



Escola Politécnica de Enxeñaría de Ferrol

UNIVERSIDADE DA CORUÑA



Trabajo Fin de Máster

CURSO 2022/2023

Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

M. U. en Informática Industrial y Robótica

AUTOR: DANIEL CARDEZO MOSQUERA

TUTORES: ALMA MARÍA MALLO CASDELO
ALEJANDRO PAZ LÓPEZ

TFM N°: 2223_MIIR_1

FECHA: FEBRERO 2023

Título Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

Índice general

Peticionario Escuela Politécnica de Ingeniería de Ferrol
Rúa Mendizábal, s/n, Campus de Esteiro,
15403, Ferrol

Fecha Febrero 2023

Autor El Alumno

Fdo. Daniel Cardezo Mosquera

Índice general

1 Índice general	1
2 Memoria	5
2.1 Objeto	8
2.2 Alcance	8
2.3 Antecedentes	8
2.3.1 Introducción a la visión artificial	8
2.3.2 Problemáticas a resolver por la Visión Artificial	11
2.3.2.1 Clasificación	12
2.3.2.2 Detección de objetos	12
2.3.2.3 Segmentación	12
2.3.2.4 Comparativa de los diferentes métodos	13
2.3.3 Etapas de un sistema de Visión Artificial	13
2.3.4 Bibliografía	15
2.4 Requisitos de diseño	16
2.4.1 Requisitos funcionales y no funcionales	16
2.4.1.1 Requisitos funcionales	17
2.4.1.2 Requisitos no funcionales	17
2.4.2 Tipos de métodos para detección de objetos en imagen/vídeo	17
2.4.2.1 Técnicas tradicionales de procesamiento de imágenes	18
2.4.2.2 Técnicas de procesamiento de imágenes basadas en Deep Learning	19
2.4.3 Detección de objetos con Deep Learning	20
2.4.3.1 YOLOv5	20
2.4.3.2 SSD (Single Shot Detector)	22
2.4.3.3 ResNet-101	25
2.4.4 Métodos basados en extractores de características clásicos	28
2.4.4.1 SIFT	28
2.4.4.2 HOG	29
2.4.5 Plataformas en la nube	30
2.4.5.1 Google Cloud	31
2.4.5.2 Amazon Web Services	33
2.4.5.3 Microsoft Azure	34
2.5 Análisis de las soluciones	34
2.5.1 Métodos de anotación de imágenes.	34
2.5.1.1 Roboflow	34
2.5.1.2 MAKESENSE	38
2.5.1.3 Label-studio	40
2.5.1.4 Conclusiones	42
2.5.2 Librerías Python para desarrollo de interfaces.	42

2.5.2.1 TkInter	42
2.5.2.2 Kivy	43
2.5.2.3 DearPyGUI	43
2.5.2.4 PySimpleGUI	44
2.5.2.5 WxPython	44
2.5.2.6 PyQT	45
2.5.2.7 Librería a emplear	45
2.5.2.8 Técnicas de detección de objetos seleccionadas	46
2.6 Resultados finales	47
2.6.1 Interfaz GUI	47
2.6.2 Mockups	48
2.6.2.1 Solución final	51
2.6.2.2 Código fuente	53
2.7 Orden de prioridad entre los documentos	53
3 Anexos	55
3.1 Contenido	57
3.1.1 Manual de usuario para manejo de la Interfaz Gráfica	57
3.1.1.1 Funcionamiento principal	57
3.1.1.2 Información común a todas las ventanas	59
4 Estudios con entidad propia	63
5 Otros anexos que justifiquen y aclaren conceptos expresados en el Trabajo Final	67
6 Planos	71
6.1 Contenido	73
7 Pliego de condiciones	75
7.1 Contenido	77
8 Mediciones	79
8.1 Contenido	81
9 Presupuesto	83
9.1 Contenido	85

Título Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

Memoria

Peticionario Escuela Politécnica de Ingeniería de Ferrol
Rúa Mendizábal, s/n, Campus de Esteiro,
15403, Ferrol

Fecha Febrero 2023

Autor El Alumno

Fdo. Daniel Cardezo Mosquera

Índice Memoria

2.1	Objeto	8
2.2	Alcance	8
2.3	Antecedentes	8
2.3.1	Introducción a la visión artificial	8
2.3.2	Problemáticas a resolver por la Visión Artificial	11
2.3.2.1	Clasificación	12
2.3.2.2	Detección de objetos	12
2.3.2.3	Segmentación	12
2.3.2.4	Comparativa de los diferentes métodos	13
2.3.3	Etapas de un sistema de Visión Artificial	13
2.3.4	Bibliografía	15
2.4	Requisitos de diseño	16
2.4.1	Requisitos funcionales y no funcionales	16
2.4.1.1	Requisitos funcionales	17
2.4.1.2	Requisitos no funcionales	17
2.4.2	Tipos de métodos para detección de objetos en imagen/vídeo	17
2.4.2.1	Técnicas tradicionales de procesamiento de imágenes	18
2.4.2.2	Técnicas de procesamiento de imágenes basadas en Deep Learning	19
2.4.3	Detección de objetos con Deep Learning	20
2.4.3.1	YOLOv5	20
2.4.3.2	SSD (Single Shot Detector)	22
2.4.3.3	ResNet-101	25
2.4.4	Métodos basados en extractores de características clásicos	28
2.4.4.1	SIFT	28
2.4.4.2	HOG	29
2.4.5	Plataformas en la nube	30
2.4.5.1	Google Cloud	31
2.4.5.2	Amazon Web Services	33
2.4.5.3	Microsoft Azure	34
2.5	Análisis de las soluciones	34
2.5.1	Métodos de anotación de imágenes.	34
2.5.1.1	Roboflow	34
2.5.1.2	MAKESENSE	38
2.5.1.3	Label-studio	40
2.5.1.4	Conclusiones	42
2.5.2	Librerías Python para desarrollo de interfaces.	42
2.5.2.1	TkInter	42

2.5.2.2 Kivy	43
2.5.2.3 DearPyGUI	43
2.5.2.4 PySimpleGUI	44
2.5.2.5 WxPython	44
2.5.2.6 PyQt	45
2.5.2.7 Librería a emplear	45
2.5.2.8 Técnicas de detección de objetos seleccionadas	46
2.6 Resultados finales	47
2.6.1 Interfaz GUI	47
2.6.2 Mockups	48
2.6.2.1 Solución final	51
2.6.2.2 Código fuente	53
2.7 Orden de prioridad entre los documentos	53

2.1. Objeto

Este TFM se enmarca en el estudio y aplicación de técnicas de visión artificial y aprendizaje máquina en el ámbito de la detección de objetos. La detección de objetos es un tema en continuo estudio que genera nuevas publicaciones cada año y con ellas nuevos métodos y librerías software para su aplicación. Muchas de las técnicas más avanzadas están disponibles a través del uso de diferentes librerías de código libre o accesibles a través de servicios de computación en la nube en plataformas como Amazon AWS o Google Cloud Platform en modalidad de pago por uso (incluyendo habitualmente opciones de uso limitado sin coste). En este trabajo, se realizará el diseño y desarrollo de un prototipo de software con una arquitectura modular fácilmente configurable por el usuario, que permita seleccionar componentes de varios tipos de herramientas relacionadas con el reconocimiento de objetos y asista al usuario en la generación de un esqueleto de solución software enlazado con las herramientas seleccionadas. A partir de este esqueleto el programador podrá centrarse en el detalle concreto de la solución garantizando que parte de una arquitectura estandarizada para su proyecto. El lenguaje de programación utilizado en las plantillas será Python, siendo actualmente el lenguaje de mayor difusión en ámbitos científico-técnicos relacionados con aprendizaje máquina y visión por computador. Esta herramienta dispondrá de una sencilla interfaz de usuario que facilitará al usuario la configuración inicial de cada nuevo proyecto que cree con la herramienta. El usuario objetivo de la herramienta será alguien con conocimiento básicos en el campo de la visión artificial y/o el aprendizaje máquina que necesite iniciarse de forma rápida en la construcción de soluciones experimentales que requieran del empleo de técnicas de reconocimiento de objetos.

Un interés añadido de este trabajo es ampliar los conocimientos en el campo de la visión artificial que se ven en el máster a través del desarrollo de un prototipo software.

2.2. Alcance

- Análisis y especificación de requisitos funcionales y no funcionales de la herramienta.
- Estudio de técnicas y herramientas de detección de objetos.
- Selección de las técnicas y desarrollo de los ejemplos a incluir en la herramienta.
- Diseño de la arquitectura de la herramienta y modelo de datos.
- Desarrollo de la herramienta.
- Pruebas finales de funcionamiento de la herramienta.
- Creación de un manual de usuario.

2.3. Antecedentes

2.3.1. Introducción a la visión artificial

Una de las grandes ramas de investigación de la Inteligencia Artificial, en la cual además están teniendo lugar una gran cantidad de avances, es la Visión Artificial. Este campo, está

centrado en replicar partes del sistema de visión del ser humano, tratando de que las computadoras sean capaces de replicar la manera que tiene el ojo de identificar y procesar objetos, tanto en imágenes como en vídeos.

Hasta hace poco, la capacidad de los ordenadores para visión artificial era muy limitada, pero gracias a los últimos avances en IA el campo ha crecido enormemente, siendo capaz de igualar en muchos casos las capacidades humanas en tareas de detección de objetos o segmentación de regiones, entre otras.

Entre estos avances, los más importantes han sido:

- El sustancial aumento de datos a los que se puede acceder en la actualidad, que permiten entrenar modelos de visión artificial, favoreciendo la mejora de esta
- El significativo aumento en materia de capacidad computacional, permitiendo mejoras en el análisis de los datos. Este apartado ha mejorado a raíz del desarrollo de algoritmos que mejoran la eficiencia y el diseño de hardware más robusto y potente.

La motivación de este proyecto viene dada por la continua evolución y desarrollo de las técnicas de detección de objetos por medio de Deep Learning.

El Deep Learning, o aprendizaje profundo, se define como una rama del Machine Learning dominada por distintas técnicas o algoritmos los cuales tras un entrenamiento utilizando una gran cantidad de datos, son capaces de conseguir realizar tareas similares a las que podría realizar un ser humano (realizar predicciones, identificar elementos en imágenes, reconocimiento del habla, etc.).

Los modelos algorítmicos más utilizados en Deep Learning son las redes neuronales. Estas redes neuronales artificiales cuentan con un número de capas ocultas que se ocupan del procesamiento de los datos, permitiendo a la máquina profundizar en su aprendizaje estableciendo conexiones y asignando ponderaciones a los diferentes datos de entrada para obtener cada vez resultados más satisfactorios.

Uno de los campos en los que ha cobrado gran importancia esta rama del Machine Learning es la visión artificial. Esta se define como la capacidad de obtener información relevante del entorno por medio de imágenes que el sistema es capaz de analizar. Para ello, hace uso de diferentes sensores y cámaras, generando unos resultados que en muchas ocasiones mejoran la precisión del ojo humano.

El reconocimiento de imágenes con Deep Learning es una aplicación clave en el campo de la visión artificial y se emplea en multitud de ámbitos en la actualidad: seguridad, medicina, control de calidad, picking y packing, procesamiento de datos, retail, etc. A continuación, se muestran algunos ejemplos de la visión artificial aplicada tanto a la industria como en recopilación de datos para realizar análisis y estudios.

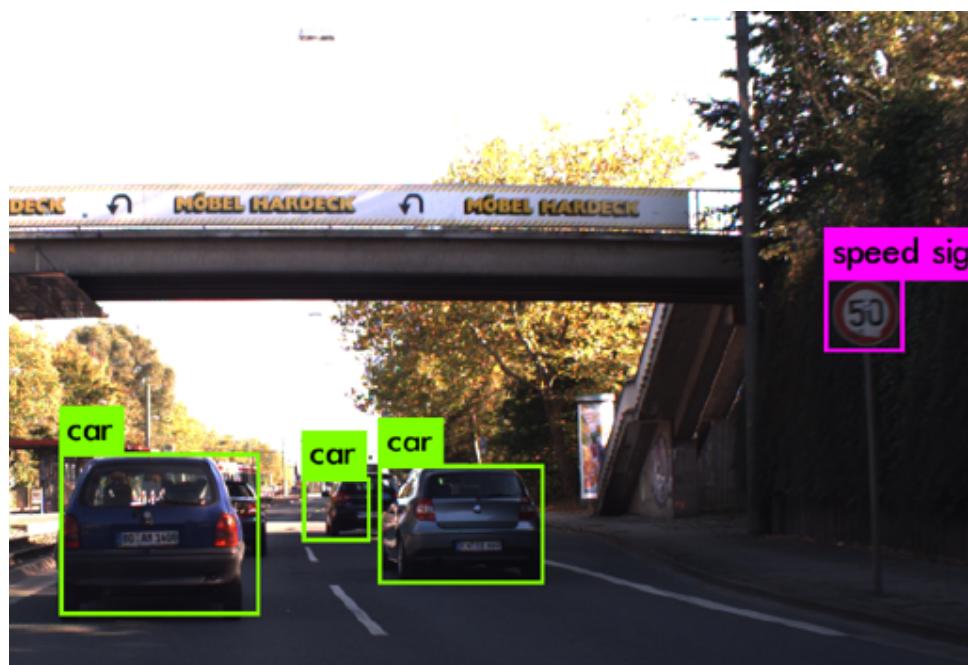


Figura 2.1: Detección de objetos en vía urbana. Referencia: [36].



Figura 2.2: Detección de persona para seguridad en edificios. Referencia: [35].

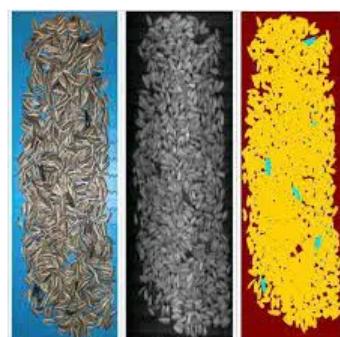


Figura 2.3: Detección de intrusos en cadena de producción alimenticia. Referencia: [34].

2.3.2. Problemáticas a resolver por la Visión Artificial

La manera de la visión artificial para abordar diversas tareas plantea un gran número de retos complicados de resolver. Principalmente, en este campo predomina el estudio acerca de las siguientes problemáticas:

- Clasificación de objetos: Para un conjunto de categorías conocido, ¿A qué categoría pertenece la siguiente imagen?
- Identificación de objetos: ¿Qué tipo de objeto se observa en la imagen?
- Verificación de objetos: Dado una categoría de objetos, ¿Cuál se corresponde con el objeto de la imagen?
- Detección de objetos: Dados una serie de objetos a detectar, ¿Dónde se encuentran los objetos en la imagen?
- Segmentación de objetos: ¿Qué píxeles corresponden al objeto de la imagen?
- Reconocimiento de objetos: ¿Qué objetos hay y donde se sitúan en la imagen?

Para tratar de esclarecer a qué se refieren los problemas que arriba se describen, se hace uso de la figura 2.4.

Computer Vision Tasks

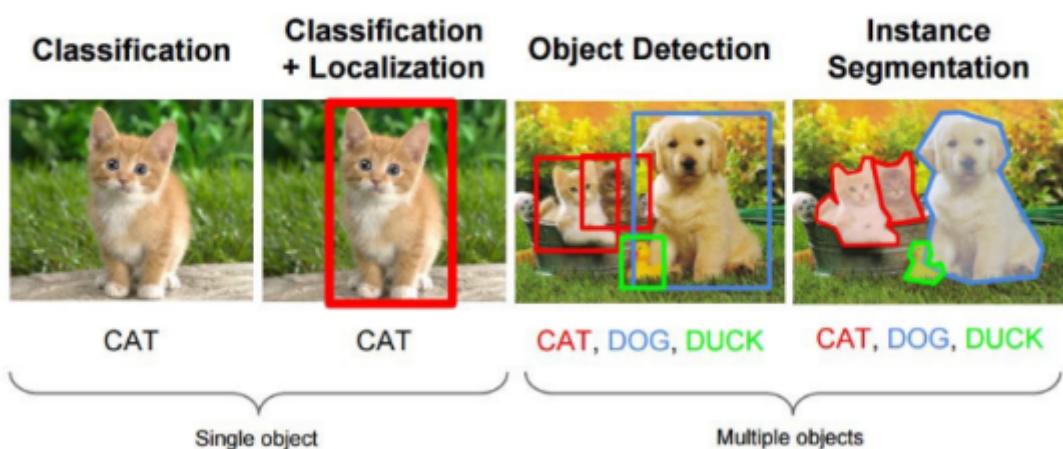


Figura 2.4: Desafíos de la Visión Artificial. Referencia : [17].

Con el fin de hacer frente o solucionar estos problemas, se han desarrollado distintas técnicas.

2.3.2.1. Clasificación

La clasificación de imágenes es una de las tareas más fundamentales de la visión artificial. Está relacionada con asociar una o más etiquetas a una imagen dada.

Existen dos tipos de clasificación:

- Clasificación supervisada, donde el algoritmo de clasificación se entrena en un conjunto de imágenes junto con sus etiquetas correspondientes.
- Clasificación no supervisada, en la cual, el algoritmo usa solamente datos sin procesar para el entrenamiento.

Para crear clasificadores de imagen fiables, es necesario utilizar un conjunto de imágenes lo suficientemente rico con datos etiquetados con precisión.

En el caso de redes convolucionales, la clasificación de imágenes funciona deslizando un kernel o filtro a través de la imagen de entrada, con el fin de capturar detalles relevantes en forma de características.

2.3.2.2. Detección de objetos

La detección de objetos es el campo de la visión artificial que se encarga de resolver la localización y clasificación de un o más objetos dentro de una imagen.

En la práctica, esto se reduce a dibujar cuadros delimitadores alrededor de los objetos detectados, lo que a su vez permite ubicarlos en la imagen.

La diferencia entre la clasificación de imágenes y la detección de objetos reside en que la clasificación solamente devuelve etiquetas de objetos en una imagen (por ejemplo, indica que en una imagen hay un gato, pero no aporta más información), mientras que la detección se encarga de obtener su posición dentro de la imagen además de ofrecer un etiquetado de los objetos (además de detectar que hay un gato en la fotografía, lo sitúa en ella con bounding boxes o cuadros delimitadores).

2.3.2.3. Segmentación

La segmentación trata de definir qué píxeles de una imagen pertenecen a un objeto en concreto.

La segmentación semántica de imágenes consiste en marcar todos los píxeles de una imagen que pertenecen a un tipo de objeto dentro de una imagen, pero no define los límites de cada objeto.

Por otro lado, como se ha descrito antes, la detección de objetos si delimita con un cuadro delimitador la ubicación exacta de cada objeto.

Combinando la segmentación semántica junto con la detección de objetos, se obtiene una segmentación de instancias, que en primer lugar detecta las instancias de objetos para posteriormente segmentar cada uno de los recuadros detectados (denominados regiones de interés).

