Universidad de La Laguna

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

TRABAJO FIN DE GRADO:

Sistema de Visión Artificial para Monitorización y Seguimiento.

Grado en Ingeniería Electrónica Industrial y Automática

Autor:
Dylan Emanuele Santos Verzilli

Tutor:
Santiago Torres Álvarez

Julio de 2023

Universidad de La Laguna

AGRADECIMIENTOS

En primera instancia quiero agradecer a toda mi familia por el apoyo incondicional que me han dado a lo largo de los años, así como por las palabras llenas de sabiduría que me ayudaron en los momentos más difíciles. Ahora mismo hay un océano que me separa de una parte importante de esta, pero no hay un solo día que no agradezca por la fortuna de tenerlos a ellos.

Así mismo, agradecer a todos mis compañeros y amigos, tanto aquellos que me acompañaron en mi país de origen, como a todos los que a lo largo de estos cuatro años me recibieron con los brazos abiertos mejorando considerablemente mi día a día.

Finalmente, quiero agradecer a todos mis profesores por la formación y los conocimientos que me fueron proporcionados, especialmente a mi tutor Santiago Torres Álvares por la orientación a lo largo del desarrollo del proyecto.

RESUMEN

El presente trabajo tiene como objetivo principal el diseño de un sistema de vigilancia para monitorización y seguimiento de una zona delimitada, para lo cual se hará uso de un algoritmo de inteligencia artificial especializado para la detección de elementos pertenecientes a un conjunto de clases previamente definidas en el mismo, el YOLOv5 de ultralytics. A final de conseguir la detección esperada se debe llevar a cabo un proceso extenso de recolección y selección de imágenes relevantes a la tarea a realizar, para poder entrenar el algoritmo eficientemente.

Una vez entrenado el algoritmo, este deberá pasar por una serie de pruebas que confirmen el buen desempeño que presenta a la hora de realizar la detección de distintas situaciones posibles en las cuales puede encontrarse trabajando dicho sistema. Para ello se utilizarán una serie de imágenes y videos que cumplan con la variedad y complejidad de las situaciones esperadas.

De la misma forma, la implementación del sistema cuenta con un placa de circuito impreso especializada, la RaspBerry Pi 3 modelo B, para utilizar el modelo entrenado sobre distinta información obtenida por medio de la captura y grabación a través de un módulo de cámara conectado a esta.

Así mismo, serán discutidas aquellas líneas de mejora y posibles aplicaciones del sistema diseñado, con el fin de proporcionar propuestas para diseños futuros que mejoren el trabajo ejecutado por medio de la adición de distintos elementos que complemente aquellos posibles aspectos que no hayan sido implementados en el presente proyecto.

ABSTRACT

The main objective of this work is the design of a surveillance system for monitoring and tracking of a delimited area, for which use will be made of a specialized artificial intelligence algorithm for the detection of elements belonging to a set of classes previously defined in it, the YOLOv5 of ultralytics. In order to achieve the expected detection, an extensive process of recollection and selection of images relevant to the task to be performed must be carried out, in order to train the algorithm efficiently.

Once the algorithm has been trained, it must undergo a series of tests to confirm its good performance in detecting different possible situations in which the system may be working. In order to achieve the expected detection, an extensive process of collection and selection of images relevant to the task to be performed must be carried out, in order to train the algorithm efficiently.

Similarly, the system implementation relies on a specialized printed circuit board, the RaspBerry Pi 3 model B, to use the model trained on different information obtained by capturing and recording through a camera module connected to it.

Likewise, those lines of improvement and possible applications of the designed system will be discussed, in order to provide proposals for future designs that improve the executed work by adding different elements that complement those possible aspects that have not been implemented in the present project.

Universidad de La Laguna

ÍNDICE

Universidad de La Laguna

ÍNDICE DE TABLAS

Universidad de La Laguna

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

"Visión es saber qué hay y dónde mediante la vista" (Aristóteles).

La visión es la capacidad de poder reconocer, localizar y caracterizar distintos elementos presentes en el mundo. El ser humano lleva a cabo esta tarea a través de un complejo sistema de visión que parte de los ojos como órganos sensoriales, cuyas células son capaces de captar información del exterior a través de la luz que incide en estos y transmitirla a la corteza visual, región del cerebro responsable del procesamiento de la información.

Debido a la facilidad con la que el ser humano realiza el proceso de visión, se creó la idea equivocada de que conseguir desarrollar un sistema informático capaz de emular las capacidades visuales presentes en las personas sería algo sencillo. De lo anterior nació un proyecto denominado: "The Summer Visión Project", que pretendía dar solución al problema de la visión artificial a través de un proyecto de verano. Sin embargo, el problema de la visión artificial resultó ser de elevada dificultad y requirió de años de investigación para conseguir los resultados esperados.

Afortunadamente, los avances tecnológicos de la última década han permitido explotar el potencial de la visión por computadora gracias a los desarrollos en inteligencia artificial. El concepto de Inteligencia Artificial (IA) fue inventado oficialmente en 1956 en la conferencia de Dartmouth por los investigadores John McCarthy, Marvin Minsky y Claude Shannon, para referirse a un conjunto de sistemas combinados que por medio de algoritmos consiguen crear máquinas capaces de imitar la inteligencia humana durante la resolución de problemas.

Como es de esperar, la IA ha revolucionado un gran números de campos de la ciencia siendo uno de los más impactados el de la visión por computador, que a través de la implementación de sistemas capaces de desarrollar distintas tareas de adquisición, procesamiento, análisis e

identificación de imágenes del mundo, ha conseguido solucionar numerosos problemas presentes en diversas disciplinas, como son:

- Imágenes médicas: Análisis de radiografías, detección de tumores y anomalías, etc.
- Inspección de calidad: Comprobación del cumplimiento de los estándares de calidad impuestos para determinados productos de una cadena de producción.
- Vigilancia: Monitorización y seguimiento a través de sistemas de vigilancia que contiene algoritmos de visión artificial.
- Detección facial: Mejora de cualidades de cámaras, aplicaciones de filtros y búsqueda en bases de datos.

Entre las muchas aplicaciones que se pueden mencionar, la vigilancia es una de las más desarrolladas. Existen sistemas que monitorizan una zona determinada con el fin el fin de garantizar el correcto cumplimiento de las normas establecidas. Estos sistemas utilizan algoritmos de visión artificial para llevar a cabo la detección de vehículos, personas, entre otros elementos que puedan encontrarse en el perímetro, además de identificar y evaluar características como puede ser la velocidad de un automóvil o si una persona está utilizando mascarilla facial en una zona donde se requiere su uso.

El objetivo principal del presente trabajo es el diseño e implementación de un sistema de visión por computador para monitorización y seguimiento. Este sistema utilizará un algoritmo de detección para reconocer a las personas y vehículos presentes en una región determinada.

Queda fuera del alcance de este proyecto el diseño de la arquitectura del algoritmo de reconocimiento.

2. MARCO TEÓRICO

El siguiente apartado busca desarrollar detalladamente los conceptos teóricos necesarios para abordar el proceso de diseño propuesto.

2.1. VISIÓN ARTIFICIAL

“La visión artificial o comprensión de imágenes describe la deducción automática de la estructura y propiedades de un mundo tridimensional, posiblemente dinámico, bien a partir de una o varias imágenes bidimensionales de ese mundo.” [V.S. Nalwa, 1993] [1]

En otras palabras, la visión artificial es una disciplina que engloba todos los procesos y elementos que enseñan al computador a ver e interpretar la información adquirida como lo haría el sistema de visión de un ser humano.

Un proceso de visión artificial puede descomponerse en las siguientes etapas: [1]

- Etapa de adquisición: Esta etapa requiere de los sensores adecuados para poder captar la imagen, así como la electrónica necesaria para su posterior digitalización.

- Transformaciones y filtrado de la imagen: Consiste en preprocesamiento que mejoren la calidad de la imagen para facilitar tareas de detección y reconocimiento.

- Etapa de segmentación: Consiste en dividir los elementos que se encuentran en la imagen, para poder trabajar con ellos de forma separada.

- Etapa de parametrización: Etapa encargada de extraer rasgos que proveen alguna información de interés.

- Etapa de clasificación: La última etapa consiste en la asignación de una etiqueta a cada objeto basándose en la información proporcionada por los descriptores.

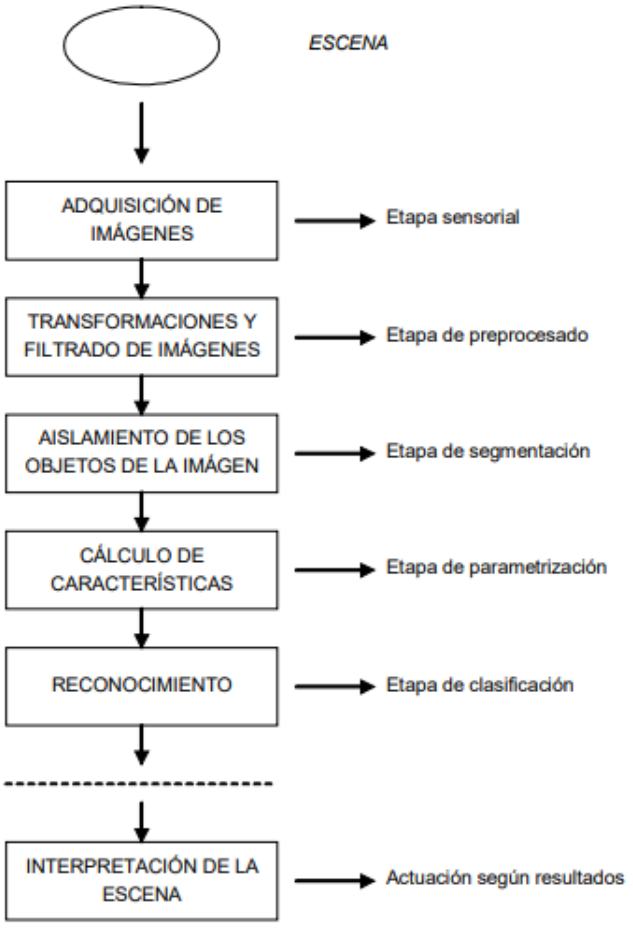


Figura 1. Esquema de etapas de los sistemas de visión artificial. [1]

Otra representación más simple de un sistema de visión artificial es a través de 3 grandes etapas:

- Procesamiento de bajo nivel.

- Procesamiento de nivel intermedio.

- Procesamiento de alto nivel.

Dichas etapas engloban los procesos anteriormente mencionados. La etapa de bajo nivel busca determinar las características más elementales de los objetos presentes en la imagen, como pueden ser el color, la textura o los bordes.

Así mismo, en el procesamiento de nivel intermedio se lleva a cabo la segmentación de los elementos, y se asignan etiquetas para poder agrupar aquellas partes de la imagen que comparten las mismas características obtenidas en el paso anterior.

Por último, la etapa final está conformada por aquellos procesos de interpretación de los entes presentes en la imagen, como puede ser identificación y reconocimiento.

De cualquier forma, el objetivo final de los sistemas de visión artificial es el de extraer características relevantes en una imagen para su posterior interpretación por el computador. [2]

2.2. PROCESAMIENTO DE IMÁGENES

El procesamiento de imágenes es un área que tiene una gran relación con la visión artificial, y que abarca todas aquellas acciones aplicadas a la

imagen con el fin de aumentar su calidad y prepararla para utilidades posteriores.

Como entrada del proceso se tiene una imagen, y a la salida se tiene otra imagen a la cual se le aplicó alguna operación que mejoró su calidad, como pueden ser mejora del contraste, eliminar defectos o problemas de desenfoque, entre otros.

Figura 2. Esquema general del procesamiento de imágenes [2]

Entre los procedimientos más comunes que pueden aplicarse a las imágenes tenemos:

-Mejora del contraste: Una imagen se representa como una matriz de números enteros usualmente comprendidos entre 0 y 255. Cada uno de estos números corresponde al valor de intensidad de los píxeles de la imagen, valores muy próximos a cero indican valores de intensidad bajos (tonos oscuros), y valores próximos a 255 intensidades altas (tonos claros).

Estos valores pueden representarse gráficamente a través de un histograma donde los valores del eje de la abscisas corresponden a la intensidad (0 a 255), y los que están presentes en el eje de las ordenadas son la cantidad de píxeles que tienen ese valor de intensidad.

La técnica más común para mejorar el contraste se denomina ecualización del histograma, que busca uniformizar los valores de intensidad de los píxeles de la imagen.

Figura 3.Mejora del contraste de la imagen a través de ecualización del histograma. [3]

-**Filtrado de imagen:** Existen dos tipos de filtrado, en el dominio del espacio y en el dominio de frecuencia. Para el caso que nos compete se va a tratar el filtrado espacial, ya que introduce un operador de gran relevancia para apartados posteriores del trabajo.

El filtrado espacial consiste en operar directamente sobre los píxeles de la imagen. Para ello se emplea una máscara, una matriz que generalmente suele ser cuadrada y consta de un conjunto de valores predeterminados en cada celda de la matriz. [2]

La máscara se centra sobre el píxel de interés de tal forma que el nuevo pixel resultado dependa del conjunto de píxeles vecinos cubiertos por la máscara. A cada celda de la máscara le corresponde un coeficiente denominado peso, donde el píxel resultante se obtiene de la sumatoria del producto de los píxeles vecinos con el peso correspondiente (Correlación).

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

Ecuación 1. Operación de correlación de la imagen y la máscara. [3]

Existe una variante a esta fórmula denominada convolución:

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l)$$

Ecuación 2. Operación de convolución de la imagen y la máscara. [3]

Esta operación es fundamental en el diseño de la arquitectura de las redes neuronales convolucionales, o *convolutional neural networks* (CNN). Estas redes son la base principal de la mayoría de algoritmos de *Deep Learning* que serán tratados en el apartado posterior.

Las máscaras también suelen llamarse *kernel* o máscara de convolución. Con esta operación puede conseguirse una gran cantidad de imágenes resultado dependiendo del valor de los pesos que tenga la máscara, como puede ser el filtrado de ruido, la detección de bordes u otro tipo de operación que genere una imagen de salida cuyas características se vieron modificadas de manera significativa posteriormente a la aplicación del operador.

De esta forma, el filtrado en el dominio de la frecuencia utiliza la transformada de Fourier para obtener el espectrograma de la imagen considerada. En el espectrograma se evidencian las componentes de frecuencia que posee la imagen. Estas componentes dan información de los cambios en tonalidad que presenta, una imagen de alta frecuencia tiene cambios bruscos de intensidad, mientras que en una imagen de baja frecuencia se puede observar que estos cambios se producen de manera suavizada a lo largo de esta.

Una vez determinada cada componente en frecuencia de la imagen, se debe diseñar un filtro que permita eliminar las porciones del espectro que no son de interés durante el procesado.

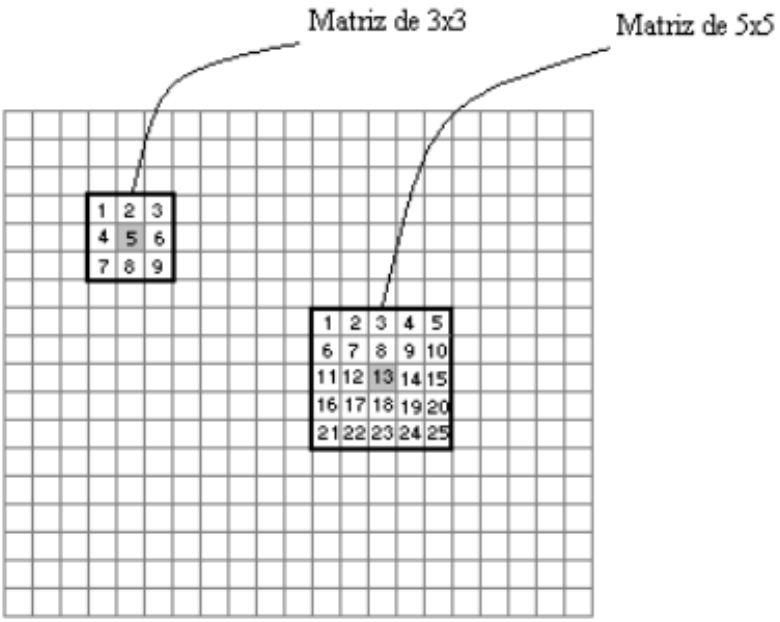


Figura 4. Operación de convolución en una imagen. [1]

2.3. DEEP LEARNING

Para la implementación del sistema de visión artificial, se utilizará un algoritmo basado en Aprendizaje Profundo o *deep learning* que consiga la correcta ejecución de los objetivos propuestos. Los algoritmos de *deep learning* son un caso particular de *machine learning* basados en redes neuronales, la gran mayoría de estos emplean algoritmos de optimización fundamentados en una técnica denominada gradiente estocástico descendente para conseguir el aprendizaje buscado. [4]

Antes de explicar este tipo de algoritmos, resulta de utilidad abordar los conceptos básicos relativos a *machine learning* para mejorar la comprensión de las técnicas que serán utilizadas más adelante.

2.3.1. FUNDAMENTOS DE MACHINE LEARNING

Machine Learning es una disciplina de la informática que se encarga del diseño de algoritmos capaces de emplear una colección de muestras (datos o información), con el fin de ejecutar una tarea particular por la que fueron diseñados. [5]

Un algoritmo de *machine learning*, generalmente es aquel que puede aprender a resolver un problema por medio de un conjunto de información proporcionada para entrenarlo.

Para construir un algoritmo de este tipo se debe poseer un conjunto de datos que fueron reunidos previamente, y que están ampliamente relacionados con la tarea que se pretende resolver, este conjunto suele denominarse *dataset*. Seguidamente, se debe conseguir la implementación algorítmica de un modelo estadístico utilizando el grupo de datos, de tal forma que el modelo diseñado consiga aprender a producir la salida esperada a partir de una entrada determinada. Sin embargo, existe la idea equivocada de considerar que un algoritmo de machine learning aprende como lo haría un ser humano, el proceso de aprendizaje en realidad consiste en determinar una fórmula matemática que al aplicarle una serie de entradas se obtengan las salidas deseadas, no solo para el conjunto de datos proporcionado para su aprendizaje, sino también para cualquier otra entrada que se aplique a este, siempre y cuando vengan de una distribución estadística similar a la que se utilizó para entrenar el algoritmo. [5]

Debido a esto, si se modifican ligeramente las entradas, existe una alta probabilidad de que las salidas obtenidas sean completamente erróneas, con lo cual la respuesta del modelo depende de que las nuevas entradas tengan una gran similitud con el dataset utilizado para su entrenamiento.

Así pues, Tom Mitchell en su libro Machine Learning (1997), proporciona la siguiente definición de algoritmo de aprendizaje:

"Se dice que un programa de computadora aprende de una experiencia E con respecto a un tipo de tarea T, siendo P una medida de su desempeño, si la medida de desempeño del algoritmo mejora con la experiencia E". [6]

Para poder entender en mejor medida cómo funcionan los algoritmos de aprendizaje, resulta de gran utilidad aprender a identificar estas tres características.

La tarea T, es el tipo de objetivo para el cual se diseña el sistema de *machine learning*, es el problema que se quiere resolver por medio del aprendizaje (entrenamiento del modelo). Una de las tareas más comunes en los sistemas de *machine learning* es la de clasificación, esta consiste en determinar, dentro de un número de categorías definidas, cual es la categoría a la que pertenece una entrada específica. [6]

Un ejemplo de un problema de clasificación es la detección y reconocimiento de objetos, proporcionar como entrada del algoritmo una imagen que contenga un elemento que pertenezca a una de las clases definidas, y se obtenga como salida la clase a la que tiene mayor probabilidad de pertenecer la entrada proporcionada.

A su vez, La experiencia E, comprende el *dataset* que contiene la información empleada para el entrenamiento del sistema con el fin de mejorar su desempeño, en función de cómo se administre el conjunto de datos, existen algoritmos de aprendizaje supervisado y no supervisado. [6]

Finalmente, el desempeño P, cantidad utilizada para poder llevar a cabo la medida del desempeño del sistema, habitualmente se utilizan dos tipos de medidas del desempeño, la precisión (*Accuracy*) y la tasa de error. La primera evalúa el porcentaje de aciertos que tiene el modelo, es decir, el número de

veces en que la salida del mismo es correcta dada una entrada determinada, la tasa de error es exactamente lo contrario. [6]

2.3.1.1 TIPOS DE APRENDIZAJES

De acuerdo al tipo de aprendizaje o experiencia, los modelos de *machine learning* pueden clasificarse en:

- Aprendizaje supervisado: Se denomina aprendizaje supervisado cuando un algoritmo aprende a asociar una entrada con una salida a través de un conjunto de datos de entrenamiento de las entradas y salidas previstas, debido a que en algunos casos las salidas pueden resultar complicadas de obtener, se proporcionan directamente por una persona "supervisor". [5]

- Aprendizaje no supervisado: Es aquel en el que la tarea de aprendizaje, para extraer información relevante acerca de una distribución, no requiere de una anotación previa por parte de una persona para conseguir que el modelo responda apropiadamente. [5]

En general, la diferencia que existe entre ambos métodos de aprendizaje es que en uno se proporcionan anotaciones de los elementos que corresponden a las salidas esperadas del modelo (aprendizaje supervisado), y en el otro no.

Como ejemplo, si se quiere identificar un animal en una imagen, un método de aprendizaje supervisado requiere que para cada conjunto de datos de entrenamiento (imágenes del animal), se proporcionen de manera adicional una serie de etiquetas que indican la ubicación del animal en la imagen.

2.3.1.2 DISEÑO DE UN ALGORITMO DE MACHINE LEARNING

A continuación, se van a describir brevemente los pasos a seguir para diseñar un programa que aprenda a ejecutar una tarea por medio de un proceso de entrenamiento.

- Selección de los datos de entrenamiento: El tipo de información que se proporcione al algoritmo tiene un impacto considerable en el nivel de desempeño con el que consiga ejecutar los objetivos propuestos, con lo cual, este paso comprende una decisión crítica de diseño. Debido a esto, es importante una correcta selección del tipo de información que utilizará el sistema para llevar a cabo el aprendizaje, una representación adecuada de esta información que produzca la retroalimentación óptima para conseguir una disminución progresiva del error a medida que aumenta su experiencia y finalmente, un mecanismo de aprendizaje que ejecute todo el proceso. [6]

- Selección de la función objetivo: Este paso comprende la definición del proceso de aprendizaje utilizado en el algoritmo. Para ello se debe determinar una función que sea capaz de producir la salida deseada a partir de una entrada proporcionada, en la mayoría de los casos se suele definir una aproximación de la función objetivo debido a la gran dificultad que puede presentarse al utilizar la función original. [6]

  Adicionalmente, se debe escoger una correcta representación de la función, una representación con una precisión considerable implica un mayor set de entrenamiento y resulta costoso computacionalmente, con lo cual este paso debe evaluarse apropiadamente de acuerdo a las herramientas disponibles.

- Selección del algoritmo de aproximación de la función: Considerando una función objetivo de tipo polinómica, tal que se define como:

$$Y(salida) = wo.x2 + w1.x1 + w2.x$$

Ecuación 3. Ejemplo de una ecuación objetivo.

siendo (w0, w1, w2) los pesos de la función, y (x2, x1 ,x) las entradas a la función.

Se presenta el problema de determinar los valores de cada peso para obtener la salida adecuada respecto a un conjunto de entradas dado. Para ello se utilizan los sets de entrenamiento, cada set de entrenamiento es un conjunto de salidas y entradas proporcionado al sistema por el usuario para completar el aprendizaje.

El ajuste de los pesos se consigue por medio de un algoritmo de optimización que minimice el error que existe entre la salida real del set de entrenamiento, y la que proporciona la función objetivo una vez administrada la entrada, siendo el error considerado:

$$Error = Y(train) - Y(modelo)$$

Ecuación 4. Error entre la salida real y la predicción hecha por el modelo.

Uno de los algoritmos más utilizados para realizar el proceso de optimización es el LMS (*least mean squares* / mínimos cuadrados).

Una vez culminado el proceso de entrenamiento, el resultado de la actualización de los pesos de la función objetivo garantiza que el algoritmo ejecute su tarea correctamente. Incluso para datos fuera del

conjunto de entrenamiento, siempre y cuando tengan una distribución parecida a la de los datos utilizados para generar los pesos.

2.3.1.3 MOTIVACIÓN PARA EL DESARROLLO DEL DEEP LEARNING

A pesar de la gran utilidad que presentan los algoritmos de *machine learning*, existen una serie de inconvenientes presentes en estos que motivaron el desarrollo de nuevas técnicas para poder hacer frente a problemas de inteligencia artificial mucho más complejos como son la detección y reconocimiento de objetos.

De igual forma, algunos de los problemas más comunes presentes en machine learning que motivaron el desarrollo de *deep learning* son los que se describen a continuación.

- **La maldición de las dimensiones**. Es un fenómeno que surge al trabajar con datos de múltiples dimensiones. Al aumentar las variables, se produce un aumento exponencial en las posibles configuraciones del set de variables, es decir, las salidas del sistema. [6]

  Por tanto, el número de datos de entrenamiento aumenta considerablemente y consigo el coste computacional, ya que se requiere procesar más información.

- **Necesidad de aprendizaje supervisado:** En la misma línea, otra problemática de gran importancia es la necesidad de emplear un supervisor en algoritmos de gran complejidad, debido a que requieren ser guiados a lo largo de cada fase de tal forma que sepan identificar cada categoría automáticamente. En otras palabras, hay que proporcionar al algoritmo la semántica necesaria para que en el futuro sea capaz de ejecutar las tareas de forma independiente. [7]

2.3.2 DEEP NEURAL NETWORKS

Las redes neuronales surgen como caso particular del aprendizaje supervisado en los algoritmos de *machine learning*. La idea general se basa en desarrollar un algoritmo cuya arquitectura sea capaz de imitar al complejo sistema de neuronas biológicas que presentan los seres humanos.

Las neuronas son las células básicas del sistema nervioso de los seres vivos, permiten el funcionamiento adecuado de los mismos a través de la transmisión de impulsos eléctricos desde un punto hasta su destino. En la misma línea, el cerebro humano está compuesto por seis capas de células nerviosas que por medio de la comunicación entre estas permiten complejos procesos análisis de información, dando como resultado que el ser humano tenga la capacidad de juicio, decisión y comprensión de lo que le rodea. [8]

Cada capa de neuronas tiene una función específica, utilizando la información proporcionada por las capas anteriores para extraer características relevantes y finalmente comunicar dicha información a las capas posteriores.

