar en la detección de objetos. En lugar de utilizar clasificadores abogaron por la predicción de cajas delimitadoras y las probabilidades de clase de imágenes completas en una sola evaluación. Consiguieron así una arquitectura muy veloz manteniendo una precisión elevada comparada a la de otros detectores.

Yolov3 [14] fue la tercera versión de este sistema, publicada en 2018 con nuevas mejoras logrando así la velocidad de procesado más alta de entre todos sus homólogos del momento (Figura 8[3]). Este hecho provocó el reconocimiento de este sistema como uno de los más punteros y se hizo conocido.

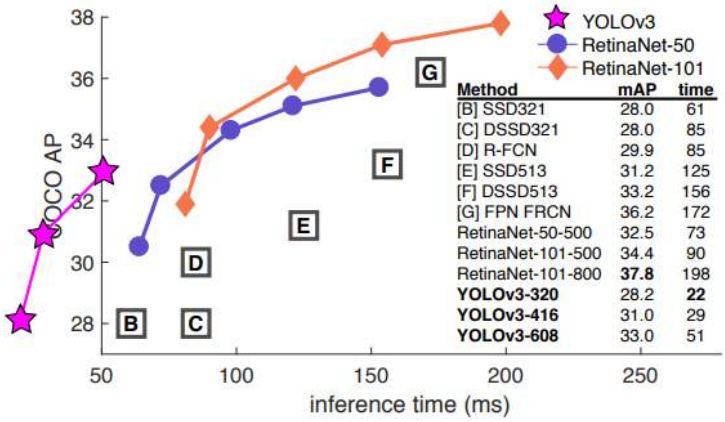

Figura 8. Gráfica comparativa de los sistemas de detección de objetos más punteros en 2018, medidos sobre la base de datos de imágenes de COCO

Chien-Yao -Wang y Hong-Yuan Mark Liao del *Institute of Information Science Academia Sinica* de Taiwán tomaron el testigo y el 23 de abril de 2020 publicaron la siguiente y, hasta la fecha,

[3] https://pjreddie.com/media/files/papers/YOLOv3.pdf

última versión del sistema, Yolov4 [15], que volvió a ser un referente en cuanto a la relación precisión/fotogramas (Figura 9[4]).

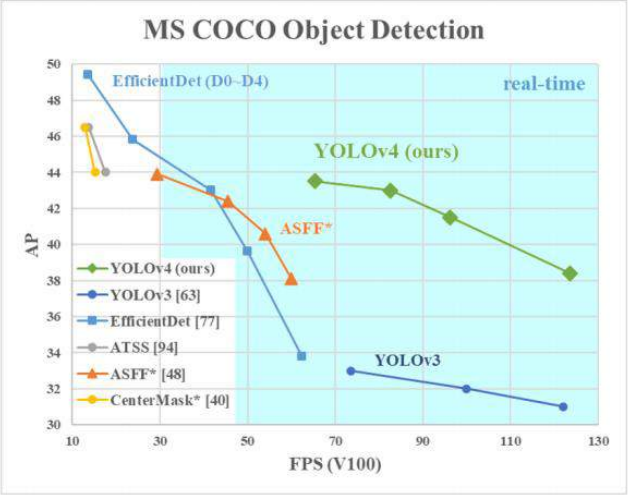

Figura 9.Gráfica comparativa de los sistemas de detección de objetos más modernos en abril de 2020

Gran parte del éxito de este sistema es debido a la red neuronal creada por los mismos desarrolladores llamada Darknet-53, que posee 53 capas convolucionales y cuya estructura es la que se puede ver en la figura 10[5]. Comparado con otras redes, la cantidad de capas es considerablemente menor, y esto es clave para garantizar una velocidad de procesado mucho mayor y clasificar así más imágenes por segundo.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 10. Estructura de la red Darknet-53

[4] https://arxiv.org/pdf/2004.10934.pdf
[5] https://pjreddie.com/media/files/papers/YOLOv3.pdf

En la tabla 1 se pueden observar los valores de precisión que brinda el sistema Yolov4:

Tabla 1. Imágenes por segundo y precisión del sistema Yolov4 ejecutado sobre un dispositivo GPU tipo PASCAL[6]

Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	54 (P)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	43 (P)	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4	CSPDarknet-53	608	33 (P)	**43.5%**	**65.7%**	**47.3%**	**26.7%**	**46.7%**	53.3%

Cabe destacar que los valores de imágenes por segundo (FPS) que se muestran en la tabla han sido obtenidos utilizando una versión de GPU concreta, lo que quiere decir que estos valores son válidos para un conjunto de tarjetas gráficas determinadas (los valores de precisión, sin embargo, no varían).

El ámbito de la detección en tiempo real comenzó con la adaptación de los clasificadores; por ello es posible que existan sistemas de detección de objetos más precisos pero que requieran de un mayor tiempo de procesado, no consiguiendo así resultados instantáneos. Actualmente Yolov4 se sitúa entre las redes más potentes con respecto a ambos parámetros.

Ya que en este proyecto no se cuenta con dispositivos de alto rendimiento, Yolov4 se considera el sistema idóneo para conseguir una detección de objetos con considerable fiabilidad y una fluidez suficiente.

Como se ha comentado anteriormente, Darknet llevará a cabo el entrenamiento de este sistema, mientras que su ejecución (en diferentes versiones) será realizada por scripts en Python mediante el uso de Tensorflow, estos scripts serán desarrollados por el autor mediante las herramientas y entornos virtuales que aporta Anaconda. Por otro lado, CUDA permitirá que el sistema tengo un rendimiento óptimo habilitando la GPU como unidad de procesado; de no ser posible se habilitará una versión menos precisa, pero con menor carga computacional.

4.2.4. Google Colaboratory

También conocido por su abreviación Google Colab [16], es una herramienta de desarrollo para la docencia e investigación de *machine learning*. Está basado en *Jupyter notebooks*[7], permite la escritura y ejecución de código de Python sin necesidad de ninguna configuración previa.

[6] Incluye los modelos Titan X, Titan Xp, GTX 1080 Ti y Tesla P100 GPU

[7] Es una aplicación web de código abierto que te permite crear y compartir documentos que contienen código que puede ejecutarse en la nube

Los notebooks se guardan en Google Drive o pueden ser cargados desde Github[8]. La ejecución se realiza sobre recursos ofrecidos por Google, suele recibir el nombre de computación en nube o *cloud*.

Este servicio es gratuito y tiene sus limitaciones. El usuario puede elegir entre una unidad de procesado TPU[9] o GPU. Automáticamente se asignará un dispositivo libre dependiendo de la demanda, la prioridad y el uso previo que haya realizado el propio usuario.

Si el cliente necesita más prestaciones existe una opción premium de pago llamada *Colab Pro* que proporciona recursos con mayor capacidad. También es posible conectar esta plataforma con servicios de computación en nube de pago. Estos servicios son más concretos y especializados: en este caso el usuario elige todos los elementos hardware que necesite y el coste se calcula en base al tiempo de uso de los recursos seleccionados. Las empresas punteras en este campo son *Google Cloud Services*, *Amazon Web Services* y *Microsoft Azure*, siendo Amazon la más importante y con mayor experiencia y Google la más orientada hacia el aprendizaje automático.

Google Colaboratory resulta una herramienta muy útil para este proyecto de cara a la gran carga computacional durante un tiempo prolongado que requiere el entrenamiento de la red neuronal.

4.3. Creación del dataset

Los conjuntos de imágenes para el entrenamiento y la validación de la red neuronal se han extraído de *Open Images Dataset V6* [17], que es un banco de imágenes libres creado por Google. Contiene más de 9 millones de imágenes de distintos objetos (o combinaciones de ellos) con distintas anotaciones como cajas delimitadoras, máscaras de segmentación, interacciones visuales…

Como este proyecto se centra en la detección de objetos se utiliza el conjunto de imágenes con cajas delimitadoras. A pesar de la gran cantidad de imágenes, para ciertos objetos no existe un número de muestras suficiente para entrenar la red de forma adecuada, por ello se escogen 10 objetos que dispongan de al menos de 3000 imágenes.

Se podría haber elegido cualquier elemento para comprobar el funcionamiento de la red, sin embargo, se han seleccionado objetos que se pueden encontrar de forma habitual.

Los objetos elegidos son:

[8] Es una plataforma de almacenamiento de código para el control de versiones y el desarrollo colaborativo
[9] Unidad de procesamiento tensorial, es un circuito integrado especializado para acelerar procesos de IA desarrollado por Google. Está enfocado principalmente en el aprendizaje automático de redes neuronales

Tabla 2. Objetos seleccionados para ser identificables por la red neuronal de este proyecto

Objetos	
Persona	Teléfono móvil
Teclado (de ordenador)	Silla
Botella	Libro
Manzana	Plátano
Naranja	Taza

Finalmente, el conjunto de imágenes de entrenamiento dispone de 30000 imágenes.