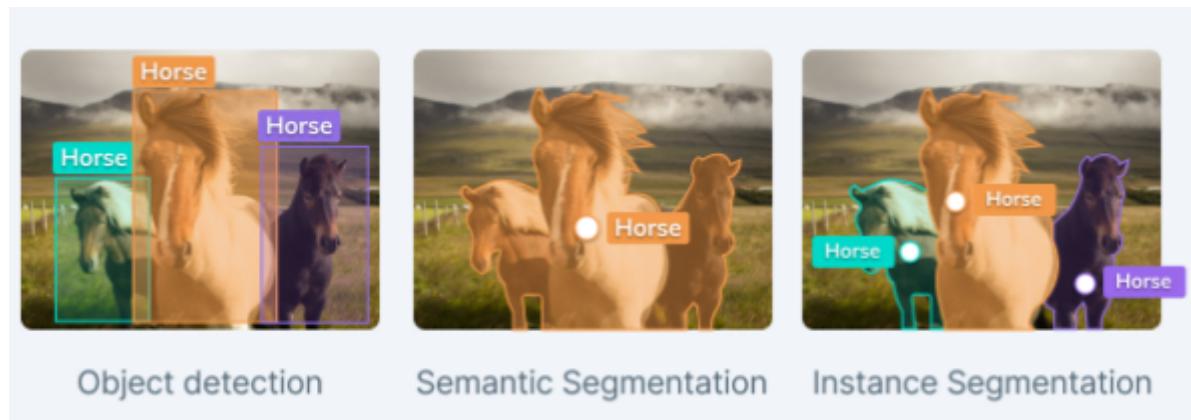


Figura 2.5: Detección de objetos frente a segmentación. Referencia: [18]

2.3.2.4. Comparativa de los diferentes métodos

MÉTODO	USO
Detección	Objetos que ocupan entre el 2 % y el 60 % de la imagen. Detección con límites claros. Detección de grupos de objetos como 1 elemento. Localización a alta velocidad.
Segmentación de instancias	Mejor para objetos largos y delgados. Elementos que a menudo están ocluidos. Elementos que ocupen menos del 10 %.
Clasificación	Elementos como la etiqueta "soleado.º "sesgado".
Segmentación semántica	Objetos sin límites claros.

Tabla 2.1: Selección del método en función del objeto a detectar.

En particular, este trabajo se centrará en las técnicas de reconocimiento de imágenes.

2.3.3. Etapas de un sistema de Visión Artificial

Para llevar a cabo una tarea de visión artificial, suelen seguirse una serie de pasos fundamentales, que se mencionan a continuación.

En primer lugar, se debe de adquirir la imagen digital.

Una vez la imagen digitalizada ha sido obtenida, se procede al preprocesamiento de la imagen, cuyo objetivo es mejorar la imagen, para tratar de obtener el mejor resultado de la tarea de visión que se está llevando a cabo.

Después del preprocesado de datos, tiene lugar la segmentación. Como se ha descrito en el apartado anterior, la segmentación se centra en dividir la imagen en las partes u objetos que la constituyen. Este es un punto bastante crítico del proceso, ya que, una buena segmentación facilitará enormemente la solución del problema, mientras que, una segmentación errónea dará lugar a un error en el proceso.

A continuación, es necesario convertir la imagen segmentada a una forma que sea apropiada para que el ordenador pueda entenderla, es decir, es necesario transformar los datos de entrada

(fase de clasificación). Aquí es donde entran en juego los extractores de características (o descriptores), los cuales, se encargan de extraer información de la imagen de modo que sea más sencillo realizar una clasificación.

Finalmente, una vez se ha extraído la información necesaria de la imagen, tiene lugar la fase de reconocimiento e interpretación. El reconocimiento se describe como el proceso de etiquetado de objetos según la información obtenida por los extractores de características en la fase de clasificación.

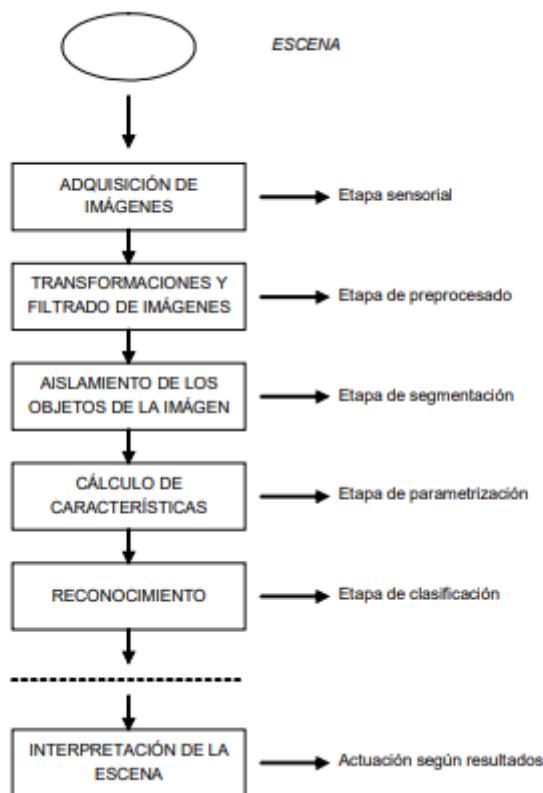


Figura 2.6: Diagrama de bloques de las fases de un proceso de visión artificial. Referencia: [19]

En esta última fase, se asocia cada objeto con su etiqueta correspondiente. Un ejemplo puede ser el uso de un SVM (Support Vector Machine), que se podría definir como un modelo lineal de aprendizaje automático para la resolución de problemas de clasificación y regresión.

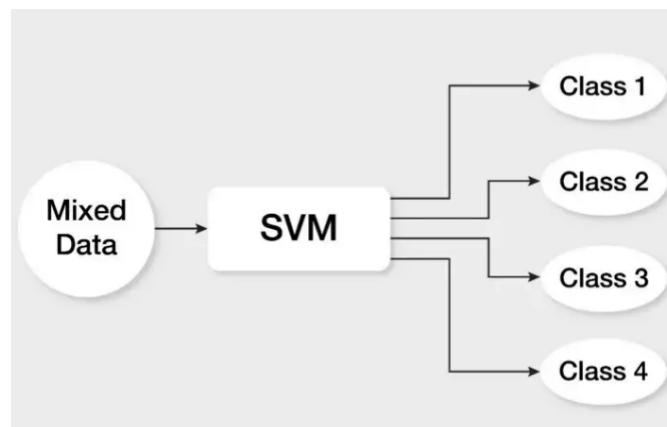


Figura 2.7: SVM. Referencia: [20]

2.3.4. Bibliografía

- [1] <https://www.tensorflow.org/hub>
- [2] <https://blog.facialix.com>
- [3] <https://www.unite.ai>
- [4] <https://tfhub.dev/tensorflow>
- [5] <https://tensorflow-object-detection-api-tutorial.readthedocs.io>
- [6] <https://github.com>
- [7] <https://datasmarts.net/es/como-extraer-features-de-una-imagen-con-hog-en-scikit-image>
- [8] <https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd>
- [9] <https://www.researchgate.net/publication/356842871>
- [10] <https://viso.ai/deep-learning/object-detection/>
- [11] <https://www.paradigmadigital.com/techbiz/inteligencia-artificial-servicio-reconocimiento-imagenes/>
- [12] <https://www.v7labs.com/blog/image-classification-guide>
- [13] <https://www.v7labs.com/blog/object-detection-guide>
- [14] <https://buleria.unileon.es/bitstream/handle/10612/11065>
- [15] <https://towardsdatascience.com/https-medium-com-pupalershikesh-svm-f4b42800e989>
- [16] [Dialnet-Técnicas Y Algoritmos Básicos De Visión Artificial-338314.pdf](#)
- [17] <https://medium.com/@SeoJaeDuk/archived-post-understanding-and-visualizing-convolutional-neural-networks-d6da3e2851cf>

- [18] <https://www.v7labs.com/blog/object-detection-guide>
- [19] Dialnet-Técnicas Y Algoritmos Básicos De Visión Artificial-338314.pdf
- [20] <https://towardsdatascience.com/https-medium-com-pupalershikesh-svm-f4b42800e989>
- [21] <https://dev.to/amigosmaker/pyqt-vs-tkinter-spanish-2n4k>
- [22] <https://stackoverflow.com/questions/29928496/kivy-look-and-feel>
- [23] <https://itnext.io/python-guis-with-dearpygui-137f4a3360f2>
- [24] <https://wxpython.org/pages/screenshots/index.html>
- [25] <https://dev.to/amigosmaker/pyqt-vs-tkinter-spanish-2n4k>
- [26] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.
- [27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed C.Y. Fu, and A.C. Berg. Single shot multibox detector. <https://arxiv.org/pdf/1512.02325.pdf>, 2016
- [28] https://medium.com/@jonathan_hui/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06
- [29] <https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.codeproject.com>
- [30] <https://aprendiendoarduino.wordpress.com/tag/arquitectura-api/>
- [31] <https://www.linkedin.com/pulse/detecci%C3%B3n-de-peatones-con-python-y-opencv-hog-svm-pardo-herrera>
- [32] <https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8>
- [33] <https://www.lisdatasolutions.com/es/blog/deep-learning-clasificando-imagenes-con-redes-neuronales>
- [34] <https://www.atriainnovation.com/aplicaciones-de-la-vision-artificial-en-la-industria>
- [35] <https://www.interempresas.net/Seguridad/Articulos/354163-Inteligencia-Artificial-no-se-utiliza-solo-aplicaciones-videovigilancia-alta-gama.html>
- [36] <https://www.abc.es/ciencia/abci-matematicas-esconden-detras-seguridad-coche-autonomo-20181120227>

2.4. Requisitos de diseño

2.4.1. Requisitos funcionales y no funcionales

Para cumplir con los objetivos planteados en el trabajo es necesario definir formalmente los requisitos del sistema desde el punto de vista del usuario que lo va a utilizar. Dichos requisitos se dividen en funcionales, si se refieren a lo que el usuario espera que le resuelva el sistema; o no funcionales, si nos referimos a propiedades o aspectos de calidad del propio sistema.

2.4.1.1. Requisitos funcionales

- RF01. Proporcionar una interfaz de usuario gráfica para la configuración de soluciones de detección de objetos.
- RF02. La interfaz de usuario funcionará a modo de asistente, guiando al usuario a través de una secuencia de pasos.
- RF03. Permitir seleccionar diferentes métodos generales de detección de objetos.
- RF04. Permitir seleccionar un modelo o servicio concreto dentro de un determinado método.
- RF05. Permitir, a través de un formulario, la parametrización de cada modelo o servicio concreto.
- RF06. Generar una solución de código en lenguaje Python de acuerdo con la configuración establecida por el usuario.

2.4.1.2. Requisitos no funcionales

- RNF01. Utilizar lenguaje de programación Python.
- RNF02. Utilizar librerías de código abierto en la medida de lo posible.
- RNF03. Multiplataforma.
- RNF04. Utilizar una librería de creación de GUIs madura, bien documentada, ampliamente extendida y de aspecto moderno.
- RNF05. Se definirán diferentes plantillas de código Python para distintos modelos/servicios utilizados en técnicas habituales de detección de objetos.
- RNF06. La aplicación prototipo proporcionará soporte para varios métodos de detección de objetos.
- RNF07. Diseñar la aplicación de modo que sea fácilmente extensible para la incorporación de nuevas técnicas habituales de detección de objetos.

2.4.2. Tipos de métodos para detección de objetos en imagen/vídeo

La detección de objetos es una parte fundamental del campo de la visión artificial, en el cual, se busca ser capaces de detectar objetos de ciertas clases determinadas (vehículos, animales, etc) en imágenes o incluidos videos. El objetivo es desarrollar modelos computacionales capaces de obtener información que posteriormente será utilizada por aplicaciones de visión artificial.

Estos se conocen como técnicas tradicionales de procesamiento de imágenes.

En los últimos años, el rendimiento de estos detectores y rastreadores de imágenes ha aumentado significativamente gracias a los grandes avances producidos en el campo del Deep Learning, una rama del Machine Learning dominada por distintas técnicas o algoritmos los cuales tras un entrenamiento haciendo uso de una gran cantidad de datos, son capaces de realizar tareas como las podría realizar un ser humano (en este caso, identificar elementos en imágenes y vídeos).

Dichos avances, a su vez, tampoco habrían sido posibles gracias a la potencia informática que ofrecen las GPU y al uso del Transfer Learning, que reduce significativamente los tiempos de entrenamiento de las redes de Deep Learning.

La disciplina del Transfer Learning está dominada por algoritmos conocidos como Redes Neuronales (en la visión artificial, las más utilizadas son las Redes Neuronales Convolucionales), que son algoritmos que cuentan con un gran número de capas ocultas que se encargan

del procesamiento de los datos, permitiendo a la máquina profundizar en su aprendizaje estableciendo conexiones y asignando ponderaciones a los diferentes datos de entrada para obtener a su vez resultados cada vez más satisfactorios. En la figura 2.8 se muestra un esquema de la estructura básica de una red neuronal.

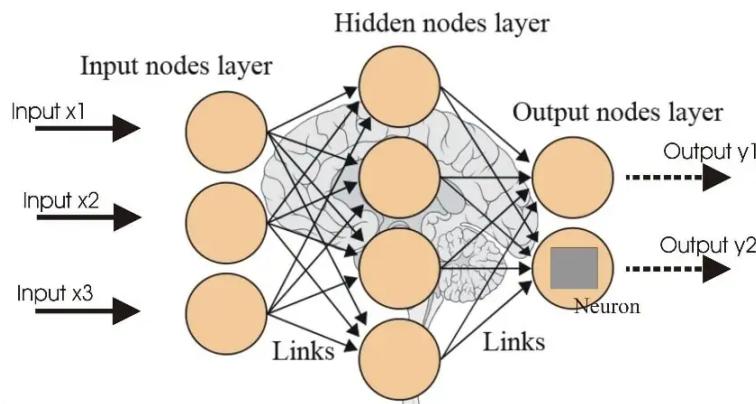


Figura 2.8: Estructura básica de la red neuronal. Referencia: [33].

Básicamente, lo que se pretende con el Transfer Learning es entrenar una red neuronal modelo a partir de un conjunto de datos y una tarea. Posteriormente, las características aprendidas por esta red son reutilizadas, transfiriéndose a una segunda red, para que esta se entrene con otro conjunto de datos diferente orientados a otra tarea distinta. Este "preentrenamiento" es el que reduce enormemente el gasto computacional, y por tanto, los tiempos de entrenamiento de nuestros algoritmos. Actualmente, existen gran cantidad de modelos que se pueden emplear, como pueden ser YOLOv5, SSD, Faster R-CNN, etc.

Estas, forman parte de las técnicas modernas de procesamiento de imágenes, o técnicas de procesamiento por Deep Learning.

La detección y el seguimiento de objetos está presente en muchos ámbitos distintos en la actualidad, permitiendo el monitoreo inteligente de pacientes de la atención médica, la conducción autónoma, la detección de anomalías, etc.

2.4.2.1. Técnicas tradicionales de procesamiento de imágenes

Se consideran técnicas tradicionales a todas aquellas empleadas antes de 2014.

Estas, generalmente no requieren datos históricos para el entrenamiento y son de naturaleza no supervisada.

VENTAJAS	No requieren imágenes anotadas.
DESVENTAJAS	Necesitan un fondo unicolor. No detectan objetos parcialmente ocultos. Frágiles ante la iluminación, las sombras y el efecto de desorden.

Tabla 2.2: Técnicas tradicionales de procesamiento de imágenes.

Como ejemplo de este tipo de técnicas podemos nombrar HOG, la cual, pese a haber sido

desarrollada hace una década, sigue utilizándose con mucha frecuencia en la actualidad debido a su gran capacidad de detección.

2.4.2.2. Técnicas de procesamiento de imágenes basadas en Deep Learning

A partir de 2014 en adelante, se han comenzado a emplear las técnicas de detección por medio de Deep Learning.

Generalmente utilizan aprendizaje supervisado, aunque también existen casos en los que se ha empleado aprendizaje no supervisado. En el caso de la visión artificial, lo más utilizado es el aprendizaje supervisado.

Este tipo de técnicas de procesamiento tiene los siguientes ventajas y desventajas:

VENTAJAS	Mayor resistencia a la oclusión. Mayor resistencia a escenas de color (más complejas que escala de grises). Mayor resistencia a la iluminación.
DESVENTAJAS	Requiere de una cantidad de datos de entrenamiento muy elevada. Proceso de anotación de imágenes altamente laborioso. Rendimiento limitado por la potencia de las GPU.

Tabla 2.3: Técnicas de procesamiento de imágenes basadas en Deep Learning.

Cabe mencionar que pese a estar limitado el rendimiento de este tipo de técnicas a la potencia de las GPU, esta aumenta significativamente con el paso de los años.

Las técnicas de procesamiento de imágenes que hacen uso de deep learning se divide a su vez en dos etapas:

- Algoritmos de detección de dos etapas (son los más importantes), por ejemplo:
 - RCNN (2014).
 - Fast RCNN y Faster RCNN (2015).
 - Pyramid Networks/FPN (2017).
 - G-RCNN (2021).
- Algoritmos de detección de una sola etapa. Algunos de ellos son:
 - Yolo (2016).
 - SSD (2016).
 - RetinaNet (2017).
 - YOLOv3 (2018).
 - YOLOv5 (2020).
 - Yolor (2021).
 - YOLOv7 (2022).

La base de estos modelos siempre se divide en dos pasos:

- Encontrar un número arbitrario de objetos.
- Clasificar cada objeto y estimar su tamaño con un cuadro delimitador.

Sin embargo, dependiendo del tipo de modelo se realiza de una forma diferente.

En el caso de los detectores de dos etapas, primero se detectan regiones de interés por medio de redes neuronales profundas, para posteriormente, realizar una clasificación de objetos basada en las características extraídas anteriormente. Estos métodos suelen ser los más precisos, aunque también son los más lentos de los dos.

Por otro lado, los detectores de una etapa predicen cuadros delimitadores sobre las imágenes directamente, sin el paso intermedio de detectar regiones de interés. Estos son más rápidos, pero no son tan precisos, sobre todo cuando se trata de reconocer objetos con formas irregulares o grupos de pequeños objetos.

2.4.3. Detección de objetos con Deep Learning

2.4.3.1. YOLOv5

YOLOv5 es un modelo perteneciente a la familia YOLO, la cual, pertenece a su vez a los algoritmos de detección de una única etapa.

Para tratar de entender la arquitectura de YOLOv5, se utilizará la estructura global de los distintos modelos YOLO.

En la figura 2.9 se puede observar la arquitectura de la estructura de red convolucional de la primera versión de YOLO, la cual, constaba de 24 capas convolucionales seguidas por dos capas totalmente conectadas.