Del mismo modo que el cerebro humano está formado por conjuntos de neuronas interconectadas entre sí, una red neuronal artificial está formada por neuronas artificiales conectadas entre sí y agrupadas en diferentes niveles llamados capas. Así pues las capas son una agrupación de neuronas cuya entrada proviene de un nivel previo y cuya salida será recibida por el nivel posterior. [9]

La arquitectura de una red neuronal artificial está constituida principalmente por:

- Capa de entrada: Aquella capa conformada por el conjunto de neuronas que reciben la información que es alimentada a la red.
- Capa de salida: Comprendida por el conjunto de neuronas del último nivel, en esta capa se proporciona el resultado de las operaciones llevadas a cabo en la red.
- Capas ocultas: Engloban a todas las capas intermedias entre la entrada y la salida donde se ejecutan el conjunto de operaciones necesarias para conseguir los objetivos del sistema.

El *deep learning* hace referencia a la utilización de un gran número de capas ocultas en las redes neuronales. [9]

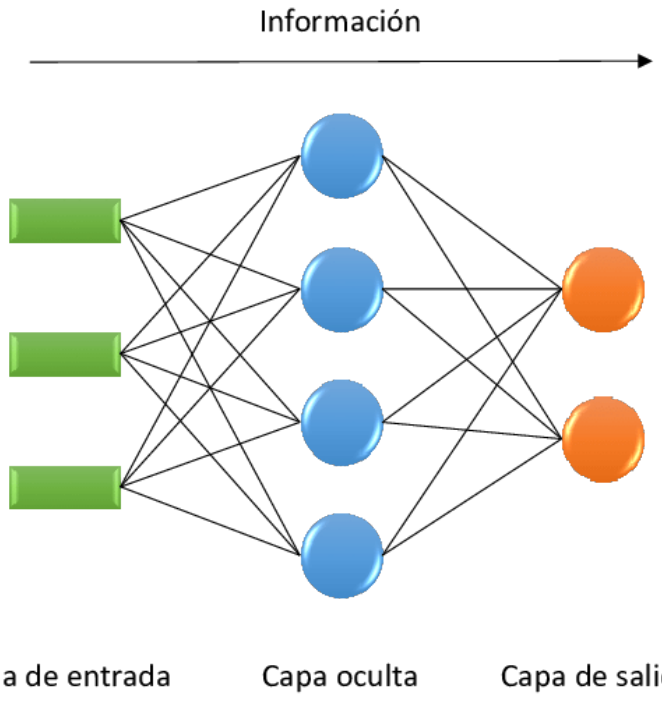


Figura 5. Esquema simple de la arquitectura de una red neuronal artificial. [10]

Como entrada a una neurona artificial se tiene un conjunto de valores que son multiplicados por el peso (coeficiente) de cada entrada, una vez hecho esto, el grupo de entradas se suma y sobre sí se aplica una función matemática

definida en la propia neurona. La salida generada se transmite a las capas posteriores para repetir el proceso hasta llegar a la capa de salida.

En resumen, cada neurona artificial recibe información de diversas unidades, computará su propia acción y transmitirá el resultado a las neuronas siguientes. El término de "*network*" hace referencia a la forma de representar estos sistemas por medio de la composición de muchas unidades (neuronas) agrupadas entre sí. La "profundidad de la red" (*depth*), indica la longitud de ésta, es decir, el número de capas ocultas presentes en la arquitectura. [4]

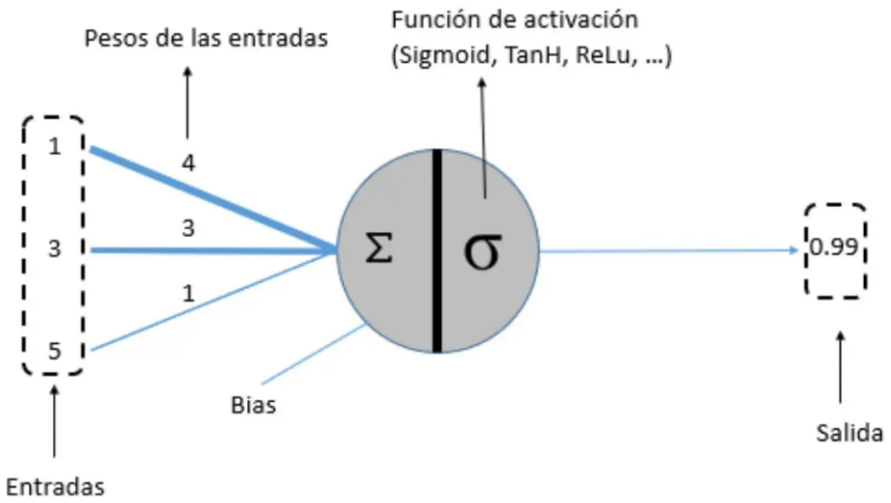


Figura 6. Esquema de una neurona artificial. [9]

Al contrario que los algoritmos de *machine learning*, las redes neuronales artificiales utilizadas en los algoritmos de *deep learning* están pensadas para ser entrenadas en su totalidad, de manera que puedan recibir

como entrada la información directamente y produzcan la salida esperada sin la necesidad de un procesamiento previo para extraer características. [3]

Para conseguir que el algoritmo lleve a cabo su función es necesario un entrenamiento que constituye el aprendizaje del algoritmo. Como paso previo al entrenamiento, debe definirse un conjunto de elementos de gran importancia para el mismo.

- La arquitectura o topología de la red (profundidad y tamaño).
- Regularización y normalización.
- Definición de la función de costes.
- Definición de un procedimiento de optimización.
- Inicialización de los pesos de la red.

El proceso de regularización y normalización abarca todas aquellas técnicas utilizadas para conseguir que el modelo de la red responda de forma adecuada a datos distintos de los empleados en el proceso de entrenamiento, es decir, busca que la red aprenda la relación existente entre los datos, siendo capaz de generalizar el proceso para un conjunto nuevo de los mismos.

El entrenamiento se centra en comparar la salida de la red neuronal con un valor de referencia (resultado esperado a la salida de la red), la comparación genera un error entre las dos cantidades. Seguidamente, se emplea un procedimiento de optimización que va ajustando el valor de los pesos de las entradas de forma iterativa hasta conseguir que el error se reduzca lo máximo posible, de tal forma que la salida se ajuste en mejor medida al valor esperado.

Dicho lo anterior, la función de costes es el error entre el valor esperado y la salida de la red, cuyo proceso de optimización da como resultado los pesos necesarios en la red para que dicha función sea mínima.

Adicionalmente, para llevar a cabo el proceso de entrenamiento es fundamental disponer de una gran cantidad de datos relevantes al objetivo que debe realizar la red neuronal.

2.3.3 REDES NEURONALES CONVOLUCIONALES (CNN)

Las redes neuronales convolucionales son un tipo particular de red neuronal artificial de gran relevancia para el campo de la visión por computador. La idea principal consiste en organizar cada una de las capas de la red a través de una estructura denominada “mapa de características”. [3]

Estos mapas están constituidos por múltiples filtros convolucionales de una o más dimensiones, conformando un conjunto capas ocultas especializadas con una jerarquía determinada que permiten la ejecución de una función específica. [12]

Las redes neuronales convolucionales están diseñadas para imitar el funcionamiento de la corteza visual primaria (V1), el conjunto de mapas permite identificar distintas características primarias de la imagen a medida que esta recorre los sucesivos planos que componen la red. Estas capas se van especializando al profundizar en la estructura hasta llegar a las capas encargadas de la detección de características de mayor complejidad. [12]

2.3.3.1 FUNCIONAMIENTO DE LAS CNN

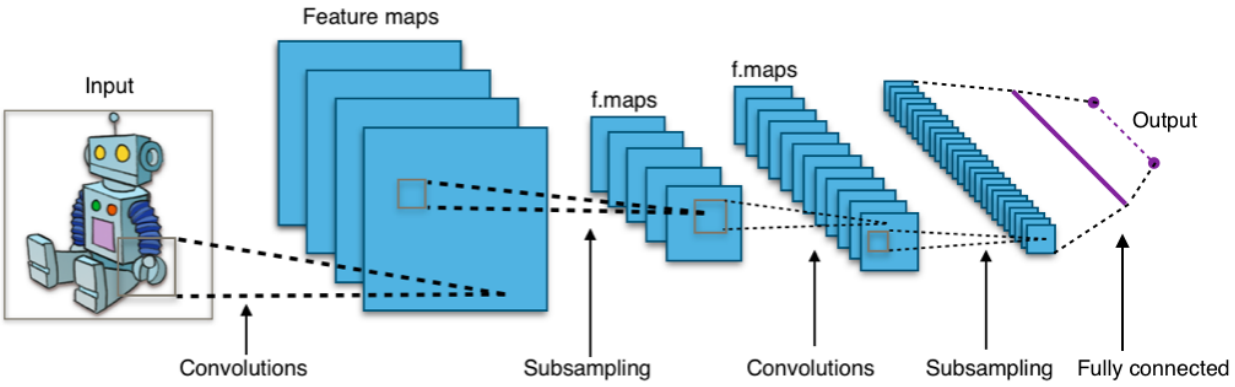


Figura 7. Estructura de una CNN. [13]

Convolución: Considerando una imagen cualquiera como entrada de la red CNN, el paso inicial consiste en realizar la operación de convolución entre la imagen y una máscara o *kernel* de convolución previamente entrenado. El resultado de esta operación es una imagen transformada en la cual cierta cantidad de píxeles tendrán un valor mayor al del resto, dando lugar a que se destaquen características de la imagen. [11]

Reducción de muestreo: Una vez hecho el paso de convolución, se produce un *subsampling* (reducción de muestreo) sobre la imagen filtrada, este paso es fundamental para reducir el número de neuronas en las capas posteriores puesto de lo contrario, el número necesario de las mismas crecería exponencialmente. Además, al reducir la resolución prevalecen en mejor forma las características principales que detectó cada filtro en la etapa de convolución y aumenta la eficiencia de la red en general, puesto que el número de elementos que deben procesar las capas a medida que se profundiza en la red se reduce en un valor significativo. [12]

Una de las técnicas más utilizadas para realizar este paso se conoce con el nombre de *maxpooling*, que consiste en buscar el máximo valor de una ventana de muestra cualquiera y pasarlo como resumen de las características de ese área. Consiguiendo simultáneamente un descenso considerable de neuronas y una mejora en el almacenamiento de la información más significativa para la detección de las características. [12]

Fase clasificatoria: A medida que se realizan más operaciones de convolución, los mapas de características serán capaces de identificar formas más complejas como pueden ser rostros o dígitos. Al llegar a la última capa oculta, los datos ya fueron depurados lo suficiente hasta una serie de características únicas de la imagen, en la última fase del proceso se clasifican las características por medio de una etiqueta en función del objetivo del entrenamiento. [11]

En resumen, las redes neuronales convolucionales tienen la peculiaridad de imitar la estructura de la corteza visual V1, organizándose en grupos de neuronas que ejecutan funciones específicas a medida que la información transcurre en la red.

Al utilizar una imagen de entrada, se aplican operaciones de convolución con máscaras de tamaño determinado que barren la imagen, generando una nueva imagen filtrada en la que pueden destacarse ciertos grupos de características. La capa siguiente se encarga de llevar a cabo un sub-muestreo, reduciendo la resolución de la imagen filtrada para adaptarla a las capas posteriores. En este paso se evidencian las características dominantes en la imagen, y también permite un aumento en la eficiencia de la red al reducir el número de neuronas.

Los pasos anteriores se ejecutan de manera continua hasta llegar a la capa final de la red, en la cual se utiliza la información relevante obtenida en las capas previas para clasificar la imagen de entrada según el objetivo deseado.

Las matrices de convolución utilizadas para obtener las características de cada imagen están constituidas por un conjunto de números denominados “pesos”, estos pesos deben ajustarse de acuerdo a la tarea a realizar por medio de un proceso de entrenamiento.

Para entrenar adecuadamente el modelo se necesita una gran cantidad de datos de entrenamiento (imágenes en este caso), la salida producto del recorrido de los datos por la red será comparada con la salida esperada y a través de un proceso de retroalimentación se proporciona el error resultado de comparar las dos imágenes, que será reducido al máximo utilizando un método de optimización.

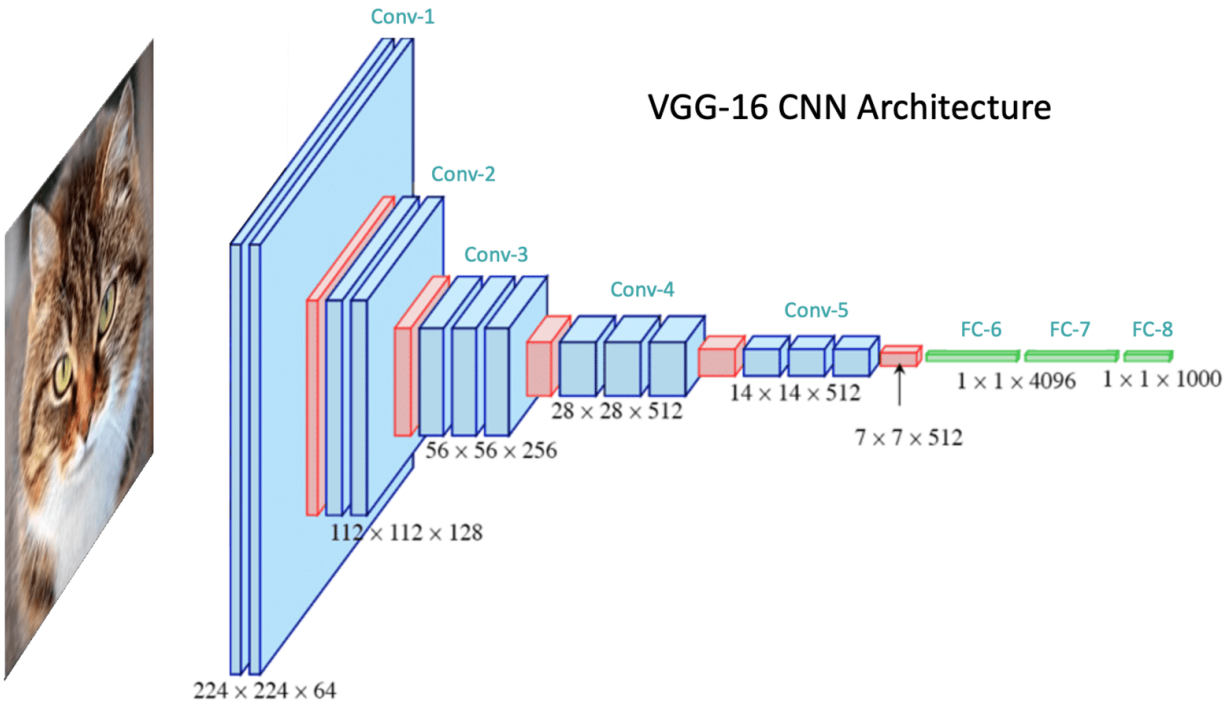


Figura 8. Estructura de una CNN. [14]

3. DESARROLLO

En este apartado serán detalladas las herramientas y tecnologías empleadas para la ejecución del trabajo, así como todos aquellos procedimientos llevados a cabo durante la ejecución del mismo.

3.1 TECNOLOGÍAS Y HERRAMIENTAS

El objetivo principal del trabajo es conseguir la correcta detección de los coches y personas que puedan estar presentes en una zona delimitada en la cual se quiere consolidar una correcta vigilancia y monitorización. Debido a esto, es necesario un sistema electrónico capaz de realizar la adquisición adecuada de las imágenes de la zona para su posterior procesamiento e identificación.

Así mismo, para que el sistema consiga ejecutar de manera óptima los pasos anteriores, es necesario un algoritmo de aprendizaje que a través de un entrenamiento previo, esté preparado para ejecutar la tarea propuesta de manera automática con un alto porcentaje de acierto. Por lo tanto, para la ejecución del trabajo fueron empleadas diversas herramientas software y hardware que permiten cumplir con lo anteriormente mencionado, este apartado tiene como finalidad cubrir todos los equipos utilizados a lo largo del desarrollo del proyecto.

3.1.1 YOLOv5

Como se comentó en los capítulos anteriores, la inteligencia artificial proporcionó un desarrollo significativo en los campos relacionados con el procesamiento y análisis de imágenes. Debido a la naturaleza del sistema

empleado (sistema de visión artificial), es conveniente considerar la utilización de un algoritmo de inteligencia artificial, más específicamente un algoritmo de *deep learning*, basado en redes neuronales convolucionales que a través de un proceso de entrenamiento sea capaz de reconocer aquellos elementos de interés en la zona monitorizada y sea capaz de llevar a cabo el correcto seguimiento de los mismos.

Debido a lo anterior, se consideró adecuado utilizar el algoritmo YOLO (*You Only Look Once*) para la detección de objetos. Así pues, la detección de objetos consiste en indicar en la imagen por medio de un recuadro (*bounding box*), el objeto perteneciente a alguna de las clases definidas previamente en el algoritmo.

YOLO es un algoritmo escrito en python, basado en redes neuronales convolucionales que se usa para la detección en tiempo real de objetos, como salida el algoritmo entrega un tensor (matriz de tres dimensiones), que contiene las coordenadas del objeto ubicado en la imagen, así como la probabilidad de que este pertenezca a una determinada clase definida con anterioridad. La gran ventaja que proporciona YOLO respecto de los demás algoritmos utilizados para este fin es su rapidez y rendimiento, puesto su arquitectura está compuesta de una única red CNN para una sola imagen, de ahí el nombre You Only Look Once. [15]

Al juntar la potente capacidad de análisis del algoritmo con su alto rendimiento, se tiene una herramienta de gran versatilidad y precisión a la hora de realizar cualquier tipo de tarea de detección.

3.1.1.1 Pasos para crear un algoritmo en YOLO:

1. Datos de entrenamiento.
2. Definición de etiquetas en el formato adecuado.
3. Definición de los conjuntos de entrenamiento, validación y prueba.
4. Definición del archivo YAML.

Datos de entrenamiento: El algoritmo de YOLO tiene una arquitectura especializada para la detección de objetos. Por lo tanto, una vez determinado el tipo de detección que se desea llevar a cabo tan solo resta entrenar el algoritmo con la suficiente información para que pueda generalizar las características fundamentales de los elementos en la imagen, y proporcione una correcta identificación de los mismos de acuerdo con los objetivos propuestos.

El proceso de entrenamiento requiere de un número significativo de imágenes en las que se encuentre presente el conjunto de objetos de los cuales se pretende realizar la detección, por lo cual, el primer paso engloba la obtención de aquellas imágenes que serán utilizadas para entrenar el algoritmo con el fin de prepararlo para trabajar de manera eficiente.

El proceso de obtención de imágenes es el paso más largo, puesto que para conseguir un entrenamiento satisfactorio que cumpla con las expectativas, será necesario un gran número de imágenes variadas de los objetos. Afortunadamente, existen bases de datos especializadas que recaban millones de imágenes de muchos elementos como pueden ser automóviles, personas, rostros, señales de tránsito, entre otros, aligerando considerablemente este paso.

Definición de etiquetas en el formato adecuado: Una vez culminado el paso anterior es necesario etiquetar las imágenes, YOLO necesita como entradas, además de las imágenes a utilizar, un conjunto de coordenadas que indican la ubicación del objeto en la imagen.

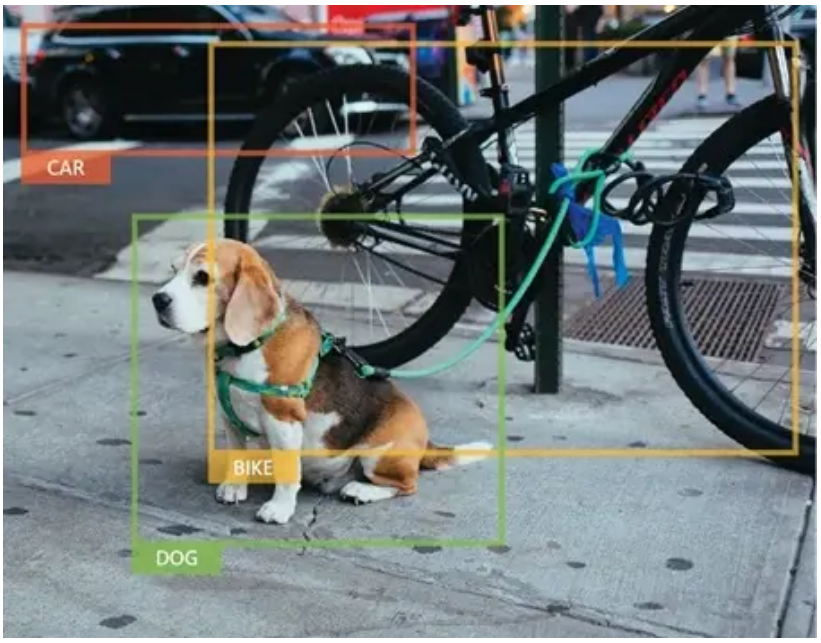


Figura 9. Ejemplo de una detección por medio de YOLO. [15]

Más concretamente, las etiquetas indican las coordenadas del vértice superior izquierdo y del inferior derecho del *bounding box*, recuadro utilizado para la ubicación del elemento de relevancia en la imagen. Adicionalmente, a las coordenadas, se debe proporcionar el número que corresponde con la clase del objeto etiquetado.

Las etiquetas deben estar en un formato adecuado para la correcta interpretación de estas por parte del algoritmo.

TABLA I				
Etiquetas de Datos de Entrenamiento				
Clase	**x**	**y**	**w**	**h**
0	0.503	0.525	0.848	0.679
1	0.498	0.456	0.995	0.746
2	0.496	0.447	0.942	0.809
1	0.508	0.563	0.855	0.839

Tabla 1. Formato adecuado de etiquetado en una imagen.

La primera columna corresponde a un número entero que indica la clase a la cual pertenece el objeto de la imagen. Las dos columnas que siguen a la clase indican la coordenada x e y del centro del *bounding box* y las últimas dos son el ancho y el alto de la *bounding box* normalizado por el ancho y alto de la imagen.

El proceso de etiquetado se puede realizar a través de programas o librerías especializadas, que utilizan el set de imágenes y generan como salida un fichero de texto en el que se encuentra la información anteriormente mencionada en el formato apropiado para YOLO.

Definición de los conjuntos de entrenamiento, validación y prueba: Por cada imagen utilizada se tendrá un archivo de texto que contenga las etiquetas de cada objeto que se pretende detectar.

A continuación, se define un árbol de directorios que agrupan el conjunto de imágenes y sus etiquetas en tres grupos, entrenamiento *(**train***), validación (***val***) y prueba (***test***).

El set de mayor tamaño es el de entrenamiento, utilizado para entrenar el algoritmo y obtener los pesos de las matrices de convolución para producir la salida esperada.

Definición del archivo YAML: En este archivo se indica la ruta que señala la localización exacta de las imágenes de entrenamiento, validación y prueba, además se definen la cantidad de clases para las cuales se va a entrenar el algoritmo y sus nombres correspondientes. Para realizar el archivo se utilizará el lenguaje de programación python, por medio del cual se proporciona un conjunto de funciones que permiten lo anterior, así mismo, la ubicación del fichero no es relevante, siempre y cuando se proporcione al algoritmo la ruta de ubicación completa de este.

```
train: /content/data/images/train/
test:  /content/data/images/test/
val:   /content/data/images/val/
nc: 2
names: ['eye', 'mouth']
```

Figura 10. Archivo YAML utilizado en una prueba de detección de ojos y boca.

Una vez terminados los pasos anteriores, se puede comenzar el proceso de entrenamiento del algoritmo.

Figura 11. Distintas arquitecturas de YOLOv5 proporcionadas por ultralytics [16].

Existen varias versiones de YOLOv5 en función de su arquitectura que van de versiones más simples pero muy rápidas, a otras de gran complejidad que requieren un coste computacional elevado. Para la elaboración del proyecto se consideró apropiada la versión YOLOv5s de ultralytics [27], siendo una versión sencilla con la cual se pueden obtener buenos resultados sin un coste computacional tan alto. El objetivo último de este trabajo es la incorporación de estas herramientas en un procesador simple, como puede ser un sistema embebido basado en Raspberry PI o similar, por lo que se requiere una herramienta de detección de objetos y personas que no introduzca una complejidad computacional elevada.

Cabe destacar que los pasos utilizados para el entrenamiento del algoritmo son exactamente los mismos independientemente de la arquitectura utilizada.

3.1.2 ORANGE PI ZERO y RASPBERRY PI 3 MODELO B

En un principio se optó por una placa Orange PI Zero como hardware especializado para la ejecución de los algoritmos elaborados, debido a que presenta una alternativa adecuada a la Raspberry PI utilizada en muchas aplicaciones de visión artificial ya que posee módulos especiales para dicho fin.