Por otro lado, la cantidad de imágenes conveniente para una validación adecuada es del 20-30% del tamaño del conjunto de entrenamiento, por desgracia, *Open Images Dataset V6* no cuenta con suficiente cantidad de imágenes, por lo que se extraen el máximo de ilustraciones disponibles por clase hasta un máximo de 600, quedando una cantidad total de 4468 imágenes.

El proceso de obtención del *dataset* se detalla a continuación:

- Se accede a la web del repositorio[10] y se accede a la opción "*explore*" (Figura 11)

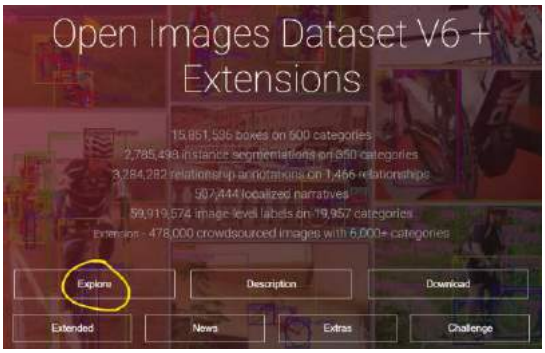

Figura 11. Página principal del repositorio de imágenes Open images Dataset V6

- En la siguiente página se selecciona que el tipo de imágenes que se desea es para detección (Figura 12)

[10] https://storage.googleapis.com/openimages/web/index.html

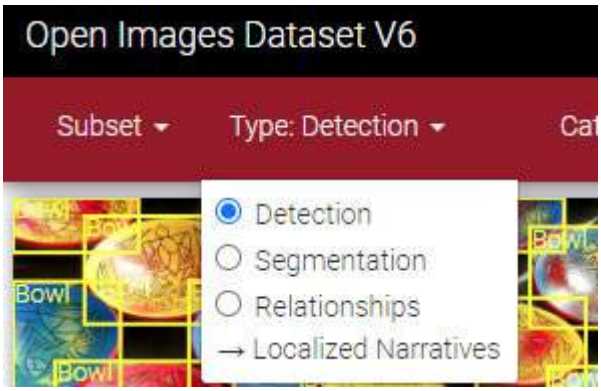

Figura 12. Selección del tipo de imágenes deseadas en el banco de imágenes Open Images Dataset V6

- A continuación, se consulta en el buscador todos los objetos disponibles (Figura 13), cabe destacar que no se puede filtrar por número de imágenes, es posible que de algún objeto haya muy pocas imágenes disponibles, sobretodo de validación.

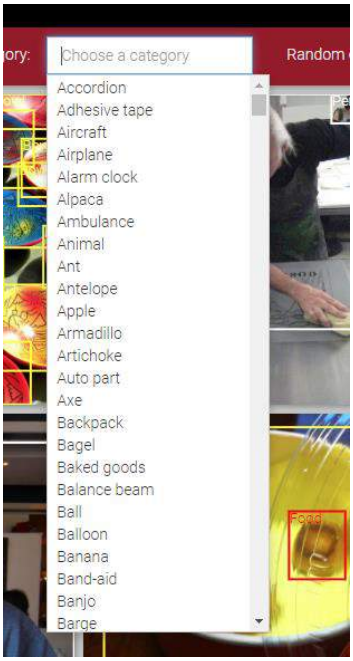

Figura 13. Buscador de objetos del repositorio de imágenes Open Images Dataset V6

- Una vez se sabe qué objetos se van a usar, hay que descargar las imágenes. Este proceso se va a realizar mediante la herramienta *OIDv4_Toolkit* [18], este kit está alojado en GitHub, de donde se pude descargar gratuitamente. Además, para facilitar este proceso, se obtendrá un script llamado "*convert_annotation.py"* desarrollado por *TheIAGuy* [19] que ayudará a adaptar las anotaciones de las imágenes al formato que sigue Yolov4.

- Se accede a la carpeta donde se encuentra la herramienta de descarga, se abre el intérprete de comandos y se introduce *"pip install –r requirements.txt"* para instalar las librerías necesarias para este procedimiento.

- Cuando hayan acabado de instalarse las librerías se debe que introducir el siguiente comando: "*main.py downloader - -classes*" seguido del nombre de los objetos que se desea descargar, seguido de "- *-type_csv train*" para descargar las imágenes destinadas al entrenamiento, seguido de "- *-limit*" con el número de imágenes máximas que se desean descargar, y, por último, "- *-multiclasses 1*" esta opción hará que todas las imágenes se descarguen en la misma carpeta ayudando así a la organización del *dataset*. En la figura 14 se puede observar un ejemplo de cómo sería el comando completo.

```
main.py downloader --classes Person Apple Orange Bottle Chair Computer_keyboard --type_csv train --limit 3000 --multiclasses 1
```

Figura 14. Comando de ejemplo para la descarga de imágenes de entrenamiento

- Una vez acabe la descarga se deberá introducir el mismo comando cambiando la opción "*train*" por "*validation*" indicando así que se quieren descargar imágenes de validación, y cambiando el límite de imágenes (Figura 15).

```
main.py downloader --classes Person Apple Orange Bottle Chair Computer_keyboard --type_csv validation --limit 600 --multiclasses 1
```

Figura 15. Comando de ejemplo para la descarga de imágenes de validación

- A continuación, se debe modificar el archivo "*clases.txt*" introduciendo el nombre de los objetos descargados (en inglés), separados por saltos de línea.

- Después de guardar el archivo se ejecuta el script "*convert_annotations.py*".

- Finalmente, cuando termine de convertir las anotaciones, se podrá eliminar la carpeta "*label*" que acompaña a las imágenes y el *dataset* estará listo para ser comprimido y usado.

El tipo de imágenes que se pueden encontrar en este repositorio y que, en particular, se han utilizado en el entrenamiento de este proyecto pueden verse en las siguientes figuras:

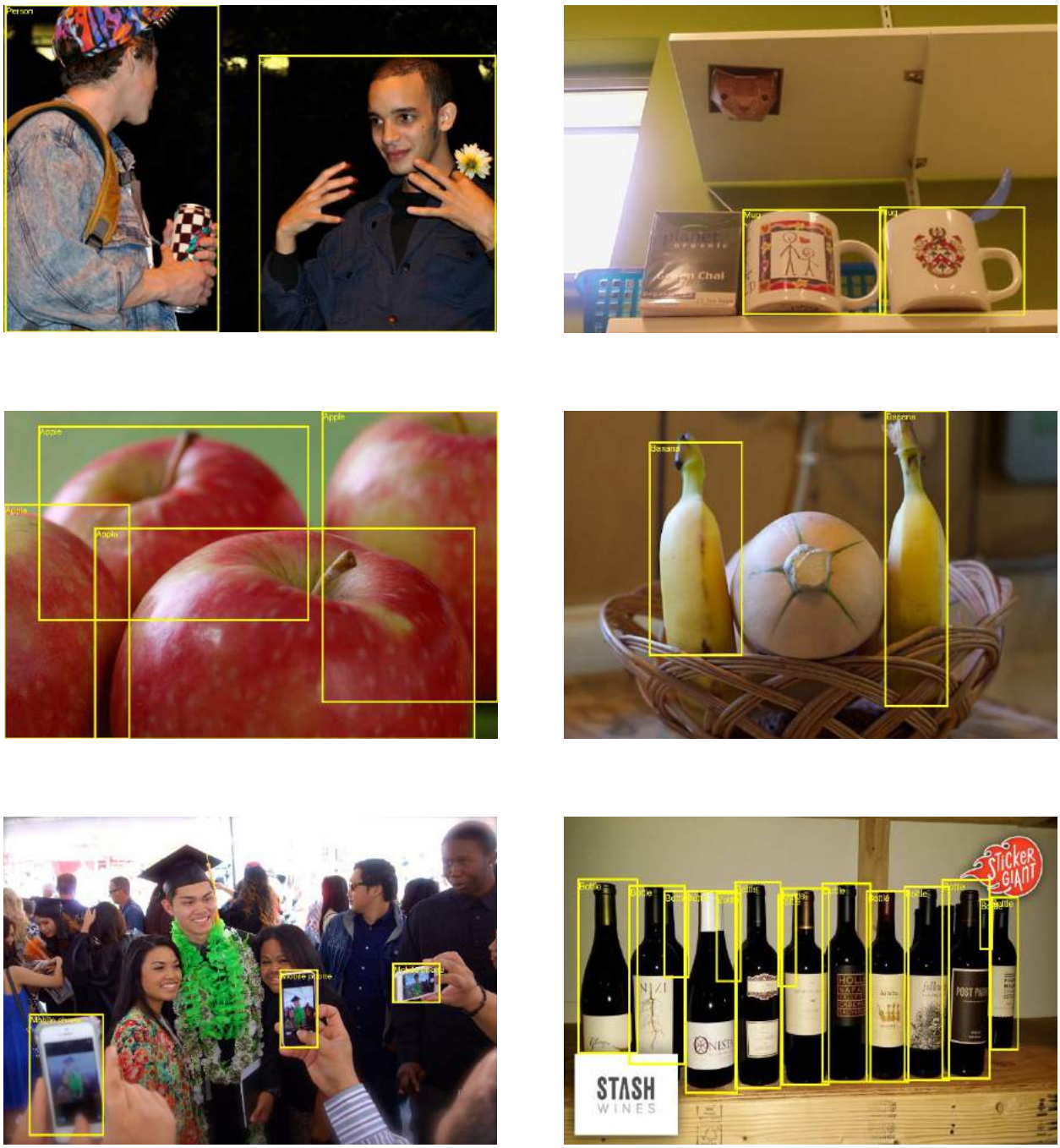

Figura 16. Imágenes extraídas de Open Images Dataset V6 y utilizadas en el entrenamiento de la red neuronal