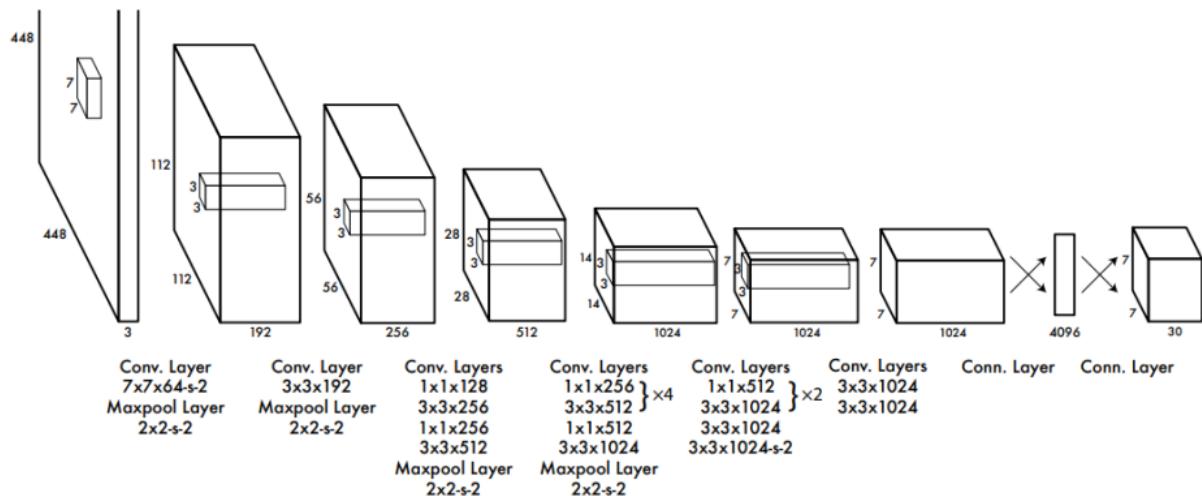


Figura 2.9: Arquitectura base de los distintos modelos de la familia YOLO. Referencia: [26]

La filosofía de la arquitectura propuesta en YOLO consiste en unificar los componentes de detección de objetos en una sola red.

En YOLO, cada imagen se divide en una malla (conjunto de celdas) y cada celda de la malla se encarga de detectar y clasificar objetos dentro de su área. De este modo, la imagen solamente se recorre una vez, de ahí su nombre (You Only Look Once). Es uno de los algoritmos

más populares en el ámbito de la detección de objetos gracias a su bien rendimiento y licencia open-source.

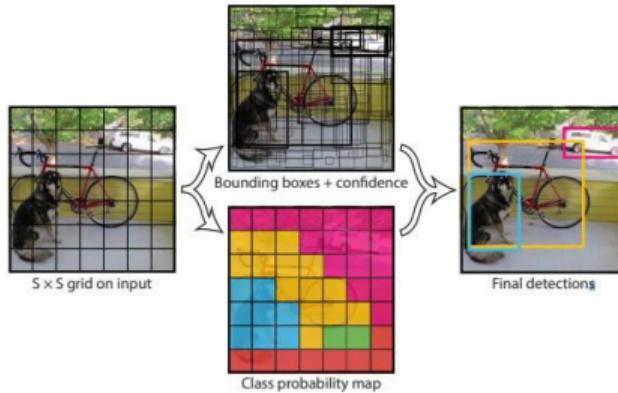


Figura 2.10: Ejemplo de funcionamiento de un modelo YOLO. Referencia: [26].

YOLO es notablemente más rápido que otros algoritmos de la familia R-CNN, sin apreciar una notable pérdida de precisión. Un ejemplo de la velocidad de YOLO frente a otras redes como puede ser Faster R-CNN o SSD (otro algoritmo de detección en una etapa) se puede observar en la figura 2.11.

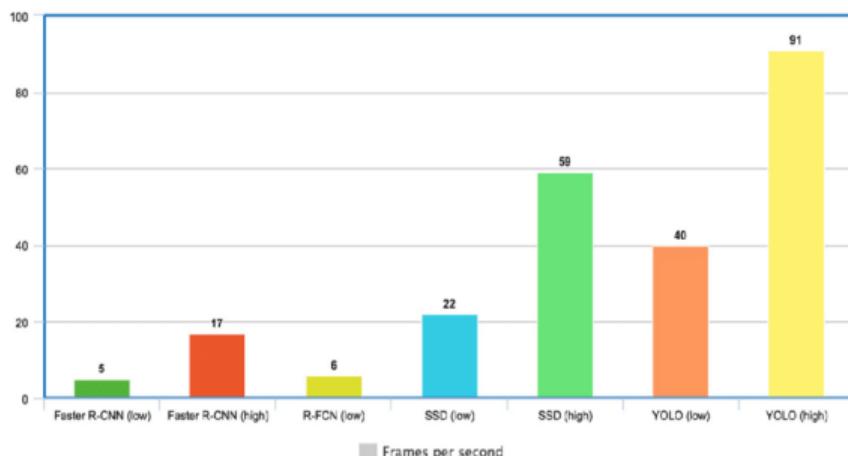


Figura 2.11: Comparativa de velocidades de distintas redes neuronales. Referencia: [26].

YOLOv5 es el primer algoritmo de la familia desarrollado en Python (todos los anteriores han sido desarrollados en C++), permitiendo al usuario un manejo más cómodo de la red neuronal.

Esta red tiene distintos modelos, los cuales, comparten la misma arquitectura y algoritmo, difiriendo únicamente en el número de capas y el número de neuronas de cada capa de la red neuronal, lo cual, influye directamente sobre el peso, la velocidad y la precisión de la red neuronal.

La gráfica que se muestra en la figura 2.12 ofrece una comparativa de los distintos modelos de YOLOv5 en función de su velocidad (eje horizontal) y de su precisión (eje vertical). En esta

gráfica se ha utilizado una GPU y se han entrenado todas las redes con el dataset de Microsoft COCO.

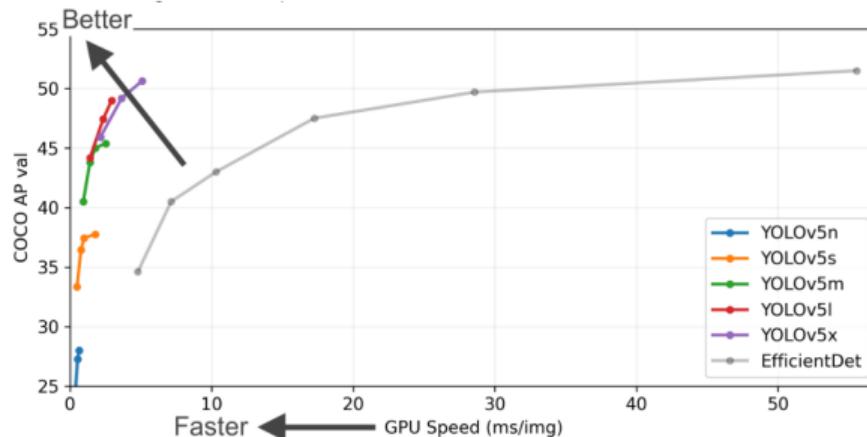


Figura 2.12: Comparativa de los distintos tipos de modelos de la familia YOLO. Referencia: [26]

2.4.3.2. SSD (Single Shot Detector)

SSD fue publicada por primera vez en un paper en el año 2015. Su principal finalidad es la detección de objetos en tiempo real.

La principal característica que permite a esta red la detección en tiempo real es la eliminación de la necesidad de la RPN (Region Proposal Network). Esta eliminación se traduce en una gran pérdida de precisión, que se ha solventado añadiendo pequeñas mejoras incluyendo cajas por defecto y características multi-escala.

Este detector de objetos está compuesto por dos partes principales:

- Extracción de las características de la imagen.
- Detección de objetos haciendo uso de filtros convolucionales.

Para tratar de explicar el proceso, se comenzará mostrando la imagen de la arquitectura de un SSD con un único predictor:

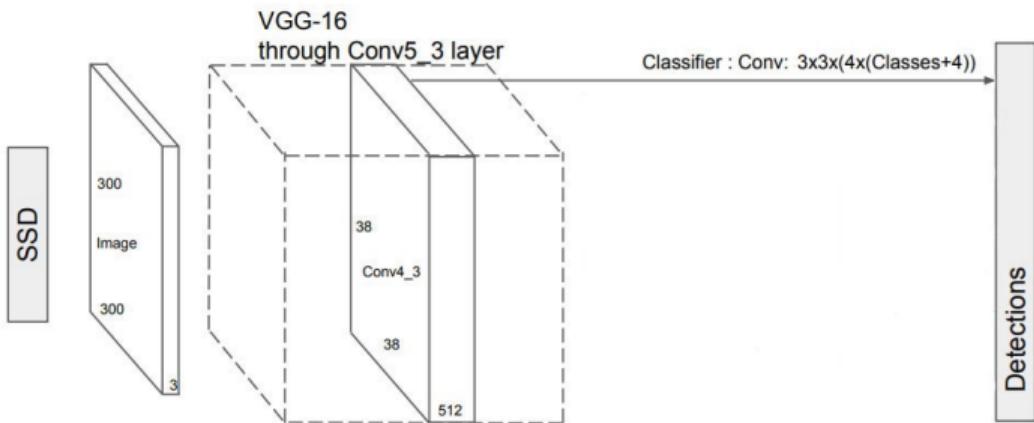


Figura 2.13: Arquitectura de una SSD con un único predictor. Referencia: [27]

La primera parte del proceso, es redimensionar la imagen de entrada, por ejemplo, en la figura 2.13 se puede observar que se redimensiona a un tamaño de 300x300 píxeles. Esto proporciona una mejora notable en cuanto a la velocidad de inferencia, aunque con el coste de perder la capacidad de detectar objetos pequeños.

Tras el redimensionamiento, tiene lugar la fase de extracción de características, que en la figura 2.13 se realiza por medio de una red neuronal convolucional VGG16.

SSD divide la imagen en celdas de igual tamaño, y cada una de ellas que representen el centro de un objeto, serán las celdas encargadas de detectarlo. Para ello, cada celda predice un número N de cajas delimitadoras y la probabilidad de que esas cajas contengan al objeto.

Como ejemplo, se puede observar la capa Conv4_3 de la figura 2.14. A modo de simplificación, se muestra a continuación un ejemplo de uso realizando 4 predicciones por casilla.



Figura 2.14: Ejemplo de detección SSD con un solo predictor. Referencia: [28].

Cada predicción está compuesta por una caja delimitadora y cuatro puntuaciones, que en este caso son ciclista, peatón, coche y una para cuando no existen objetos. Se asigna a la

predicción la clase con mayor puntuación, en el caso de esta imagen, ciclista.

Una de las grandes limitaciones de esta red es el número de celdas en las que se puede dividir la imagen , ya que, si son demasiado grandes no podrán detectar objetos pequeños al existir la posibilidad de que haya más de un objeto en la celda.

Pese a este ejemplo con un detector, la realidad es que SSD consta de múltiples detectores, los cuales hacen capaz la detección de más de un objeto en una imagen, de forma independiente.

Aquí reside uno de los grandes puntos fuertes de esta arquitectura, y es que, SSD es capaz de aprovechar el problema de que las CNN reducen la dimensión espacial de forma gradual, por lo que la resolución de las capas de extracción de características también se ve reducida. Esto lo consigue aprovechando dicha reducción para detectar objetos a distintas escalas, detectando objetos de distintos tamaños gracias a su estructura piramidal. Así, a modo de ejemplo, en la figura 2.15 la capa 4x4 es empleada para detectar objetos de una gran escala, mientras que la capa 8x8 se usa para la detección de objetos más pequeños.

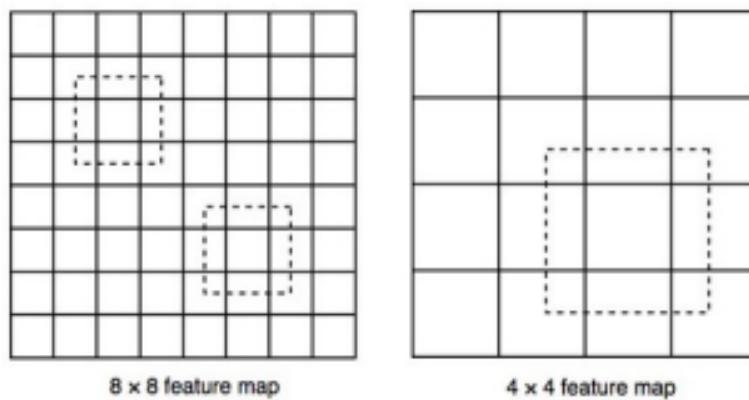


Figura 2.15: Detección de objetos de distintos tamaños con múltiples detectores. Referencia: [28]

La representación final de la arquitectura SSD se muestra en la figura 2.16.

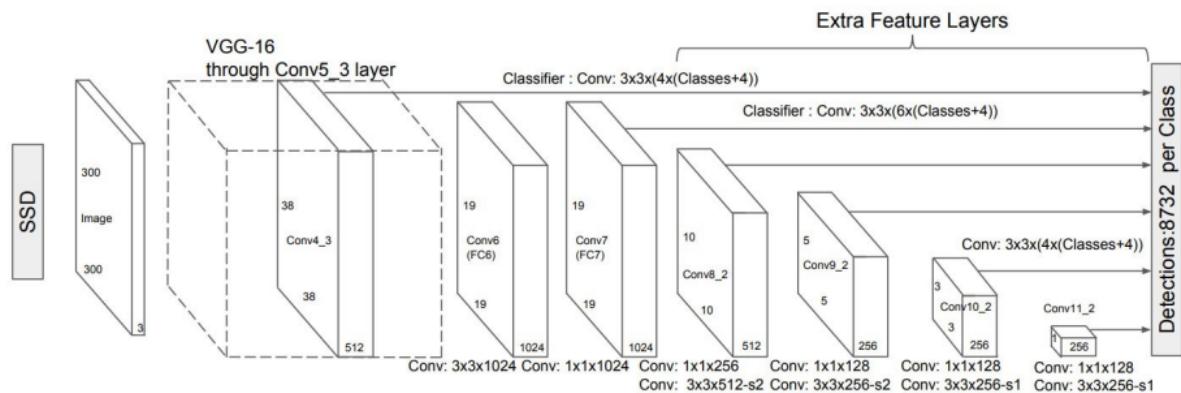


Figura 2.16: Arquitectura del modelo SSD. referencia: [27]

2.4.3.3. ResNet-101

La red neuronal ResNet aparece por primera vez en 2015, como una nueva aproximación para las redes neuronales convolucionales.

Se lanzaron tres arquitecturas a la vez: ResNet-50, ResNet-101 y ResNet-152. El número indica el número de capas de la red neuronal.

La mayor ventaja de esta red frente a las anteriores es que, a pesar de contar con un número de capas de redes neuronales significativamente superior, la complejidad del modelo es menor, esto se debe a que el número de operaciones realizadas es menor.

Por otro lado, supuso un gran avance debido a que anteriormente, se sabía que existía un máximo en la profundidad de los modelos de redes convolucionales. En la figura 2.17 se puede ver el porcentaje de error generado en una red neuronal con 20 capas frente a otra con 56.

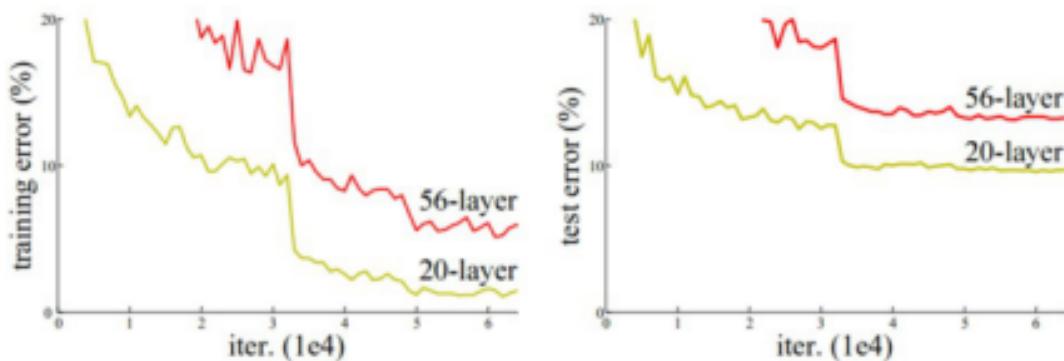


Figura 2.17: Comparativa de error entre redes neuronales con diferente número de capas. Referencia: [32].

Como se puede observar, el error es significativamente mayor en la red de 56 capas pese a ser la mayor de las dos. ResNet, sin embargo, es capaz de reducir o mitigar este error para las redes neuronales con más capas, utilizando para ello una nueva técnica de aprendizaje que no se había implementado hasta la fecha, la cual, está basada en minimización de residuos.

Dicha minimización de residuos consiste en que cada vez que se añaden a la red capas que incluyan los pesos, se añade un “atajo” que vaya desde la entrada de la primera de las capas a la salida de la última.

Así, si una X es una imagen de entrada, y una imagen de salida y , $F(x)$ las transformaciones realizadas por las distintas capas, y el objetivo de una red neuronal se puede definir como el de optimizar una función $F(x)+X$ resultado de pasar la imagen a través de sucesivas capas, lo que haría una red neuronal anterior sería tratar de predecir la función que hace que X se convierta en y , la cual se puede nombrar como $H(x)$, por ejemplo ($H(x)=y$), mientras que ResNet trata de predecir cuánto le queda a x para convertirse en y .

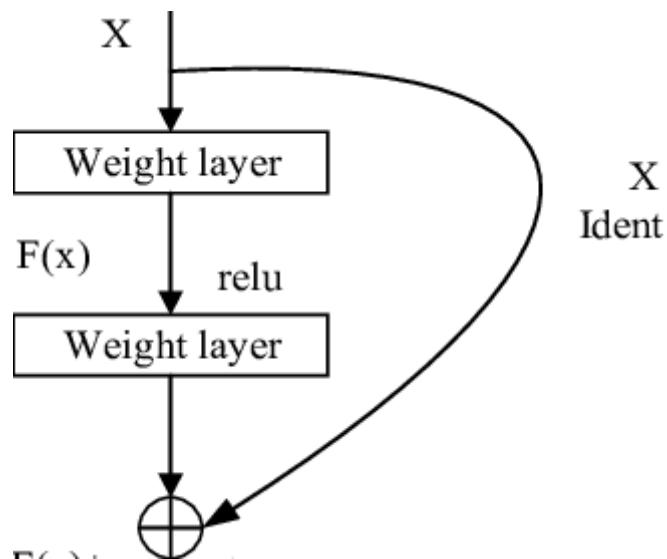


Figura 2.18: Teoría de residuos utilizada en ResNet. Referencia: [32].