La Orange PI Zero es un ordenador monoplaca de código abierto que puede ejecutar diversos sistemas operativos como son Android 44, Ubuntu y Debian, la placa tiene un diseño compacto que cuenta con unas dimensiones de 48 x 46 mm. [18]

Otros aspectos relevantes que forman parte de las especificaciones de la placa son que tiene una DDR3 SDRAM de 256/512, siendo el modelo

utilizado el de 512 MB. el cual utiliza un AllWinner H2 SoC y se alimenta a través de un USB OTG (On The Go) que permite una mayor flexibilidad en el intercambio de información, un puerto de conexión Ethernet 100, integra una tarjeta TF y posee un puerto de expansión de hasta 26 pines.[18]

La Orange Pi Zero es capaz de realizar las mismas funciones que un ordenador, desarrollando una gran variedad de aplicaciones como son juegos, música y sonido, aplicaciones *android*, servidores, entre muchas otras. [18]

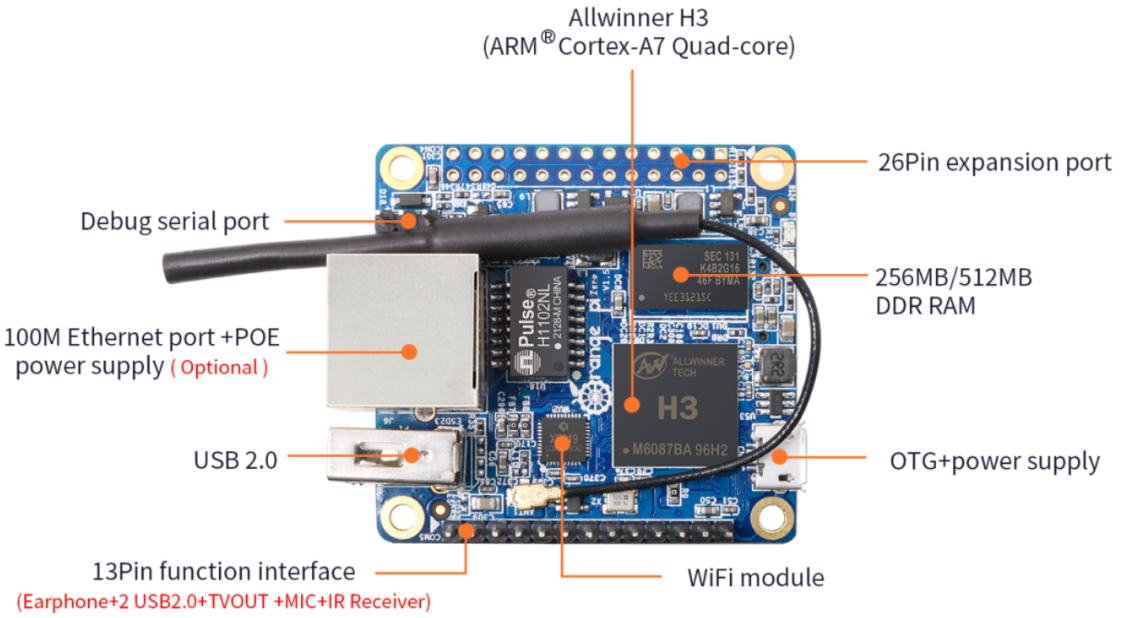


Figura 12. Vista superior de la Orange PI Zero. [18]

3.1.2.1 ESPECIFICIACIONES DE HARDWARE (ORANGE PI ZERO)

TABLA II	
CPU	H2 Quad-core Cortex-A7 H.265/HEVC 1080P
GPU	Mali400MP2 GPU @600MHz
Soporte para OpenGL ES 2.0	
Memoria (SDRAM)	512 MB DDR3
Conexión en línea	10/100 MB Ethernet RJ45
WIFI	XR819, IEEE 802.11 b/g/n
Entrada de audio	MIC
Salida de video	AV OUT
Alimentación	USB OTG
Puertos USB 2.0	USB 2.0 HOST, USB 2.0 OTG
Periféricos de bajo nivel	26 Pins Header, compatibles con Raspberry PI B+
	13 Pins Header (2x USB, IR pink Audio (MIC, AV)
LED	led de alimentación, led de estado
Sistemas operativos soportados	Android, Lubuntu, Debian
Tamaño	46 mm x 48 mm
Peso	26g
Alimentación	5V, 3a
Temperatura de trabajo	-10 a 65 ℃

Tabla 2. Especificaciones de Hardware de Orange PI Zero. [18]

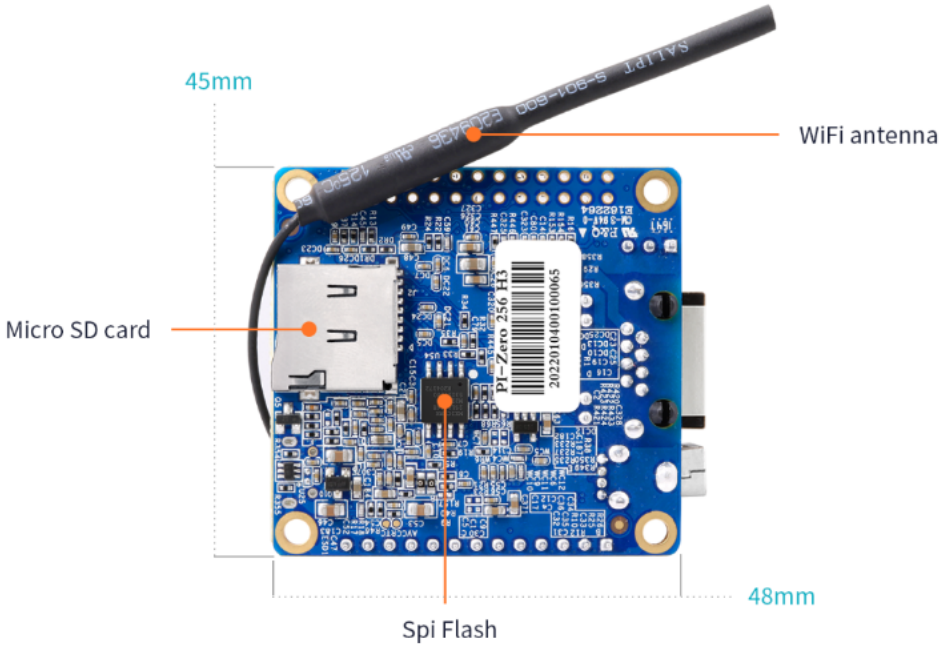


Figura 13. Vista inferior de la Orange PI Zero. [18]

Sin embargo, durante el desarrollo del proyecto se identificó un inconveniente a la hora de utilizar dicho hardware para la tarea propuesta. La Orange PI Zero presenta una serie de pines que son completamente compatibles con la Raspberry PI, no obstante, no posee un puerto de conexión de cámara para la cámara módulo 2 de la Raspberry PI.

Como alternativa existe la posibilidad de configurar la placa para utilizar una webcam a través del puerto USB. Sin embargo, el método de operación de esta difiere significativamente del empleado para la cámara habitual de la RaspBerry Pi, con lo cual constituye una complicación añadida que se aleja del objetivo inicial del trabajo.

Por lo tanto, para conseguir evaluar los resultados de dicho algoritmo se optó por utilizar una placa Raspberry PI 3 modelo B proporcionada por la Universidad de La Laguna, utilizada previamente en un Trabajo de Fin de Grado para la obtención de un video una vez se cumplan una serie de condiciones estipuladas.

Importante destacar que la placa proporcionada presenta una serie de defectos que pueden ser relevantes en ciertas aplicaciones. Entre los fallos presentes tenemos que el módulo WIFI de la placa es defectuoso siendo incapaz de conectarse a través de red WIFI, con lo cual no se puede llevar a cabo la comunicación de información por medio de esta vía.

Adicionalmente, la placa presenta fallos a la hora de conseguir una comunicación serie a través del protocolo I2C. No obstante, debido a que para el proyecto realizado estos problemas no afectan la obtención de los resultados esperados, es viable emplear la placa para el prototipo del sistema.

La Raspberry PI 3 modelo B constituye la tercera generación de placas de Raspberry, de la misma forma que sus predecesoras, es un potente ordenador monoplaca de reducido tamaño que tiene un gran número de utilidades en diversas aplicaciones. De la misma forma, presenta una mayor potencia que las generaciones anteriores, sustituyendo a las placas originales del modelo B + y a las Raspberry PI 2 modelo B, esta nueva generación de placas trae un procesador más potente y hasta diez veces más rápido, además de añadir conexión inalámbrica LAN y Bluetooth, siendo la opción ideal para diseños que requieran elevadas prestaciones. [19]

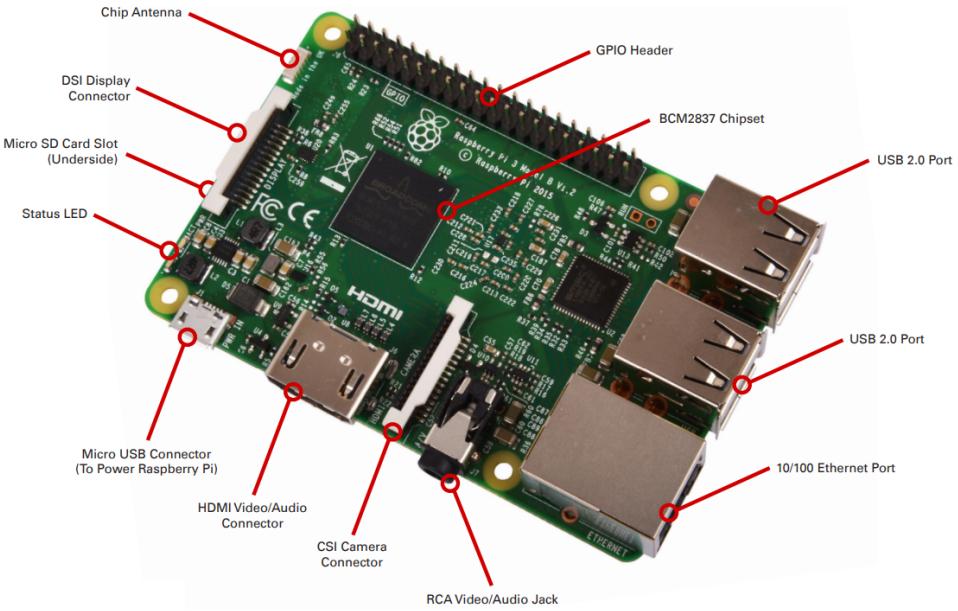


Figura 14. Vista superior de la Raspberry PI 3 Modelo B. [19]

3.1.2.2 ESPECIFICACIONES DE HARDWARE (RASPBERRY PI 3 B)

TABLA III	
Dimensiones	85 x 56 x 17 mm
SoC	BCM2837
Núcleo del Procesador	ARM Cortex-A53
Poder de Procesado	1.2 GHz
Memoria	1GB LPDDR2
Puertos USB	4x USB 2.0
Puerto Ethernet	1x 10/100 Ethernet
GPIO (Pines de propósito general)	40

Tabla 3. Especificaciones de Hardware de Raspberry PI 3 Modelo B. [19]

A pesar de que la placa Orange PI Zero no será utilizada para la ejecución de este trabajo, es una gran sustituta de la Raspberry PI y por tanto, será añadida en el apartado correspondiente a las propuestas de mejora como alternativa para diseñar el sistema utilizando una configuración de webcam por USB para la toma de imágenes del entorno.

3.1.3 LENGUAJES DE PROGRAMACIÓN

El lenguaje de programación utilizado para la síntesis de los códigos de entrenamiento del algoritmo y de la captura de video es el Python. El cual es un lenguaje de gran potencia que destaca frente a muchos otros por su flexibilidad y la facilidad que presenta a la hora de su lectura, haciendo de este uno de los lenguajes más fáciles para aprender así como uno de los más utilizados del momento. [20]

Como ventajas frente a otros lenguajes de programación puede mencionarse que presenta un tipado dinámico, un efectivo sistema de programación orientado a objetos, que es de naturaleza interpretativa. Además de tener a disposición un número importante de librerías de libre distribución que por su variada naturaleza pueden utilizarse para un sin fin de aplicaciones como procesamiento de imágenes, inteligencia artificial, manejo de ficheros, y operaciones matemáticas, entre muchas otras.

A continuación se indicarán aquellas librerías utilizadas en los códigos del proyecto, así como una breve descripción de las mismas.

- **librería numpy:** Es una librería especializada en el cálculo numérico y análisis de datos de gran volumen, proporciona una función especial que

fue utilizada durante el proceso de grabación y visualización del video que contiene la detección llevada a cabo.

- **librería matplotlib.pyplot:** Es una librería empleada para la representación gráfica de información contenida en arrays o listas, fue utilizada para la visualización de imágenes durante el entrenamiento y prueba del algoritmo.

- **librería os:** Módulo especializado para la manipulación de archivos, debido a que para entrenar el algoritmo se necesita un gran número de imágenes ordenadas en grupos específicos, esta librería aporta una serie de funciones apropiadas para trabajar con directorios, consiguiendo así ordenar la información de la forma buscada.

- **librería cv2:** Open cv es una librería de código abierto de visión artificial utilizada para el análisis y tratamiento de imágenes, tuvo un protagonismo importante en el código realizado puesto la misma fue empleada para la lectura, visualización y tratamiento de las imágenes de estudio.

- **librería torch:** Es una librería de código abierto de aprendizaje automático utilizada para una gran variedad de proyectos relacionados con inteligencia artificial, permite cargar el modelo del algoritmo entrenado previamente con sus respectivos pesos para poder ejecutar una detección directa tanto para imágenes como videos.

3.2 OBTENCIÓN DE DATOS DE ENTRENAMIENTO

Antes de desarrollar el código de entrenamiento del algoritmo es necesario recabar suficiente información para poder entrenarlo de manera

eficiente, para el caso trabajado, se necesita un número importante de imágenes de los objetos a los cuales se les desea llevar a cabo la detección (personas y vehículos). Para conseguir esto fueron seleccionadas un total de trescientas imágenes que contienen vehículos y personas, estas imágenes fueron recabadas de distintas fuentes como pueden ser bases de datos públicas elaboradas por diversas personas. [17] [26]

Una vez obtenidas las imágenes se procede a realizar el proceso de etiquetado de las mismas, por medio del cual se indican los objetos que se quieren detectar así como el tipo de clase al que pertenecen.

Para realizar este paso se utilizó una página web denominada Make Sense, por la cual se consigue llevar a cabo el etiquetado indicando manualmente con el ratón los objetos deseados en la imagen del conjunto de datos proporcionado a la página.

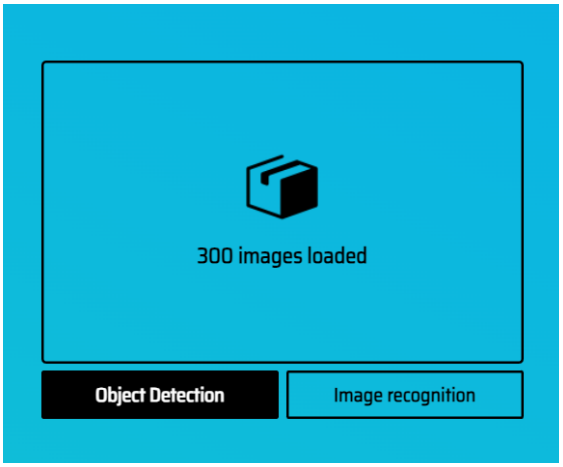


Figura 15. Proceso de etiquetado de las imágenes para la detección de objetos.

Una vez hecho esto, se debe indicar el objetivo con el cual se pretende entrenar al algoritmo y seguidamente se procede a definir el conjunto de clases para las cuales se realizará la detección.

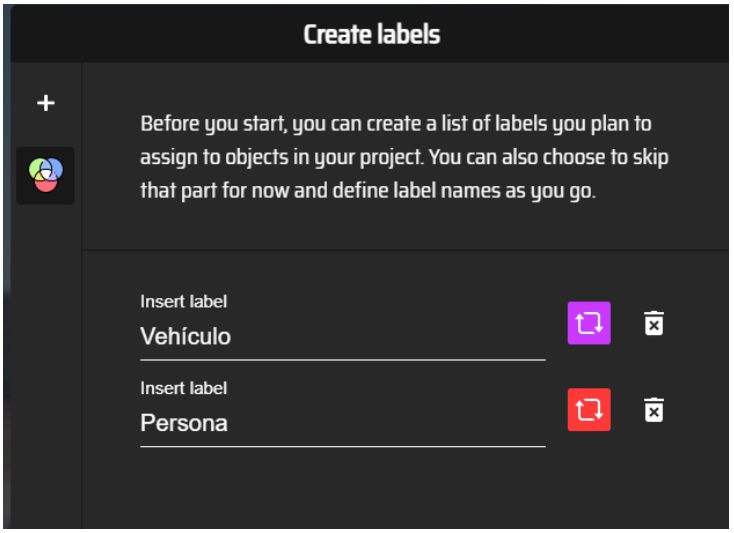


Figura 16. Creación de clases a las cuales pertenecen los objetos a detectar.

El número de clases a detectar puede ser de gran variedad, para efectos del trabajo a ejecutar, se consideró apropiado conseguir un resultado satisfactorio con un número reducido de clases para facilitar el proceso de etiquetado y prueba. Sin embargo, en el apartado de propuestas de mejora se destaca la posibilidad de incluir diversas clases más específicas.

Una vez hecho esto, se procede a llevar a cabo el proceso de etiquetado con cada una de las imágenes seleccionadas, indicando en cada una la clase a la que pertenece el objeto en cuestión.

Figura 17. Etiquetado de objetos a través de la página web Make Sense.

Así pues, la misma página web proporciona las etiquetas de cada imagen en archivos de texto con el formato adecuado para que puedan ser entendidos por el algoritmo YOLO. Por lo cual, una vez culminado se deben exportar las etiquetas indicando dicho formato.

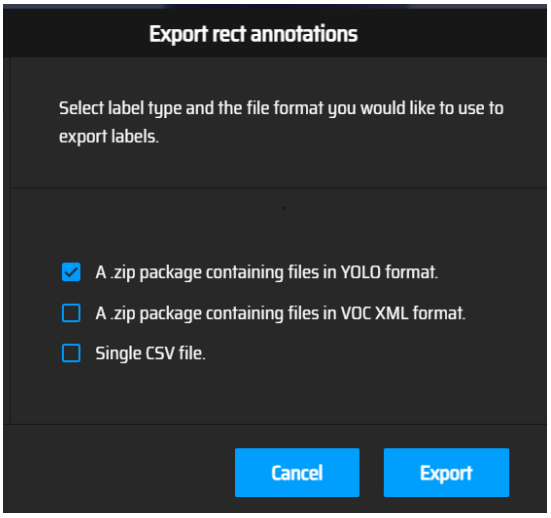


Figura 18. Exportación de etiquetas de la página web Make Sense.

El proceso de obtención de imágenes y etiquetado que constituyen la etapa crítica de creación del *dataset*, es uno de los pasos más importantes a la hora de trabajar con cualquier algoritmo de aprendizaje automático. La calidad de los resultados dependerá de la cantidad de imágenes utilizadas así como también de la variedad de las mismas, permitiendo que el algoritmo sea más preciso a la hora de llevar a cabo la detección.

Debido a que la detección a realizar forma parte de un sistema de vigilancia que puede estar ubicado en vías públicas transitadas o regiones bien delimitadas, se utilizaron imágenes de diversa naturaleza para garantizar buenos resultados.

Este paso es el más difícil y laborioso a la hora de entrenar los algoritmos, por lo cual en la mayoría de casos se suele contratar anotadores que recaben la información necesaria y elaboren las etiquetas para entrenar adecuadamente al algoritmo.

En las figuras 19 a 24 podemos observar algunas imágenes etiquetadas utilizadas para el entrenamiento del algoritmo:

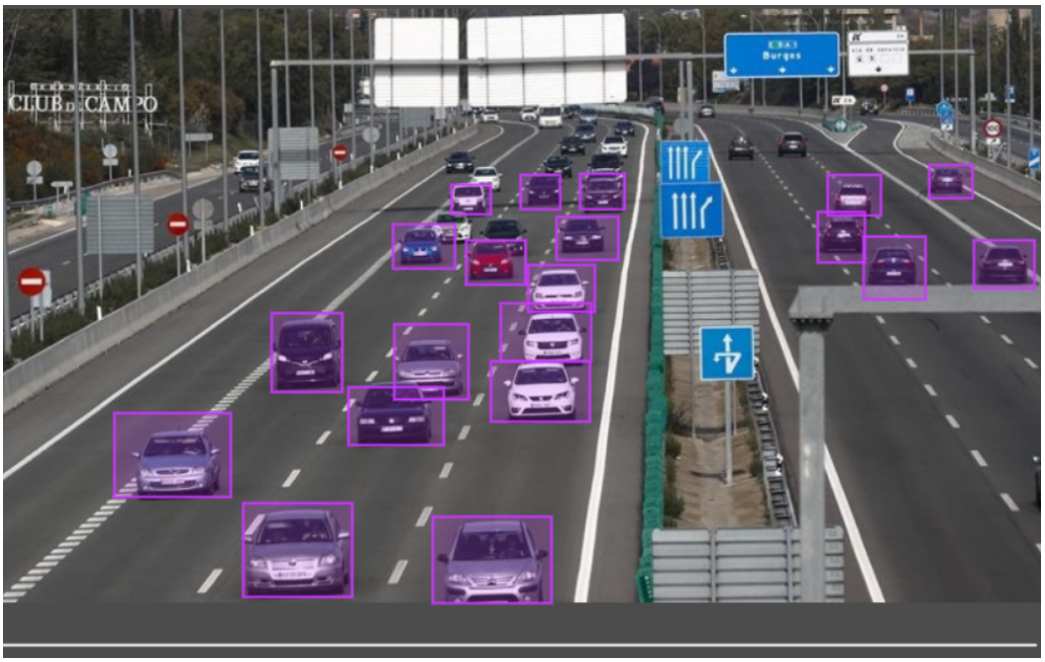


Figura 19. Imagen etiquetada utilizada para entrenar el algoritmo.

Figura 20. Imagen etiquetada utilizada para entrenar el algoritmo.

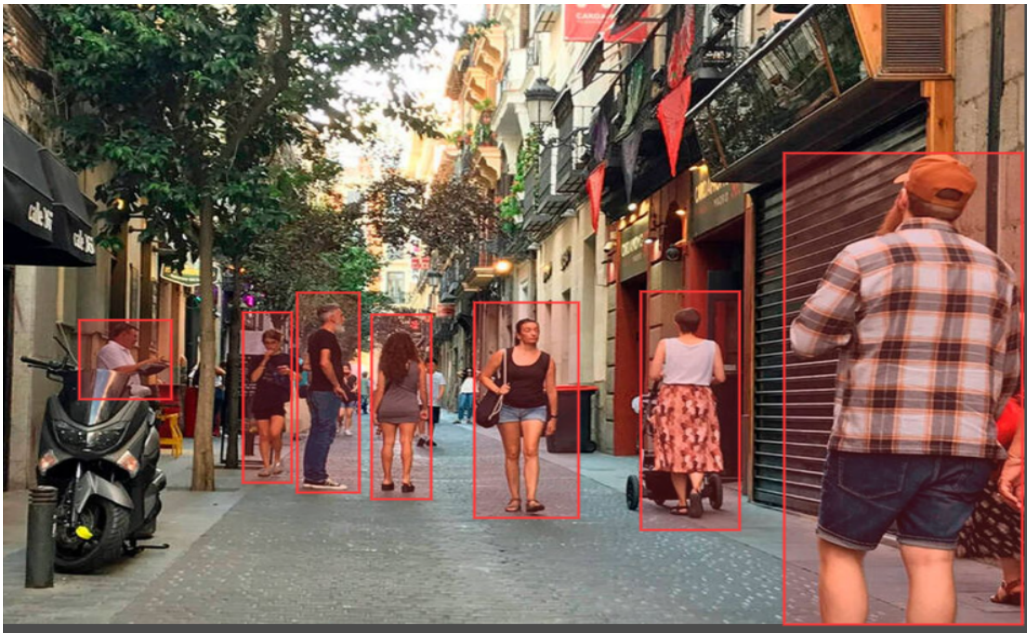


Figura 21. Imagen etiquetada utilizada para entrenar el algoritmo.

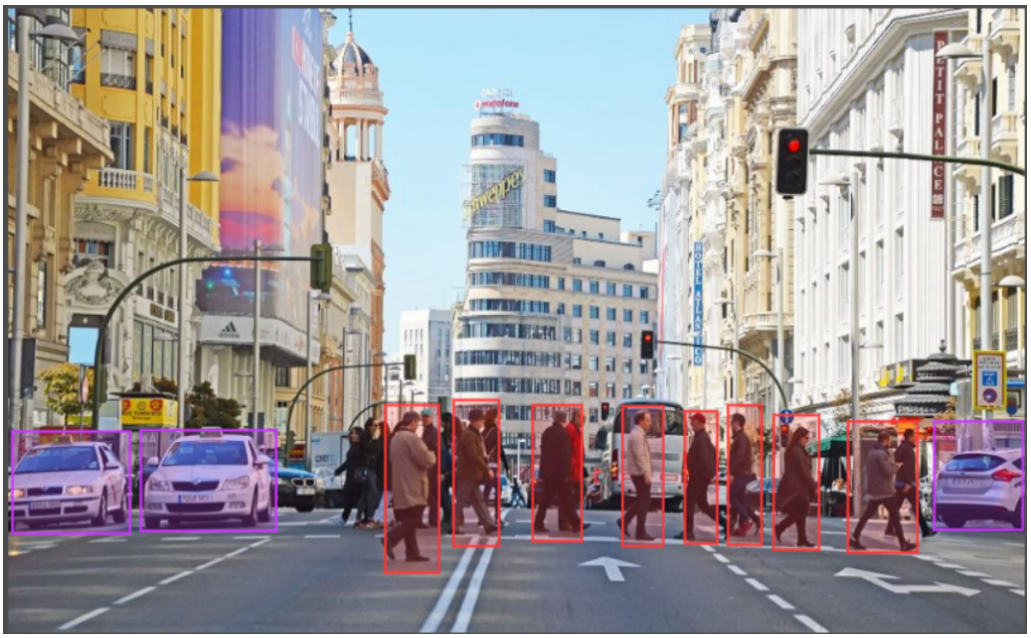


Figura 22. Imagen etiquetada utilizada para entrenar el algoritmo.

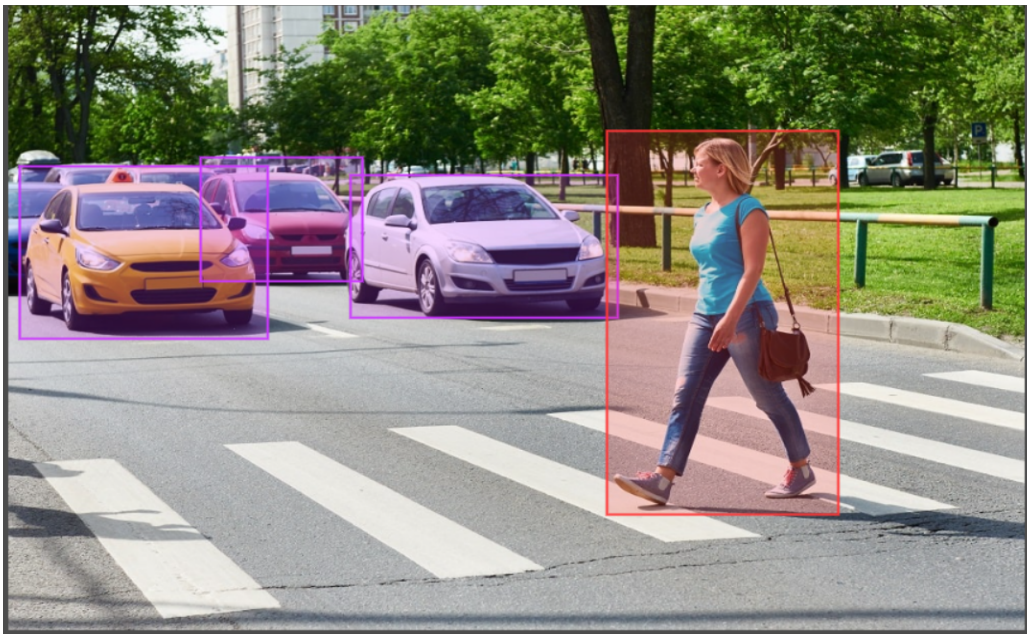


Figura 23. Imagen etiquetada utilizada para entrenar el algoritmo.