4.4. Entrenamiento de la red neuronal

Para que una red neuronal aprenda a detectar objetos es necesario introducir como datos de entrada imágenes que contengan objetos y la información de dónde se encuentran exactamente. Esta información viene dada por cuatro vértices que forman una caja que delimita el objeto en cuestión.

Como se comentó en apartados anteriores, cuanta más información se introduzca más precisos serán los resultados. Por ello, en este caso, aun omitiendo la fase de validación la red seguiría siendo bastante precisa. Sin embargo, se opta realizar el proceso de validación para obtener mejores resultados.

Para el entrenamiento se ha seguido un *notebook* de Google Colab [20] redactado por *The IA guy* que ha seguido las indicaciones del repositorio oficial de Yolov4 y Darknet [21] . A continuación, se traduce y se sintetiza parte del contenido del mismo:

- En primer lugar, como punto de partida, se deben subir los siguientes archivos a Google Drive en una carpeta llamada "Yolov4":
    - "Obj.zip": carpeta comprimida con las imágenes de entrenamiento
    - "Test.zip": carpeta comprimida con las imágenes de validación
    - Los scripts "generate_train.py" y "generate_test.py" que se pueden encontrar en el repositorio de *TheIAGuy* [22]
    - Una carpeta llamada "backup"

- Se abre el notebook de Google Colab y se habilita la GPU como acelerador del hardware

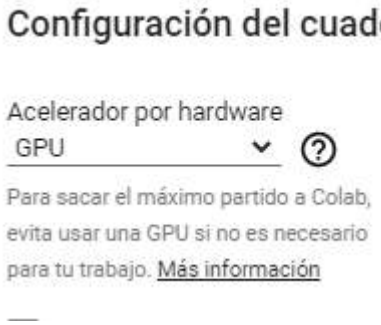

Figura 17. Captura de la configuración de Entorno de ejecución de un notebook de Google Colab

- Se clona el repositorio de Darknet mediante git[11]

[11] Es un sistema de control de versiones de código abierto

Figura 18. Comando en Google Colab para la descarga del repositorio de GitHub de AlexeyAB

- Se cambian ciertas variables para habilitar el modo de entrenamiento

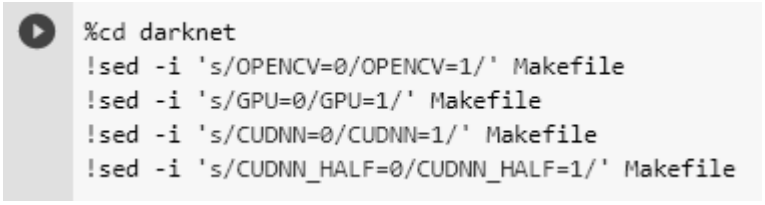

Figura 19. Comandos en Google Colab para modificar un fichero y habilitar el modo entrenamiento

- Se compila Darknet para que se pueda usar el archivo ejecutable para realizar el entrenamiento

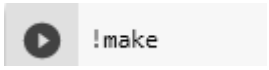

Figura 20. Comando en Google Colab para la compilación de Darknet

- Se enlaza el repositorio temporal de Google Colab con Google Drive

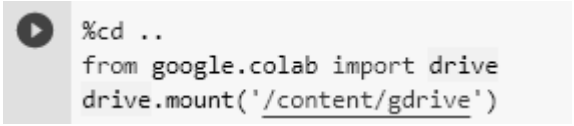

Figura 21. Comando en Google Colab para sincronizar Google Drive con el almacenamiento de Colab

- Se copian del drive la carpeta comprimida que contiene las imágenes de entrenamiento “obj.zip” y la carpeta comprimida que contiene las imágenes de validación “test.zip”

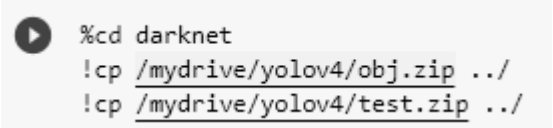

Figura 22. Comandos en Google Colab para copiar los ficheros “obj.zip” y “test.zip” del Drive al Colab

- Se descomprimen dichas capetas en el directorio “data”

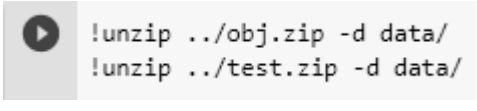

Figura 23. Comandos en Google Colab para descomprimir los archivos “obj.zip” y “test.zip”

- Se copia a Google Drive el archivo de configuración "yolov4-obj.cfg"

```
!cp cfg/yolov4-custom.cfg /mydrive/yolov4/yolov4-obj.cfg
```

Figura 24. Comando en Google Colab para copiar el archivo "yolov4-obj.cfg" de Colab al Drive

- Se modifica dicho archivo de la siguiente manera para obtener un entrenamiento óptimo:
  - Las dimensiones ancho y alto (*width, height*) deben tener el valor 416
  - El parámetro *max_batches* debe ser igual al número de clases (número de objetos) que se quiere entrenar multiplicado por 2000, en el caso de 10 clases, 20.000.
  - El parámetro *steps* debe tener un primer valor igual al 80% de *max_batches* y un segundo valor igual al 90% de *max_batches*, en este caso 16.000 y 18.000.
  - Por último, el número de filtros (*filters*) debe ser el número de clases más cinco, y multiplicado por 3, en este caso 45.
- A continuación, se copia el archivo de configuración modificado de vuelta a la carpeta de Google Colab

```
!cp /mydrive/yolov4/yolov4-obj.cfg ./cfg
```

Figura 25. Comando en Google Colab para copiar el archivo "yolov4-obj.cfg" del Drive al Colab

- Se crea un archivo de texto llamado "obj.names" que contenga los nombres de las clases, uno por línea

obj.names

```
1 Persona
2 Manzana
3 Teclado
4 Silla
5 Libro
```

Figura 26. Contenido de ejemplo del fichero "obj.names"

- Se crea otro archivo de texto llamado "obj.data" con el siguiente contenido:

obj.data

```
1 classes = 10
2 train = data/train.txt
3 valid = data/test.txt
4 names = data/obj.names
5 backup = /mydrive/yolov4/backup
```

Figura 27. Contenido de ejemplo del fichero "obj.data"

- Se copian los archivos anteriores en la carpeta “data”

```
!cp /mydrive/yolov4/obj.names ./data
!cp /mydrive/yolov4/obj.data  ./data
```

Figura 28. Comandos en Google Colab para copiar los archivos “obj.names” y “obj.data” del Drive a Colab

- Se copian los archivos “generate_train.py” y “generate_test.py” a Google Colab

```
!cp /mydrive/yolov4/generate_train.py ./
!cp /mydrive/yolov4/generate_test.py ./
```

Figura 29. Comandos en Google Colab para copiar los scripts “generate_train.py” y “generate_test.py” del Drive al Colab

- Se ejecutan dichos archivos

```
!python generate_train.py
!python generate_test.py
```

Figura 30. Comandos en Google Colab para la ejecución de los scripts “generate_train.py” y “generate_test.py”

- Se descargan los pesos pre-entrenados para las capas convolucionales. De esta manera se aplica el aprendizaje por transferencia obteniéndose mejores resultados y reduciendo el tiempo de computación

```
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137
```

Figura 31. Comando en Google Colab para la descarga de unos pesos pre-entrenados de Yolov4

- Finalmente se lleva a cabo el entrenamiento

```
!./darknet detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv.137 -dont_show -map
```

Figura 32. Comando en google Colab para realizar el entrenamiento de la red neuronal

Cada 100 iteraciones los pesos se guardarán en la carpeta “*backup*” por si por cualquier motivo el entrenamiento se detiene, y de esta manera retomar el entrenamiento desde un punto más avanzado y no empezar de nuevo.