En cuanto a la arquitectura de este modelo, se utiliza una arquitectura de red simple de 34 capas, la cual, está inspirada en VGG-19 con la diferencia de que sobre ella se añade la conexiones de .^atajo". Estos .^atajos" son los que finalmente transforman la arquitectura en la red residual. Se puede observar en la figura 2.19.



Figura 2.19: Base de ResNet y su transformación aplicando Teoría de residuos. Referencia: [32].

2.4.4. Métodos basados en extractores de características clásicos

2.4.4.1. SIFT

SIFT (Scale Invariant Feature Transform) es un tipo de descriptor que permite detectar puntos característicos dentro de una imagen para posteriormente describirlos por medio de un histograma orientado de gradientes (HOG).

Para obtener un conjunto de descriptores SIFT de una imagen se necesita, primeramente, obtener los puntos característicos, para después calcular el vector descriptor para cada punto de interés, partiendo de la información que aportan los píxeles que lo rodean. Cabe mencionar también que el vector de características consta de 128 elementos.

Finalmente, este vector ofrece información de su posición en coordenadas de la imagen, la escala a la que se encontró y la orientación dominante de la región alrededor de dicho punto.

Uno de los puntos fuertes de SIFT es que esto lo realiza de forma que se muestra invariante a la posición, la escala y la orientación.

SIFT ha sido propuesto para imágenes en escala de grises, por lo que el vector descriptor contiene información sobre como se distribuyen los niveles de intensidad alrededor de cada punto de interés.

Así, se definen dos partes en el algoritmo:

- Obtención de puntos característicos.
- Descripción de la región alrededor de cada punto de interés.

La obtención de los puntos característicos se consigue analizando los puntos de la imagen en los que se producen a ambos lados diferencias de gradiente significativas. En este descriptor se propone detectar blobs.

Un blob puede definirse como una región de la imagen que se caracteriza porque algunas de sus propiedades se mantienen aproximadamente constantes. En este caso, la propiedad que se suele tener en consideración es el nivel de gris. De este modo, se detectan regiones superiores a un punto.

Por otro lado, para obtener el descriptor de cada punto característico, se propone realizar un cálculo del histograma de direcciones del gradiente local alrededor del punto de interés.

Este descriptor se convierte en invariante a la escala y a la rotación, y se utiliza dicha información para conseguir orientar la rejilla que se emplea a la hora del cálculo del histograma.

Se muestra un ejemplo de la aplicación del SIFT en la figura R2.20.

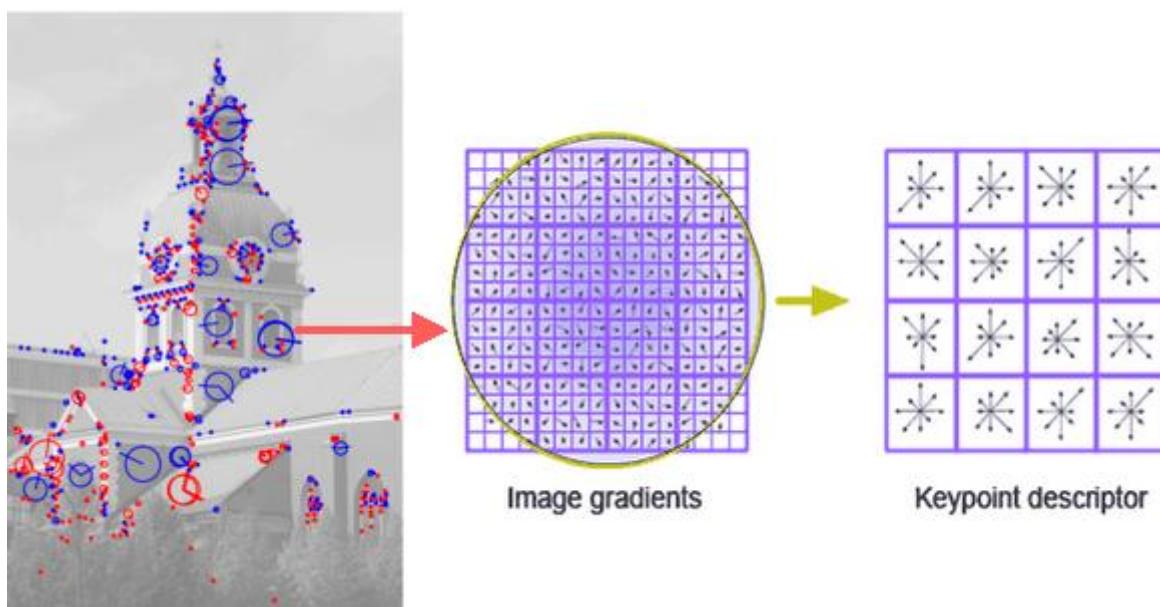


Figura 2.20: SIFT. Referencia: [29]

2.4.4.2. HOG

HOG (Histograma de Gradientes Orientados) es un tipo de descriptor de características.

El reconocimiento de objetos por medio de descriptores HOG es muy utilizado en el campo de la visión artificial, ya que a pesar de ser una técnica que se ha desarrollado hace más de una década, ofrece unos grandes resultados.

Habitualmente, se utilizan en conjunto con una LSVM (máquina virtual de soporte lineal), lo cual ofrece resultados muy satisfactorios en cuanto a precisión a la hora de entrenar clasificadores para detección de personas y rostros, principalmente.

Esta técnica se basa en la orientación de los gradientes localizados en un área de una imagen, teniendo como fundamento que la forma y apariencia de los objetos pueden describirse por medio de la distribución de gradientes de intensidad o direcciones de los bordes.

La idea es que la imagen se descompone en celdas, y sobre los píxeles de cada celda se aplica un Histograma de direcciones de gradientes. Los descriptores son la concatenación de estos histogramas.

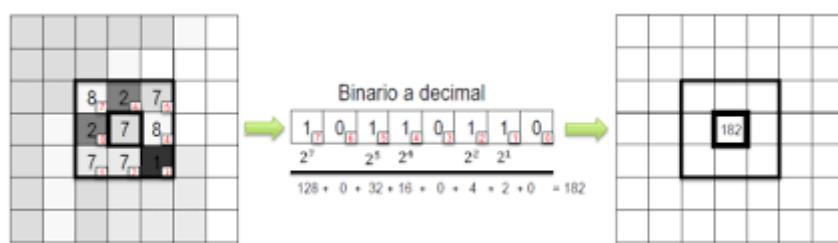


Figura 2.21: Ejemplo de funcionamiento de HOG. Referencia: [31]

La gran ventaja de los descriptores HOG frente a otros descriptores es que, al operar en celdas locales, son invariantes ante transformaciones geométricas o fotométricas, excepto a la rotación de los objetos.

Esto sumado al muestreo espacial amplio, la normalización fotométrica y el muestreo de orientación más fino permite ignorar los movimientos corporales de las personas siempre y cuando estas estén erguidas. Esto es lo que hace a HOG una técnica muy potente para la detección de personas en imágenes, entre otras cosas.

Un ejemplo de detección de personas se puede ver en la figura 2.22. En ella, se ve que la imagen de la izquierda detecta inicialmente más recuadros que personas en la imagen, esto es porque inicialmente detecta potenciales personas, para posteriormente utilizar una aplicación de supresión y obtener la imagen de la derecha, con la correcta detección de las dos personas presentes en la imagen.

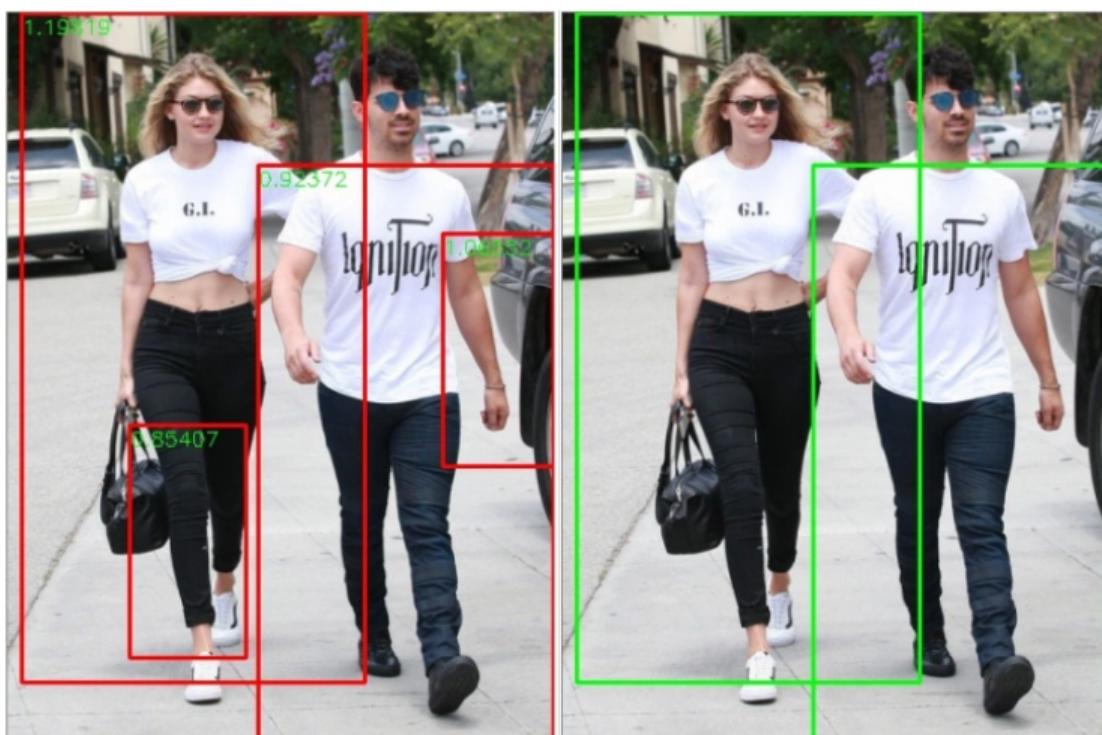


Figura 2.22: Detección de personas con HOG. Referencia: [31]

2.4.5. Plataformas en la nube

Este tipo de servicios surgen bajo el nombre de MLaaS (Machine Learning as a Service), y son principalmente plataformas basadas en la nube que ofrecen soluciones de preprocesado y entrenamiento de modelos de visión artificial (redes convolucionales, entre otras).

Las plataformas cloud más importantes son AWS (Amazon Web Services), Google Cloud o Microsoft Azure.

Este tipo de servicios hace accesible el campo del reconocimiento de imágenes ya no solo a perfiles muy específicos, sino también a cualquier usuario que desee probar un modelo de reconocimiento de objetos.

Las plataformas en la nube se diferencian las unas de las otras teniendo en cuenta además de su capacidad o potencial para detección de imágenes otros factores como la integración de otras herramientas y servicios en la nube, las limitaciones del API y los costes.

En cuanto a los servicios que ofrece cada una, en la figura 2.23 se puede observar una comparativa de los servicios que ofrece cada una:

FUNCIONALIDAD	AWS REKOGNITION	GOOGLE CLOUD VISION API	AZURE COMPUTER VISION API/FACE API
ETIQUETADO DE IMÁGENES (OBJETOS Y ESCENAS)	SI	SI	SI
DETECCIÓN DE CARAS	SI	SI	SI
RECONOCIMIENTO DE RASGOS FACIALES	SI	SI	SI
DETECCIÓN DE SENTIMIENTO	SI	SI	SI
OCR	NO	SI	SI
DETECCIÓN DE LOGOS Y MARCAS	NO	SI	NO
DETECCIÓN DE CONTENIDO INAPropiado	SI	SI	SI
COMPARACIÓN DE CARAS	SI	NO	SI
BÚSQUEDA DE CARAS	SI	NO	NO
RECONOCIMIENTO DE CELEBRIDADES	SI	NO	SI
ANÁLISIS DE VIDEO	NO	SI	SI

Figura 2.23: Comparativa de servicios de plataformas en la nube. Referencia: [30]

2.4.5.1. Google Cloud

La API de Google Cloud que se encarga de proporcionar soluciones de Computer Vision a los usuarios se llama Google Vision API.

Google Vision API, desarrollada por Google, permite a sus usuarios realizar un análisis profundo de sus imágenes. Para esto, se hace uso de modelos de redes neuronales preentrenadas accesibles a través de esta API.

Se puede definir una API o interfaz de programación de aplicaciones como un conjunto de definiciones y protocolos usados para diseñar software. Dicho en otras palabras, son un conjunto de normas o reglas para definir cómo se intercambian los datos.

Cloud Vision es una API de transferencia de estado representacional (REST). REST se conoce como un conjunto de reglas de arquitectura apoyado en el estándar HTTP. En la figura REF se puede ver un ejemplo de la arquitectura REST.

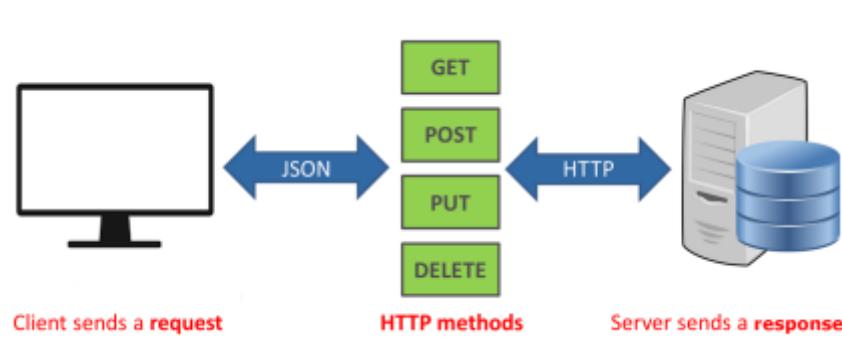


Figura 2.24: Arquitectura REST. Referencia: [30]

Como funcionamiento básico, el usuario debe de enviar una solicitud en formato JSON, la cual contiene una lista de Requests además de un o más objetos de AnnotateImageRequest. En la lista, es necesario especificar el archivo de la imagen, los tipos de anotaciones que se realizarán en esta, y si fuese necesario alguna sugerencia para ayudar al servicio durante el proceso (idioma, etc). Todas las solicitudes deben de autenticarse haciendo uso de un token OAuth o una clave API.

Las imágenes que se pueden enviar al servicio pueden estar codificadas en base64, como una URL de Cloud Storage o como una URL de acceso público.

La respuesta a la solicitud también es en formato JSON, pero esta vez de tipo AnnotateImageResponse.

En la figura 2.53 se puede observar un ejemplo de una respuesta.

```

Labels:
mid: "/m/09j2d"
name: "Clothing"
score: 0.8392713665962219
bounding_poly {
  normalized_vertices {
    x: 0.12546376883983612
    y: 0.6115875840187073
  }
  normalized_vertices {
    x: 0.9814369678497314
    y: 0.6115875840187073
  }
  normalized_vertices {
    x: 0.9814369678497314
    y: 0.9928443431854248
  }
  normalized_vertices {
    x: 0.12546376883983612
    y: 0.9928443431854248
  }
}

mid: "/m/01g317"
name: "Person"
score: 0.8092928528785706
bounding_poly {
  normalized_vertices {
    x: 0.13958807289600372
    y: 0.23619818687438965
  }
  normalized_vertices {
    x: 0.9913316369056702
    y: 0.23619818687438965
  }
  normalized_vertices {
    x: 0.9913316369056702
    y: 0.9859361052513123
  }
  normalized_vertices {
    x: 0.13958807289600372
    y: 0.9859361052513123
  }
}

```

Figura 2.25: Respuesta de Google Cloud Vision API. Referencia: Elaboración propia.

2.4.5.2. Amazon Web Services

El servicio de AWS para reconocimiento de objetos es Amazon Rekognition.

Amazon Rekognition está a su vez subdividido en dos APIs: Amazon Rekognition Image, para el procesado de imágenes, y Amazon Rekognition Video, para el procesamiento de video.

Por un lado, Rekognition Image hace uso de técnicas de Deep Learning, haciendo posible la detección de rostros, famosos, escenas y texto, entre otras. Igual que Google Cloud, tanto las entradas como las respuestas tienen formato JSON.

Una de las limitaciones de Amazon Rekognition Image es que las imágenes de entrada tienen que estar en formato JPEG o PNG. Estas imágenes, pueden ser cargadas desde Amazon S3 (Amazon Simple Storage Service), o desde el sistema de archivos local del usuario.

En cuanto a Amazon Rekognition Video, este servicio hace posible el análisis de videos por medio de técnicas de Aprendizaje Automático o Machine Learning, y es capaz de detectar personas en movimiento, rostros, contenido inapropiado en transmisiones de video en directo o almacenadas, etc.

Cabe mencionar que Amazon Rekognition Image realiza sus operaciones de forma síncrona, mientras que Amazon Rekognition Video de forma asíncrona.

En cuanto a las operaciones realizadas con Amazon Rekognition, existen dos tipos de categorías en función de si se desea que la aplicación guarda los resultados (operaciones con almacenamiento) o no (operaciones sin almacenamiento).

2.4.5.3. Microsoft Azure

Microsoft Azure utiliza el servicio Computer Vision Azure para reconocimiento de imágenes. Proporciona acceso a algoritmos avanzados capaces de procesar imágenes, y devolver datos en base a características visuales de interés.

Este servicio de análisis de imágenes se divide en tres marcos o apartados:

- Stream Analytics, que se encarga de analizar los datos que se van a procesar.
- Azure Machine Learning, que realiza las técnicas de aprendizaje automático sobre los datos procesados.
- Data Lake Analytics, que es un servicio de Big Data para procesado de datos.

Dentro de Computer Vision, el servicio de Azure más relevante es el de reconocimiento óptico de caracteres (OCR), cuya finalidad es la extracción de texto de imágenes.

2.5. Análisis de las soluciones

2.5.1. Métodos de anotación de imágenes.

Los métodos de anotación de imágenes probados son 3: makesense, labelstudio y roboflow.

2.5.1.1. Roboflow

Herramienta para favorecer el etiquetado de conjuntos de imágenes. Pasos a seguir:

1. Escoger u obtener un conjunto de imágenes y/o videos.
2. Integrar etiquetas (labels) con los mecanismos de etiquetado que nos ofrece roboflow.
3. Aumento del conjunto de datos. Roboflow ofrece una gran multitud de opciones para aumentar el conjunto de imágenes y hacerlo más rico, como puede ser rotar las imágenes, pasárlas a b/w, difuminarlas, hacer un efecto espejo...
4. También permite exportar los conjuntos de datos en el formato que se desee, como para usarlo en yolov5, por ejemplo. Ofrece diferentes opciones.

Primeramente, se debe de crear un nuevo proyecto de anotación dentro de una carpeta. Para crear el proyecto se debe de presionar sobre el botón que se muestra en la figura 2.26.