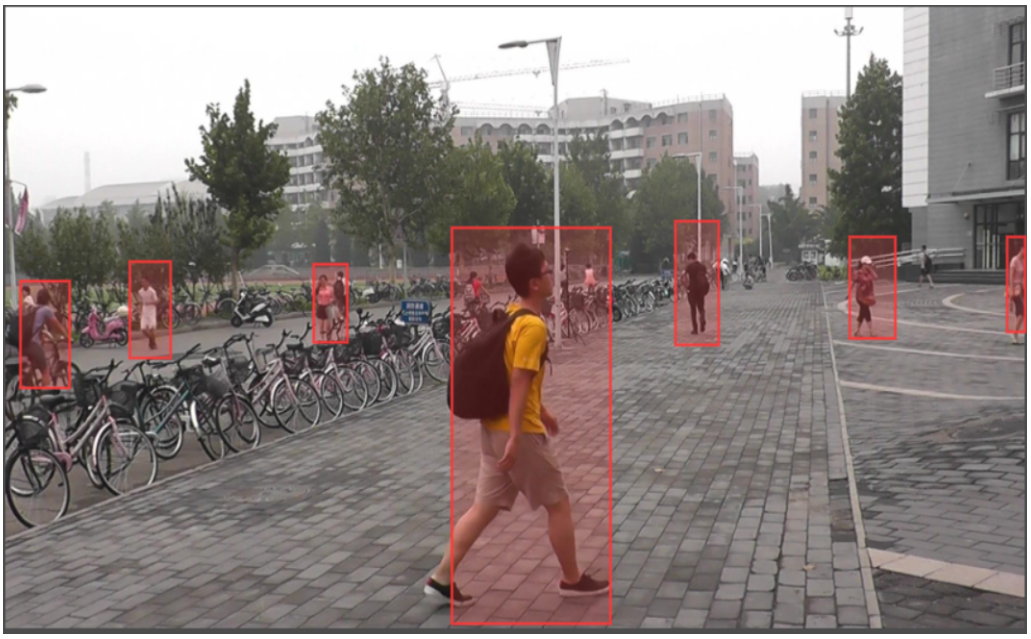


Figura 24. Imagen etiquetada utilizada para entrenar el algoritmo.

El conjunto de datos para entrenar el algoritmo quedará organizado de la siguiente manera:

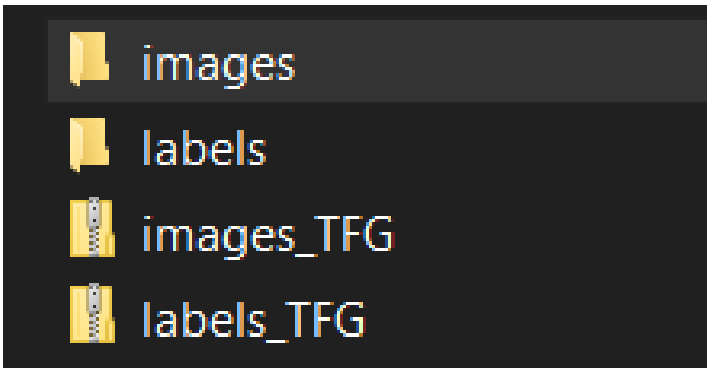


Figura 25. Imágenes y etiquetas para entrenar el algoritmo.

La carpeta "*images*" contiene las trescientas imágenes seleccionadas para el entrenamiento, de la misma forma, "*labels*" contiene los archivos .txt de las etiquetas de cada objeto creadas en el paso anterior.

De lo anterior fueron generados dos archivos .zip para facilitar el envío del dataset.

3.3 CÓDIGO DE ENTRENAMIENTO

A continuación se procederá a detallar el código creado para entrenar al algoritmo YOLO.

- **Importación de librerías:** En la primera parte del código se importan las librerías previamente mencionadas que serán utilizadas a lo largo de la elaboración del mismo.

```
import torch
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
```

Figura 26. Importación de librerías utilizadas.

- **Carga de ficheros de imágenes y etiquetas:** Los ficheros comprimidos que contienen todos los datos necesarios para el entrenamiento, prueba y validación del modelo de YOLO deben cargarse al código para su posterior manipulación.

```
#CARGAMOS ARCHIVOS .ZIP DE IMAGENES Y ETIQUETAS

path = '/content/gdrive/MyDrive/YOLO_TFG/'

path_img = path + 'images_TFG.zip'
path_lab = path + 'labels_TFG.zip'

#SE COPIAN LOS ARCHIVOS PARA TRABAJAR CON ELLOS EN EL CODIGO

!cp {path_img} {'/content/'}
!cp {path_lab} {'/content/'}
```

Figura 27. Carga de ficheros de imágenes y etiquetas.

- **Extracción de archivos comprimidos:** Una vez cargados los archivos se debe proceder a descomprimir los mismos para poder acceder al contenido del dataset elaborado.

```
#DESCOMPRIMIMOS LOS ARCHIVOS

!unzip -qq images.zip
!unzip -qq labels.zip
```

Figura 28. Extracción de archivos comprimidos.

- **Movimiento de los archivos para su posterior manipulación:** Primero fueron creados dos directorios, uno para las imágenes empleadas y otro para las etiquetas respectivamente.

```
#CREARMOS DIRECTORIOS PARA ALMACENAR LAS ETIQUETAS E IMAGENES

!mkdir /content/labels_TFG
!mkdir /content/images_TFG
```

Figura 29. Directorios para almacenar imágenes y etiquetas.

Este paso permite ubicar cada parte del *dataset* en un directorio específico de tal forma que el acceso a la información se haga más intuitivo. Hecho esto se definió una función que produzca el movimiento de los archivos al destino deseado, esta función tiene tres argumentos de entrada, el primer y segundo argumento corresponden tanto a la dirección de origen como a la dirección de destino, el último argumento es la terminación del archivo, es decir, un archivo con terminación .txt corresponde con una etiqueta, uno que termine en .jpg será una de las imágenes utilizadas, esto permite discriminar entre archivos que no formen parte de las clases del dataset.

```
#DEFINIMOS UNA FUNCIÓN PARA MOVER TODOS LOS ARCHIVOS RELEVANTES A LOS DIRECTORIOS CREADOS

def move_fich(path, destino, term):

 content = os.listdir(path)

 for k in content:

  if os.path.isfile(os.path.join(path, k)) and k.endswith(term):

    !mv {path + k} {destino}
```

Figura 30. Definición de la función de movimiento de los archivos.

La función definida utiliza funciones de la librería **os**. La primera línea define una lista que contiene todos los archivos, tanto ficheros como directorios, comprendidos en la dirección de origen (*path*). Seguidamente se define un bucle ***for*** que recorre dicha lista y por medio de una acción condicional determina si el elemento (k) de la lista de archivos es un fichero, si además de cumplirse esa condición, el fichero posee la terminación indicada (.txt o .jpg), se produce el movimiento del mismo desde la dirección de origen a la dirección de destino.

Una vez invocada la función, puede comprobarse que se ejecutó adecuadamente definiendo una lista de los archivos contenidos en los directorios de destino y comprobando que la longitud de la lista es igual al total de imágenes empleado (300).

```
#MOVEMOS LOS FICHEROS .JPG (IMAGENES)

move_fich('/content/images/', '/content/images_TFG/', '.jpg')

#comprobamos que se realizó correctamente

pifile = os.listdir('/content/images_TFG/')

print(len(pifile))
```

```
300
```

Figura 31. Comprobación del movimiento de archivos .

- **Definición del árbol de directorios para entrenamiento prueba y validación:**

```
#CREACION DEL ARBOL DE DIRECTORIOS NECESARIOS PARA EL ENTRENAMIENTO

!mkdir /content/data
!mkdir /content/data/images
!mkdir /content/data/images/train
!mkdir /content/data/images/test
!mkdir /content/data/images/val
!mkdir /content/data/labels
!mkdir /content/data/labels/train
!mkdir /content/data/labels/test
!mkdir /content/data/labels/val
```

Figura 32. Creación del árbol de directorios para el entrenamiento.

- **Distribución del dataset entre los directorios de entrenamiento, prueba y validación:** Para el siguiente paso se debe distribuir el número de imágenes y etiquetas del *dataset* en

tres grupos, se optó por repartir el *dataset* entre los directorios de la siguiente manera, setenta por ciento del dataset corresponde a la cantidad utilizada para el entrenamiento, veinte por ciento para la prueba del algoritmo y el diez por ciento restante para validación del mismo.

Una vez determinado el porcentaje de distribución, se definió una función para repartir los archivos en los porcentajes que fueron discutidos anteriormente.

```
#CREACION DE UNA FUNCIÓN PARA DISTRIBUIR EL CONJUNTO DE DATOS EN TRAIN/TEST/VAL


def fichero_ordenar(content, path, path_destino1, path_destino2, path_destino3, term):

  content = sorted(content)

  p_train = 0.7
  p_val = 0.1
  p_test = 0.2

  n_train = round(len(content) * p_train)
  n_val = round(len(content) * p_val)
  n_test = round(len(content) * p_test)

  print('cantidad del set de entrenamiento: ', n_train, '\n')
  print('cantidad del set de validación: ', n_val, '\n')
  print('cantidad del set de prueba: ', n_test, '\n')

  for i in range(len(content)):

    if i <= n_train and content[i].endswith(term):

      !mv {path + content[i]} {path_destino1}

    elif i > n_train and i <= n_train + n_val and content[i].endswith(term):

      !mv {path + content[i]} {path_destino2}

    elif i > n_train + n_val and content[i].endswith(term):

      !mv {path + content[i]} {path_destino3}
```

Figura 33. Función encargada de la distribución del dataset.

La función tiene seis argumentos de entrada, el primer argumento es una lista de los archivos contenidos en la dirección de origen,

seguidamente se tiene cuatro direcciones la primera corresponde a la dirección de origen en la cual están contenidos todas las imágenes o etiquetas, las otras tres direcciones son de los directorios de ***train***, ***test*** y ***val***, el último argumento es la terminación del archivo para manipular un solo tipo de archivo a la vez.

En la primera parte de la función se ordena la lista de archivos por medio de la función ***sorted()***, seguidamente se define la proporción esperada de archivos para los tres directorios de destino y posteriormente se calcula la cantidad correspondiente al total de archivos que serán trasladados a cada uno de los directorios.

En la parte final de la función se define un bucle ***for*** que recorre la lista de archivos y por medio de una serie de condiciones impuestas se asegura que sean transferidos adecuadamente a cada uno de los directorios de destino manteniendo las proporciones calculadas anteriormente.

Se procede a ejecutar la función de acuerdo a lo indicado en párrafos anteriores por medio de las siguientes líneas de código:

```
#ORDENAMOS LOS DATOS DE IMAGENES Y ETIQUETAS UTILIZANDO LA FUNCION PREVIAMENTE DEFINIDA

fich_img = os.listdir('/content/images_TFG/')

fich_lab = os.listdir('/content/labels_TFG/')

#Para imagenes

fichero_ordenar(fich_img, '/content/images_TFG/', '/content/data/images/train/', '/content/data/images/val/', '/content/data/images/test/', '.jpg')

#Para etiquetas

fichero_ordenar(fich_lab, '/content/labels_TFG/', '/content/data/labels/train/', '/content/data/labels/val/', '/content/data/labels/test/', '.txt')
```

Figura 34. Distribución del dataset en los directorios correspondientes.

Terminado el proceso de distribución, puede comprobarse que se ejecutó correctamente, por medio de la comparación del número de archivos contenidos en los directorios definidos. La cantidad de archivos debe coincidir tanto para las imágenes como para las etiquetas.

```
#COMPROBAMOS QUE EL PROCESO SE HIZO CORRECTAMENTE PARA IMAGENES

destino1 = '/content/data/images/train/'
destino2 = '/content/data/images/val/'
destino3 = '/content/data/images/test/'

content_train = os.listdir(destino1)
content_val = os.listdir(destino2)
content_test = os.listdir(destino3)

print(len(content_train))
print(len(content_val))
print(len(content_test))
```

```
211
30
59
```

```
#COMPROBAMOS TAMBIE PARA ETIQUETAS

destino1 = '/content/data/labels/train/'
destino2 = '/content/data/labels/val/'
destino3 = '/content/data/labels/test/'

fichero1 = os.listdir(destino1)
fichero2 = os.listdir(destino2)
fichero3 = os.listdir(destino3)


print(len(fichero1), 'train')
print(len(fichero2), 'val')
print(len(fichero3), 'test')
```

```
211 train
30 val
59 test
```

Figura 35. Comprobación de la correcta ejecución de la función de la figura 33.

- **Definición del archivo .YAML**: La siguiente parte del código cuenta con la creación del archivo .YAML, utilizado para indicar las clases que van a ser detectadas por medio del algoritmo.

Primero se definen las direcciones de cada uno de los directorios que contienen las imágenes de ***train***, ***test*** y ***val***, así como también el número de clases a detectar y el nombre de cada una que estará contenido en un lista de strings.

Al finalizar se escribe un fichero en el que se incluye toda la información mencionada anteriormente, en la figura 36 puede observarse el código empleado en este paso.

```
#CREAMOS ARCHIVO .YAML INDICANDO LAS CLASES A DETECTAR (vehículo y persona)
path_train = '/content/data/images/train/'
path_test  = '/content/data/images/test/'
path_val   = '/content/data/images/val/'

nro_clases = 2

nombre = ['vehiculo', 'persona']

f = open('/content/data/data.yaml', 'w+')
f.write('train: ' + path_train + '\n')
f.write('test:  ' + path_test  + '\n')
f.write('val:   ' + path_val   + '\n')

f.write('nc: ' + str(nro_clases) + '\n')
f.write('names: ' + str(nombre)  + '\n')

f.close()
```

Figura 36. Creación del archivo .YAML.

Se puede comprobar que el archivo se creó correctamente como indica la figura 37:

```
#COMPROBAMOS LA CREACION DEL FICHERO

!cat /content/data/data.yaml

train: /content/data/images/train/
test:  /content/data/images/test/
val:   /content/data/images/val/
nc: 2
names: ['vehiculo', 'persona']
```

Figura 37. Comprobación del archivo .YAML.

- **Instalación del modelo de YOLOv5s proporcionado por ultralytics:** Para llevar a cabo la instalación del modelo es necesario ejecutar un conjunto de sentencias indicadas en la página de ultralytics. [27]

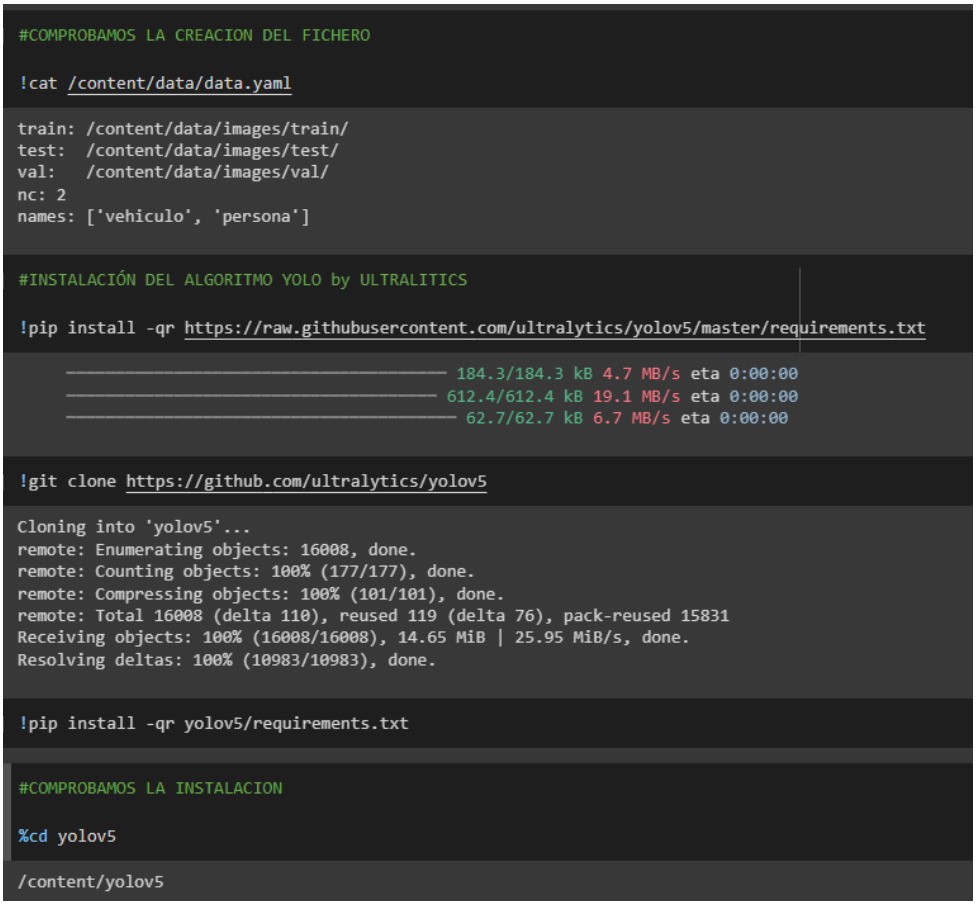


Figura 38. Instalación y comprobación del modelo de YOLO.

La instalación culmina a los pocos segundos y ya se encuentra preparado para comenzar el entrenamiento del algoritmo de detección.

- **Definiciones previas al entrenamiento:** Antes de comenzar el proceso de entrenamiento se definen una serie de variables que serán proporcionadas al algoritmo.

```
#DEFINICIONES PARA EL ENTRENAMIENTO

YOLO_MODEL = 'yolov5s' #versión del modelo a emplear (small)
EPOCHS = 200
CFG = YOLO_MODEL + '.yaml'
TRAIN_WEIGHTS = YOLO_MODEL + '.pt' #pesos utilizados para el entrenamiento
BATCH = 8
IMG_SIZE = 256
YAMLFILE = '/content/data/data.yaml'
```

Figura 39. Definiciones previas al entrenamiento del modelo de YOLO.

Es necesario definir el tipo de modelo de YOLOv5 que será empleado, para el caso estudiado se utilizará el modelo s (*small*).

Seguidamente, el número de épocas con las que será entrenado el algoritmo, un número elevado de épocas implica mejores resultados, sin embargo, la duración se el proceso de entrenamiento se extiende de manera significativa. Se optó por utilizar doscientas épocas para no hacer el proceso de entrenamiento tan largo, de tal forma que pueda comprobarse si el *dataset* utilizado resulta apropiado para la tarea que se desea llevar a cabo.

Los *batch* son las subdivisiones que se hacen al conjunto de datos para aligerar el proceso de entrenamiento.

Adicionalmente, debe indicarse la ubicación del archivo .YAML definido anteriormente, y los pesos de entrenamiento que serán utilizados para generar los pesos óptimos. Generalmente se

emplean los pesos por defecto que vienen con la instalación del algoritmo.

- **Entrenamiento del modelo de YOLOv5s:** Para entrenar el algoritmo se ejecuta la siguiente sentencia proporcionando las variables definidas en el paso anterior, si la información administrada es correcta comenzará el proceso de entrenamiento hasta culminar con el número de épocas.

```
#ENTRENAMOS EL ALGORITMO PARA DETECCION

!python train.py --img {IMG_SIZE} --batch {BATCH} --epochs {EPOCHS} --data {YAMLFILE} --cfg {CFG} --weights {TRAIN_WEIGHTS} --nosave --cache
```

Figura 40. Entrenamiento del algoritmo YOLO.

El entrenamiento finalizó en aproximadamente dos horas y quince minutos. Como resultado de este proceso, se tienen aquellos pesos que proporcionan los mejores resultados de acuerdo al tipo de tarea a ejecutar.

Así pues, es de suma importancia guardar los pesos calculados, para que puedan ser utilizados cada vez que se desee emplear el algoritmo para la detección de las clases definidas en el trabajo.

Una vez finalizado el entrenamiento el algoritmo y guardados los pesos correspondientes, este se encontrará listo para ser sometido a proceso de prueba para evaluar el desempeño de la detección realizada. Las estadísticas de testeo y validación serán comentadas en el apartado correspondiente a discusión y resultados.

3.4 INVOCACIÓN DEL ALGORITMO

Una vez se tenga a disposición el archivo (.pt) que contiene los pesos entrenados del algoritmo para identificar las clases ya indicadas, es posible llevar a cabo la detección de varias formas.

La primera forma es por medio de la invocación del *script* perteneciente al repositorio de ultralytics, ***detect.py***. El cual recibe como argumentos los pesos a emplear y la dirección en la cual se encuentra la imagen sobre la cual realizará la detección en cuestión.

```
!python detect.py --weights {PESOS_TEST} --img {IMG_SIZE} --conf 0.4 --source {'path'} --save-txt
```

Figura 41. Detección sobre una imagen determinada.

Sin embargo, si bien la opción anteriormente descrita proporciona buenos resultados, existe otra forma de invocar el modelo entrenado utilizando la librería de aprendizaje automático ***torch***. A través de la cual por medio de una serie de funciones se puede descargar y utilizar el modelo personalizado para ejecutar sobre los archivos de interés.

```
#CARGAMOS EL MODELO A TRAVÉS DE PYTORCH HUB

model = torch.hub.load('ultralytics/yolov5', 'custom', path = '/content/yolov5/runs/train/exp/weights/last.pt')

img = cv2.imread('/content/gdrive/MyDrive/YOLO_TFG/2.jpg')

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

detecta = model(img)

detecta.show()
```

Figura 42. Carga del modelo de YOLO a través de PYTORCH.

La primera sentencia utilizada invoca el modelo de YOLO, con lo cual se debe proporcionar como parámetros el tipo de modelo de YOLO utilizado, siendo el caso particular el modelo de yolov5 creado por ultralytics. Así como también los pesos que utilizará el algoritmo, que en caso de proceder de un entrenamiento personalizado, debe indicarse su procedencia como "*custom*" y proporcionar la dirección de ubicación del archivo de estos.

De la misma forma, las siguientes líneas cargan una imagen de prueba y cambian su espacio de color de BGR a un RGB para garantizar la correcta visualización de la imagen.

Finalmente, la detección se lleva a cabo ejecutando el modelo sobre la imagen de prueba como puede observarse en la figura 41. Seguidamente, puede visualizarse los resultados por medio del método ***.show()*** sobre el objeto que contiene la imagen sobre la que se realizó la detección, o bien obtener las coordenadas de las *bounding box* que ubican el objeto detectado sobre la imagen.

3.5 GRABACIÓN Y DETECCIÓN DE VIDEO

A continuación, será detallado el código utilizado para la grabación de videos en el cual de forma adicional se ejecuta la detección de los objetos por medio del algoritmo entrenado en los apartados previos.

- **Importación de librerías:** Las primeras líneas de código cargan las librerías a utilizar, dichas librerías son las mismas utilizadas en el código de entrenamiento.

- **Cargar el modelo YOLO:** Se carga el modelo YOLO al código por medio de la librería torch ejecutando la sentencia mostrada en la figura 43.

```
model = torch.hub.load('ultralytics/yolov5', 'custom', path='C:/Users/HP/Desktop/PESOS_TFG.pt')
```

Figura 43. Carga del modelo personalizado de YOLO.

- **Definiciones previas a la grabación:** Para poder grabar un video por medio de la librería Opencv, es necesario definir un conjunto de sentencias agrupadas en la figura 44.

```
cap = cv2.VideoCapture('C:/Users/HP/Desktop/video3_car.mp4')

ret, frame = cap.read()

h,w,c = frame.shape

forucc = cv2.VideoWriter_fourcc(*'mp4v')

out = cv2.VideoWriter('output3.mp4', forucc, 30, (w,h))
```

Figura 44. Definiciones previas a la grabación.

La primera sentencia guarda el archivo de video en una variable determinada, a efectos del proyecto se utilizaron videos obtenidos de internet. Con lo cual como argumento a la función ***cv2.VideoCapture()***, se proporciona la ruta de ubicación del archivo.

Seguidamente se lee un *frame* (segmento de video) por medio del método ***read()***, esto permite obtener las dimensiones (alto y

ancho del video) que definen la resolución de este y que serán utilizadas en la grabación final.

Las últimas 2 sentencias crean la variable en la que será grabado el video final, para ello primero se indica el formato que tendrá el video por medio de la función ***cv2.VideoWrite_fourcc()***. Finalmente se crea la variable en la cual se pasa como argumento el conjunto de especificaciones que tendrá el video resultado, nombre del archivo, formato, *fps* y resolución.

- **Bucle de grabación:** Para finalizar, se requiere de una instrucción tipo bucle que permita la lectura de los sucesivos *frames* del video.

```
while True:

    ret, frame = cap.read()

    if not ret:
        print('Error en la apertura del video o camara')
        break


    result = model(frame)
    out.write(np.squeeze(result.render()))

    t = cv2.waitKey(27)
    if t == 27:
        break

print('end of process')
cap.release()
cv2.destroyAllWindows()
```

Figura 45. Bucle de grabación del video.

La instrucción ***while*** permite la ejecución del set de instrucciones internas de tal forma que se consiga dicho objetivo, así mismo, la

primera instrucción del bucle ***while*** *lee* el frame correspondiente por medio del método ***read()***. De forma adicional, este método proporciona una variable que indica el estado de la lectura (ret), si su valor es 0 indica que se produjo algún tipo de error en la apertura del archivo y por lo tanto, a través de una sentencia ***if*** se finaliza la ejecución del código.