4.5. Aplicación final

La aplicación final se ha desarrollado con la biblioteca *tkinter* [23] de Python y consiste en una sencilla interfaz gráfica mediante la cual se puede utilizar el sistema de identificación de objetos.

Como se puede ver en la figura 33, la aplicación está compuesta por 3 bloques principales:

El **bloque de instalación**, que configura el software necesario para poder usar el sistema.

El **bloque de detección con cámara web**, que permite la identificación de objetos sobre el vídeo capturado por la cámara predeterminada del equipo y la visualización del resultado de manera instantánea (Especificación E1).

La opción de "Alto rendimiento" hace referencia al modo GPU, se recomienda instalar CUDA y CUDnn si se quiere utilizar esta opción, de lo contrario la tasa de imágenes por segundo será muy baja.

La opción de "Bajo rendimiento" requiere de mucho menos procesado y provee de una tasa aceptable de imágenes por segundo incluso sin tener habilitado el modo GPU, es decir, utilizando la CPU del equipo.

Se ha habilitado un mecanismo para tomar capturas mientras se está utilizando esta opción (leer manual).

El **bloque de detección de ficheros** está destinado a aplicar la identificación de objetos sobre una imagen o vídeo que esté almacenado en el equipo.

Por último, en la zona inferior derecha se pueden observar tres botones que corresponden a tres funciones distintas:

El primer botón "Establecer" define el umbral del índice de confianza; es decir, en los resultados solo se mostrarán las detecciones que superen ese valor

El botón "Manual de Uso" muestra una nueva ventana con un resumen de las funcionalidades de las que dispone la aplicación.

Y, por último, el botón "Cerrar aplicación" termina con la ejecución de la aplicación y cierra todas las ventanas que tengan relación con ella.

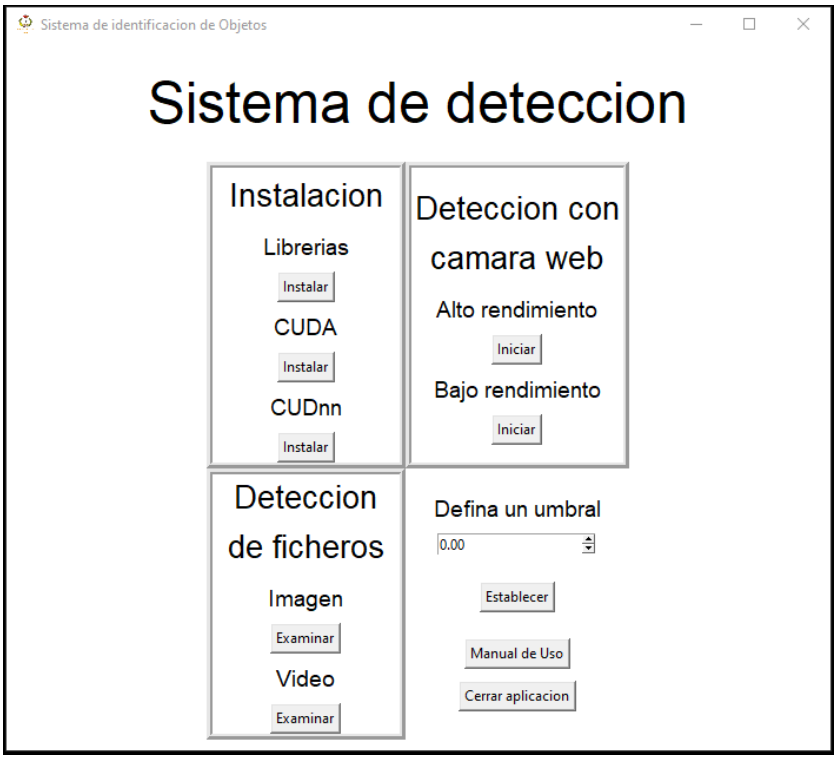

Figura 33. Captura de la aplicación del sistema de identificación de objetos desarrollado en este proyecto

Capítulo 5. Resultados

5.1. Resultados del entrenamiento

Para evaluar los resultados del entrenamiento se ha utilizado el comando de la figura 34, que usa el *dataset* de validación para calcular la precisión de la identificación de objetos.

```
[ ]  !./darknet detector map data/obj.data cfg/yolov4-obj.cfg /mydrive/yolov4/backup/yolov4-obj_last.weights
```

Figura 34. Comando en Google Colab para evaluar la precisión de los pesos obtenidos

En un primer lugar se realizó un entrenamiento que constaba de 7 objetos, con una base de datos de imágenes de aproximadamente 2000 imágenes por clase para el entrenamiento y de un 20% de esa cantidad para la validación. Se obtuvieron los resultados de la tabla 3.

Tabla 3.Resultados de la precisión del primer intento de entrenamiento de la red

Objeto	AP	VP	FP
Libro	38,64%	284	451
Botella	35,29%	192	352
Silla	28,66%	290	722
Bebida	29,62%	186	442
Fruta	42,57%	716	966
Teléfono móvil	76,40%	123	38
Persona	37,80%	561	923

En total, se obtuvo una precisión del 38%, y una sensibilidad del 45%, con una media de IsU de 30,52%. Estos datos junto a una prueba de la aplicación con estos pesos, determinó que el rendimiento del sistema de detección no era suficiente.

Por consiguiente, se volvió a realizar el entrenamiento con los datos y el procedimiento descritos en el apartado 4.4. Se aumentó el número de imágenes por clase y se decidió ampliar y cambiar algunas clases hasta un total de 10 (Especificación E2).

Los resultados obtenidos del nuevo entrenamiento son los reflejados en la siguiente tabla:

Tabla 4. Resultados del segundo entrenamiento de la red neuronal

Objeto	AP	VP	FP
Botella	45,52%	488	584
Libro	41,95%	451	624
Manzana	38,79%	327	516
Naranja	42,07%	361	497
Persona	38,76%	854	1349
Plátano	44,81%	367	452
Silla	33,17%	469	945
Taza	56,53%	398	306
Teclado	46,96%	193	218
Teléfono móvil	89,57%	249	29

En total se obtuvo una precisión media del 47% y una sensibilidad de 52%, con una media de IsU del 37%. En comparativa se puede observar una clara mejora en el rendimiento de la aplicación. Se prueba a utilizar el sistema de detección a través de la cámara web y se comprueba que el sistema trabaja de manera suficientemente adecuada como para dar el entrenamiento como válido.

Es importante destacar que la suma de VP y FP no coincide con el número de imágenes de validación porque en cada imagen puede haber más de un objeto de la misma clase (véase la figura 16).

5.2. Resultados de la aplicación

En este apartado se va a comprobar la efectividad del sistema a través de la aplicación teniendo en cuenta las diferentes salidas que proporciona, individualizando para cada objeto la mayoría de los casos particulares y excepciones posibles. También se comprobará su rendimiento en modo CPU y GPU.

Para cada detección la aplicación dibuja una caja delimitadora que trata de abarcar todo el objeto. Dicha caja cambia de color según el objeto y va acompañada del nombre de la clase y del índice de confianza o seguridad, éste es un valor entre 0 y 1 que indica la certeza que tiene el sistema de que el objeto en cuestión pertenece a la clase mostrada (Especificación E4 y E6).

- Botella

La complejidad de identificar una botella viene dada sobre todo por dos factores, la transparencia de la mayoría de ellas, como suelen ser las botellas de agua de plástico o vidrio, y, por otro lado, sus múltiples formas.

En la figura 35 se puede observar que la transparencia de las botellas no es un impedimento para su detección. El principal motivo es porque la red neuronal se centra más en los bordes a la hora de detectar, por ello se obtienen puntuaciones de confianza muy altas.

Figura 35. Imagen de botellas procesada por el sistema de identificación de objetos

También, en la figura 36 se obtienen índices de confianza muy altos a pesar de que las botellas son de diferentes formas y colores. Como es lógico, los objetos que no se vean en su forma completa tendrán un índice más bajo, en este caso la botella de color marrón rojizo del centro de la imagen tiene un índice que no supera el umbral y por ello su caja no está representada.

Figura 36. Imagen de botellas procesada por el sistema de identificación de objetos

En la figura 37 se encuentra un caso particular con el que el sistema de identificación de objetos tiene grandes conflictos, son los grupos de objetos apilados de forma masiva sin ningún orden. En este tipo de casos una de las posibles salidas es la que se ve en la imagen, el sistema detecta algunos de los objetos.

Éstos suelen ser los objetos más visibles o los que se pueden apreciar de forma íntegra, sin embargo, al estar tan juntos unos objetos de otros es posible que un contorno que una persona pueda detectar fácilmente sea interpretado por el sistema como una continuación del objeto, considerando así que la forma que está evaluando no coincide con ninguna de las que puede identificar. Es por ello que a pesar de que en la imagen se ven botellas claramente, los índices de las botellas detectadas con bastante bajos.