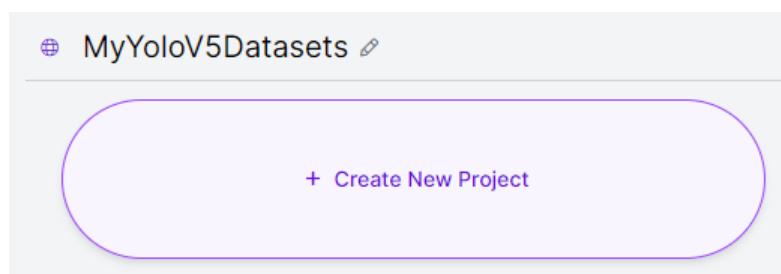


Figura 2.26: Nuevo proyecto en Roboflow. Referencia: Elaboración propia.

A continuación, se le debe dar un nombre, seleccionar si se desea que el dataset sea de dominio público o privado, nombrar el tipo de proyecto que se pretende desarrollar (en este caso, será detección de objetos por medio de bounding boxes), y definir el tipo de objeto que se pretenderá detectar.

The screenshot shows the 'Create Project' interface. At the top, it says 'MyYoloV5Datasets / New Public Project'. The 'Project Name' field contains 'cell_dataset'. The 'License' dropdown is set to 'Public Domain'. The 'Project Type' dropdown is set to 'Object Detection (Bounding Box)'. The 'What will your model predict?' field contains 'cells'. At the bottom, there are 'Cancel' and 'Create Public Project' buttons.

Figura 2.27: Descripción de un nuevo proyecto. Referencia: Elaboración propia.

Acto seguido, se debe de pasar a la pestaña "Generate", donde se irá diseñando paso a paso el dataset. El primer paso, que se muestra en la FIGURA 2.28, es subir el conjunto de imágenes sobre el que se realizarán las anotaciones.

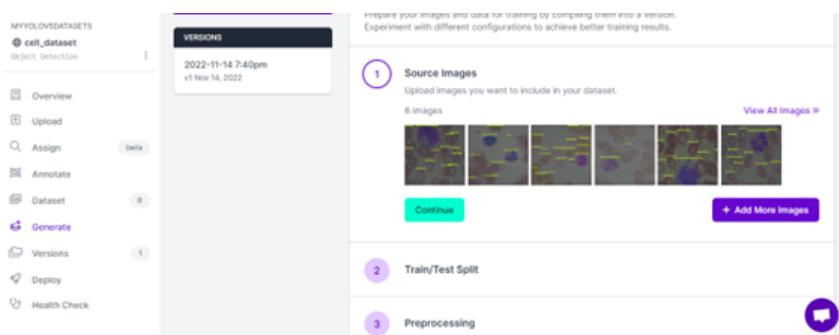


Figura 2.28: Carga del conjunto de datos en Roboflow. Referencia: Elaboración propia.

El siguiente paso consiste en dividir el conjunto de datos anteriormente cargado en tres subconjuntos, uno para el futuro entrenamiento de la red neuronal, otro para su posterior validación y el último para realizar el test final que refleje los resultados del entrenamiento (figura 2.29).

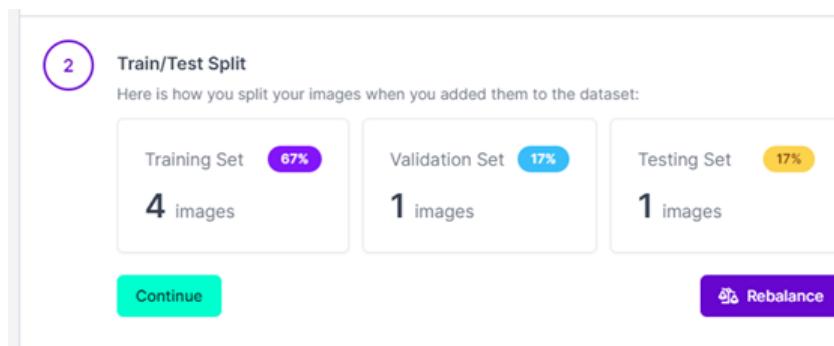


Figura 2.29: División del conjunto de datos. Referencia: Elaboración propia.

Tras haber realizado la segmentación anterior, debe de realizarse la anotación de las imágenes. En la figura 2.30 se muestra un apartado en el que se deben de ir seleccionando las diferentes imágenes para colocar un bounding box sobre el objeto que se pretende detectar. Las imágenes que se vayan anotando, pasarán a la pestaña ".^annnotated".

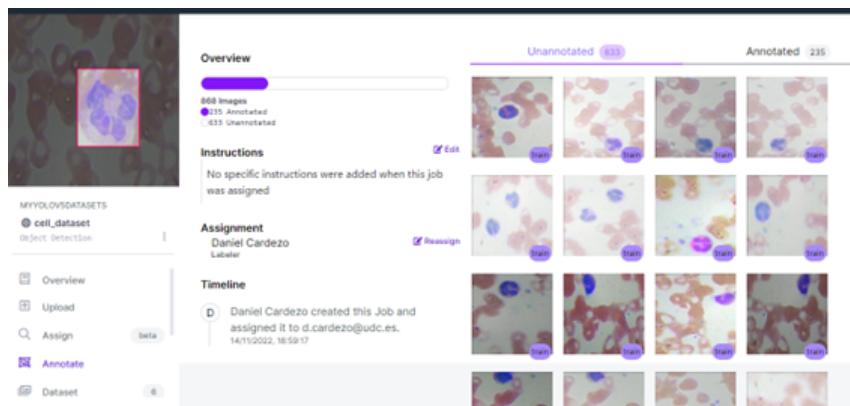


Figura 2.30: Sección para anotación de imágenes. Referencia: Elaboración propia.

En la figura 2.31 hay un ejemplo del proceso que se debe de llevar a cabo para la anotación de una imagen. Se deben de seguir los siguientes pasos:

1. Seleccionar en la barra de tareas de la derecha el recuadrado de objetos.
2. Colocar un rectángulo (bounding box) sobre el objeto de la imagen que se desea detectar.
3. Asociar este objeto a la clase a la que pertenezca. En caso de querer detectar más de un objeto, deberá de crearse una clase para cada tipo. Para crear una nueva clase, basta con escribir el nombre que se le desee atribuir en el ".^annotation Editor". En este caso, existen dos clases distintas: WBC (White Blood Cell) y RBC (Red Blood Cell).

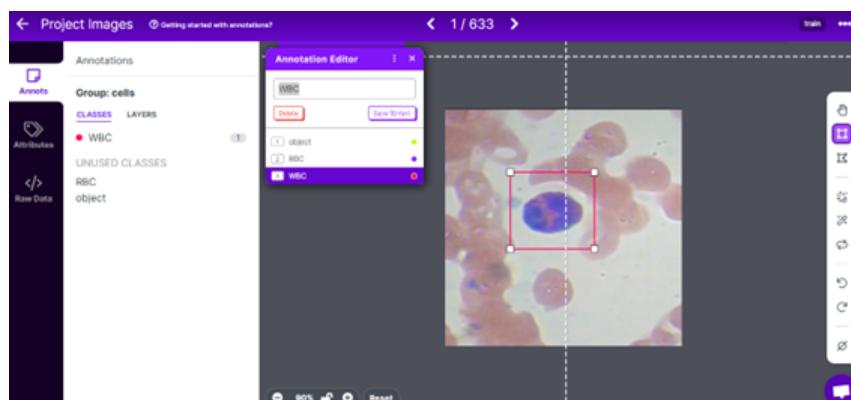


Figura 2.31: Ejemplo de anotación de una imagen en Roboflow. Referencia: Elaboración propia.

Una vez todas las imágenes estén anotadas, Roboflow ofrece la opción de realizar un preprocesado de imágenes, en donde se pueden seleccionar distintos pasos de preprocesado, en este caso (figura 2.32) se utiliza una auto orientación y un reajuste con el fin de obtener un tamaño de imágenes de 640x640.

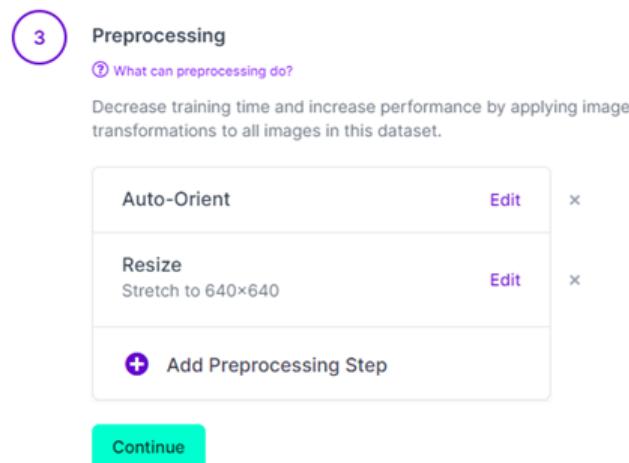


Figura 2.32: Fase opcional de preprocesamiento de datos. Referencia: Elaboración propia.

Finalmente, y antes de poder exportar el dataset, el programa ofrece la posibilidad de aumentar el número de imágenes mediante diferentes técnicas como pueden ser: pasar a escala de grises, girar las imágenes 90º o 180º, realizar una rotación de las bounding boxes, etc. (Figura 2.33).

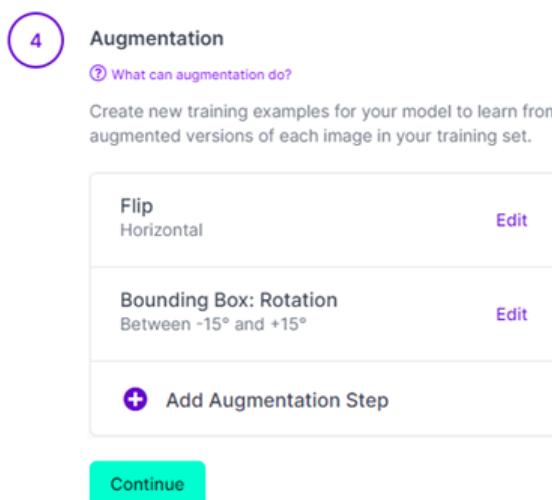


Figura 2.33: Fase opcional de aumento del dataset. Referencia: Elaboración propia.

Cabe destacar que Roboflow permite distintos modos o formatos para exportar los conjuntos de datos, incluso, permite exportar en .zip el conjunto de datos en el ordenador o generar un link que permite acceder a este sin necesidad de descargarlo, y al ejecutarlo en código se accede a él de forma remota.

2.5.1.2. MAKENSENSE

Inicialmente, se debe cargar el conjunto de imágenes, se han seleccionado unas imágenes de un tablero de ajedrez con sus piezas. Es necesario seleccionar si se va a realizar una detección de objetos o reconocimiento de imágenes. En este caso, se escogerá reconocimiento de objetos.

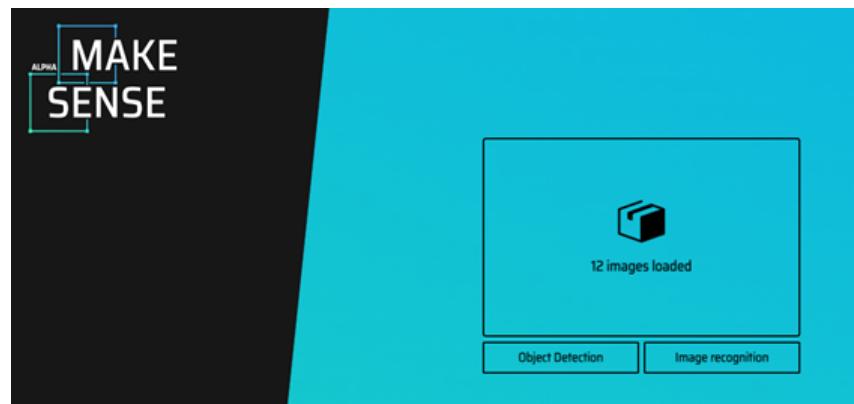


Figura 2.34: MAKENSENSE. Referencia: Elaboración propia.

Antes de empezar con la anotación de imágenes, debemos seleccionar las clases de nuestras etiquetas. En este ejemplo, tendremos una clase para los peones y una clase para los caballos.

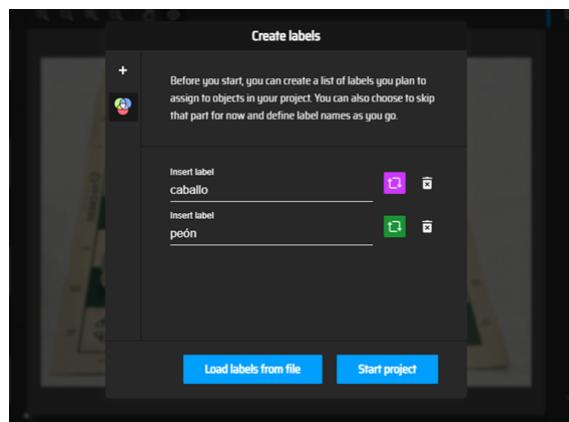


Figura 2.35: Selección de las clases que se van a usar para la anotación de imágenes. Referencia: Elaboración propia.

Acto seguido, se irán seleccionando las imágenes que se deseen etiquetar y se rodearán con rectángulos las distintas piezas, con las etiquetas correspondientes.

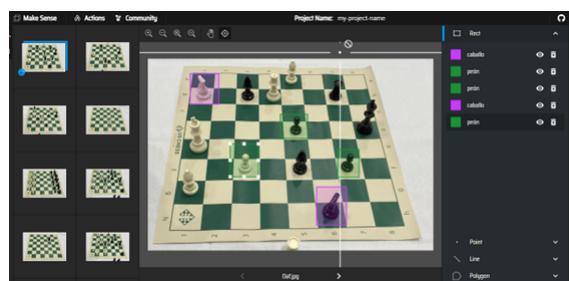


Figura 2.36: Ejemplo de una imagen anotada en MAKESENSE. Referencia: Elaboración propia.

Finalmente, una vez anotadas las imágenes, se pueden exportar n el menú acciones.

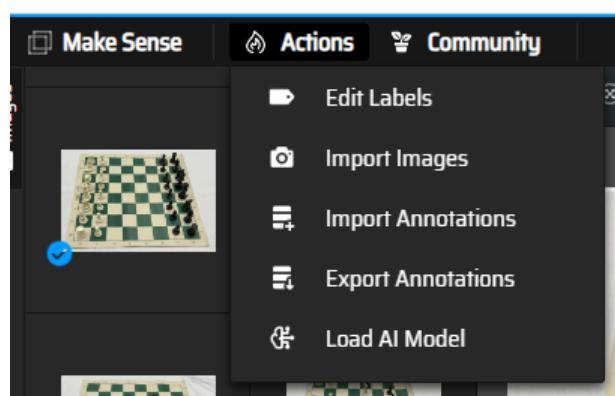


Figura 2.37: Exportar dataset. Referencia: Elaboración propia.

También permite la exportación en distintos formatos, aunque en menos que roboflow.

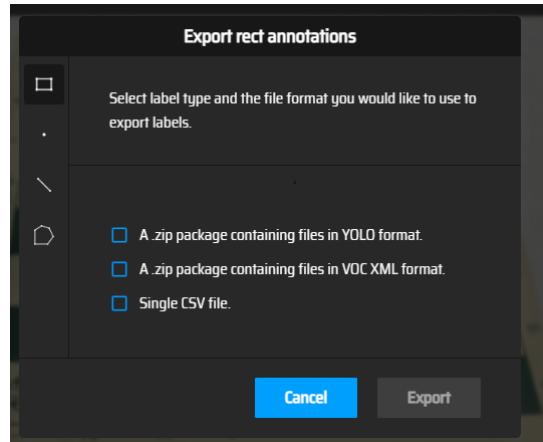


Figura 2.38: Formatos disponibles para exportar el conjunto de imágenes. Referencia: Elaboración propia.

2.5.1.3. Label-studio

Label-studio es un programa para anotación de imágenes que se puede descargar directamente desde cmd haciendo uso del comando "pip install label-studio".

Para acceder al programa, también debe de escribirse el siguiente comando en el cmd: "label-studio".

Una vez dentro del entorno, la aplicación solicitará introducir el nombre del proyecto, importar las imágenes que compondrán el conjunto y definir los distintos tags que se utilizarán para anotar los objetos en las imágenes.

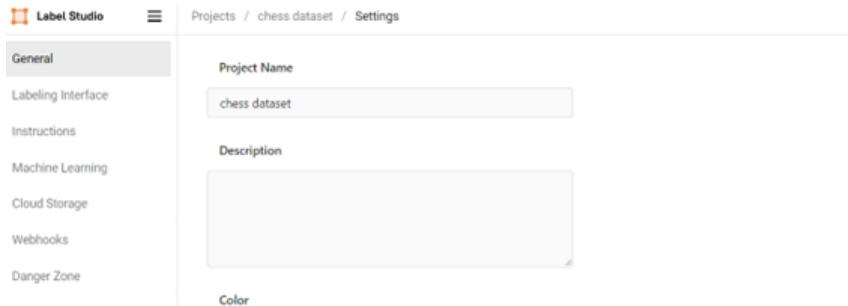


Figura 2.39: Ejemplo de definición de un proyecto. Referencia: Elaboración propia.

En el apartado "Labeling interface", se debe de seleccionar el tipo de anotación que se desea, además de especificar el tipo de conjunto de datos introducido (en este caso, imágenes).

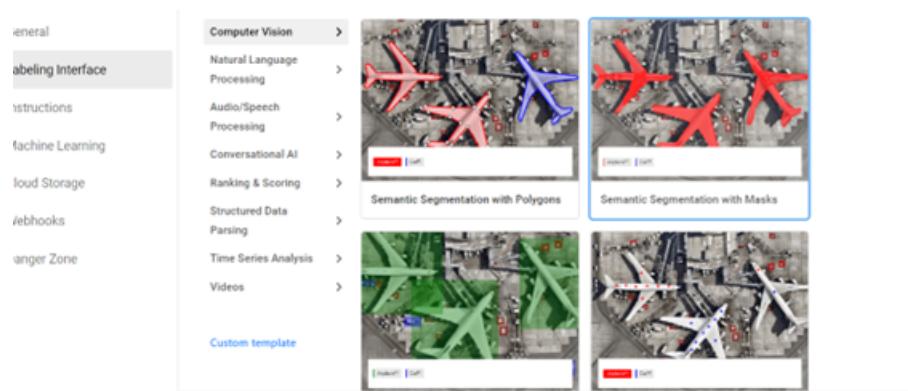


Figura 2.40: Selección del tipo de conjunto a anotar y del tipo de anotación. Referencia: Elaboración propia.

Una vez definido, se continúa con el etiquetado de imágenes con los tags definidos anteriormente.