En la misma línea, si la apertura se realizó correctamente, el siguiente paso consiste en la ejecución del modelo sobre el *frame* designado, de tal forma que el resultado que arroja el modelo se escriba en el video de salida por medio del método ***write()***. El proceso continúa hasta abarcar la totalidad del video utilizado como entrada, en dicho momento se tendrá un nuevo video idéntico al anterior en el que están representados los objetos de la detección por medio de *bounding boxes*.

La última sentencia del bucle permite abandonar el proceso antes de culminar el número de frames si el usuario presiona la tecla número veintisiete, que corresponde con la tecla *escape*.

3.6 MONTAJE DEL SISTEMA

Comprobado el algoritmo, el siguiente paso es realizar el montaje del sistema y verificar la ejecución del algoritmo en la placa seleccionada, la RaspBerry Pi 3 modelo B. De la figura 46 a la 49 puede observarse los elementos necesarios para el montaje:

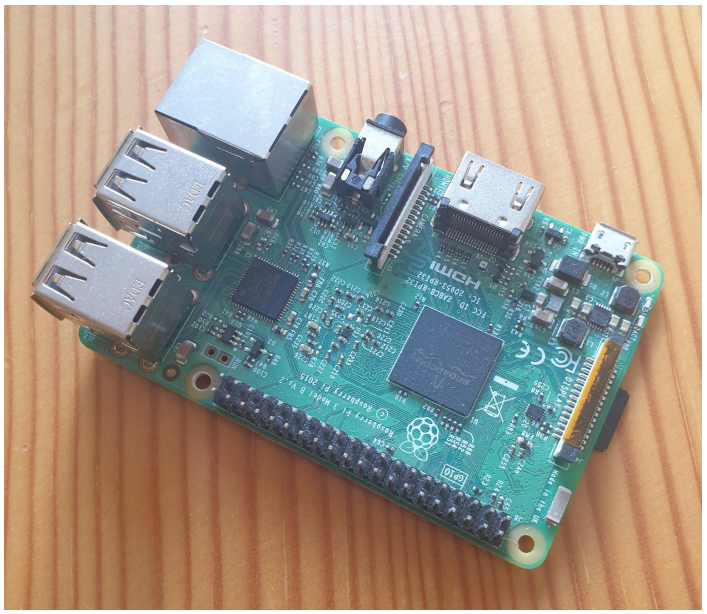


Figura 46. RaspBerry Pi 3 modelo B.

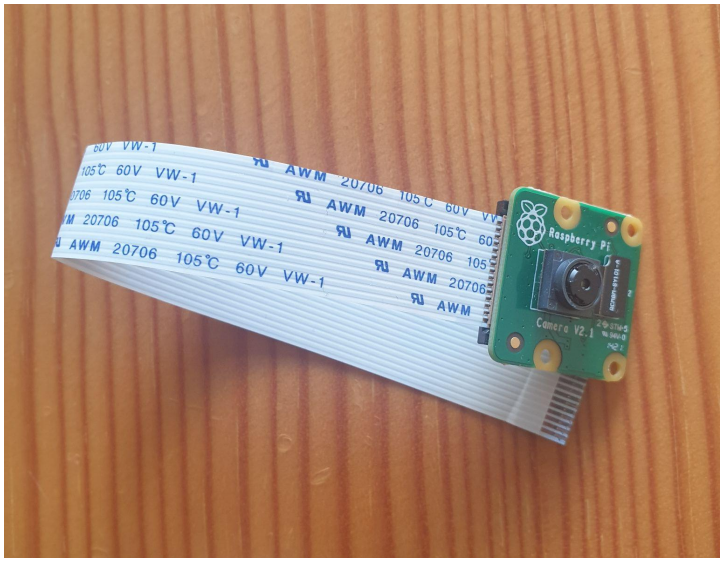
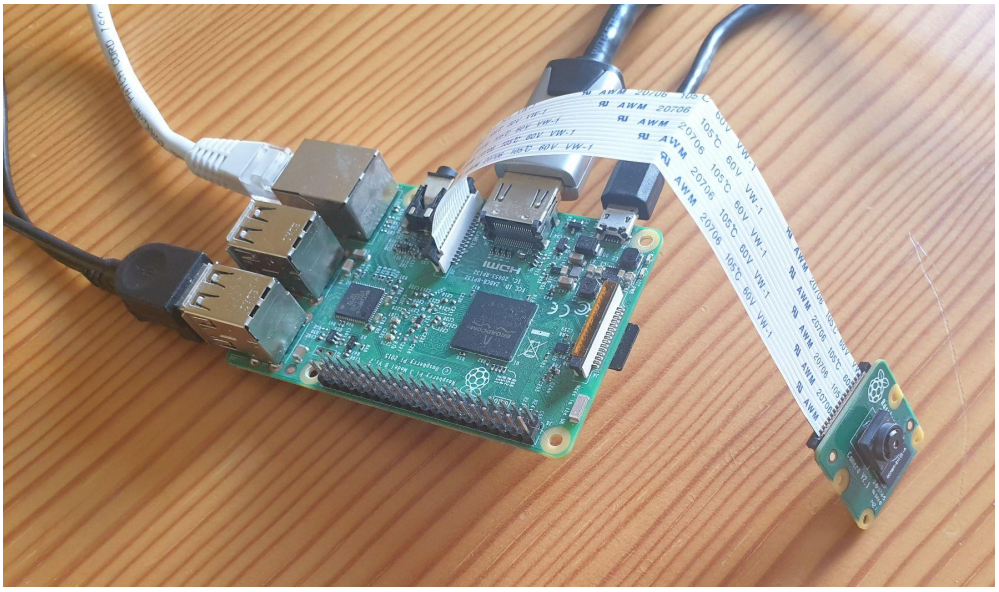


Figura 47. Módulo de cámara RaspBerry Pi V2.

Figura 48. Cables USB 2.0 (OTG) y Ethernet.

Figura 49. Periféricos USB y Cable HDMI.



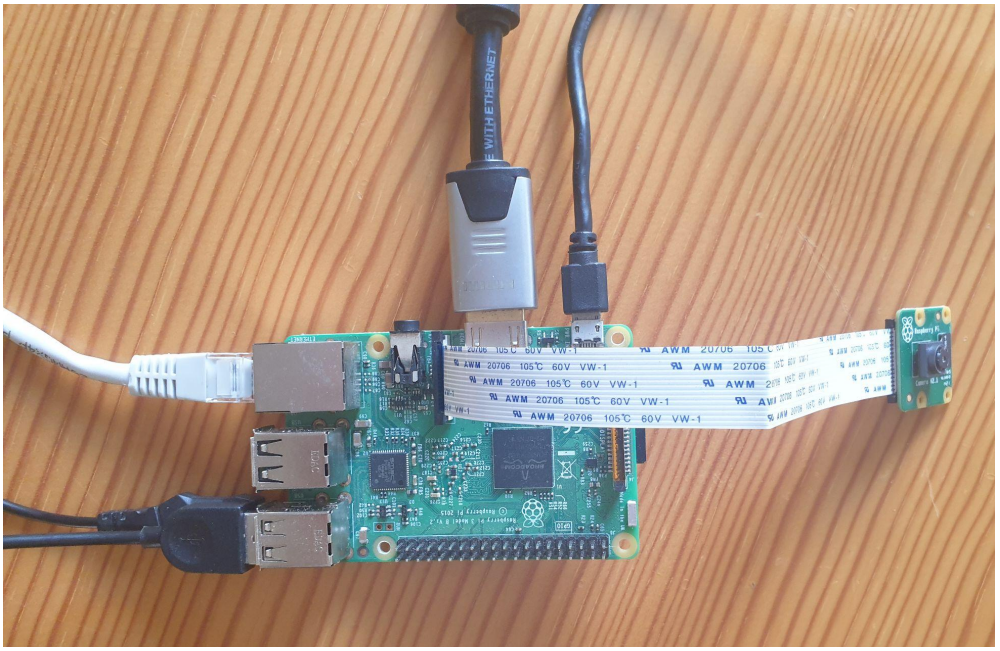


Figura 50. Montaje final de la RaspBerry Pi.

3.7 PROBLEMAS EN LA IMPLEMENTACIÓN Y SOLUCIÓN

Culminado el montaje del sistema, el siguiente paso consiste en comprobar que el código diseñado para la grabación y detección de objetos funciona de acuerdo a lo esperado. Sin embargo, a lo largo de las pruebas realizadas surgieron una serie de inconvenientes que obligaron a cambiar la forma en que trabajará el sistema, a continuación serán comentados los problemas que surgieron en la implementación así como las soluciones consideradas para los mismos:

- **Cámara RaspBerry PI V2 y Opencv:** Para operar la cámara de la RaspBerry Pi por medio de python, existe una librería especializada para ese fin llamada **picamera**, esta librería contiene un conjunto de funciones preparadas para capturar imágenes, grabar videos y configurar el conjunto de propiedades de la cámara. Debido a esto, resulta redundante utilizar **Opencv** para dicho fin, con lo cual se consideró apropiado el diseño de un nuevo código de captura de imágenes y video que utilice la librería correspondiente a la cámara de la RaspBerry Pi.

- **Incompatibilidad con torch:** Un inconveniente de mayor gravedad surgió a la hora de instalar la librería de aprendizaje automático **torch**, esta librería desarrolla un papel fundamental a la hora de trabajar con el algoritmo puesto permite invocar y obtener los resultados de la detección, sin embargo, al intentar proceder con la instalación de la librería en la placa resultaba en una excepción que indicaba la incompatibilidad de la versión existente. Así pues, para tratar de resolver el problema se intentó descargar una versión anterior de la librería para la cual existe un *port* para procesadores ARN (núcleo del procesador

utilizado por la RaspBerry Pi), a pesar esto, la instalación no pudo completarse indicando nuevamente la incompatibilidad con el sistema.

Después de investigar y agotar todas las posibles opciones, se identificó la causa del problema: la librería **torch** sólo está disponible para sistemas operativos de 64 bits. Por lo tanto, cualquier sistema operativo de 32 bits es incompatible con la librería incluyendo el de la propia placa utilizada (RaspBerry PI OS de 32 bits).

Una posible solución al problema es descargar e instalar un nuevo sistema operativo de 64 bits en la placa, sin embargo, debido a que los códigos de video sufrirán cambios a raíz de la necesidad de trabajar con la librería **picamera**, se consideró apropiado utilizar al sistema compuesto por la RaspBerry Pi únicamente para la captura y grabación, es decir, el proceso de detección será llevado a cabo por un sistema externo que recibirá los videos tomados por la placa. De la misma forma, la placa utilizará el código para grabar los videos y posteriormente los subirá a una nube por la cual se tendrá acceso para la posterior detección.

3.8 CÓDIGO DE GRABACIÓN Y ENVÍO

A continuación se mostrarán los códigos utilizados para la captura y envío de video.

Las nuevas librerías que serán utilizadas en la herramienta final son las siguientes:

- **picamera:** Librería utilizada para la manipulación del módulo de cámara de la RaspBerry Pi V2, con un conjunto de funciones que permiten captura de imágenes, grabación, entre otro tipo de utilidades. [24]

- **pydrive:** Librería que permite acceder a la nube de google drive para poder subir, descargar o crear archivos en la misma, a través de *scripts* de python. [25]

El objetivo por tanto, es hacer un código que permita capturar imágenes o grabar vídeos según los requerimientos prefijados. Posteriormente, por medio de la librería **pydrive** se consiga enviar la información obtenida de la captura a la nube de google drive en la cual se tendrá acceso a esta en cualquier momento.

Código implementado en la RaspBerry Pi:

- **Configuración de las propiedades de la cámara:** La librería picamera permite crear un objeto de la clase ***PiCamera*** con el cual se tendrá control del módulo de cámara de la placa, el primer paso una vez creado dicho objeto es utilizar una serie de métodos proporcionados por la librería para poder configurar la resolución y la tasa de frames de la cámara.

```
#CONFIGURACIÓN DE LA CÁMARA

cam = PiCamera()
cam.resolution = (320, 240)
cam.framerate = 30
```

Figura 51. Configuraciones preliminares de la cámara.

- **Modos de operación:** A efectos del sistema implementado, se consideró la adición de dos posibles modos de operación indicados por la variable “tarea”, si la variable tiene un valor de uno, entonces el modo de operación realiza la captura esporádica de una serie de imágenes y las guarda en su directorio correspondiente. Si la variable tiene el valor de dos, entonces procede a grabar un segmento de video con un tiempo determinado, de cualquier forma, el resultado será almacenado en un directorio especificado para posteriormente proceder al envío de la información.

```
#INDICAR MODO DE EMPLEO (1 --> CAPTURAS PERIÓDICAS), (2 --> GRABACIÓN)

tarea = int(input('Indicar acción a realizar: 1 (capturas periódicas), 2 (grabacion continua): '))

#-----------------------------------------------------------------------------------------------------------

#Bucle captura periódica

if tarea == 1:
    i = 0
    while True:
        cam.capture('/home/pi/Desktop/YOLO_TFG/capturas_cam/img'+str(i)+'.jpg', resize = (640, 480))
        time.sleep(2)
        i += 1
        print(i)
        if i == 10:
            break

#Grabacion de video

elif tarea == 2:

    t = int(input('Indicar duracion video: '))

    cam.start_recording('/home/pi/Desktop/YOLO_TFG/video_cam/video.h264')
    cam.wait_recording(t)
    cam.stop_recording()

else:
    print('error argumento no valido')

print('end process')
cam.close()
```

Figura 52. Segmento de código de captura y grabación.

La captura de imágenes se realiza por medio un bucle ***while*** y el método de la clase ***capture()***, una vez culminado este sale del bucle a esperas de la siguiente instrucción.

De la misma forma, la grabación se consigue por medio del método ***start_recording()*,** al cual se le pasa como argumento la ruta en la cual se desea guardar el video así como el nombre que tendrá este. De forma adicional, el método ***wait_recording()*** permite indicar el tiempo de duración del video, y para finalizar la grabación se utiliza otro método ***stop_recording()***, que será ejecutado al culminar la instrucción anterior.

- **Envío de la información a Google Drive:** Terminada el paso anterior, se tendrá a disposición un archivo de video (.h264) o bien un conjunto de imágenes resultado de la captura. Así mismo, el paso siguiente es enviar la información a la nube de google drive, para ello será utilizada una librería de gran flexibilidad que permite subir, descargar y manipular estos archivos, la librería pydrive.

```
#ACCESO AL SERVIDOR DE GOOGLE
gauth = GoogleAugth()
drive = GoogleDrive(gauth)

#-------------------------------------------------------------------------------

#DEFINICIÓN ID FICHERO DE DESTINO

id_destino = '1El_OSLvCaT9W1VJ7wxaFj5xkPEy79yw9'

#-------------------------------------------------------------------------------

#RUTA DE ARCHIVOS DE ORIGEN, SI TAREA = 1 --> CAPTURAS, SI TAREA = 2 --> VIDEO

if tarea == 1:

    carpeta_origen = '/raspberry/pi/Desktop/YOLO_TFG/captura_cam'

elif tarea == 2:

    carpeta_origen = '/raspberry/pi/Desktop/YOLO_TFG/video_cam'


#-------------------------------------------------------------------------------
```

Figura 53. Código para subir archivos a google drive.

Las primeras dos líneas de código crean el acceso al servidor de google para poder subir los archivos, seguidamente se define la variable que contiene el identificador del fichero de google drive en la cual se quiere subir el archivo. Así mismo, la variable denominada “carpeta_origen” contiene la ruta de memoria en la cual se encuentra el conjunto de archivos que se desea enviar a la nube, bien el video o el grupo de capturas tomadas por la cámara en el paso anterior.

```
#Crear una lista que contenga los archivos a enviar

filename = os.listdir(carpeta_origen)

#bucle para el conjutno de archivos

for i in filename:

    #Se guarda la ruta completa de cada archivo y se sube a google drive

    f_origen = os.path.join(carpeta_origen, i)
    googledrive = drive.CreateFile({'parents' : [{'id' : id_destino}], 'title' : i})
    googledrive.SetContentFile(f_origen)
    googledrive.Upload()
```

Figura 54. Código para subir archivos a google drive.

Seguidamente se crea una lista que contendrá aquellos archivos almacenados en el directorio de origen, debido a que puede existir más de un archivo en esa carpeta, se procede a implementar un bucle que recorra la lista de archivos.

Finalmente, las últimas líneas de código indican el directorio de destino y el contenido que será enviado a este para posteriormente subir el archivo a google drive.

Al finalizar la implementación del código, se realizó una primera prueba que resultó exitosa. No obstante, a pesar de esto, no se pudo volver a utilizar la placa puesto que la misma presentó un problema que le impedía iniciar.

Al volver a intentar iniciar la placa, esta no generaba salida de vídeo ni suministraba corriente a los periféricos USB conectados a esta. La única respuesta que presentaba era el led frontal rojo que permanecía encendido en todo momento, de esta forma, con el fin de dar con la causa del problema se procedió a investigar al respecto para dar con una posible solución.

Figura 55. Problema al iniciar la RaspBerry Pi.

Así mismo, después de investigar en profundidad entre las posibles causas del suceso se consideró que las más probables de acuerdo a la situación son las siguientes:

- **Caída de voltaje:** En caso de que la alimentación de la RaspBerry Pi caiga por debajo de 3.3 V, esta no iniciará adecuadamente, además, el cable para alimentar la placa no corresponde con el cable oficial distribuido por RaspBerry, con lo cual también es posible que se produzcan fluctuaciones de voltaje que impidan que inicie.

- **Corrupción del sistema operativo:** En el extremo frontal de la placa existen dos led que indican el estado de esta, el led rojo indica que la placa recibe energía, el siguiente led es de color verde, e indica que se está ejecutando el sistema operativo. De esta forma, si la luz verde no parpadea en ningún momento al conectar la alimentación, puede tratarse de un problema con el sistema operativo.

Para intentar solucionar el problema se probaron varías alternativas a la alimentación, se cambiaron los cables utilizados y se conectó directamente a una *powerbank* que estabiliza adecuadamente el voltaje. De la misma forma se utilizó una nueva tarjeta SD para instalar nuevamente el sistema operativo utilizado en la placa, con el fin de comprobar que una versión más reciente del mismo resolvería el problema.

Sin embargo, pese a todo lo anterior, la placa no inició en ningún momento, con lo cual es probable que se deba a una avería interna de la placa de la misma forma que los fallos del módulo wifi y la conexión serial I2C. Debido a la imposibilidad de probar con otra placa similar, se dio por finalizado el proyecto y serán presentados los resultados obtenidos hasta este punto, los cuales cumplen con los objetivos planteados inicialmente en el proyecto.

4. RESULTADOS Y DISCUSIÓN

El siguiente apartado tiene como finalidad mostrar los resultados obtenidos a lo largo de la ejecución del proyecto, así como también comentar los aspectos relevantes de cada uno.

4.1 TESTEO DEL ALGORITMO ENTRENADO

Finalizado el entrenamiento del modelo YOLO, puede ejecutarse este sobre todo el conjunto del *dataset* reservado para la prueba del mismo. En la figura 56 puede observarse la ejecución del *script* ***detect.py*** que una vez ejecutado, el algoritmo procede a analizar cada una de las imágenes contenidas en el directorio de '*test*' indicando en cada una de las imágenes si existe un objeto de las clases definidas previamente, así como el tiempo estimado para su detección y guardando los resultados para poder acceder a estos más adelante con el fin de confirmar la eficacia de la detección.

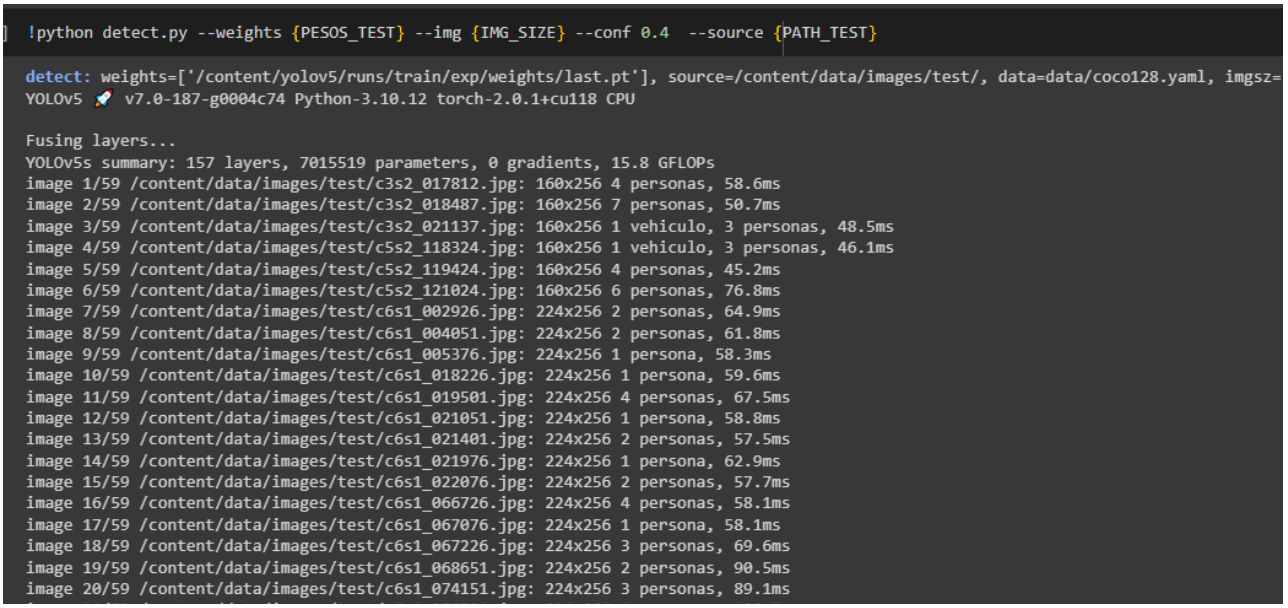


Figura 56. Testeo del algoritmo YOLO.

De la misma manera, es conveniente indicar que, por cada objeto detectado, el algoritmo calcula una probabilidad de que este objeto forme parte de la clase correspondiente de acuerdo a las características extraídas. Por lo tanto, para obtener únicamente los resultados que correspondan a la clase definida se le indica un umbral a partir del cual todo objeto detectado que posea una probabilidad menor a dicho umbral es automáticamente descartado,

para el caso se utilizó un umbral de 0.4 con el cual se obtuvieron buenos resultados.

En las figuras 57 a la 60 puede observarse algunas de las imágenes de testeo resultado del proceso anterior:

Figura 57. Imagen empleada en el testeo del algoritmo.

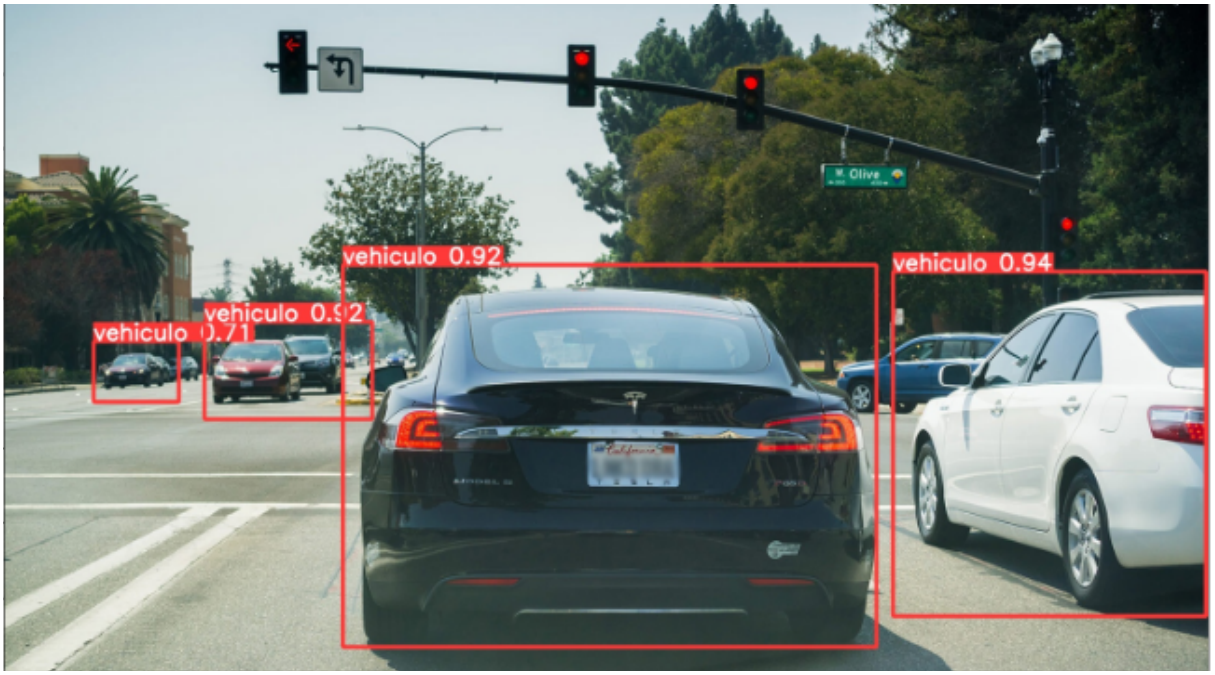


Figura 58. Imagen empleada en el testeo del algoritmo.

Figura 59. Imagen empleada en el testeo del algoritmo.