Figura 37. Imagen de botellas procesada por el sistema de identificación de objetos

- **Libro**

En esta clase se encuentran varios casos particulares muy interesantes de comentar. Primeramente, en la figura 38 se puede apreciar una gran caja delimitadora que abarca gran parte de imagen, esta es otra posible salida para el caso en el que el sistema se encuentra una gran cantidad de objetos de la misma clase muy próximos y no es capaz de localizar bien los contornos.

Por otro lado, en la parte central izquierda de la imagen se visualizan 4 detecciones correctas o parcialmente correctas. Este resultado indica que el difuminado afecta negativamente a la identificación de objetos, pero no la imposibilita.

Por último, los libros que se encuentran en primer plano son reconocidos con un grado de confianza bastante considerable, a pesar de que estén apilados sus contornos son bastante claros, lo que permite al sistema trazar unas cajas delimitadoras con precisión.

Otra casuística se encuentra en la figura 39, donde la identificación es correcta pero la distribución es tan compleja para el sistema que dibuja cajas delimitadoras erróneas.

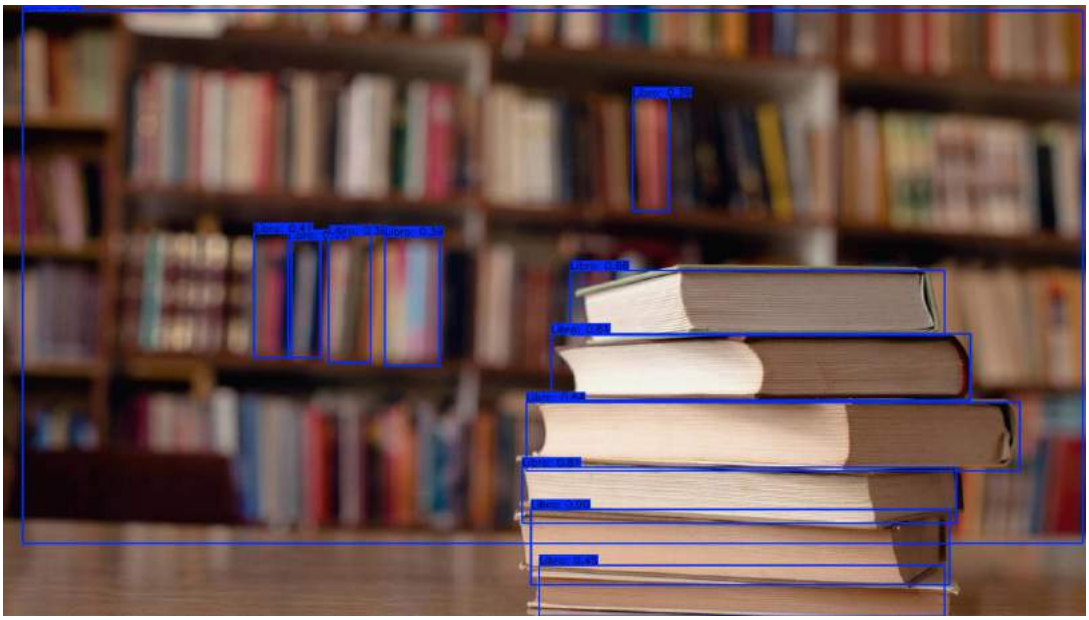

Figura 38.Imagen de libros procesada por el sistema de identificación de objetos

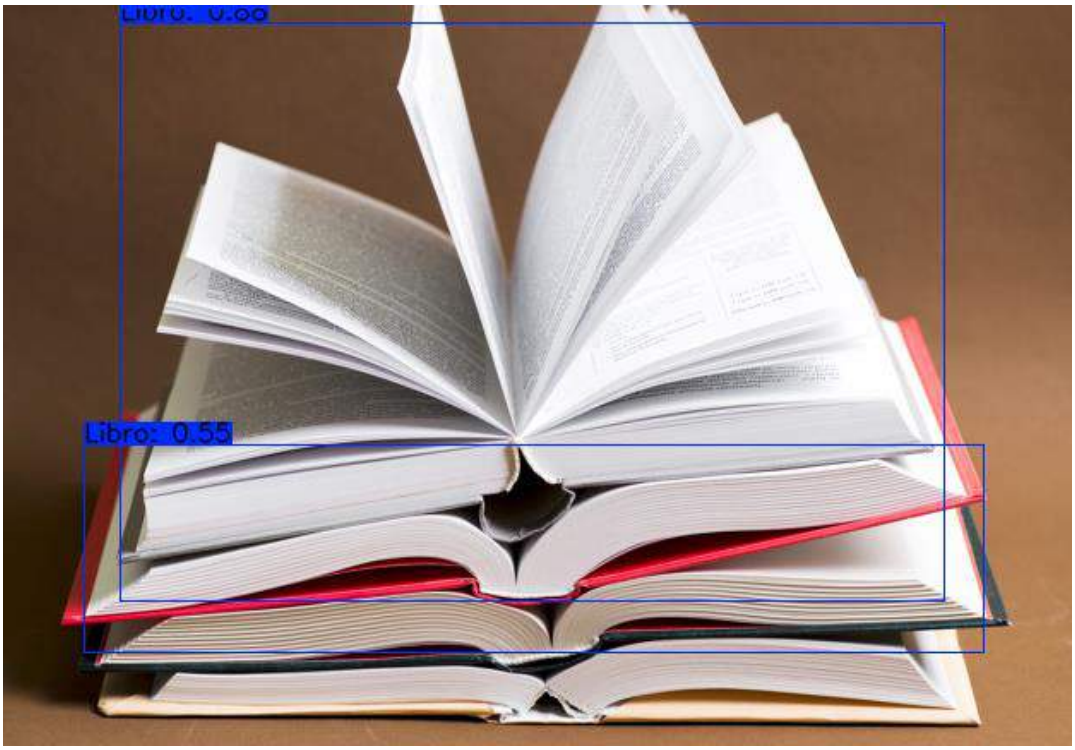

Figura 39. Imagen de libros procesada por el sistema de identificación de objetos

Como último ejemplo de esta clase se encuentra la figura 40, en esta ocasión aparecen tres cuadernos apilados de los cuales dos son identificados como libros. Este error es común en muchos objetos por dos motivos, ambos tienen una apariencia visual muy similar y, por otra parte, la red no ha sido entrenada para identificar cuadernos. Es por esto que el sistema no ha tenido la oportunidad de elaborar patrones para distinguir entre un objeto u otro, de haber sido así

seguramente también existirían falsos positivos, pero tendrían un índice de confianza considerablemente menor.

Figura 40. Imagen de cuadernos procesada por el sistema de identificación de objetos

- **Manzana, naranja y plátano**

Se ha decidido englobar a estas clases ya que las tres son frutas y presentan casos particulares similares.

En las figuras 41 y 42 se verifica que, aunque el tipo de fruta y su estado de maduración varíen se siguen identificando correctamente y con cajas delimitadoras bastante precisas.

Figura 41. Imagen de manzanas procesada por el sistema de identificación de objetos

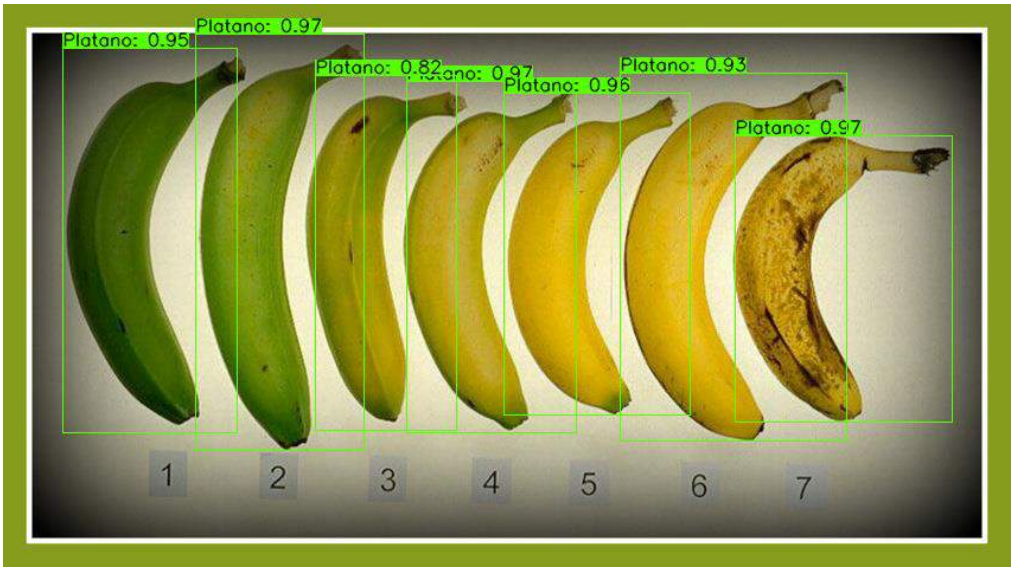

Figura 42. Imagen de plátanos procesada por el sistema de identificación de objetos

En las figuras 43 y 44 se aprecia un error parecido al del libro y el cuaderno, en ambas imágenes aparecen melocotones (rojos y amarillos respectivamente) y son identificados como manzanas.