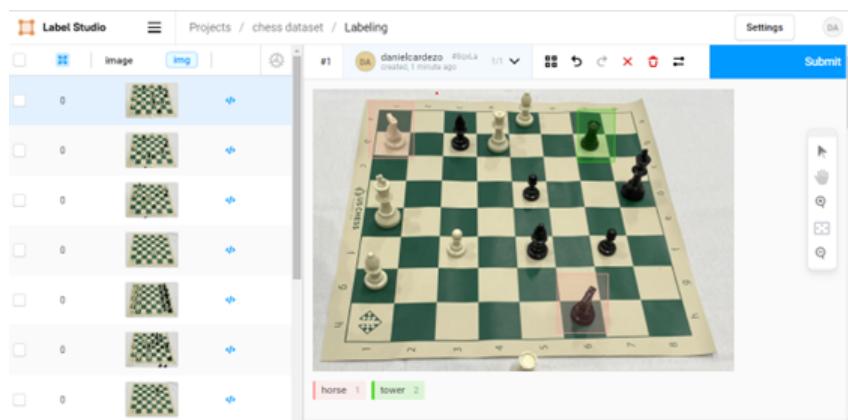


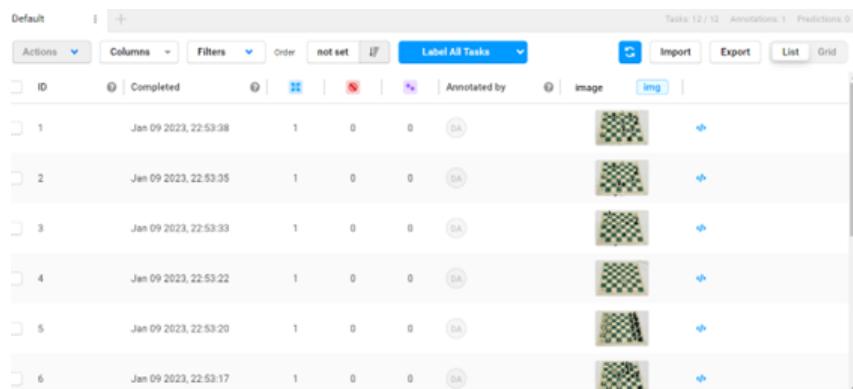
Figura 2.41: Ejemplo de anotación de una imagen en Label-studio. Referencia: Elaboración propia.

Finalmente, cuando todas las imágenes han sido anotadas correctamente, es necesario presionar el botón "Submit".



Figura 2.42: Botón submit, parte superior derecha del entorno. Referencia: Elaboración propia.

Presionando el botón .^{Export}.en la parte superior derecha de la imagen, se pueden seleccionar distintos formatos para exportar el conjunto de datos (Pascal XML, Yolov5, etc).



The screenshot shows a software interface for managing a dataset. At the top, there are buttons for 'Actions', 'Columns', 'Filters', 'Order', 'Label All Tasks', 'Import', 'Export', 'List', and 'Grid'. The main area displays a table with 6 rows, each representing a task. The columns are: ID, Completed (with a progress bar), Order (set to 'not set'), Annotated by (labeled 'DA'), and Image (a thumbnail of a chessboard pattern). Each row has a delete icon at the end.

ID	Completed	Order	Annotated by	Image
1	Jan 09 2023, 22:53:38	1	DA	
2	Jan 09 2023, 22:53:35	1	DA	
3	Jan 09 2023, 22:53:33	1	DA	
4	Jan 09 2023, 22:53:22	1	DA	
5	Jan 09 2023, 22:53:20	1	DA	
6	Jan 09 2023, 22:53:17	1	DA	

Figura 2.43: Imagen del entorno con dataset anotado. Referencia: Elaboración propia.

2.5.1.4. Conclusiones

Roboflow es un programa muy intuitivo al tener todos los pasos ordenados y una interfaz muy amigable con muchas ayudas, tiene datasets de ejemplo, permite aumento del conjunto de datos mediante distintas técnicas, se puede exportar en muchos formatos, y genera un enlace que se puede incluir directamente sobre las redes neuronales como yolov5. Sin embargo, tiene un número limitado de imágenes a incluir en el dataset, lo que merma notablemente su potencial.

Makesense es más sencillo, no permite aumentar las imágenes del conjunto, la exportación solo es posible en un .zip (no permite extraer una URL) y con cuatro tipos diferentes de exportación, menos completo que roboflow y que Label-studio. Por otro lado, no permite almacenar los conjuntos de imágenes, por lo que no puedes guardar los conjuntos para editarlos en situaciones futuras.

Label-studio es el más completo de todos. Como ventajas, se pueden destacar las siguientes:

- No tiene límite de imágenes a utilizar, por lo que se puede generar un dataset tan grande como se desee.
- Permite un gran número distintos de formas de anotación.
- Además de conjuntos de imágenes, permite anotar audios, vídeos, lenguaje natural, etc.
- Igual que Roboflow, tiene gran cantidad de formatos de exportación.

2.5.2. Librerías Python para desarrollo de interfaces.

2.5.2.1. TkInter

Es la librería más popular y más utilizada en el mundo del desarrollo de aplicaciones Python. Este marco GUI viene preinstalado con Python, por lo que no es necesario instalarlo por separado.

Se utiliza para crear aplicaciones GUI para computadoras, no es compatible para el desarrollo de aplicaciones móviles.

Con Tkinter, los elementos visuales reciben el nombre de widgets. También ofrece una amplia gama de elementos de uso común como marcos, botones, botones de verificación, etiquetas, etc.

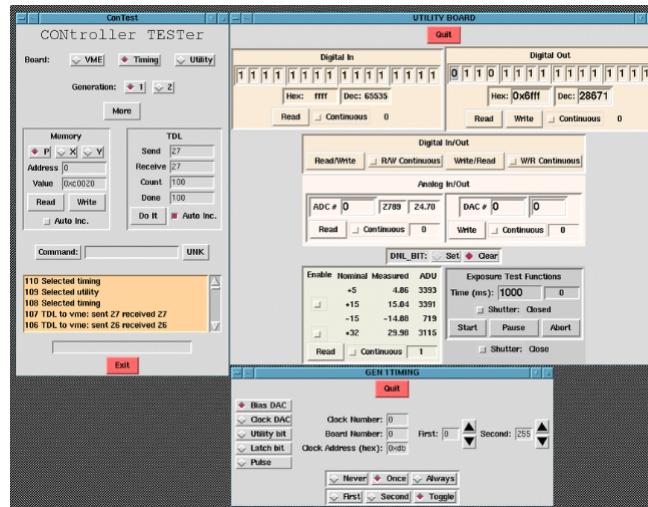


Figura 2.44: Ejemplo de GUI desarrollado con Tkinter. Referencia: [21]

2.5.2.2. Kivy

Kivy es un marco acelerado de OpenGL ES 2. Este soporte, permite técnicas y gráficos modernos.

Ha sido escrito con una combinación de Python y Cython, y permite el desarrollo de interfaces de usuario más intuitivas gracias a aplicaciones multitáctiles.

Esta da soporte a Windows, Linux, Android, Mac e IOS, entre otras.

También cabe destacar que, esta librería de código abierto incluye más de 20 widgets.



Figura 2.45: Ejemplo de GUI desarrollado con Kivy. Referencia: [22]

2.5.2.3. DearPyGUI

Kit de herramientas de interfaz gráfica de usuario multiplataforma, acelerado por GPU y de fácil uso para Python.

Incluye elementos GUI tradicionales como botones, botones de radio, menús, etc. También contiene una gran variedad de diagramas dinámicos, tablas, dibujos, etc.

Para mejorar el rendimiento, la mayor parte del código está escrita en lenguaje C++ utilizando la biblioteca Dear ImGui.



Figura 2.46: Ejemplo de GUI desarrollado con DearPyGUI. Referencia: [23]

2.5.2.4. PySimpleGUI

Esta librería ha sido desarrollada en 2018. Su finalidad principal reside en facilitar a los nuevos usuarios de Python comenzar a involucrarse en el desarrollo de GUI.

En resumen, PySimpleGUI permite al usuario comenzar de inmediato, eliminando o reduciendo las complejidades de algunas otras librerías.

Está basada en cuatro marcos de GUI: Qt, Tkinter, wxPython y Remi. Se puede elegir el marco de GUI que se desee de los anteriores y tener así un fácil acceso a elementos visuales que vienen en ellos.

2.5.2.5. WxPython

Funciona con variedad de entornos diferentes como Mac OS, Windows, Linux y sistemas basados en Unix.

WxPython incluye gran cantidad de widgets, uno de sus mayores puntos fuertes. Cabe destacar que, al verse muy bien en todas las plataformas, no requiere de grandes modificaciones personalizadas, lo que se traduce en que su curva de aprendizaje es mucho más pronunciada que la de otras librerías, como Tkinter.

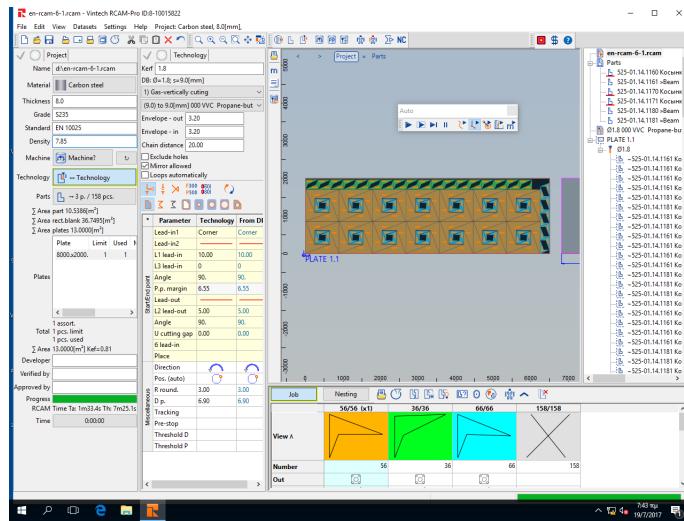


Figura 2.47: Ejemplo de GUI desarrollado con WxPython. Referencia: [24]

2.5.2.6. PyQt

Es uno de los marcos más populares de Python para el desarrollo de GUI. Está basado en el marco multiplataforma Qt.

El desarrollador puede insertar elementos visuales simplemente con arrastrarlos y soltarlos gracias a los módulos QtGui y QtWidgets, aunque también es posible insertar los elementos por medio de código.

Se usa para desarrollar aplicaciones en una gran variedad de plataformas como Mac, Windows, Linux, iOS y Android.

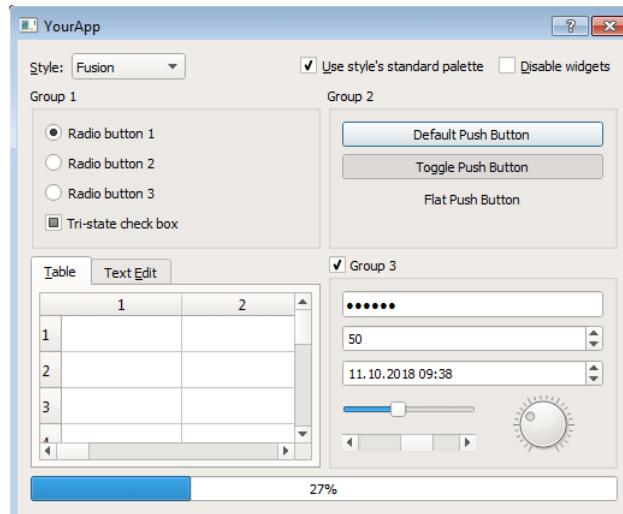


Figura 2.48: Ejemplo de GUI desarrollado con PyQt. Referencia: [25]

2.5.2.7. Librería a emplear

A continuación, se muestra una tabla destacando las ventajas de cada una de las librerías mencionadas anteriormente:

LIBRERÍA	VENTAJAS
PyQt5	Versatilidad de codificación. Varios componentes de la interfaz de usuario. Varios recursos de aprendizaje. Amplia variedad de API de plataforma nativa para redes. Fácil de dominar.
Kivy	Basado en Python El código escrito una vez se puede usar en todos los dispositivos. Widgets fáciles de usar con soporte multitáctil. Implementable en multitud de plataformas.
DearPyGUI	Toda la documentación está escrita en Python. Dispone de Python 2 y 3. Admite extensiones como OpenGL y GTK. Código abierto multiplataforma.
PySimpleGUI	Fácil para principiantes. No requiere complejidades avanzadas de otros marcos. Uyiliza QT, Tkinter, wxPython y Remi. Permite crear interfaces de usuario en cualquier marco de los anteriores.
WxPython	Gran biblioteca de Widgets. Apariencia nativa. Altamente flexible. Comunidad de usuarios útil.
Tkinter	Fácil de usar y rápido de implementar. Flexible y estable. Incluído en Python. Proporciona una sintaxis simple. Disponible sin cargo para uso comercial.

Tabla 2.4: Ventajas ofrecidas por librerías Python.

En este trabajo, se ha seleccionado la librería Tkinter, ya que, pese a carecer de widgets avanzados y de un aspecto nativo, es la más extendida y utilizada de todas, se presenta como una biblioteca Python subyacente y es sencilla de entender y dominar, de ahí que sea la más conocida de todas.

2.5.2.8. Técnicas de detección de objetos seleccionadas

El prototipo software contará con diferentes modelos de detección de objetos. Se han seleccionado los siguientes modelos de visión artificial:

- Google Cloud API

Se ha seleccionado esta técnica debido a que existe un número de peticiones a la plataforma de forma gratuita, además de haberse visto en la asignatura de Visión Artificial. Es una de las plataformas en la nube más relevantes en la actualidad.

- YOLOv5

Se ha seleccionado ya que es un modelo que usa redes convolucionales (CNN) y deep learning, además de ser uno de los modelos más rápidos en lo que a detección de objetos se refiere, debido a su arquitectura.

- HOG

HOG ha sido seleccionada por ser uno de los modelos más utilizados en cuanto a técnicas clásicas de detección de objetos se refiere. En la actualidad, sigue utilizándose para reconocimiento de rostros.

Cabe mencionar que este prototipo ha sido desarrollado de modo que sea fácil añadir más modelos en un futuro, gracias a su estructura modular, aunque debido al alcance limitado del proyecto se han restringido el número de ellas que se han integrado.

2.6. Resultados finales

2.6.1. Interfaz GUI

Se ha desarrollado un prototipo de software que consiste en una interfaz gráfica, cuya finalidad es facilitar al usuario la configuración inicial de cada nuevo proyecto de visión artificial que se realice con ella.

Para el desarrollo de la herramienta, se ha utilizado como lenguaje de programación Python, por ser el lenguaje con mayor difusión en ámbitos científico-técnicos relacionados con Machine learning y visión artificial. Para el diseño gráfico de la interfaz, se hace uso de las librerías tkinter y custom-tkinter.

La herramienta destaca por una arquitectura modular, fácilmente configurable por el usuario. Esto se consigue en gran parte gracias a las plantillas generadas (archivos .tplpy), las cuales, contienen de forma ordenada diversas partes de código, que la interfaz se encargará de juntar para crear un script final .py.

Además, estas plantillas se editan por la propia herramienta introduciendo los campos cubiertos por el usuario con los datos deseados (ruta de imágenes para entrenamiento, para almacenar los resultados, etc). De esta manera, el script resultante no necesita ser modificado, sino que es la propia interfaz la que se encarga de configurarlo y generarla para que pueda ser utilizado directamente.

En este prototipo, se han introducido a modo de ejemplo tres técnicas de detección de objetos diferentes, con un modelo de detección cada una, aunque la interfaz está pensada para que se puedan añadir más en un futuro si se desease. Estas son:

- Plataformas en la nube: Google Cloud Vision API.
- Redes neuronales: YOLOv5.
- Técnicas clásicas: HOG + SVM.

2.6.2. Mockups

A modo de aproximación inicial del prototipo, se han desarrollado unos bocetos a mano, con los que posteriormente se han creado unos mockups, que se constituyen como un estándar que la aplicación debe de seguir para los diferentes modelos introducidos en ella.

La estructura del prototipo será la siguiente:

- Pantalla de inicio de la aplicación. En ella, se explica brevemente cuál es la finalidad de la interfaz.

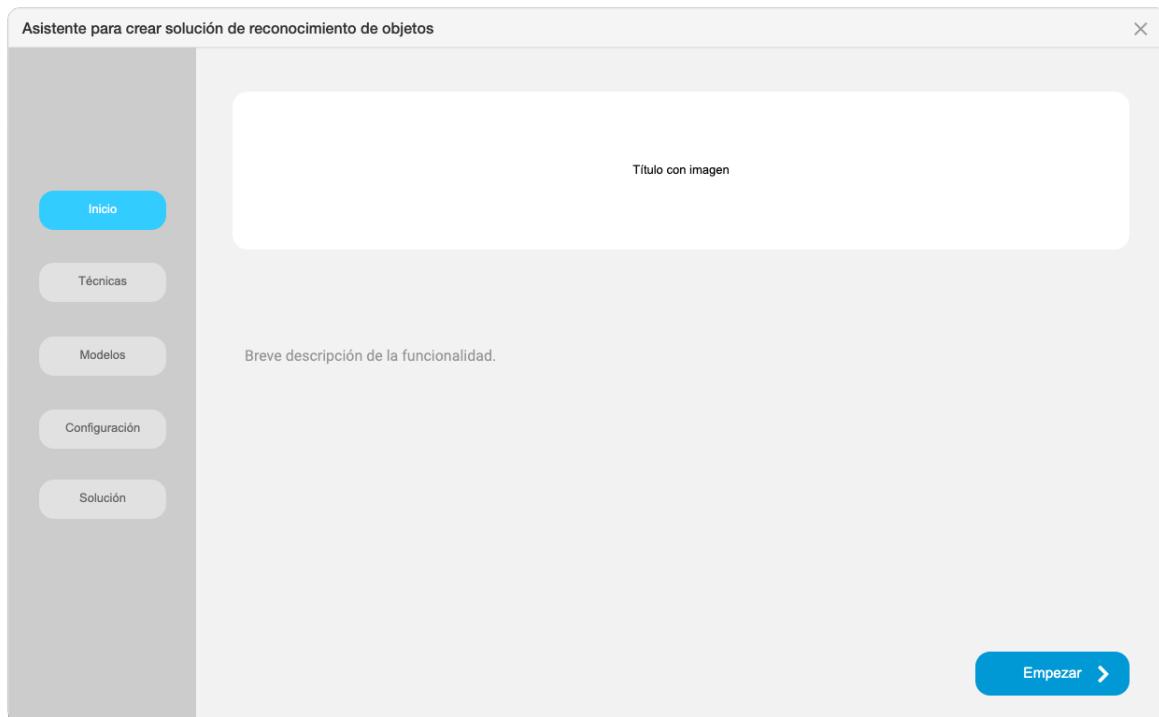


Figura 2.49: Pantalla de inicio de la aplicación. Referencia: Elaboración propia.

- Pantalla de selección de técnica. En esta pantalla se podrá seleccionar la técnica con la que se desea llevar a cabo la detección del objeto.