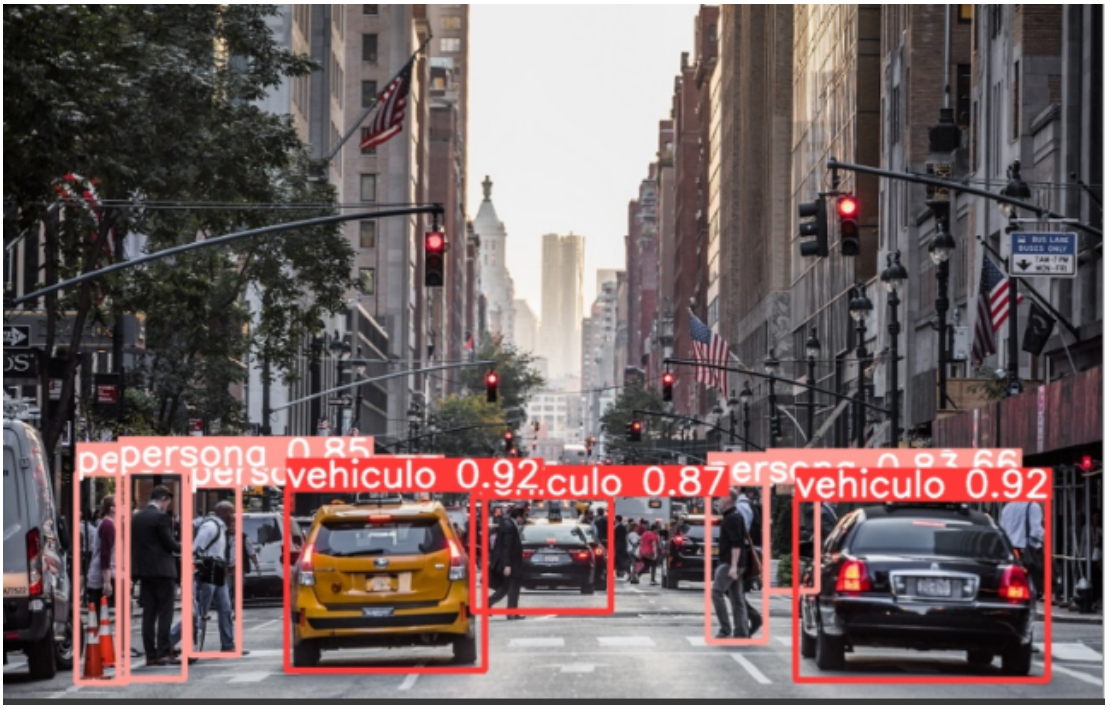


Figura 60. Imagen empleada en el testeo del algoritmo.

Como puede observarse en las imágenes de prueba, la detección se lleva a cabo de manera satisfactoria para la gran mayoría de objetos relevantes en imágenes de naturaleza variada, con lo cual puede confirmarse que la primera fase de prueba arroja resultados adecuados a lo esperado.

4.2 PRUEBA CON IMÁGENES FUERA DEL DATASET

Para continuar la fase de prueba, fueron seleccionadas un conjunto de imágenes obtenidas de la web que presentaban distintas situaciones en las que podría encontrarse el sistema de vigilancia basado en YOLO. De la misma forma, la invocación del modelo se llevó a cabo por medio de Pytorch, en las figuras 61 a la 63 pueden observarse los resultados obtenidos.

Figura 61. Imagen de prueba del algoritmo.

Figura 62. Imagen de prueba del algoritmo.

Figura 63. Imagen de prueba del algoritmo.

De los resultados obtenidos en la prueba, puede apreciarse que el algoritmo entrenado realiza la detección de manera correcta en gran parte de los casos para imágenes que divergen ligeramente de las utilizadas durante el entrenamiento. Sin embargo, es evidente que para imágenes de gran complejidad como es el caso de la figura 63, el algoritmo puede presentar alguna confusión a la hora de detectar objetos que se encuentren a una distancia significativa de la cámara y que puedan estar parcialmente cubiertos por otro tipo de elemento, no obstante, para efectos del trabajo a ejecutar los resultados obtenidos resultan adecuados y puede concluirse que la detección se llevó a cabo de manera satisfactoria.

Adicionalmente, como todo tipo de algoritmos de aprendizaje automático, la eficiencia con la cual consiga realizar las tareas propuestas dependerá en gran medida del entrenamiento administrado. Para mejorar los resultados es conveniente tener un gran número de imágenes cuya variedad

permita al algoritmo considerar muchas situaciones diferentes en las cuales pueda trabajar, caso contrario, pueden suceder pequeños errores como los presentes en la figura 63 en la que la detección identifica de manera errónea algunos objetos, puesto que se alejan considerablemente del tipo de datos utilizado para su entrenamiento.

4.3 VALIDACIÓN Y ESTADÍSTICAS DE TESTEO

De manera adicional, una forma de evaluar el desempeño del algoritmo entrenado es a través de las curvas PR (*Precision-Recall*), de esta forma, es necesario conocer los conceptos de '*Precision*' y de '*Recall*' para poder entender la forma con la cual se utilizan para evaluar el rendimiento a través de la contabilización de las predicciones hechas.[21]

TP (*True positives*): Indica el número de objetos pertenecientes a una clase que fueron detectados correctamente.

FP (*False positives*): Caso contrario al TP, conforman aquel conjunto de imágenes en las cuales se detectó como correcto un objeto que no pertenece a la clase a detectar.

TN (*True negatives*): Corresponde a aquellos casos en los cuales no existe ningún objeto en la imagen que forme parte de la clase definida, y por tanto, el algoritmo no arroja una detección sobre dicha imagen.

FN (*False negatives*): De la misma forma que en los casos anteriores, este indicador proporciona una medida del número de imágenes que contienen objetos de la clase a detectar, en los cuales no se produce una correcta detección.

Estos indicadores se utilizan para definir los ratios de *precision* y *recall* de la siguiente manera:

***Precision*:** La precisión es el ratio de predicciones correctas por parte del detector, es decir, de todo lo que detecta el algoritmo tanto correcta como incorrectamente, que parte corresponde con la detección correcta . [21]

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

Figura 64. Ratio de precisión. [21]

***Recall*:** También conocido como sensibilidad, es el ratio de positivos detectados en el set de datos por el algoritmo. En otras palabras, de todos los objetos contenidos en las imágenes del dataset, detectados o no, cual es el ratio de positivos detectados. [21]

$$recall = \frac{TP}{TP + FN}$$

Figura 65. Ratio de recall. [21]

Así pues, los indicadores anteriormente definidos suelen representarse gráficamente en una curva llamada curva de *precision-recall*, en esta curva puede observarse la relación existente entre dichas magnitudes. Idealmente la curva debe acercarse lo máximo posible a la esquina superior derecha, es decir, para un recall muy alto se tiene una precisión igual de alta, cosa que nunca sucede en la práctica.

En la misma línea, las métricas de *precision y recall* están relacionadas de tal forma que exista una disminución de una de ellas a medida que aumenta la otra, en otras palabras, un modelo con una alta precisión será capaz de obtener un mayor número de aciertos en la detección. Sin embargo, no tendrá una sensibilidad suficiente como para identificar ligeras variaciones en el set de datos proporcionado, con lo cual no será capaz de detectar positivos en determinadas imágenes del *dataset*.

Dicho lo anterior, se busca obtener un equilibrio entre las dos curvas con el fin de aprovechar al máximo las ventajas proporcionadas por ambas métricas, normalmente el modelo utiliza un umbral para indicar si la detección es correcta, si el valor devuelto por el modelo es superior a dicho umbral entonces se toma como positivo, caso contrario es descartado.

En las figuras 66 y 67 puede observarse las curvas PR obtenidas para el modelo entrenado, tanto para validación como estadísticas de testeo:

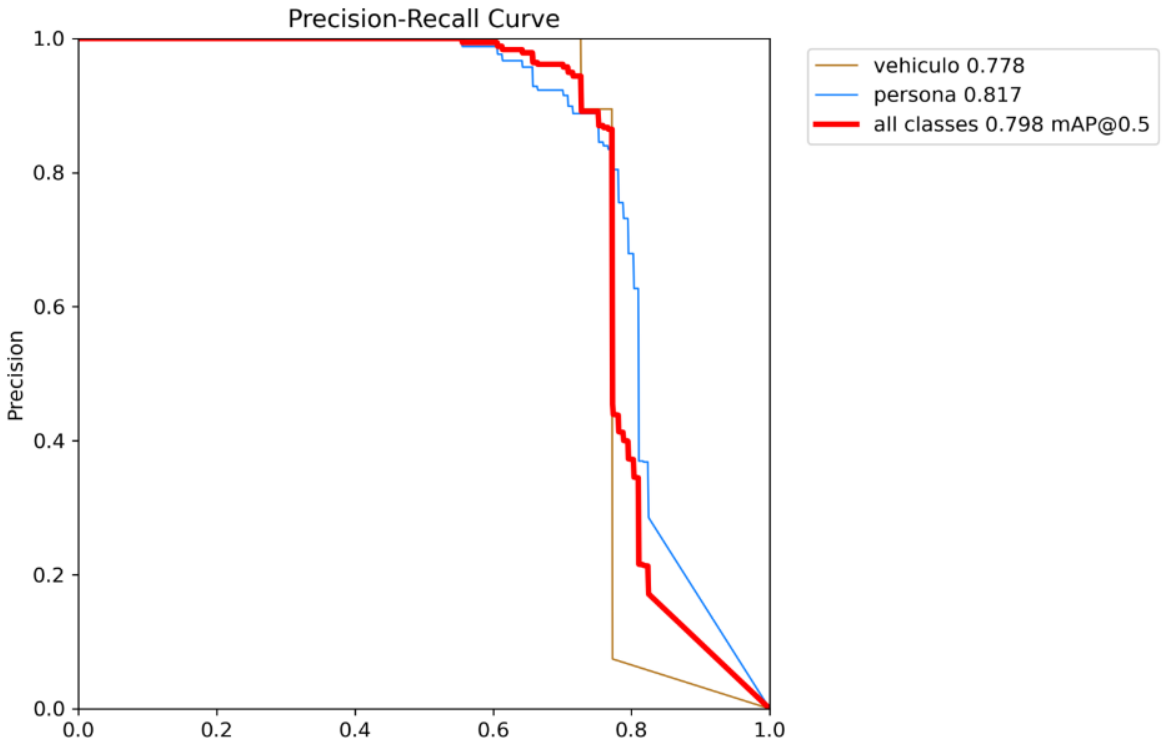


Figura 66. Curva PR de validación.

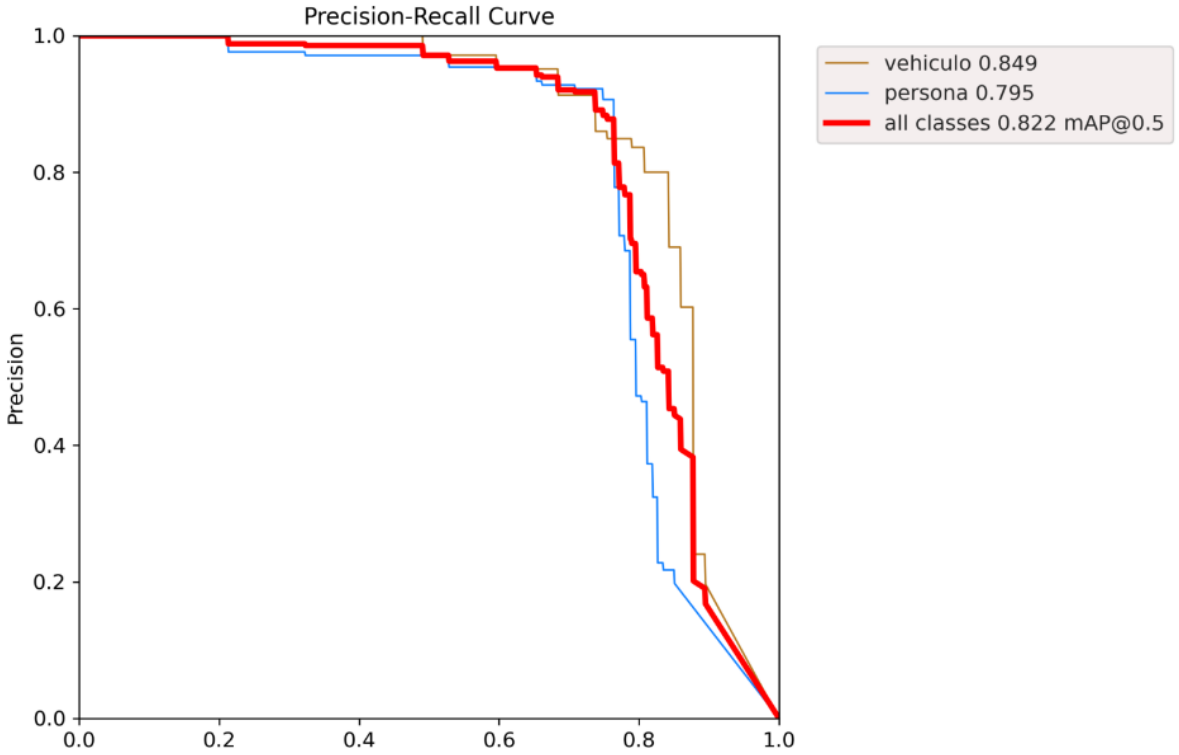


Figura 67. Curva PR de estadísticas de testeo.

En ambas curvas puede observarse la forma en que interactúan las métricas entre sí, a medida que disminuye la precisión del modelo aumenta la sensibilidad del mismo, con lo cual se comporta de acuerdo a lo esperado por la razón comentada en los párrafos previos. Ambas magnitudes están relacionadas de forma inversa con lo cual no existe ningún posible escenario en el que las dos crezcan simultáneamente.

Dicho esto, es importante determinar un umbral de confianza para el cual la detección llevada a cabo por el modelo se vea balanceada en ambos aspectos, de esta forma se consigue un alto porcentaje de acierto así como cierto grado de robustez ante posibles variaciones del set de datos. A pesar de que el umbral de confianza puede ajustarse a conveniencia, el modelo una vez entrenado suele determinar aquel que mejor se ajusta a la distribución de datos proporcionada para desarrollar dicha tarea.

Para el caso estudiado, se tiene un porcentaje de precisión considerable para ambas clases definidas, como es de esperar, si se desea aumentar la robustez del modelo es necesario un entrenamiento de mayor complejidad que consiga preparar al algoritmo para enfrentar nuevos escenarios.

No obstante, se considera que para efectos del proyecto desarrollado el resultado obtenido es satisfactorio y cumple con las características propias de los modelos de inteligencia artificial utilizados para desempeñar tareas similares a esta.

4.4 PRUEBA DE VIDEO

La última prueba realizada para comprobar la eficacia de la detección realizada sobre las clases definidas consiste en un prueba de *tracking*, en la cual el algoritmo llevará a cabo la detección de los objetos en un video determinado. A efectos prácticos, se utilizaron distintos videos obtenidos directamente de Youtube donde se presentan diferentes situaciones cotidianas en las cuales el algoritmo tendrá que detectar aquellos elementos que pertenecen a sus respectivas clases.

Video 1. Prueba simple, sistema de vigilancia. [28]

Figura 68. Tracking de un peatón por medio de YOLO .

Figura 69. Tracking de un peatón por medio de YOLO.

El video anterior presenta a una única persona realizando una ruta en la que a medida que transcurre el video, esta se va acercando cada vez más a la cámara, como resulta evidente, el algoritmo consigue la correcta detección del transeúnte así como el seguimiento del mismo en todo el video.

Video 2. Prueba de mayor complejidad. Sistema Dashcam. [29]

Figura 70. Ejemplo sistema Dash-Cam con detección incorporada.

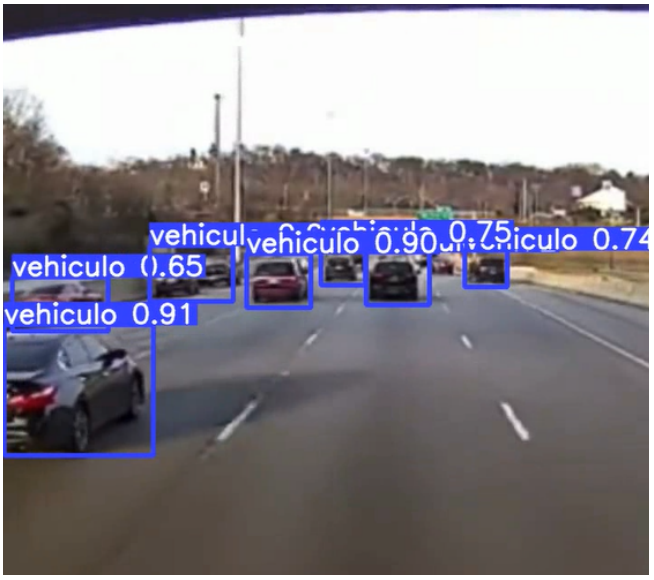


Figura 71. Ejemplo sistema Dash-Cam con detección incorporada.

El video en cuestión muestra un sistema dashcam que se encuentra filmando un tramo de una autopista, el modelo arroja buenos resultados siendo

capaz de detectar un número importante de vehículos en distintas proximidades y ángulos. Si bien pueden producirse errores a la hora de aumentar de forma significativa el número de vehículos presentes, a efectos del trabajo ejecutado, los resultados obtenidos son satisfactorios.

Video 3. Prueba de gran complejidad. Sistema de vigilancia público. [30]

Figura 72. Ejemplo sistema vigilancia con detección incorporada.

La prueba anterior contiene la detección conjunta de ambas clases utilizadas en el proyecto, así pues, de la misma manera que en las otras pruebas, la detección se consigue en su gran mayoría. Existen algunos errores de detección que pueden deberse en gran medida a la ausencia de imágenes adicionales que contemplen un mayor número de casos, pero en primera

instancia el algoritmo consigue el objetivo propuesto sin mayores complicaciones.

Video 4. Prueba anexo. Detección de ojos y boca. [31]

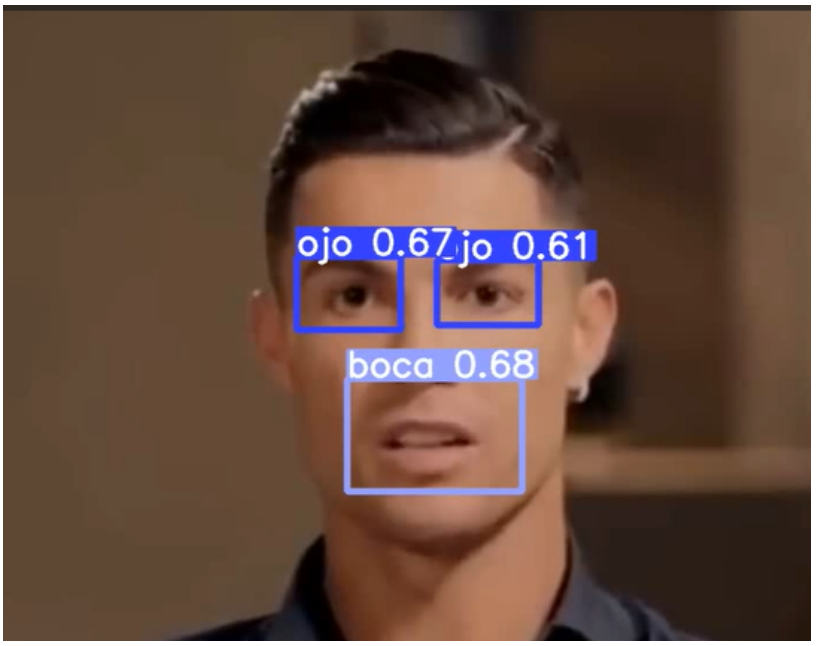


Figura 73. Ejemplo detección de ojos y boca.

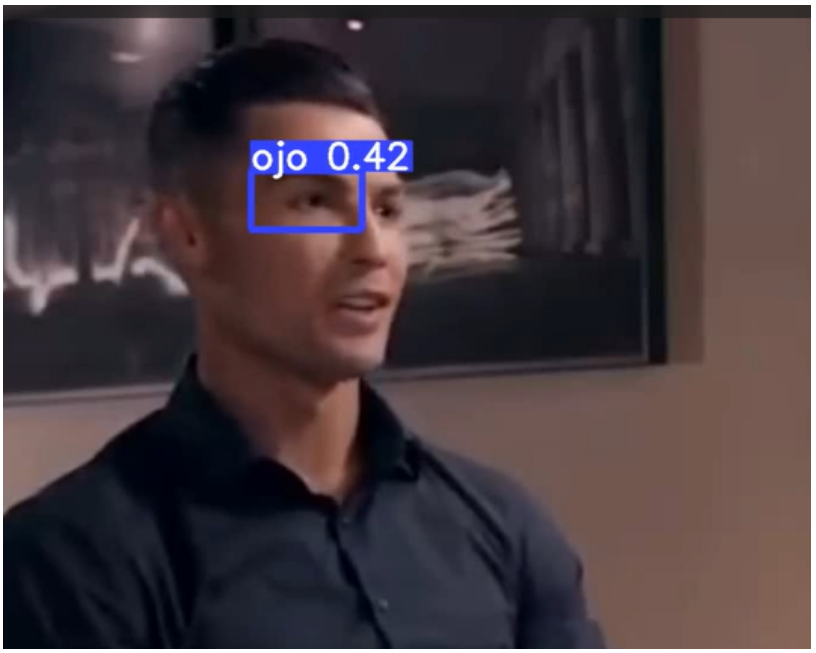


Figura 74. Ejemplo detección de ojos y boca.

La prueba número 4 constituyó el primer contacto con el entrenamiento y prueba del algoritmo de inteligencia artificial, mucho antes de llevar a cabo la recolección del *dataset* de imágenes relevantes para el proyecto. Para este caso se emplearon imágenes sencillas de diversas personas las cuales se encontraban mirando fijamente a la cámara, este ejemplo no solo tuvo gran importancia a la hora de aprender a trabajar con el algoritmo, sino que además sirve para destacar la gran importancia que tiene la recolección y correcto etiquetado de las imágenes a utilizar.

En el video utilizado se presenta una entrevista con el jugador de fútbol Cristiano Ronaldo, el algoritmo en la gran mayoría del vídeo consigue detectar y realizar el seguimiento de manera correcta de los ojos y boca del jugador. Sin embargo, en el momento que se produce un cambio de perfil que no fue contemplado en el set de entrenamiento, el algoritmo se ve incapaz de cumplir su función, fallando en la detección esperada.

4.5 GRABACIÓN (RASPBERRY PI)

A pesar de que no pudo realizarse un gran número de pruebas, la placa sí consiguió de manera correcta la toma de capturas, grabación de video y subida a la nube de los archivos generados.

No se puede proporcionar gran cantidad de imágenes, pero es suficiente para comprobar la veracidad de lo comentado.

Imágenes de prueba:

Figura 75. Imágenes de prueba tomadas por la placa.

Así mismo,en la figura 76 puede confirmarse que efectivamente se enviaron los archivos correctamente a google drive:

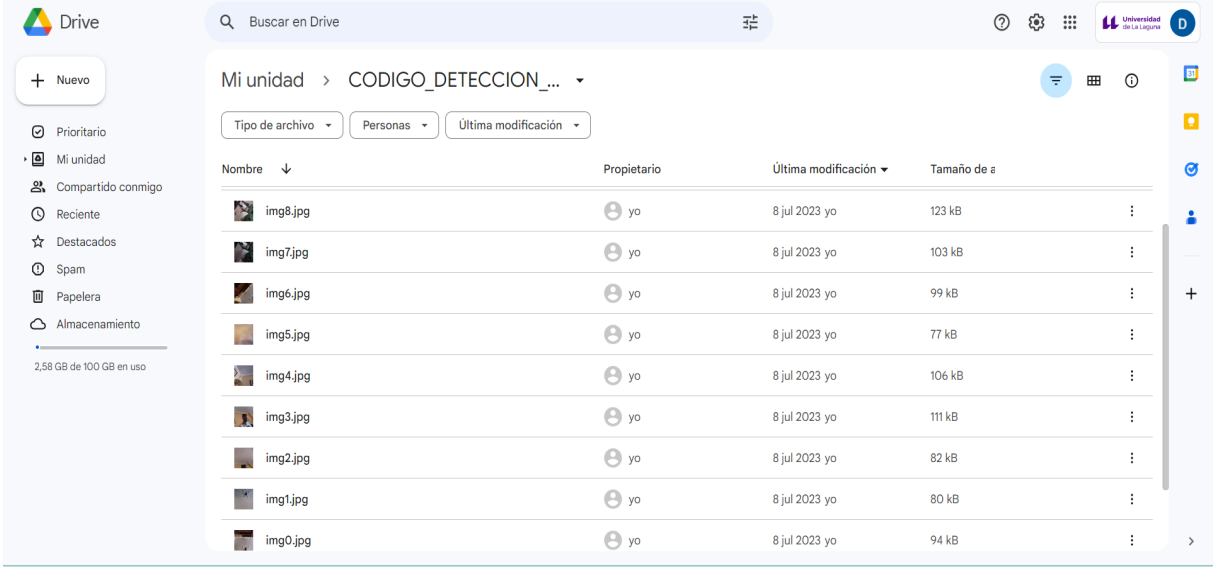


Figura 76. Imágenes subidas a google drive.

5.PROPUESTAS DE MEJORA Y AMPLIACIÓN

El objetivo de este apartado final es englobar todas aquellas mejoras y propuestas adicionales que no fueron implementadas en el diseño final, pero resulta interesante poner en conocimiento para futuras implementaciones en el diseño del sistema.

5.1 UTILIZACIÓN ORANGE PI ZERO

Como fue comentado anteriormente en el apartado de desarrollo del proyecto, la placa RaspBerry Pi presenta el inconveniente de que a día de hoy resulta complicado encontrar una debido más que nada a la escasez de

semiconductores producto de la situación política territorial que se encuentra atravesando el mundo.

Debido a lo anterior, fue propuesto como hardware inicial del sistema a diseñar una placa denominada Orange PI Zero, la cual presenta una importante alternativa a la RaspBerry Pi, siendo además compatible con muchas aplicaciones de esta. Sin embargo, el modelo de Orange Pi disponible para el proyecto no cuenta con un módulo de conexión compatible con la cámara módulo V2 de la RaspBerry Pi, dicho lo anterior, sí que es posible utilizar la placa para la captura y grabación por medio de una webcam usb.

5.2 IMPLEMENTACIÓN DE OCR (RECONOCIMIENTO ÓPTICO DE CARACTERES)

Para reforzar la solidez del sistema de vigilancia y monitorización, puede incluirse de manera adicional a la detección de vehículos la identificación de las matrículas de los mismos, de tal forma que sea capaz de asignarse a cada uno de los vehículos identificados un identificador que lo vincule al número de matrícula que fue detectado con anterioridad. Esto facilitará en mayor medida la búsqueda de un vehículo determinado una vez sea detectada una actividad sospechosa en la región sobre la cual se pretende conseguir la vigilancia.

Adicionalmente, esta implementación tendrá una gran utilidad para otro tipo de sistemas no solo de vigilancia, de la misma forma, en el *parking* de un centro comercial al mismo tiempo que se consigue una vigilancia óptima puede automatizarse el sistema de *parking*, de tal manera que por medio de visión artificial se identifique aquellos vehículos que tienen permitido salir una vez el propietario realizará el pago respectivo.