Una nueva observación a destacar es la identificación del melocotón que se encuentra más arriba en la figura 44, donde el índice de confianza es mucho más alto(0.83) que el de las otras dos identificaciones(0.49 y 0.54). Al contrario que en identificaciones anteriores donde el objeto no se veía de forma íntegra y por ello tenía un índice de confianza más bajo, en este caso la omisión de parte del objeto (precisamente la zona donde existe mayor diferencia entre las dos frutas) ayuda a la detección de un falso positivo

Figura 43. Imagen de melocotones procesada por el sistema de identificación de objetos

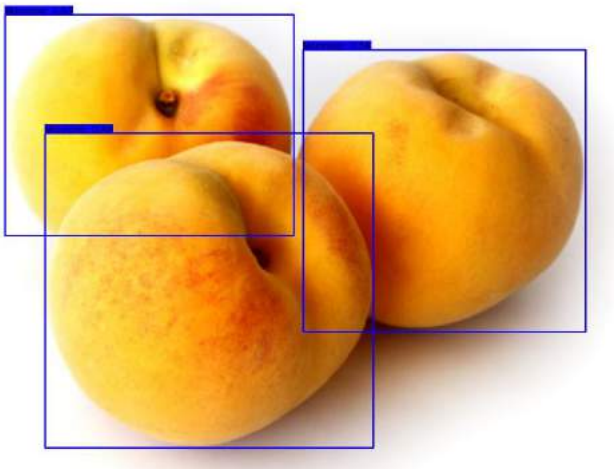

Figura 44. Imagen de melocotones procesada por el sistema de identificación de objetos

- **Persona**

En esta clase no existen casuísticas particulares que comentar. En la figura 45 se verifica que se identifican todo tipo de personas independientemente de su raza, y en la figura 46 que se traza una buena delimitación mediante cajas menos los casos en los que los cuerpos están muy juntos.

Figura 45. Imagen de personas procesada por el sistema de identificación de objetos

Figura 46. Imagen de personas procesada por el sistema de identificación de objetos

- **Silla**

En la figura 47 se observa una buena detección de sillas además de que la clase "silla" engloba también al taburete. En la figura 48 se comprueba que no solo detecta sillas clásicas. Como imagen final, en la figura 49 se observa una silla estilo *gaming* reclinada que ha sido procesada con un umbral de confianza de 0.1 y el sistema no ha sido capaz de identificarla. Esto es debido a que la posición concreta de la silla no encaja con el patrón de reconocimiento del sistema.

Figura 47. Imagen de sillas procesada por el sistema de identificación de objetos

Figura 48. Imagen de sillas gaming procesada por el sistema de identificación de objetos

Figura 49. Imagen de silla gaming reclinada procesada por el sistema de identificación de objetos sin éxito

- **Taza**

En este caso en la figura 50 se observa una buena detección en una imagen con tazas, detectando incluso la taza con forma cuadrada (aunque con índice de confianza bajo – 0.40 –).

Figura 50. Imagen de tazas procesada por el sistema de identificación de objetos

Por otro lado, en la figura 51 se puede observar como el sistema detecta como “taza” a tazones y vasos. Esto se debe principalmente a que la forma del objeto taza suele variar poco de una a otra, y para dar versatilidad a la red la base de datos de entrenamiento contenía una gran variedad de tazas con formas y disposiciones singulares. Eso sin duda alguna ayuda a la detección de todas las tazas posibles, pero también propicia la detección de falsos positivos.

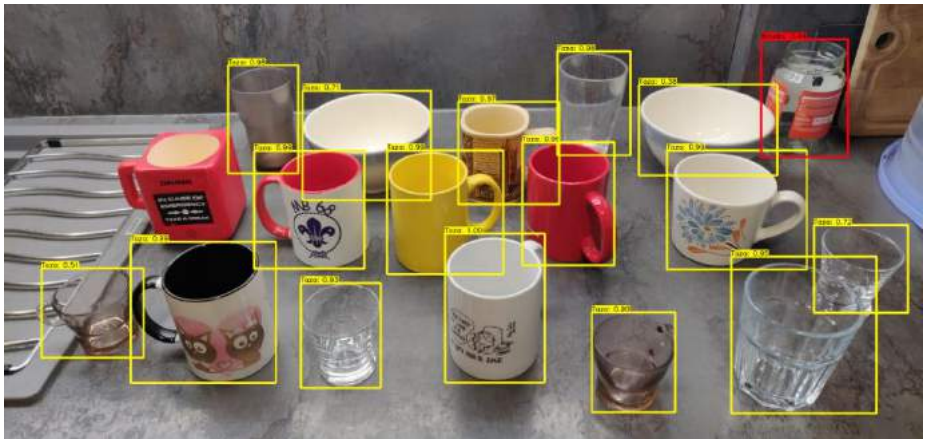

Figura 51. Imagen de tazas, vasos y tazones procesada por el sistema de identificación de objetos

- **Teclado**

Este objeto no presenta ninguna particularidad que destacar. En la figura 52 se comprueba una verificación bastante certera. Como se especificó en el apartado 4.3 la clase “teclado” hace referencia a teclados de ordenador, por ello el sistema no identifica como “teclado” un piano ni similares.

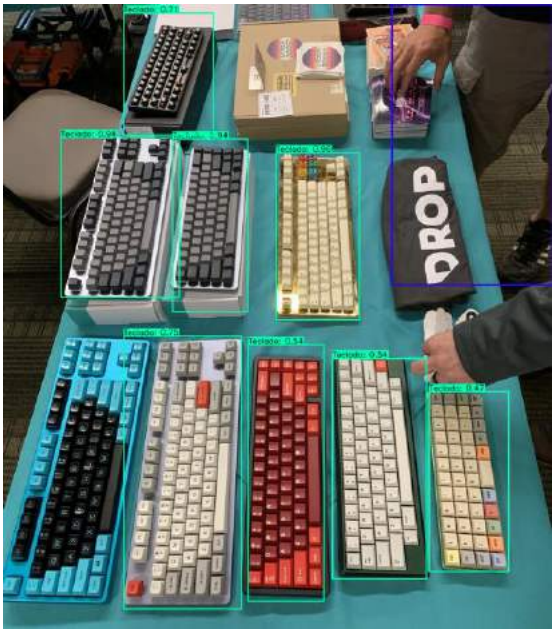

Figura 52. Imagen de teclados de ordenador procesada por el sistema de identificación de objetos de manera exitosa menos el teclado de color azul y negro

- **Teléfono móvil**

En la figura 53 se puede observar como se realizan unas detecciones correctas para esta clase teniendo en cuenta móviles de varias generaciones y distintos modelos.

La forma característica que tienen los teléfonos móviles permiten que sean fácilmente identificables, es por ello que en el apartado 5.1 el porcentaje de precisión en esta clase es tan elevada. Sin embargo, esta particularidad presenta una gran desventaja en el caso concreto de los smartphone. La apariencia de estos suele ser de una pantalla rectangular, con o sin bordes, negra cuya única variabilidad es que aparezca un reflejo. Esto ocasiona que el sistema pueda generar falsos positivos con objetos de apariencia similar como carteras, manos a distancia, discos duros… (Figura 54)

Figura 53. Imagen de móviles procesada por el sistema de identificación de objetos

Figura 54. Imagen de una cartera procesada por el sistema de identificación de objetos

- **Multi-objeto**

En este apartado se muestra la identificación que logra el sistema sobre imágenes con varios objetos de distintas clases (personas, sillas y botellas) en la misma imagen (Especificación E3):

Figura 55. Imagen con varios objetos identificados por el sistema de detección de objetos

- **Rendimiento de la aplicación (fotogramas por segundo)**

Como se ha comentado a lo largo del proyecto, la aplicación dispone de un modo llamado de alto rendimiento que hace referencia a la ejecución del sistema de identificación de objetos utilizando la GPU (en este caso una tarjeta gráfica NVIDIA GTX 1050) como procesador. Con este equipo se han obtenido los fotogramas de la figura:

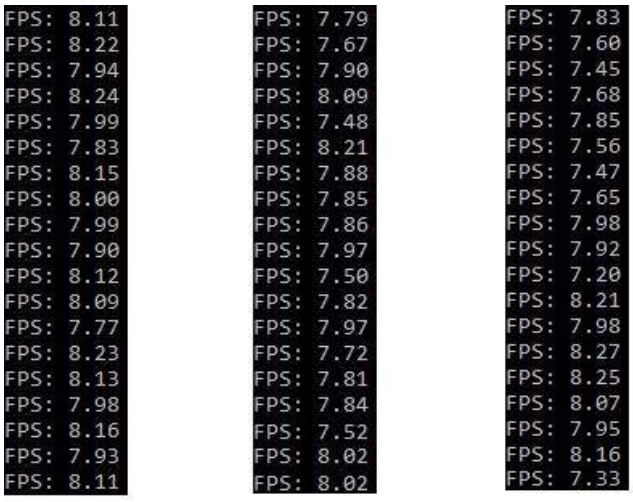

Figura 56. Captura de la consola que muestra los registros de la aplicación ejecutándose en modo Alto rendimiento, entre ellos los fotogramas por segundo que procesa

Por otro lado, el modo de bajo rendimiento se refiere al uso de una versión del sistema menos precisa pero de computación más rápida. Esta rapidez permite que sea viable ejecutar el sistema con una CPU como procesador, las imágenes por segundo obtenidas para este caso son los de la figura:

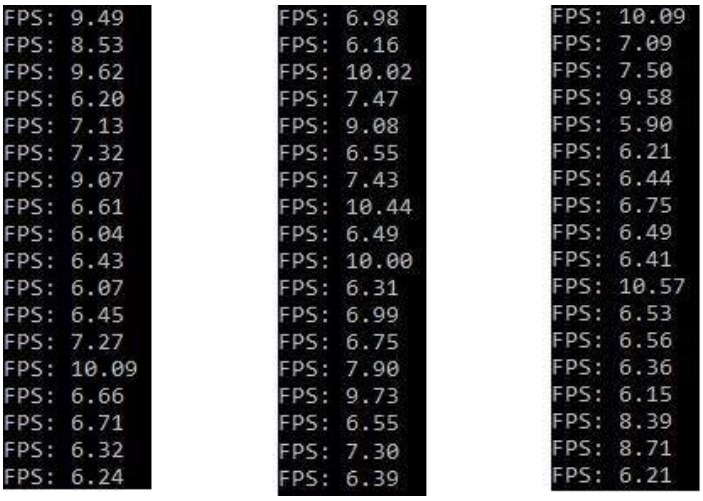

Figura 57. Captura de la consola que muestra los registros de la aplicación ejecutándose en modo Bajo rendimiento, entre ellos los fotogramas por segundo que procesa

Ambas opciones permiten una visualización fluida (o prácticamente fluida) de los resultados como se pretendía en la especificación E5.

Capítulo 6. Presupuesto

En este apartado se presenta una tabla con el desglose de los costes asociados a los recursos que han sido necesarios para la elaboración de este proyecto:

Recurso	Precio	Unidades	Precio total
Personal de desarrollo	9,95€/h	300	2.925,00 €
Anaconda	Gratuito	X	0,00 €
Tensorflow	Gratuito	X	0,00 €
CUDA Toolkit	Gratuito	X	0,00 €
Plataforma Google Colab	Gratuito	30h	0,00 €
Yolov4	Gratuito	X	0,00 €
Portátil Asus i7-7700HQ GTX 1050	900 €	1	900,00 €
Cámara web Logitech C270	50 €	1	50,00 €
Presupuesto total			3.875,00 €

El coste relativo a los recursos humanos se ha calculado a partir del documento "Convenio colectivo del sector de empresas de ingeniería y oficina de estudios técnicos" [24] considerando un trabajador de nivel 2.

Capítulo 7. Conclusiones

Con el fin de complementar un sistema de detección, localización y clasificación de objetos mediante sonidos se ha logrado desarrollar un sistema de identificación de objetos mediante imágenes obtenidas por una cámara web, habilitando así una posible fusión de estos dos sistemas.

Se puede afirmar que se han cumplido todos los objetivos planteados en la introducción; en primer lugar, el uso de una cámara web estándar como es la que se ha utilizado en este proyecto (Logitech C270), la obtención de una salida con un número de imágenes por segundo suficientes para crear una visualización con fluidez (en el modo de detección con cámara web), la habilitación de un modo de bajo rendimiento destinado a dispositivos con prestaciones técnicas bajas, y, por último, la implantación de funciones complementarias para el procesado de imágenes y videos almacenados.

Todo ello ha sido posible gracias a las tecnologías empleadas, haciendo especial hincapié en Darknet que ha permitido un entrenamiento de la red eficiente, y Tensorflow, que ha sido el *framework* encargado de ejecutar el sistema y que, al estar desarrollado en Python, ha posibilitado un desarrollo más sencillo de la interfaz gráfica y la complementación con otras librerías.

Ambas herramientas son de código abierto y libres. Esto sumado a la gran cantidad de información técnica y guiada que se aporta en los apartados anteriores verifica el objetivo secundario propuesto de desarrollar un proyecto escalable y de fácil asimilación.

Mediante el uso de la aplicación desarrollada se han obtenido unos resultados satisfactorios, las identificaciones son bastante precisas teniendo en cuenta la complejidad que ello conlleva, al igual que el trazado de cajas delimitadoras, con las que el sistema suele tener dificultades cuando existen objetos muy próximos entre sí.

Como líneas de mejora de este sistema se encuentra principalmente la ampliación del conjunto de entrenamiento y validación, bien extrayendo imágenes de bancos de imágenes o elaborando propias. También, en un futuro se publicará la siguiente versión o una modificación del sistema de detección YOLO con nuevas características que se podrían adaptar a este proyecto. Además, existe una comunidad activa de desarrolladores (tanto profesionales como amateur) que publican sus pequeñas aportaciones en distintas plataformas, las cuales pueden servir de apoyo para nuevas mejoras o prestaciones.

Este proyecto se ha realizado con la idea de que cada día son más las aplicaciones tecnológicas que tratan de aprovechar las redes neuronales y el *Deep learning* a su favor, funcionalidades como las desarrolladas en este proyecto son la realidad de un futuro cercano que tendrá gran relevancia en el ámbito tecnológico y. por consiguiente, en el económico y social.

Capítulo 8. Referencias

[1] «COCO Dataset,» [En línea]. Disponible: https://cocodataset.org/#home. [Último acceso: 11 08 2020].

[2] Mathworks, «Mathworks,» [En línea]. Disponible: https://es.mathworks.com/products/deep-learning.html. [Último acceso: 2 06 2020].

[3] Mathworks, «Mathworks,» [En línea]. Disponible: https://es.mathworks.com/help/imaq/. [Último acceso: 2 06 2020].

[4] J. Redmon, «pjreddie.com,» [En línea]. Disponible: https://pjreddie.com/darknet/. [Último acceso: 22 08 2020].

[5] «benchmarksgame-team,» [En línea]. Disponible: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python3-gcc.html. [Último acceso: 4 09 2020].

[6] Google LLC, «Tensorflow,» [En línea]. Disponible: https://www.tensorflow.org/. [Último acceso: 3 06 2020].

[7] Google LLC, «Research Google,» [En línea]. Disponible: https://research.google/teams/brain/. [Último acceso: 03 06 2020].

[8] OpenCV Team, «OpenCV,» [En línea]. Disponible: https://opencv.org/. [Último acceso: 04 06 2020].

[9] Anaconda Inc., «Anaconda,» [En línea]. Disponible: https://www.anaconda.com/. [Último acceso: 04 06 2020].

[10] NVIDIA Corporation, «Developer NVIDIA,» [En línea]. Disponible: https://developer.nvidia.com/cuda-toolkit. [Último acceso: 6 07 2020].

[11] «Wikipedia,» [En línea]. Disponible: https://en.wikipedia.org/wiki/CUDA. [Último acceso: 26 08 2020].

[12] NVIDIA Corporation, «Developer NVIDIA,» [En línea]. Disponible: https://developer.nvidia.com/cudnn. [Último acceso: 16 07 2020].

[13] S. D. R. G. A. F. Joseph Redmon, «Arvix.org,» 9 05 2016. [En línea]. Disponible: https://arxiv.org/abs/1506.02640. [Último acceso: 07 06 2020].

[14] J. Redmon, «pjreddie.com,» [En línea]. Disponible: https://pjreddie.com/media/files/papers/YOLOv3.pdf. [Último acceso: 5 06 2020].

[15] C.-Y. W. H.-Y. M. L. Alexey Bochkovskiy, «Arvix,» [En línea]. Disponible: https://arxiv.org/pdf/2004.10934.pdf. [Último acceso: 1 06 2020].

[16] G. Research, «Google Colaboratory,» [En línea]. Disponible: https://colab.research.google.com/. [Último acceso: 20 09 2020].

[17] Google LLC, «Open Images Dataset V6,» [En línea]. Disponible: https://storage.googleapis.com/openimages/web/index.html. [Último acceso: 03 09 2020].

[18] «Github,» [En línea]. Disponible: https://github.com/EscVM/OIDv4_ToolKit. [Último acceso: 24 08 2020].