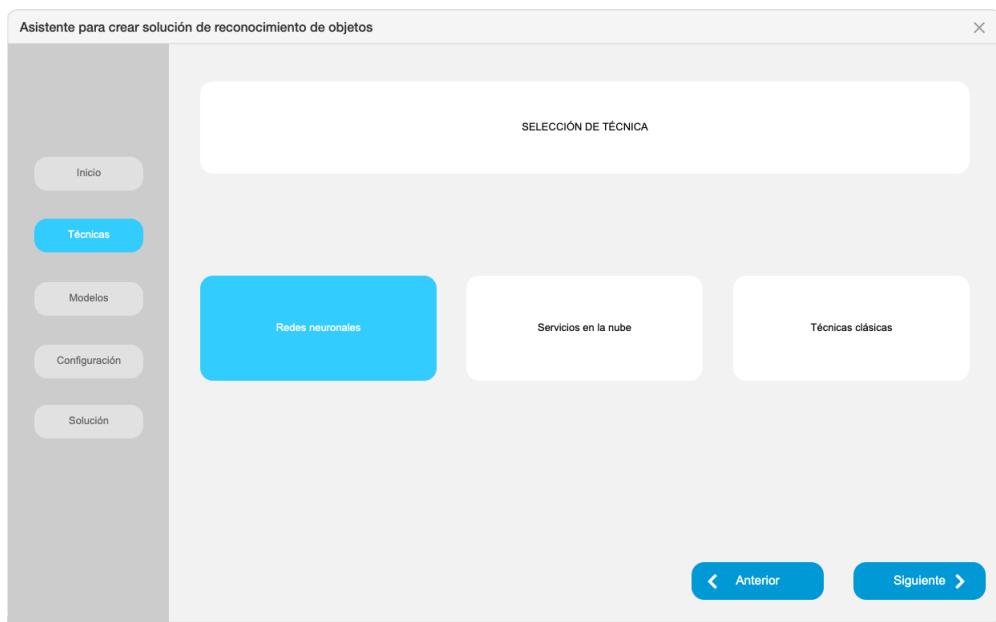


Figura 2.50: Pantalla de selección de técnica. Referencia: Elaboración propia.

- Pantalla de selección del modelo. Esta pantalla se replicará para cada una de las técnicas. En ella, se podrá elegir el modelo que se desee utilizar para la aplicación de reconocimiento de objetos dentro de la técnica seleccionada anteriormente.

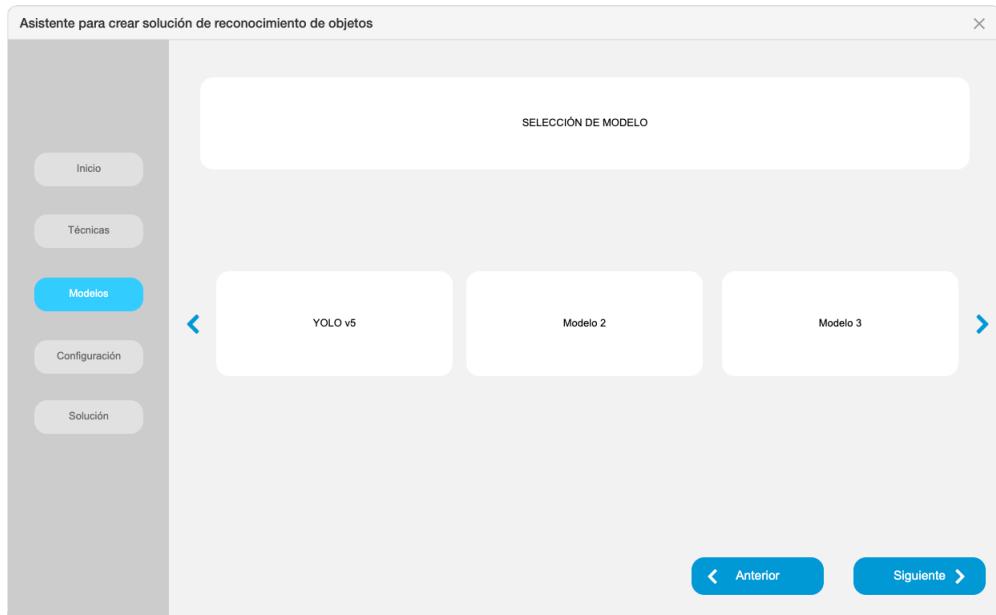


Figura 2.51: Pantalla de selección del modelo. Referencia: Elaboración propia.

- Pantalla de configuración. El usuario utilizará esta pantalla para configurar o personalizar su modelo de detección de objetos, seleccionando la carpeta de imágenes de entrenamiento, la carpeta de resultados, y diferentes opciones disponibles para cada modelo.

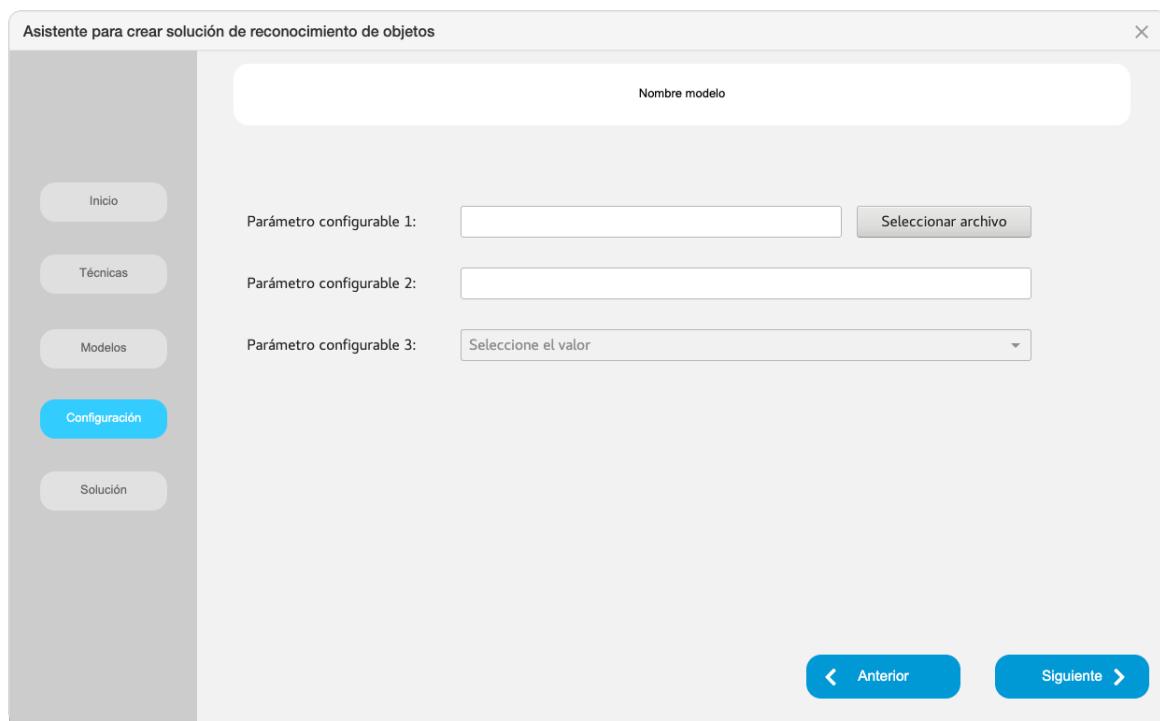


Figura 2.52: Pantalla de configuración. Referencia: Elaboración propia.

- Pantalla de solución. Pantalla final, donde se indica que el proceso ha concluido con éxito.

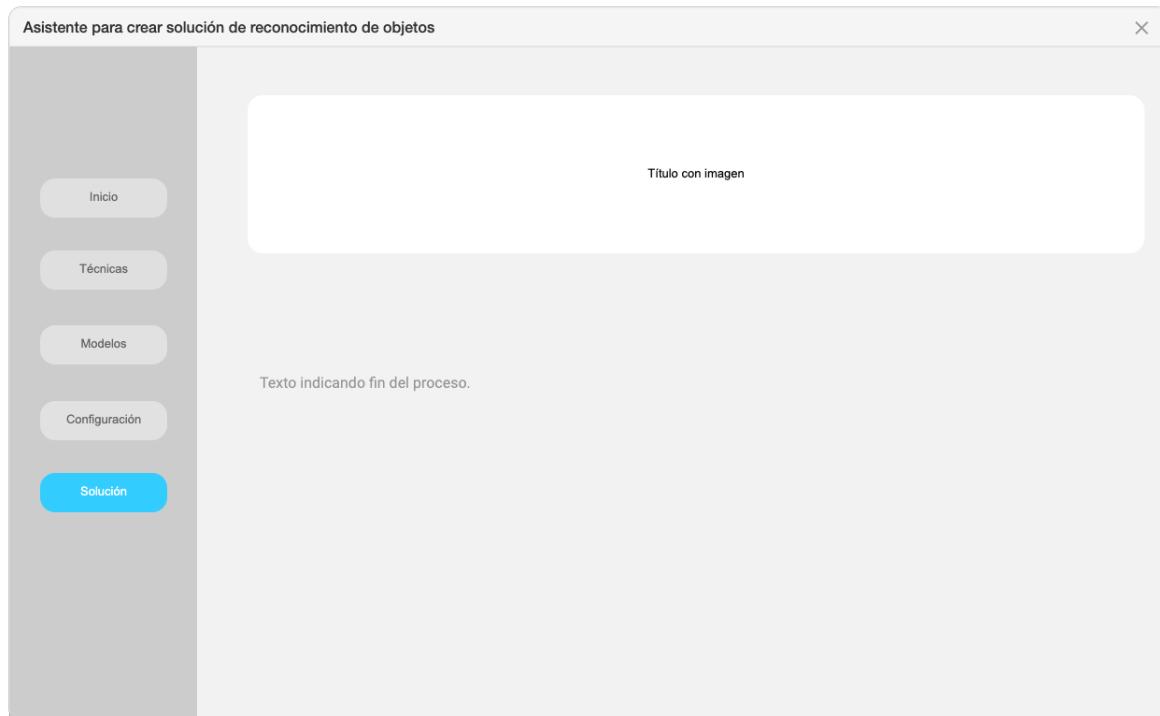


Figura 2.53: Pantalla de solución. Referencia: Elaboración propia.

Mencionar además que, paralelamente a las pantallas, en cada una de ellas figura un menú en la parte izquierda, donde se puede seleccionar la ventana a la que se desea acceder y se indica en otro color la ventana en la que el usuario se encuentra en todo momento.

2.6.2.1. Solución final

El resultado final se muestra a continuación, conserva la misma estructura planteada anteriormente en los mockups, aunque se han modificado o añadido algunas funcionalidades.

- Pantalla de inicio de la aplicación.

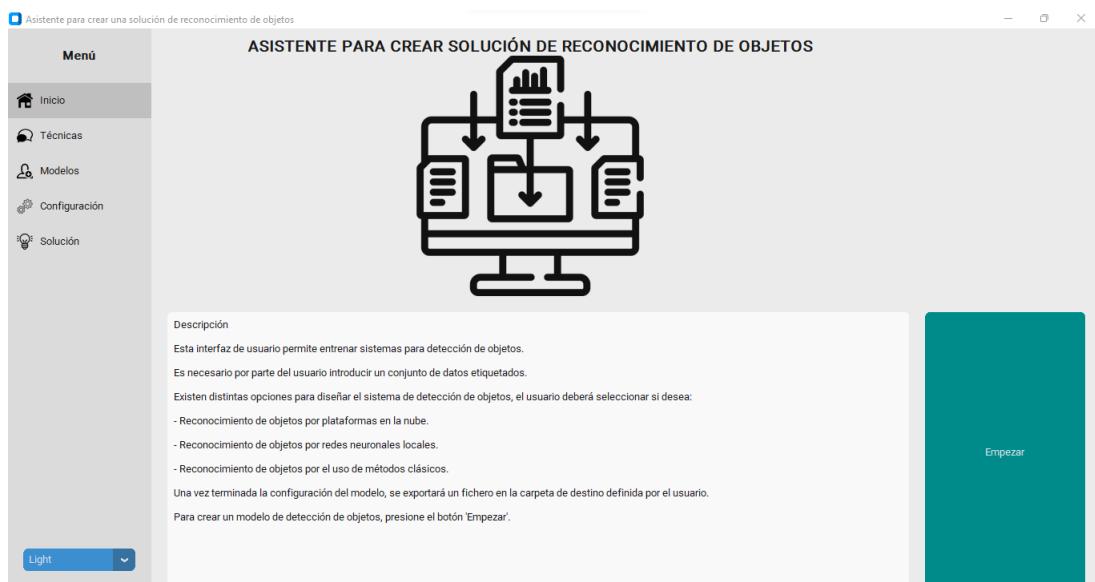


Figura 2.54: Pantalla de inicio de la aplicación. Referencia: Elaboración propia.

- Pantalla de selección de técnica.

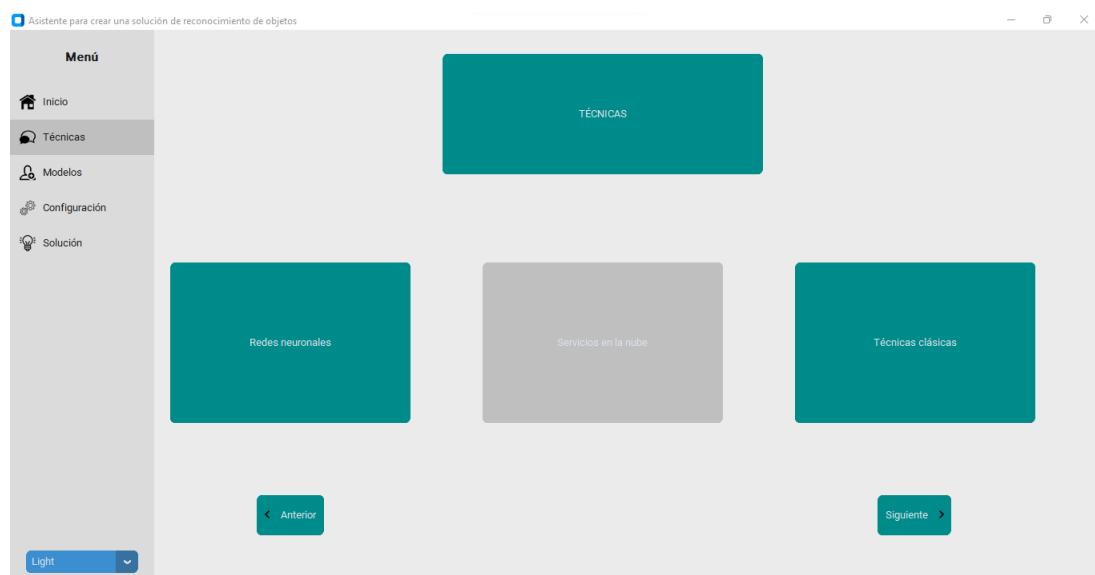


Figura 2.55: Pantalla de selección de técnica. Referencia: Elaboración propia.

- Pantalla de selección del modelo.

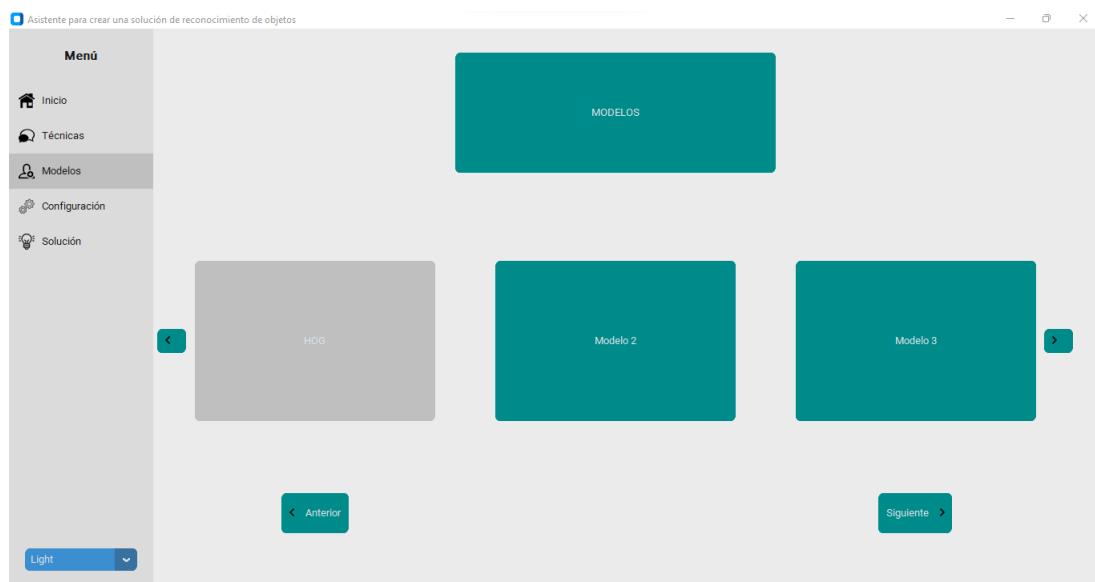


Figura 2.56: Pantalla de selección del modelo. Referencia: Elaboración propia.

- Pantalla de configuración.

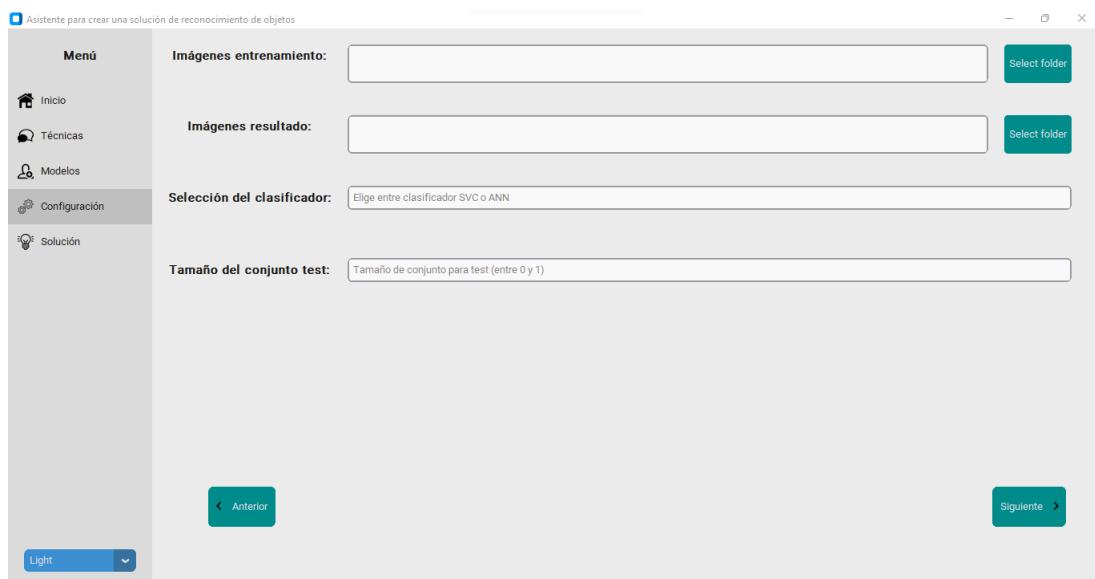


Figura 2.57: Pantalla de configuración. Referencia: Elaboración propia.