El OCR puede implementarse con YOLO de la misma forma que la fase principal de detección tratada en este proyecto, sin embargo, de manera

adicional se suele combinar con Teseract, un potente motor de reconocimiento óptico de código abierto utilizado en mayor medida para transcribir texto en imágenes. Este modelo se encuentra completamente entrenado y los pesos necesarios para su utilización están disponibles en internet a través del github “Custom-OCR-YOLO” [22].

Figura 77. Modelo para la detección de caracteres. [22]

5.3 UTILIZACIÓN DE NUEVAS CLASES

De la misma forma, el aumento del número de clases a detectar permite una mejor vigilancia del perímetro, puesto que el sistema será capaz de identificar una gran variedad de elementos que serán relevantes en el momento de analizar un suceso ocurrido en la zona.

A efectos del proyecto, la definición de una única clase denominada “vehículos”, limita enormemente la tarea de monitorización, con lo cual una gran variedad de clases que tomen en cuenta las diferencias existentes entre cada uno de los posibles medios de transporte que se encuentren rondando las

cercanías de la región de interés, permite tener un mejor control y facilita la interpretación de los vídeos adquiridos por el sistema.

5.4 SISTEMA DASHCAM PARA VEHÍCULOS

Un sistema Dashcam o de cámara "*on board*", es un elemento que consta de una cámara colocada en el parabrisas de un vehículo con la finalidad de realizar una grabación continua del recorrido del mismo, de tal forma que si existiera un posible accidente o suceso fuera de lo común que involucre al vehículo en donde se colocó previamente el sistema, se tenga evidencia de lo sucedido y puedan identificarse a los posibles culpables. [22]

En España no existe un uso extendido de estos sistemas, y si bien la toma de imágenes en este tipo de casos está regulada por la Ley de Protección de Datos. Es perfectamente legal instalar un sistema dashcam en un vehículo, siempre y cuando la información recabada por este sea utilizada adecuadamente.

Dicho lo anterior, utilizar las grabaciones proporcionadas por dicho sistema como prueba de un accidente está permitido y lo ampara la regla del interés legítimo contemplada en el Artículo 6 del Reglamento General de Protección de Datos, siempre y cuando se adopten una serie de precauciones para realizar la grabación: [22]

- Activar la grabación solo en el momento del suceso.
- La grabación solo se limita a la parte frontal del vehículo.
- Difuminar datos de personas o vehículos que no estén involucrados en el accidente.

Debido a las pequeñas dimensiones del sistema diseñado, puede emplearse perfectamente como dashcam que recabe información dada una situación determinada, y posteriormente la suba a una nube en internet para poder almacenar la evidencia en caso de un posible accidente.

Adicionalmente, puede implementarse a través de un algoritmo de detección como el utilizado en el proyecto, la identificación de los posibles involucrados por medio del reconocimiento de las matrículas de los vehículos, con lo que se agilizaría enormemente el proceso de análisis de la evidencia.

Figura 78. Sistema Dashcam implementado en un vehículo. [22]

5.5 CONTROL DE CÁMARAS

Utilizando un controlador, como puede ser una placa arduino, se puede conseguir el control de un servomotor solidario a una webcam, dicha placa tendrá como función producir la salida apropiada para garantizar el correcto movimiento del servomotor, de tal forma que la webcam mantenga al objetivo buscado a lo largo de la grabación. Adicionalmente, la grabación captada por la webcam será procesada por la Raspberry PI de la forma explicada en el proyecto.

De esta forma, se tiene un sistema de vigilancia capaz de llevar a cabo el seguimiento de sus objetivos gracias a la adición del controlador mencionado. Este sistema puede implementarse en un gran número de aplicaciones como puede ser robótica, videojuegos o incluso en drones de seguimiento empleados por la policía.

Figura 79. Imagen de un dron de seguimiento. [23]

5.6 CÁMARA INTERNA PARA VEHÍCULOS

Como proyecto adicional al sistema dashcam, podría ser de interés en determinadas aplicaciones colocar una cámara interna en los vehículos para saber el estado del conductor en el momento del accidente, con lo cual podría identificarse posibles negligencias por parte del mismo.

Una posible aplicación de este sistema consiste en colocar cámaras internas en los camiones de una empresa de transporte, así pues, por medio del correcto entrenamiento de un algoritmo de visión artificial se puede detectar cuando un conductor cierra los ojos. Una vez se produzca dicha acción, se activará un temporizador con un umbral determinado, si el conductor mantiene los ojos cerrados el tiempo suficiente se concluye que este se ha quedado dormido y se notifica a la estación, la cual en respuesta activará una alarma en la cabina del conductor para despertarle y notificar que no puede continuar con el trayecto a fin de evitar posibles accidentes.

6. PRESUPUESTO

Concepto	Precio	Unidades	Total
Costes Materiales	-	-	**89,42 €**
Componentes Electrónicos	-	-	**89,42 €**
RaspBerry Pi 3 Modelo B	32,55 €	1	32,55 €
Módulo Cámara RaspBerry Pi V2	29,97 €	1	29,97 €
Tarjeta microSD Kingston 64 GB	14,83 €	1	14,83 €
Cable adaptador USB - microUSB	12,07 €	1	12,07 €
Costes de Ejecución	-	-	**6000,00 €**
Trabajo de Ingeniería	-	-	**6000,00 €**
Mano de obra	20,00 €	300	6000,00 €
TOTAL 6089,42 €			

Concepto		TOTAL
Costes Materiales		89,42 €
Costes de Ejecución		6000,00 €
TOTAL DE COSTES		**6089,42 €**
Gastos Generales	**13%**	791,62 €
Beneficio Industrial	**6%**	365,37 €
Suma de G.G. y B.I.		1156,99 €
COSTE ESTIMADO		**7246,41 €**
I.G.I.C.	**7%**	507,25 €
COSTE TOTAL DEL PROYECTO		**7753,66 €**
EL COSTE TOTAL DEL PROYECTO ES DE SIETE MIL SETECIENTOS CINCUENTA Y TRES EUROS CON SESENTA Y SEIS CÉNTIMOS		

7. CONCLUSIONES

En este trabajo se llevó a cabo el diseño de un sistema de vigilancia por medio de un algoritmo de inteligencia artificial, que tiene la capacidad de detectar eficazmente objetos por medio de un entrenamiento previo. A su vez, por medio de un elemento hardware especializado se realiza la captura y grabación previa del entorno determinado que se desea monitorizar, para posteriormente ser procesadas por el algoritmo y conseguir la identificación de los elementos relevantes en la zona delimitada como son vehículos y personas que transiten el perímetro.

Por una parte, los códigos desarrollados para el entrenamiento del algoritmo obtuvieron resultados satisfactorios para cada una de las diferentes pruebas a las que fueron sometidos. Además, el proceso de validación y prueba del algoritmo demostró tener un comportamiento de acuerdo a lo esperado con un equilibrio adecuado para la precisión y sensibilidad del modelo, obteniendo así un modelo capaz de detectar un gran número de positivos y poseer una robustez suficiente para enfrentar situaciones de diversa naturaleza.

Por otro lado, la implementación de la RaspBerry Pi 3 como elemento principal de captura y grabación resultó exitosa, por medio de los códigos diseñados se consiguió captar información del exterior a través de diversos métodos y posteriormente enviar esta información a google drive. Por medio de lo cual la información se encuentra segura, evitando así la pérdida de ésta por daños en el sistema, y además permitiendo su posterior procesamiento por parte del modelo obteniendo así la detección buscada.

Como conclusión general del trabajo, resulta fundamental destacar la gran potencialidad que tiene los algoritmos de inteligencia artificial, con los cuales se puede conseguir el aprendizaje automático de tareas monótonas

como pueden ser la observación y reconocimiento, aligerando un gran número de actividades y desarrollando mejoras significativas en el rendimiento de muchos procesos en diversos campos, sobre todo en lo industrial. Así mismo, se consiguió una gran comprensión de los conceptos tratados, familiarizando al estudiante con todas las diversas etapas a desarrollar para implementar un algoritmo funcional de este tipo, en general, la experiencia resultó instructiva y permitió conocer nuevos campos de la ingeniería que están en constante expansión en la industria a lo largo del tiempo.

8. CONCLUSIONS

In this work, the design of a surveillance system was carried out by means of an artificial intelligence algorithm, which has the ability to effectively detect objects by means of a previous training. In turn, by means of a specialized hardware element, the previous capture and recording of the specific environment to be monitored is carried out, to be subsequently processed by the algorithm and achieve the identification of the relevant elements in the delimited area such as vehicles and people passing through the perimeter.

On the one hand, the codes developed for training the algorithm obtained satisfactory results for each of the different tests to which they were subjected. In addition, the process of validation and testing of the algorithm showed a behavior according to expectations with an adequate balance for the accuracy and sensitivity of the model, thus obtaining a model capable of detecting a large number of positives and possessing sufficient robustness to face situations of diverse nature.

On the other hand, the implementation of the RaspBerry Pi 3 as the main element of capture and recording was successful, by means of the designed codes it was possible to capture information from the outside through various methods and then send this information to google drive. By means of which the information is secure, thus avoiding its loss due to damage to the system, and also allowing its subsequent processing by means of the model to obtain the desired detection.

As a general conclusion of the work, it is essential to highlight the great potential of artificial intelligence algorithms, with which it is possible to achieve automatic learning of monotonous tasks such as observation and recognition, lightening a large number of activities and developing significant improvements in the performance of many processes in various fields, especially in the

industrial field. Likewise, a great understanding of the concepts discussed was achieved, familiarizing the student with all the various stages to be developed to implement a functional algorithm of this type, in general, the experience was instructive and allowed to know new fields of engineering that are constantly expanding in the industry over time.

9. BIBLIOGRAFÍA

[1] González et al, “Técnicas y algoritmos básicos de visión artificial,” Universidad de La Rioja Servicio de Publicaciones, 2006.

[2] L. E. Sucar, G. Gómez, “Visión computacional,” 2017.

[3] R. Szeliski, “*Computer vision algorithms and applications*,” 2010.

[4] I. Goodfellow, Y. Bengio, A. Courville, “*Deep Learning*,” The MIT Press, 2016.

[5] A. Burkov, “*The Hundred-Page Machine Learning Book*,” *Publicación Original,* 2019

[6] T. Mitchell, “Machine Learning,” McGraw Hill, 1997.

[7] J. Valencia, Deep Learning y Machine Learning: Diferencias, ventajas e inconvenientes, 2017 [en línea], disponible en: https://blog.fromdoppler.com/deep-learning-machine-learning-ventajas-e-inconvenientes/#:~:text=El%20principal%20inconveniente%20del%20Machine,necesita%20de%20un%20aprendizaje%20supervisado.

[8] I. R. Salvador, Corteza Cerebral: Sus capas áreas y funciones, 2018 [en línea], disponible en: https://psicologiaymente.com/neurociencias/corteza-cerebral

[9] Redes Neuronales Artificiales, qué son y cómo se entrenan?, 2019 [en línea], disponible en: https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i

[10] L. Manjarrez, Relaciones Neuronales Para Determinar la Atenuación del Valor de la Aceleración Máxima en Superficie de Sitios en Roca Para Zonas de Subducción, 2014 [en línea], disponible en: https://www.researchgate.net/figure/Figura-III4-Capas-de-una-Red-Neuronal-Capa-de-entrada-neuronas-que-reciben-datos-o_fig3_315762548

[11] Red Neuronal Convolucional, 2023 [en línea], disponible en: https://es.wikipedia.org/wiki/Red_neuronal_convolucional

[12] A. M. Álvaro, Clasificación de imágenes usando redes neuronales convolucionales en Python, 2019 [en línea], disponible en: https://idus.us.es/handle/11441/89506

[13] Redes Neuronales Convolucionales, 2019 [en línea], disponible en: https://pochocosta.com/podcast/redes-neuronales-convolucionales-explicadas/

[14] *Understanding Convolutional Neural Networks (CNNs): A Complete Guide*, 2023 [en línea], disponible en: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

[15] Universidad Tecnológica de San Luís Potosí: Redes Neuronales, 2021 [en línea], disponible en: https://inteligencia-artificial654.webnode.mx/

[16] *Comprehensive Guide to Ultralytics YOLOv5*, 2023 [en línea], disponible en: https://docs.ultralytics.com/yolov5/

[17] M. Ali, Vehicle images dataset for make and model recognition, 2022 [en línea], disponible en: https://data.mendeley.com/datasets/hj3vvx5946/1

[18] Orange Pi Zero [en línea], disponible en: http://www.orangepi.org/html/hardWare/computerAndMicrocontrollers/details/Orange-Pi-Zero.html

[19 Datasheet RaspBerry Pi 3 Modelo B [en línea], disponible en: https://us.rs-online.com/m/d/4252b1ecd92888dbb9d8a39b536e7bf2.pdf

[20] Documentación de Python [en línea], disponible en: https://www.python.org/

[21] J. Ramirez, Curvas PR y ROC, 2018 [en línea], disponible en: https://medium.com/bluekiri/curvas-pr-y-roc-1489fbd9a527

[22] ¿Es legal instalar una dashcam o cámara de salpicadero en el coche? [en línea], disponible en: https://www.motorpasion.com/seguridad/legal-instalar-dashcam-camara-salpicadero-coche-depende-que-hagamos-ella

[23] Drones de Seguimiento [en línea], disponible en: https://bextremeboards.com/blog/lo-ultimo-en-drones-seguimiento-automatico/

[24] Documentación librería picamera [en línea], disponible en: https://picamera.readthedocs.io/en/release-1.13/

[25] Documentación librería pydrive [en línea], disponible en: https://pythonhosted.org/PyDrive/pydrive.html#module-pydrive.drive

[26] Z. Liang et al, PRW (Person Re-identification in the Wild) Dataset, 2016 [en línea], disponible en: http://zheng-lab.cecs.anu.edu.au/Project/project_prw.html

[27] G. Jocher, Ultralytics YOLOv5 Algorithm, 2020 [en línea], disponible en: GitHub - ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite

[28] Video prueba sistema de vigilancia: https://www.kaggle.com/datasets/smeschke/pedestrian-dataset

[29] Video prueba sistema dashcam: https://www.youtube.com/shorts/QKdPRlQrpmI

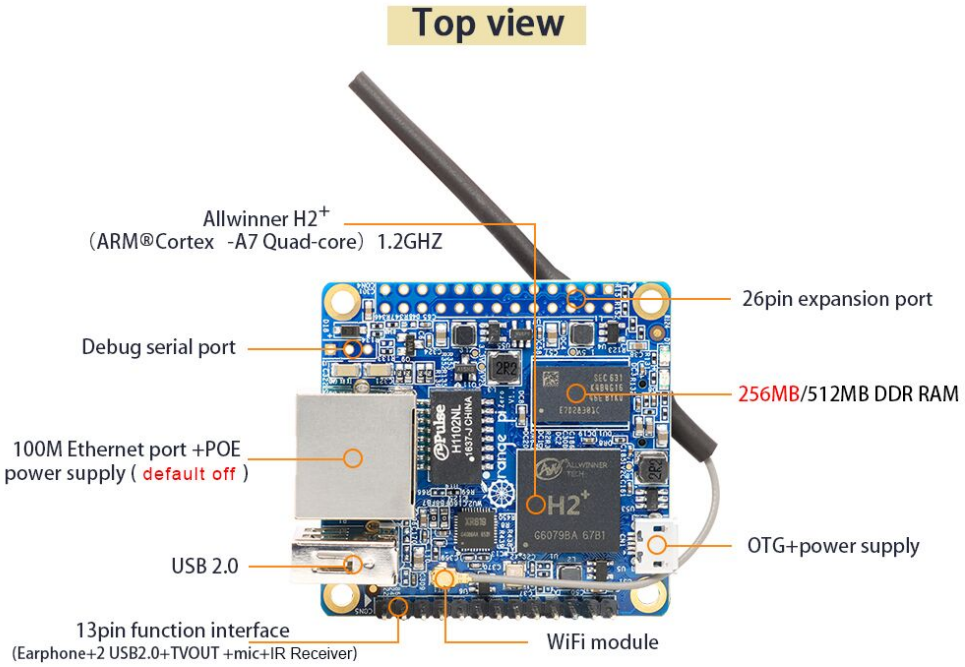
[30] Video prueba sistema de vigilancia público: https://www.youtube.com/shorts/xdcxjWT_prc

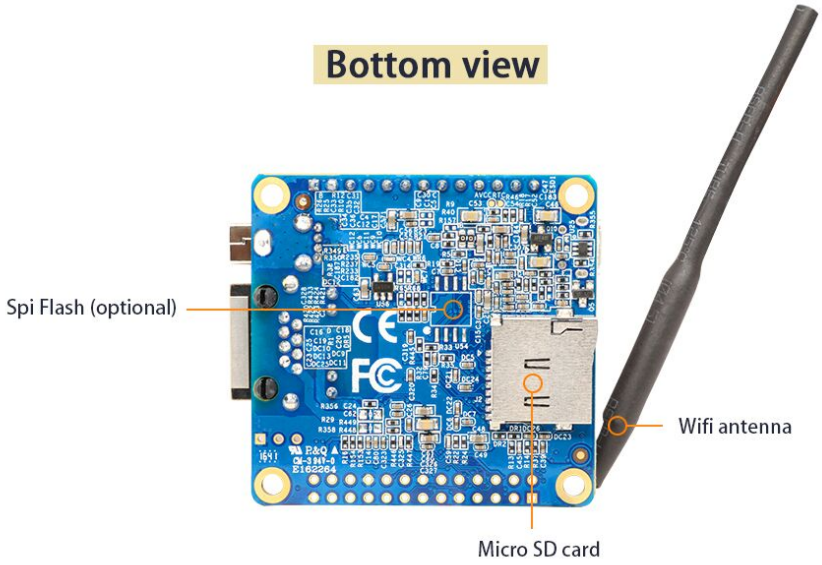
[31] Video utilizado para la prueba anexo: https://www.youtube.com/shorts/EoHppsBAvec

10. ANEXOS

Orange Pi Zero

It’s an open-source single-board computer. It can run Android 4.4, Ubuntu, Debian. It uses the AllWinner H2 SoC, and has 256MB/512MB DDR3 SDRAM(256MB version is Standard version





What can I do with Orange Pi Zero?

Build…

- A computer
- A wireless server
- Games
- Music and sounds
- HD video
- A speaker
- Android
- Scratch
- Pretty much anything else, because Orange Pi Zero is open source

Who's it for?

Orange Pi Zero is for anyone who wants to start creating with technology – not just consuming it. It's a simple, fun, useful tool that you can use to start taking control of the world around you.

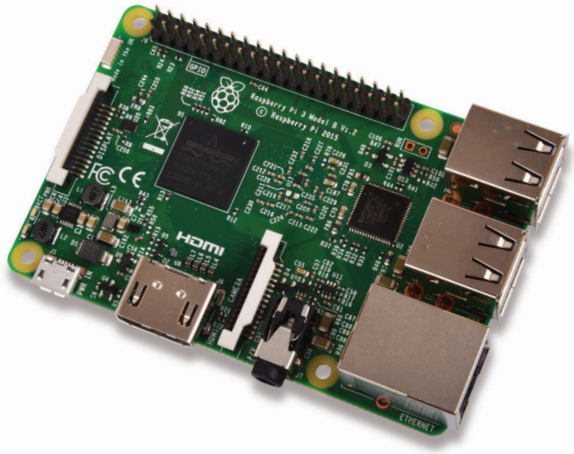
Hardware specification	
CPU	H2 Quad-core Cortex-A7 H.265/HEVC 1080P.
GPU	·Mali400MP2 GPU @600MHz ·Supports OpenGL ES 2.0
Memory (SDRAM)	256MB/512MB DDR3 SDRAM(Share with GPU)(256MB version is Standard version)
Onboard Storage	TF card (Max. 32GB)/ Spi Flash
Onboard Network	10/100M Ethernet RJ45 POE is default off.
Onboard WIFI	XR819, IEEE 802.11 b/g/n
Audio Input	MIC
Video Outputs	Supports external board via 13pins

Power Source	USB OTG can supply power
USB 2.0 Ports	Only One USB 2.0 HOST, one USB 2.0 OTG
Buttons	Power Button
Low-level peripherals	26 Pins Header, compatible with Raspberry Pi B+ 13 Pins Header, with 2x USB, IR pin, AUDIO(MIC, AV)
LED	Power led & Status led
Supported OS	Android, Lubuntu, Debian, Raspbian

Interface definition

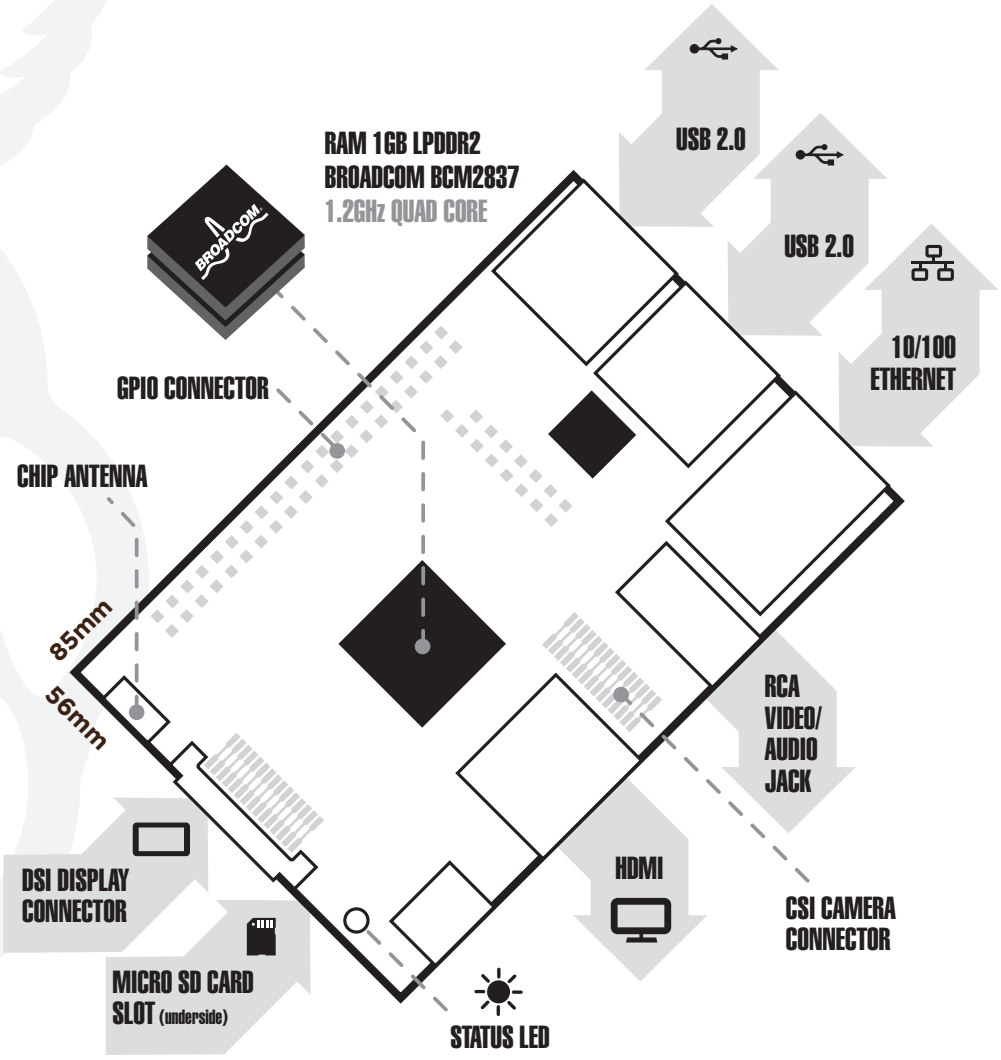
Product size	48 mm × 46mm
Weight	26g

Raspberry Pi



Raspberry Pi 3 Model B

Product Name	**Raspberry Pi 3**
Product Description	The Raspberry Pi 3 Model B is the third generation Raspberry Pi. This powerful credit-card sized single board computer can be used for many applications and supersedes the original Raspberry Pi Model B+ and Raspberry Pi 2 Model B. Whilst maintaining the popular board format the Raspberry Pi 3 Model B brings you a more powerful processer, 10x faster than the first generation Raspberry Pi. Additionally it adds wireless LAN & Bluetooth connectivity making it the ideal solution for powerful connected designs.
RS Part Number	**896-8660**



Raspberry Pi 3 Model B

Specifications

Processor	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	1GB LPDDR2
Operating System	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V1, 2.5A

Connectors:

Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4 Composite RCA (PAL and NTSC)
Audio Output	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Push/pull Micro SDIO

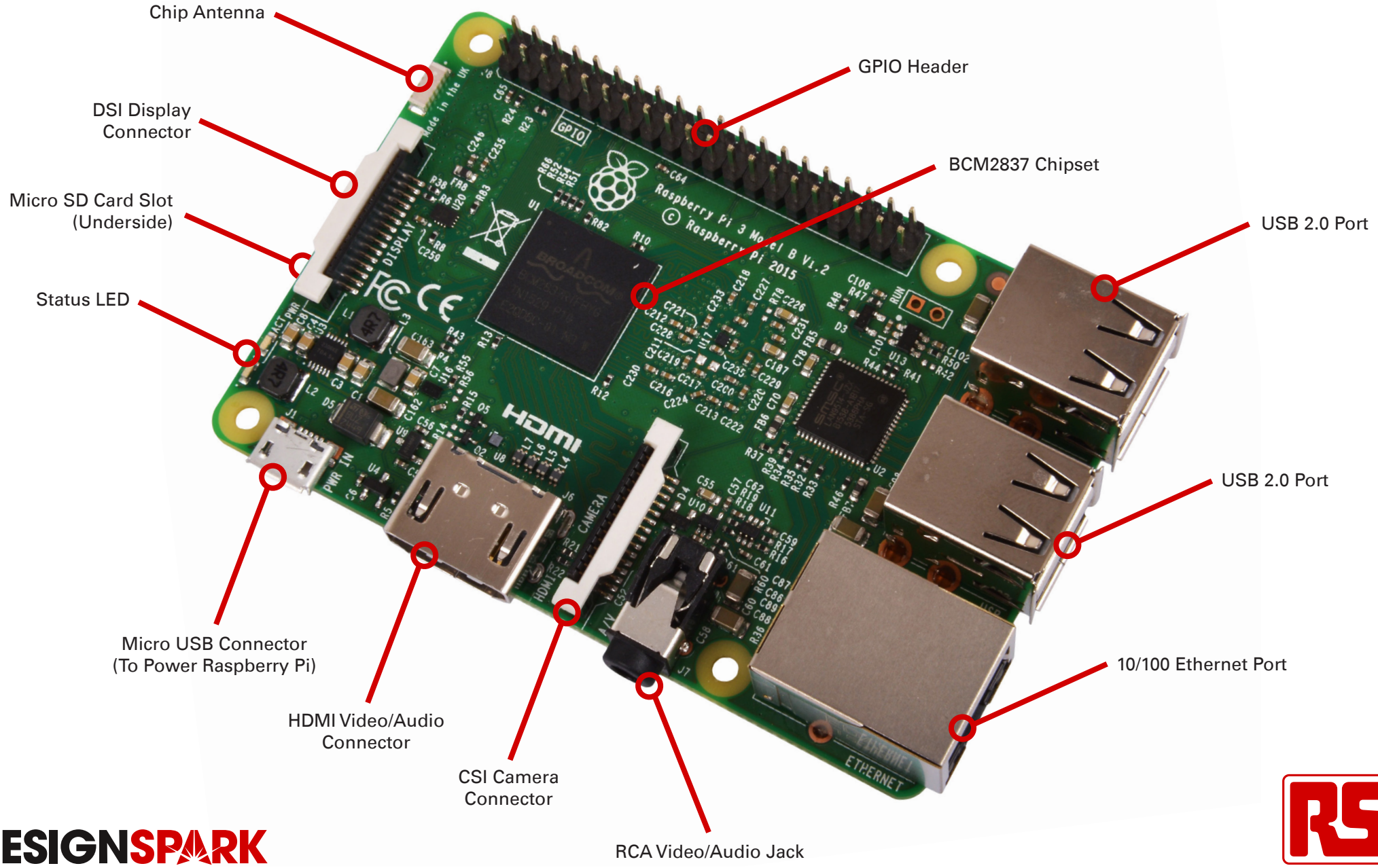
Key Benefits

- Low cost
- 10x faster processing
- Consistent board format
- Added connectivity

Key Applications

- Low cost PC/tablet/laptop
- Media centre
- Industrial/Home automation
- Print server
- Web camera
- Wireless access point
- IoT applications
- Robotics
- Server/cloud server
- Security monitoring
- Gaming
- Environmental sensing/monitoring (e.g. weather station)

Raspberry Pi 3 Model B



DESIGNSPARK

RS

Raspberry Pi
Frequently Asked Questions

What is a Raspberry Pi?