[19] TheIAguy, «Youtube - TheIAguy,» [En línea]. Disponible: https://www.youtube.com/channel/UCrydcKaojc44XnuXrfhlV8Q/. [Último acceso: 20 09 2020].

[20] TheIAguy, «Google Colab,» [En línea]. Disponible: https://colab.research.google.com/drive/1_GdoqCJWXsChrOiY8sZMr_zbr_fH-0Fg#scrollTo=imc0NP19hLuq. [Último acceso: 20 09 2020].

[21] A. Bochkovskiy, «Github,» [En línea]. Disponible: https://github.com/pjreddie/darknet. [Último acceso: 23 09 2020].

[22] TheIAGuy, «GitHub,» [En línea]. Disponible: https://github.com/theAIGuysCode/YOLOv4-Cloud-Tutorial. [Último acceso: 20 09 2020].

[23] «Python Docs,» [En línea]. Disponible: https://docs.python.org/3/library/tkinter.html. [Último acceso: 15 09 2020].

[24] «BOE,» [En línea]. Disponible: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-14977. [Último acceso: 20 09 2020].

[25] «Interactive Chaos,» [En línea]. Disponible: https://www.interactivechaos.com/manual/tutorial-de-machine-learning/arquitectura-de-redes-neuronales. [Último acceso: 16 07 2020].

Anexo: Manual de usuario

Instalación

En primer lugar, el usuario debe asegurarse de que tiene la versión 3.7 de Python instalada en su equipo y tiene agregadas las variables de entorno correspondientes. **Importante:** Es conveniente ejecutar la aplicación en modo administrador para que la configuración pueda ser llevada a cabo de forma exitosa.

Una vez ejecutada la aplicación es necesario instalar las librerías necesarias mediante el botón "Instalar" situado debajo de "Librerías", que las instalará automáticamente.

Si se dispone de un equipo con tarjeta gráfica de la marca NVIDIA se recomienda la instalación de la versión 10.1 de la herramienta *NVIDIA CUDA*, que se realiza mediante el botón "Instalar" debajo de "CUDA". Al pulsar aparecerá la ventana de la figura siguiente:

Figura 58. Ventana de instalación de NVIDIA CUDA

Se aceptan los términos del acuerdo y en la siguiente ventana se seleccionan los componentes que se desean instalar. Para ello se selecciona la opción de "Personalizar" y se seleccionan únicamente los componentes marcados en la Figura 59.

A continuación, se continúa con el proceso de instalación y se selecciona la ruta donde se quiere instalar la herramienta.

Finalizada la instalación de CUDA se recomienda cerrar y volver a abrir la aplicación para que los ajustes se hagan efectivos.

El siguiente paso consiste en añadir la librería CUDnn a las librerías que tiene CUDA, para ello el usuario únicamente debe pulsar el botón “Instalar” y seleccionar la ruta donde se ha instalado CUDA.

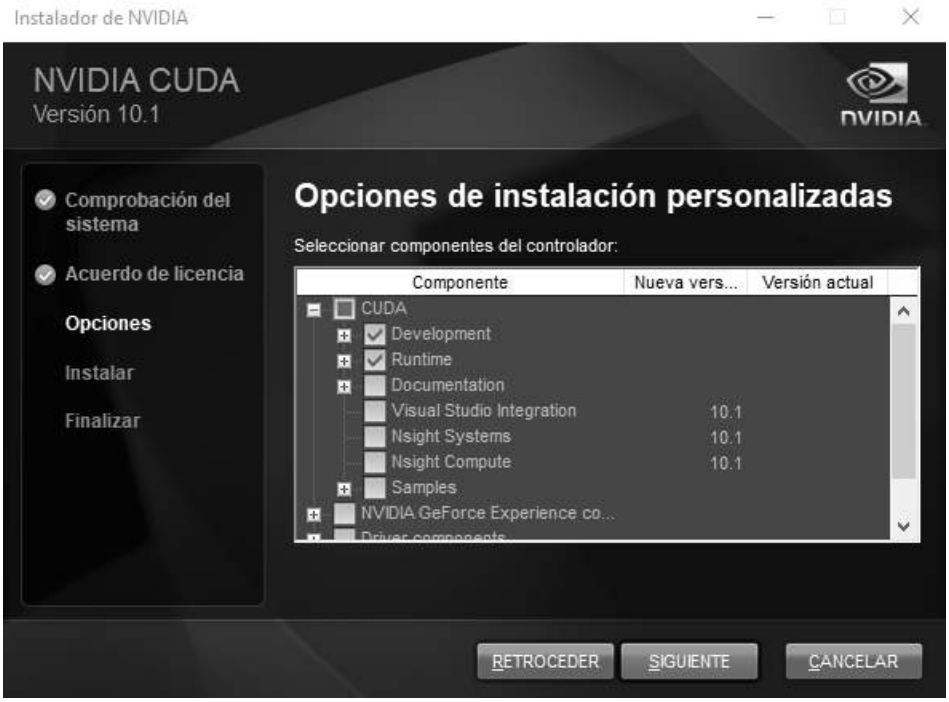

Figura 59. Ventana del instalador de NVIDIA CUDA

Detecciones

Las detecciones realizadas por este sistema son como las de la figura 60.

Los objetos detectados serán recuadrados por una caja que tratará de abarcar los mismos en toda su extensión. A la caja le acompañará el nombre del objeto que el sistema ha detectado y al lado se encontrará el índice de confianza; esta cifra da una noción de la seguridad que tiene el sistema de la identificación que ha hecho para cada objeto en particular

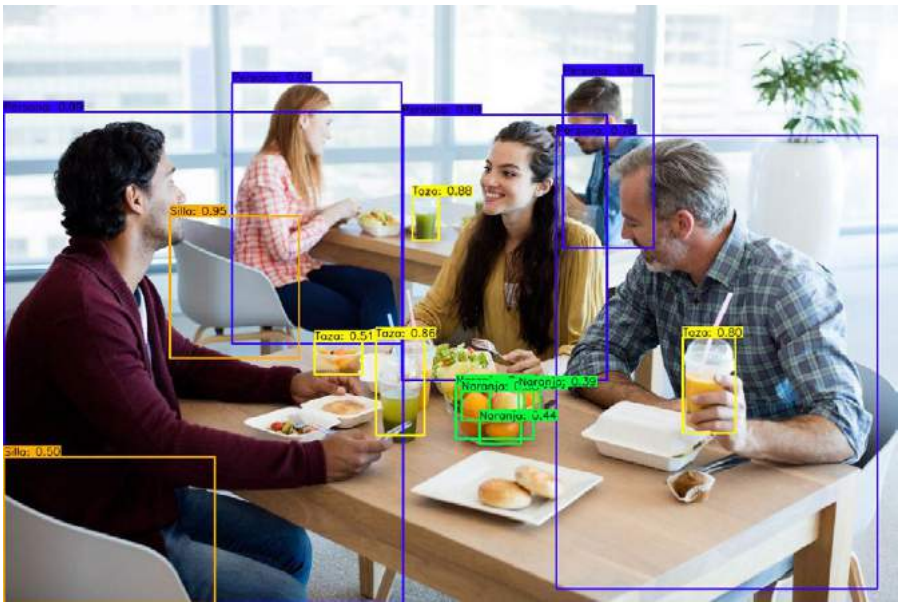

Figura 60. Imagen de prueba procesada por el sistema de identificación de objetos

Detección con cámara web

La opción de alto rendimiento ejecuta la versión más precisa de este sistema, solo se recomienda su uso si se dispone de todo el software previo. Al iniciar esta opción empezarán a aparecer registros en la consola del sistema y finalmente se abrirá una nueva ventana donde se visualizará la detección a través de las imágenes capturadas por la cámara web predeterminada por el equipo.

La opción de bajo rendimiento funciona de la misma manera que la anterior pero su precisión es considerablemente menor. Solo se aconseja su uso si el equipo es incompatible con el software mencionado anteriormente

Detección de ficheros

Mediante esta función el sistema aplica la identificación de objetos sobre la imagen o vídeo seleccionado en el equipo del usuario. Al seleccionar el archivo aparecerán registros en la consola del equipo, se procesará y finalmente se guardará en la carpeta “Detecciones/Imagen” o “Detecciones/Video” respectivamente.

En el caso de ser una imagen, al finalizar el procesado se visualizará en pantalla la imagen procesada mediante el explorador de imágenes predeterminado. En el caso del vídeo, será posible ver el procesado de éste en tiempo real en pantalla.

Defina un umbral

Mediante esta opción se establece el mínimo valor de índice de confianza con las que se quieren obtener las detecciones. Funciona para todos los tipos de detecciones.

Manual de uso

Abre una nueva ventana donde se puede leer una versión resumida del manual de la aplicación

Cerrar aplicación

Detiene la ejecución de la aplicación.