- Pantalla de solución.



Figura 2.58: Pantalla de solución. Referencia: Elaboración propia.

Se ha añadido en la parte inferior del menú izquierdo una opción para seleccionar el fondo de la aplicación, que puede ser claro u oscuro.

2.6.2.2. Código fuente

El código fuente del prototipo software desarrollado se encuentra en el siguiente repositorio:
https://github.com/GII/student_projects/tree/main/2023_daniel_cardezo

2.7. Orden de prioridad entre los documentos

En este capítulo de la Memoria el autor del Trabajo Final, frente a posibles discrepancias, debe establecer el orden de prioridad de los documentos del Trabajo Final. Si no se especifica otra cosa, el orden de prioridad debe ser el siguiente:

1. Memoria.
2. Presupuesto.
3. Anexos.

Título Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

Anexos

Peticionario Escuela Politécnica de Ingeniería de Ferrol
Rúa Mendizábal, s/n, Campus de Esteiro,
15403, Ferrol

Fecha Febrero 2023

Autor El Alumno

Fdo. Daniel Cardezo Mosquera

Índice Anexos

3.1 Contenido	57
3.1.1 Manual de usuario para manejo de la Interfaz Gráfica	57
3.1.1.1 Funcionamiento principal	57
3.1.1.2 Información común a todas las ventanas	59

El documento Anexos está formado por la documentación que desarrolla, justifica o aclara apartados específicos de la Memoria u otros documentos del Trabajo Final.

3.1. Contenido

3.1.1. Manual de usuario para manejo de la Interfaz Gráfica

3.1.1.1. Funcionamiento principal

Inicialmente, la aplicación se abrirá en su pantalla inicial, en la cual, se muestra una breve descripción de la finalidad y el uso de la interfaz.

Para avanzar hacia la selección de la técnica a utilizar se puede presionar el botón de Empezar, o también se puede presionar el botón "Técnicas" del menú principal.

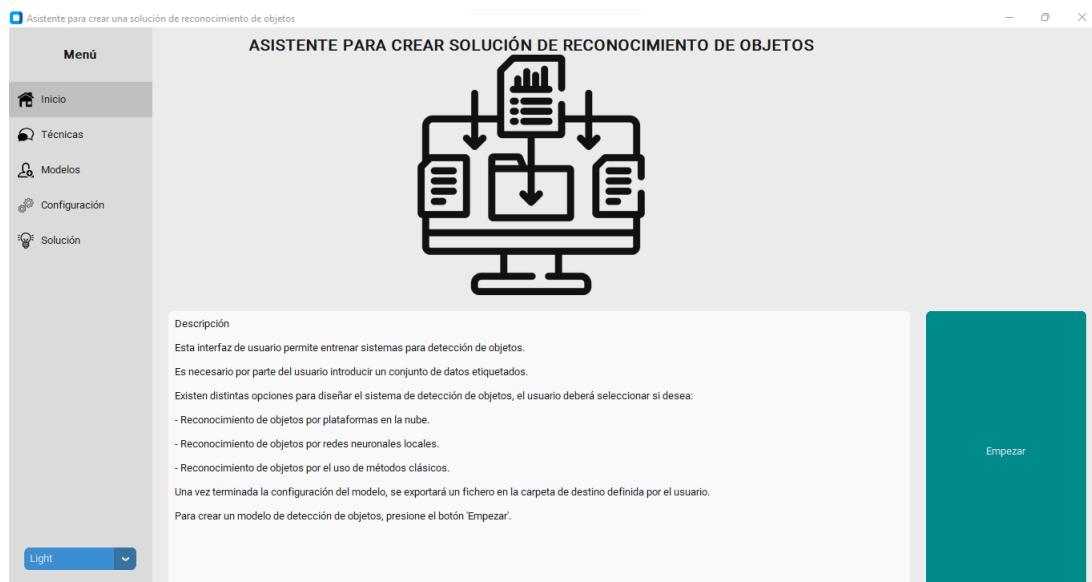


Figura 3.1: Pantalla de inicio de la aplicación. Referencia: Elaboración propia.

Una vez en la ventana "Técnicas", se debe de seleccionar la técnica deseada entre las tres disponibles ("Redes neuronales", "Servicios en la nube", "Técnicas clásicas"), haciendo click sobre el botón correspondiente. El botón seleccionado en cada momento estará teñido de un color diferente a los otros dos. Acto seguido, es necesario presionar el botón "Siguiente" para pasar a la pantalla donde se seleccionará el modelo deseado dentro de los disponibles para cada técnica.

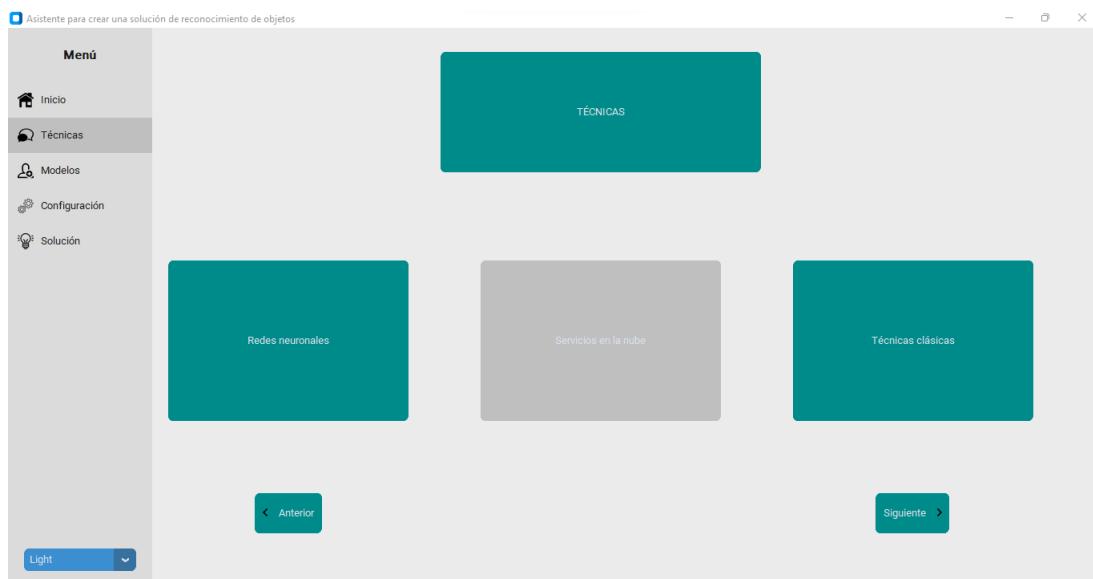


Figura 3.2: Pantalla de selección de técnicas. Referencia: Elaboración propia.

En la ventana "Modelos" se debe de escoger de la misma manera que en la ventana anterior el modelos deseado para realizar el algoritmo de detección de objetos. Esta ventana está configurada de modo que se puedan añadir diferentes modelos en un futuro. Además, en caso de querer añadir más de los que caben en la ventana, se han incorporado unos botones laterales que permitirían desplazarse y acceder a modelos inicialmente ocultos.

Pulsar sobre el botón "Siguiente" para avanzar hasta la ventana de configuración.

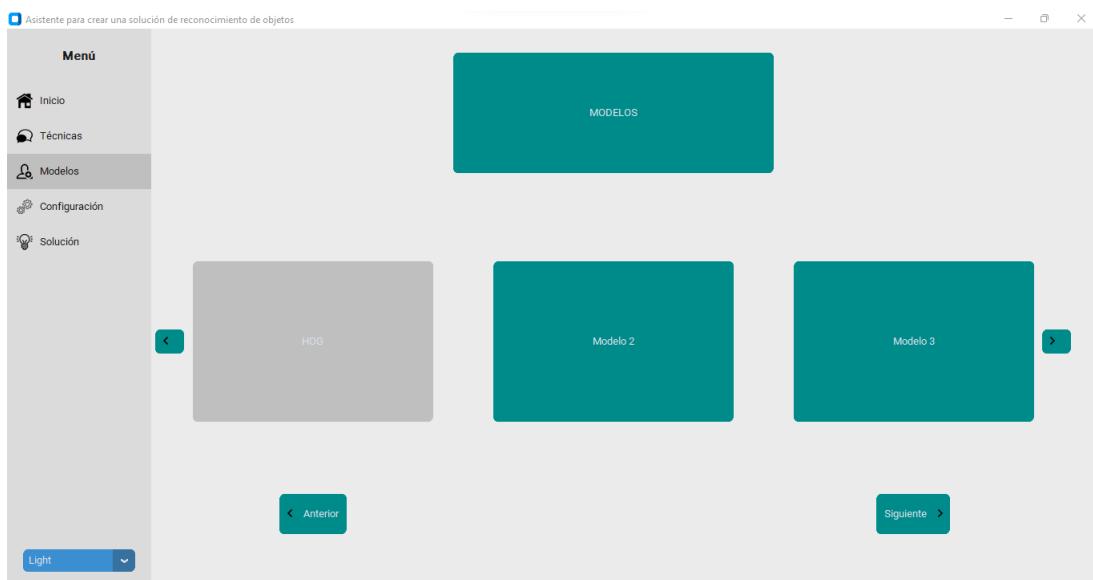


Figura 3.3: Pantalla de selección de modelos. Referencia: Elaboración propia.

En configuración, se pide al usuario que cubra los distintos campos en blanco que se pidan por parte de la interfaz.

En aquellos campos que sean para pedir una ruta de archivos, presionar sobre el botón que se encuentra a su derecha "Select folder". Se desplegará una ventana en la que el usuario

podrá seleccionar cómodamente la carpeta deseada.

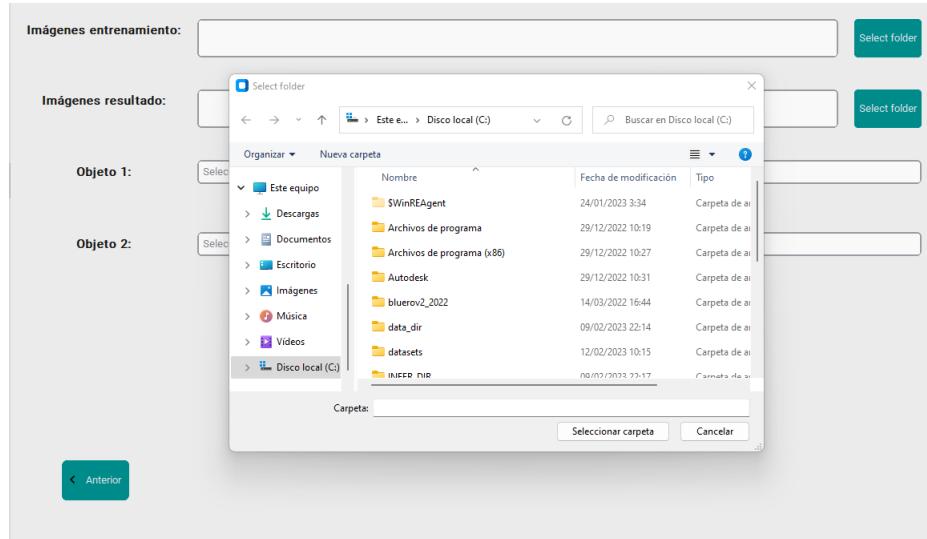


Figura 3.4: Ventana desplegable para selección de ruta de archivos. Referencia: Elaboración propia.

Aquellos campos que no sean para seleccionar una ruta, sino para configuración de los distintos modelos, tienen en los distintos espacios en blanco una pequeña información sobre qué escoger o cómo hacerlo. Por ejemplo, en el modelo Google Cloud, se pueden escoger los objetos a detectar por el modelo (deben de introducirse en inglés). Se indican las dos opciones disponibles.

Una vez realizada toda la configuración, se debe presionar el botón "Siguiente", situado en la esquina inferior derecha de la pantalla.

Finalmente, aparece la ventana de solución, en la que, se muestra que el proceso ha sido realizado correctamente.

3.1.1.2. Información común a todas las ventanas

En todas las ventanas existen elementos comunes que tienen la misma finalidad en todas ellas. Estos son:

- Botón Anterior.
- Menú principal.

Los botones que se encuentran en la zona inferior izquierda de todas las pantallas (Anterior) permiten regresar a la ventana inmediatamente anterior. De este modo, en caso de que se desee cambiar la elección del modelo o de la técnica, por ejemplo, sería posible regresar a la pantalla de selección y modificar la elección.



Figura 3.5: Menú principal de la aplicación. Referencia: Elaboración propia.

El menú principal se encuentra situado en el margen izquierdo de todas las pantallas. Aporta información de la ventana en la que se encuentra el usuario en todo momento. Además, permite desplazarse de unas ventanas a otras presionando los botones correspondientes a cada ventana. Mencionar que no es posible avanzar a la ventana de modelos sin haber seleccionado previamente una técnica. Lo mismo ocurre con la ventana configuración, es necesario seleccionar antes un modelo. Una vez se haya seleccionado todo ello por primera vez, estos botones se habilitarán, permitiendo regresar a las distintas pantallas para realizar modificaciones en la selección.

Además de estos botones, en este menú existe un selector en su parte inferior, que permite seleccionar el modo de visualización deseado, con el que el usuario puede seleccionar si desea que la aplicación se ejecute en modo claro (fondo blanco) o en modo oscuro (fondo negro).



Figura 3.6: Botón de selección de color de fondo. Referencia: Elaboración propia.

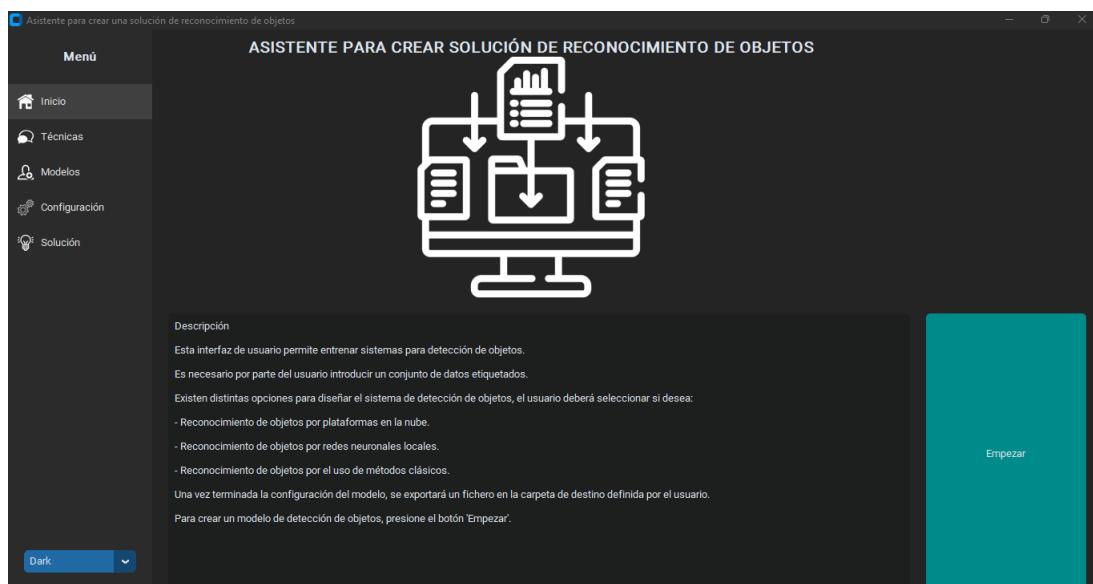


Figura 3.7: Ejemplo de pantalla con fondo oscuro. Referencia: Elaboración propia.

El código del prototipo software se puede ver en el enlace:

https://github.com/GII/student_projects/tree/main/2023_daniel_cardezo

Título Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

Estudios con entidad propia

Peticionario Escuela Politécnica de Ingeniería de Ferrol
Rúa Mendizábal, s/n, Campus de Esteiro,
15403, Ferrol

Fecha Febrero 2023

Autor El Alumno

Fdo. Daniel Cardezo Mosquera

Índice Estudios con entidad propia

No aplica.

Título Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

Otros anexos que justifiquen y aclaren conceptos expresados en el Trabajo Final

Peticionario Escuela Politécnica de Ingeniería de Ferrol
Rúa Mendizábal, s/n, Campus de Esteiro,
15403, Ferrol

Fecha Febrero 2023

Autor El Alumno

Fdo. Daniel Cardezo Mosquera

Índice Otros anexos que justifiquen y aclaren conceptos expresados en el Trabajo Final

No aplica.

Título Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

Planos

Peticionario Escuela Politécnica de Ingeniería de Ferrol
Rúa Mendizábal, s/n, Campus de Esteiro,
15403, Ferrol

Fecha Febrero 2023

Autor El Alumno

Fdo. Daniel Cardezo Mosquera

Índice Planos

6.1 Contenido	73
-------------------------	----

6.1. Contenido

No aplica.

Título Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

Pliego de condiciones

Peticionario Escuela Politécnica de Ingeniería de Ferrol
Rúa Mendizábal, s/n, Campus de Esteiro,
15403, Ferrol

Fecha Febrero 2023

Autor El Alumno

Fdo. Daniel Cardezo Mosquera

Índice Pliego de condiciones

7.1 Contenido	77
-------------------------	----

7.1. Contenido

No aplica.

Título Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

Mediciones

Peticionario Escuela Politécnica de Ingeniería de Ferrol
Rúa Mendizábal, s/n, Campus de Esteiro,
15403, Ferrol

Fecha Febrero 2023

Autor El Alumno

Fdo. Daniel Cardezo Mosquera

Índice Mediciones

8.1 Contenido	81
-------------------------	----

8.1. Contenido

No aplica.

Título Desarrollo de un prototipo software para asistir en la creación de soluciones de detección de objetos con visión por computador.

Presupuesto

Peticionario Escuela Politécnica de Ingeniería de Ferrol
Rúa Mendizábal, s/n, Campus de Esteiro,
15403, Ferrol

Fecha Febrero 2023

Autor El Alumno

Fdo. Daniel Cardezo Mosquera

Índice Presupuesto

9.1 Contenido	85
-------------------------	----

9.1. Contenido

SOFTWARE	
Producto	Importe
Librerías python de código abierto	0,00€
Servicios en la nube (capa gratuita)	0,00€
IMPORTE TOTAL	0,00€

Tabla 9.1: Presupuesto para desarrollo software.

MANO DE OBRA			
Producto	HORAS	Precio/Hora	Importe
Desarrollo del proyecto	100	35,00€	3.500,00€
IMPORTE TOTAL			3.500,00€

Tabla 9.2: Presupuesto para desarrollo software.