Created by the Raspberry Pi Foundation, the Raspberry Pi is an open-source, Linux based, credit card sized computer board. The Pi is an exciting and accessible means of improving computing and programming skills for people of all ages. By connecting to your TV or monitor and a keyboard, and with the right programming, the Pi can do many things that a desktop computer can do such as surf the internet and play video. The Pi is also great for those innovative projects that you want to try out - newer models are ideal for Internet of Things projects due to their processing power. With Pi 3, Wireless LAN and Bluetooth Low Energy are on-board too.

What are the differences between the models?

Current versions of the Raspberry Pi are the Pi A+, Pi B+, Pi 2 B, Pi 3 B and Compute Module.

	Pi A+	Pi B+	Pi 2 B	Pi 3 B	Compute Module
Dimensions	66 x 56 x 14mm	85 x 56 x 17mm	85 x 56 x 17mm	85 x 56 x 17mm	67.5 x 30mm
SoC	BCM2835	BCM2835	BCM2836	BCM2837	BCM2835
Processor Core	ARM11	ARM11	ARM Cortex-A7	ARM Cortex-A53	ARM11
Processing Power	700 MHz	700 MHz	900 MHz	1.2 GHz	700 MHz
Memory	256 MB	512 MB	1 GB	1GB LPDDR2	512 MB
Ports	1x USB 2.0	4x USB 2.0 1x 10/100 Ethernet	4x USB 2.0 1x 10/100 Ethernet	4x USB 2.0 1x 10/100 Ethernet	N/A
GPIO	40	40	40	40	N/A

What do I get with my Raspberry Pi?

A Raspberry Pi board only.

Each Raspberry Pi customer is unique. You may already have cables, power supplies, keyboards, SD memory cards or monitors. However, if you do require additional products to start with your Pi or to really get creative, we can help.

Our expanding range of accessories includes:

Protective Cases	Power Supplies	NOOBS microSD Cards	Keyboards & Mice	Printers
Cables	Displays & Camera Boards	Wireless Connectivity	Add-on Boards	RS Pi Bundles

How do I get connected?

To get started with your Pi you will need;

- A monitor or TV screen to set-up your Pi
- A keyboard to interact with your Pi
- A mouse to navigate your Pi
- A power supply
- An SD card with the latest version of New Out Of Box Software (NOOBS), to install the operating system that you would like to use.

To get **sound** and **video** you will need cables to suit what your screen or monitor accepts. For those with monitors that accept VGA, a HDMI to VGA adaptor is needed in addition to a HDMI cable, unless you use the composite video output from the Pi.

For an **internet connection**, the Pi B+ and Pi 2 B have an ethernet port. You also have the option of adding a WiFi Adapter/Dongle which may mean that you need a USB Hub if you have run out of USB ports. The Pi 3 already has 802.11 b/g/n wireless LAN and Bluetooth 4.1 (Bluetooth Classic and Low Energy).

Powering my Pi

The Pi has a 5 V microUSB power socket, located on the bottom left hand corner of your Pi board.

Version	Recommended Power Supply Current Capacity
Pi B	1.2 A
Pi A+	700 mA
Pi B+	1.8 A
Pi 2 B	1.8 A
Pi 3 B	2.5 A

Generally, the more USB ports and interfaces you use on your Pi, the more power you are going to need - be careful.

We advise to look at buying a powered USB hub - this means less pressure on your Pi whilst still being able to incorporate all the features and functionality that you want to. When connecting any devices to your Pi, it is advisable to always check the power rating.

Batteries are not a recommended power supply for your Pi.

Note: The Official Raspberry Power Supply Unit for Pi 3 is not a general purpose power supply and must only be used for the Pi 3.

What is the user name and password for the Raspberry Pi?

The user name for Raspbian is **pi**

The password for Raspbian is **raspberry**

Operating Systems, Programming Languages & SD Cards

You will need an **operating system** to start using your Pi. An operating system is vital software that acts as a computer manager.

To download an operating system you will need an **SD card** between 8-32 GB. We have SD cards with New Out Of Box Software (NOOBS) pre-installed, so you don't have to do all of the work. NOOBS helps you to set up your Pi and has six operating systems that you can download;

Raspbian (recommended)	**Pidora**	**OpenElec**	**Windows 10 IoT**
RaspBMC	**RISC OS**	**Arch Linux**	

Of course, you don't have to use NOOBS. The Raspberry Pi Foundation regularly updates other available 'distros' in the downloads section of their website.

The Raspberry Pi Foundation - Downloads

Python — Python is the recommended **programming language** — particularly if you are new to programming or want to refresh your programming knowledge.

Scratch — Scratch is a great interactive programming language for children who want to learn to code through creating games, stories and animations.

Other programming languages you can get on your Pi include C, C++, Java and Ruby.

What educational material/resources can I use?

There is so much information out there to support you with Raspberry Pi due to it's collaborative nature.

DESIGNSPARK

Here at RS, we recommend DesignSpark, our own support gateway filled with blogs, forums, useful tools, product reviews and much more. You can also let us know how you get on with your projects.

Visit DesignSpark

We have a range of Raspberry Pi support books, written by Pi experts such as it's co-founder Eben Upton and Carrie Philbin.

See our Range of Books

Other great Pi resources

Raspberry Pi Foundation

MagPi - The official Pi magazine

Piweekly - Newsletter you can subscribe to

The Raspberry Pi Guy

geekgurldiaries

Not answered your query?

DesignSpark or The Raspberry Pi Foundation website may be able to help you further.

Visit DesignSpark

Visit The Raspberry Pi Foundation website

T5875DV

Raspberry Pi Power Supply

Features:

- Official Raspberry Pi Power Supply
- 1.5M Micro USB B Lead
- ErP Level 6 Efficiency Rating
- 50,000 Hour MTBF
- 1 Years Warranty

Output

Output Voltage	+5.1Vdc
Minimum Load Current	0A
Nominal Load Current	2.5A
Nominal Output Power	13W
Output Regulation	+/-5%
Line Regulation	+/-2%
Ripple & Noise	120mVp-p Maximum
Rise Time	100mS Maximum at nominal input
Turn-on Delay	3 Seconds Maximum at nominal input
Protection	Short circuit, over current, over voltage
Efficiency	80.86%
Output Cable	1500mm Micro USB B 5 Pin

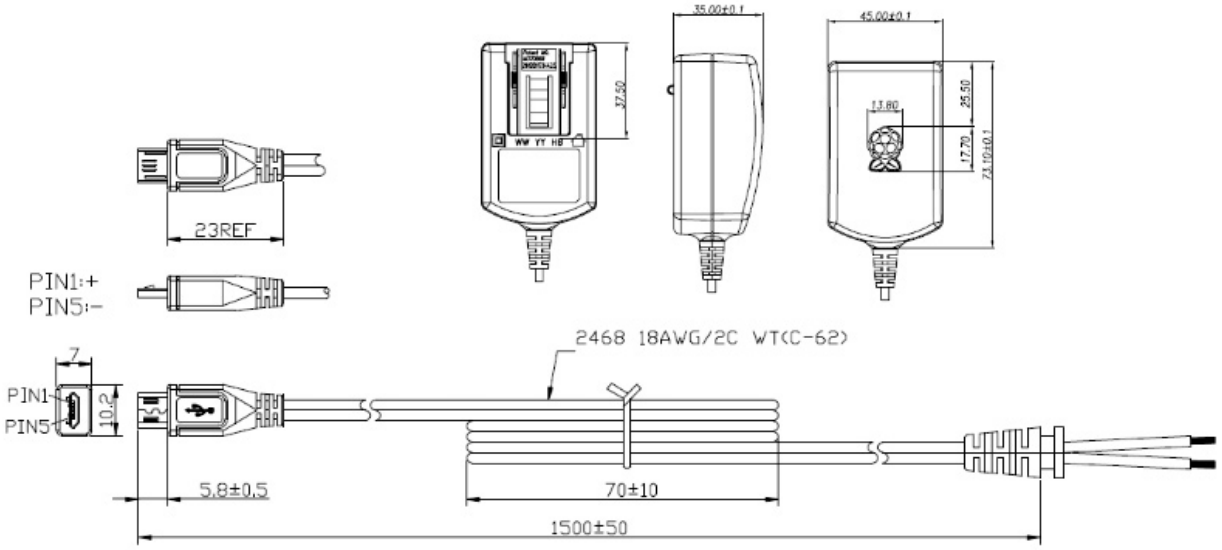
Input

Input Voltage Range	90-264VAC
Input Frequency	47-63Hz
Input Current	0.5A Max
Inrush Current	No damage and IP fuse will not blow
AC Inlet	UK, Euro, Aus & US changeable heads

Other

Dimensions	73.2 (L) * 45.1 (W) * 35.1 (H) mm
Weight	Approx 150g
Operating Temp.	0 °C to 40 °C
Storage Temp.	-20 °C to +60 °C
Opertating Humidity	20 ~ 85 % RH. Non-Condesning
MTBF	50,000 Hours

Diagrams



STONTRONICS
THE POWER TO SUPPLY

Chancerygate Business Centre, Cradock Road, Reading, Berkshire, RG2 0AH.

Tel: +44 (0) 118 931 1199 • Fax: +44 (0) 118 931 1145 • Email: info@stontronics.co.uk • Web: www.stontronics.co.uk

```
# -*- coding: utf-8 -*-
"""ENTRENAMIENTO_YOLO_DETECCION_PERSONAS_VEHICULOS.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1GarqelswhYmp7_Z53S-72sHTjLekPWtf
"""

#IMPORTACIÃ N DE LIBRERIAS

import torch
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os

#CARGAR DRIVE PARA PODER ACCEDER A LOS FICHEROS DE IMAGENES Y ETIQUETAS

from google.colab import drive
drive.mount('/content/gdrive')

#CARGAMOS ARCHIVOS .ZIP DE IMAGENES Y ETIQUETAS

path = '/content/gdrive/MyDrive/YOLO_TFG/'

path_img = path + 'images_TFG.zip'
path_lab = path + 'labels_TFG.zip'

#SE COPIAN LOS ARCHIVOS PARA TRABAJAR CON ELLOS EN EL CODIGO

!cp {path_img} {'/content/'}
!cp {path_lab} {'/content/'}

#DESCOMPRIMIMOS LOS ARCHIVOS

!unzip -qq images_TFG.zip
!unzip -qq labels_TFG.zip

#CREARMOS DIRECTORIOS PARA ALMACENAR LAS ETIQUETAS E IMAGENES

!mkdir /content/labels_TFG
!mkdir /content/images_TFG

#DEFINIMOS UNA FUNCIÃ N PARA MOVER TODOS LOS ARCHIVOS RELEVANTES A LOS DIRECTORIOS CREADOS

def move_fich(path, destino, term):

 content = os.listdir(path)

 for k in content:

  if os.path.isfile(os.path.join(path, k)) and k.endswith(term):

    !mv {path + k} {destino}

#MOVEMOS LOS FICHEROS .TXT (ETIQUETAS)

move_fich('/content/', '/content/labels_TFG/', '.txt')

#COMPROBAMOS QUE SE REALIZO CORRECTAMENTE --> SON UN TOTAL DE 300 ETIQUETAS

pfile = os.listdir('/content/labels_TFG/')

print(len(pfile))

#MOVEMOS LOS FICHEROS .JPG (IMAGENES)

move_fich('/content/images/', '/content/images_TFG/', '.jpg')

#comprobamos que se realizÃ³ correctamente

pifile = os.listdir('/content/images_TFG/')

print(len(pifile))

#CREACION DEL ARBOL DE DIRECTORIOS NECESARIOS PARA EL ENTRENAMIENTO

!mkdir /content/data
!mkdir /content/data/images
!mkdir /content/data/images/train
!mkdir /content/data/images/test
!mkdir /content/data/images/val
!mkdir /content/data/labels
!mkdir /content/data/labels/train
!mkdir /content/data/labels/test
!mkdir /content/data/labels/val

#CREACION DE UNA FUNCIÃ N PARA DISTRIBUIR EL CONJUNTO DE DATOS EN TRAIN/TEST/VAL


def fichero_ordenar(content, path, path_destino1, path_destino2, path_destino3, term):

  content = sorted(content)

  p_train = 0.7
```

```
  p_val = 0.1
  p_test = 0.2

  n_train = round(len(content) * p_train)
  n_val = round(len(content) * p_val)
  n_test = round(len(content) * p_test)

  print('cantidad del set de entrenamiento: ', n_train, '\n')
  print('cantidad del set de validaciÃ³n: ', n_val, '\n')
  print('cantidad del set de prueba: ', n_test, '\n')

  for i in range(len(content)):

    if i <= n_train and content[i].endswith(term):

      !mv {path + content[i]} {path_destino1}

    elif i > n_train and i <= n_train + n_val and content[i].endswith(term):

      !mv {path + content[i]} {path_destino2}

    elif i > n_train + n_val and content[i].endswith(term):

      !mv {path + content[i]} {path_destino3}

#ORDENAMOS LOS DATOS DE IMAGENES Y ETIQUETAS UTILIZANDO LA FUNCION PREVIAMENTE DEFINIDA

fich_img = os.listdir('/content/images_TFG/')

fich_lab = os.listdir('/content/labels_TFG/')

#Para imagenes

fichero_ordenar(fich_img, '/content/images_TFG/', '/content/data/images/train/', '/content/data/images/val/', '/content/data/images/test/', '.jpg')

#Para etiquetas

fichero_ordenar(fich_lab, '/content/labels_TFG/', '/content/data/labels/train/', '/content/data/labels/val/', '/content/data/labels/test/', '.txt')

#COMPROBAMOS QUE EL PROCESO SE HIZO CORRECTAMENTE PARA IMAGENES

destino1 = '/content/data/images/train/'
destino2 = '/content/data/images/val/'
destino3 = '/content/data/images/test/'

content_train = os.listdir(destino1)
content_val = os.listdir(destino2)
content_test = os.listdir(destino3)

print(len(content_train))
print(len(content_val))
print(len(content_test))

#COMPROBAMOS TAMBIE PARA ETIQUETAS

destino1 = '/content/data/labels/train/'
destino2 = '/content/data/labels/val/'
destino3 = '/content/data/labels/test/'

fichero1 = os.listdir(destino1)
fichero2 = os.listdir(destino2)
fichero3 = os.listdir(destino3)


print(len(fichero1), 'train')
print(len(fichero2), 'val')
print(len(fichero3), 'test')

#CREAMOS ARCHIVO .YAML INDICANDO LAS CLASES A DETECTAR (vehÃ culo y persona)
path_train = '/content/data/images/train/'
path_test  = '/content/data/images/test/'
path_val   = '/content/data/images/val/'

nro_clases = 2

nombre = ['vehiculo', 'persona']

f = open('/content/data/data.yaml', 'w+')
f.write('train: ' + path_train + '\n')
f.write('test:  ' + path_test  + '\n')
f.write('val:   ' + path_val   + '\n')

f.write('nc: ' + str(nro_clases) + '\n')
f.write('names: ' + str(nombre)  + '\n')

f.close()

#COMPROBAMOS LA CREACION DEL FICHERO

!cat /content/data/data.yaml

#INSTALACIÃ N DEL ALGORITMO YOLO by ULTRALITICS

!pip install -qr https://raw.githubusercontent.com/ultralytics/yolov5/master/requirements.txt

!git clone https://github.com/ultralytics/yolov5
```

```
!pip install -qr yolov5/requirements.txt

# Commented out IPython magic to ensure Python compatibility.
#COMPROBAMOS LA INSTALACION

# %cd yolov5

#DEFINICIONES PARA EL ENTRENAMIENTO

YOLO_MODEL = 'yolov5s' #versiÃ³n del modelo a emplear (small)
EPOCHS = 200
CFG = YOLO_MODEL + '.yaml'
TRAIN_WEIGHTS = YOLO_MODEL + '.pt' #pesos utilizados para el entrenamiento
BATCH = 8
IMG_SIZE = 256
YAMLFILE = '/content/data/data.yaml'

#ENTRENAMOS EL ALGORITMO PARA DETECCION

!python train.py --img {IMG_SIZE} --batch {BATCH} --epochs {EPOCHS} --data {YAMLFILE} --cfg {CFG} --weights {TRAIN_WEIGHTS} --nosave --cache

#REALIZAMOS EL PROCESO DE TESTEO

PESOS_TEST = '/content/yolov5/runs/train/exp/weights/last.pt'
PATH_TEST = '/content/data/images/test/'

!python detect.py --weights {PESOS_TEST} --img {IMG_SIZE} --conf 0.4  --source {PATH_TEST}

#LOS RESULTADOS SE GUARDARON EN runs/detect/exp/. Ahora se va a mostrar una imagen para ver lo obtenido

img_prueba = cv2.imread('/content/yolov5/runs/detect/exp/mnc2.jpg')
plt.figure(figsize=(20,20))
plt.imshow(cv2.cvtColor(img_prueba, cv2.COLOR_BGR2RGB))
plt.show()

img_prueba = cv2.imread('/content/yolov5/runs/detect/exp/traffic-light-car-copy.jpg')
plt.figure(figsize=(20,20))
plt.imshow(cv2.cvtColor(img_prueba, cv2.COLOR_BGR2RGB))
plt.show()

img_prueba = cv2.imread('/content/yolov5/runs/detect/exp/c6s2_123943.jpg')
plt.figure(figsize=(20,20))
plt.imshow(cv2.cvtColor(img_prueba, cv2.COLOR_BGR2RGB))
plt.show()

img_prueba = cv2.imread('/content/yolov5/runs/detect/exp/new-york.jpg')
plt.figure(figsize=(20,20))
plt.imshow(cv2.cvtColor(img_prueba, cv2.COLOR_BGR2RGB))
plt.show()

#PROCESO DE VALIDACION

!python val.py --weights {PESOS_TEST} --data {YAMLFILE} --img {IMG_SIZE} --iou 0.25

#LOS RESULTADOS DE VALIDACION PUEDEN OBSERVARSE POR MEDIO DE LAS CURVAS DE VALIDACION

curva_val = cv2.imread('/content/yolov5/runs/val/exp/PR_curve.png')
plt.figure(figsize=(20,20))
plt.imshow(curva_val)
plt.show()

#PODEMOS OBTENER LAS ESTADÃ STICAS EN EL TESTING

!python val.py --weights {PESOS_TEST} --data {YAMLFILE} --img {IMG_SIZE} --iou 0.25 --task test

curva_test = cv2.imread('/content/yolov5/runs/val/exp2/PR_curve.png')
plt.figure(figsize=(20,20))
plt.imshow(curva_test)
plt.show()

#GUARDAREMOS LOS PESOS PARA POSTERIOR USO

from google.colab import files
files.download('/content/yolov5/runs/train/exp/weights/last.pt')

#PRUEBA DE DETECCION CON OTRAS IMAGENES

img_p1 = cv2.imread('/content/gdrive/MyDrive/YOLO_TFG/1.jpeg')
img_p2 = cv2.imread('/content/gdrive/MyDrive/YOLO_TFG/2.jpg')
img_p3 = cv2.imread('/content/gdrive/MyDrive/YOLO_TFG/3.jpg')

plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(img_p1, cv2.COLOR_BGR2RGB))
plt.show()

plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(img_p2, cv2.COLOR_BGR2RGB))
plt.show()

plt.figure(figsize=(10,10))
plt.imshow(cv2.cvtColor(img_p3, cv2.COLOR_BGR2RGB))
plt.show()

#CARGAMOS EL MODELO A TRAVÃ S DE PYTORCH HUB
```

```
model = torch.hub.load('ultralytics/yolov5', 'custom', path = '/content/yolov5/runs/train/exp/weights/last.pt')

img = cv2.imread('/content/gdrive/MyDrive/YOLO_TFG/2.jpg')

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

detecta = model(img)

detecta.show()

#CARGAMOS EL MODELO A TRAVÃ S DE PYTORCH HUB

model = torch.hub.load('ultralytics/yolov5', 'custom', path = '/content/yolov5/runs/train/exp/weights/last.pt')

img = cv2.imread('/content/gdrive/MyDrive/YOLO_TFG/3.jpg')

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

detecta = model(img)

detecta.show()

#CARGAMOS EL MODELO A TRAVÃ S DE PYTORCH HUB

model = torch.hub.load('ultralytics/yolov5', 'custom', path = '/content/yolov5/runs/train/exp/weights/last.pt')

img = cv2.imread('/content/gdrive/MyDrive/YOLO_TFG/1.jpeg')

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

detecta = model(img)

detecta.show()

#REFERENCIAS

@software{yolov5,
  title = {YOLOv5 by Ultralytics},
  author = {Glenn Jocher},
  year = {2020},
  version = {7.0},
  license = {AGPL-3.0},
  url = {https://github.com/ultralytics/yolov5},
  doi = {10.5281/zenodo.3908559},
  orcid = {0000-0001-5950-6979}
}
```

```
import torch
import cv2
import matplotlib.pyplot as plt
import numpy as np
import time


model = torch.hub.load('ultralytics/yolov5', 'custom', path='C:/Users/HP/Desktop/PESOS_TFG.pt')



cap = cv2.VideoCapture('C:/Users/HP/Desktop/video3_car.mp4')

ret, frame = cap.read()

h,w,c = frame.shape

forucc = cv2.VideoWriter_fourcc(*'mp4v')

out = cv2.VideoWriter('output3.mp4', forucc, 30, (w,h))


while True:

    ret, frame = cap.read()

    if not ret:
        print('Error en la apertura del video o camara')
        break


    result = model(frame)
    out.write(np.squeeze(result.render()))

    t = cv2.waitKey(27)
    if t == 27:
        break

print('end of process')
cap.release()
cv2.destroyAllWindows()
```

```
import cv2
import matplotlib.pyplot as plt
from picamera import PiCamera
import time
import os

#-------------------------------------------------------------------------------------------------------------------------------------------------------------

#CAPTURA Y GRABACION

#-------------------------------------------------------------------------------------------------------------------------------------------------------------

cam = PiCamera()

tarea = int(input('Indicar acciÃ³n a realizar: 1 (capturas periÃ³dicas), 2 (grabacion continua): '))

cam.resolution = (320, 240)
cam.framerate = 30
cam.start_preview()

if tarea == 1:
 i = 0
 while True:
  cam.capture('/home/pi/Desktop/YOLO_TFG/capturas_cam/img'+str(i)+'.jpg', resize = (640, 480))
  time.sleep(2)
  i += 1
  print(i)
  if i == 10:
   break



elif tarea == 2:

 t = int(input('Indicar duracion video: '))

 cam.start_recording('/home/pi/Desktop/YOLO_TFG/video_cam/video.h264')
 cam.wait_recording(t)
 cam.stop_recording()

else:
 print('error argumento no valido')

print('end process')
cam.close()

#-------------------------------------------------------------------------------------------------------------------------------------------------------------

#ENVIO VIDEO A GOOGLE DRIVE

#-------------------------------------------------------------------------------------------------------------------------------------------------------------

from pydrive.auth  import GoogleAugth
form pydrive.drive import GoogleDrive


gauth = GoogleAuth()
drive = GoogleDrive(gauth)


id_destino = '1El_OSLvCaT9WlVJ7wxaFj5xkPEy79yw9'

if tarea == 1:

 carpeta_origen = '/raspberry/pi/Desktop/YOLO_TFG/captura_cam'

elif tarea == 2:

 carpeta_origen = '/raspberry/pi/Desktop/YOLO_TFG/video_cam'



filename = os.listdir(carpeta_origen)

for i in filename:

 f_origen = os.path.join(carpeta_origen, i)
 googledrive = drive.CreateFile({'parents' : [{'id' : id_destino], 'title' : i})
 googledrive.SetContentFile(f_origen)
 googledrive.Upload()


#-------------------------------------------------------------------------------------------------------------------------------------------------------------
```