Trabajo de Fin de Grado en Ingeniería de las Tecnologías de la Telecomunicación

Viabilidad y rendimiento de YOLOv5 en Raspberry Pi 4 modelo B

Autor: Luis Muñiz García

Tutor: Antonio Jesús Sierra Collado

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Final de Grado
Ingeniería de Telecomunicación

Viabilidad y rendimiento de YOLOv5 en Raspberry Pi

Autor:
Luis Muñiz García

Tutor:
Antonio Jesús Sierra Collado
Profesor

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Grado: Viabilidad y rendimiento de YOLOv5 en Raspberry Pi

Autor: Luis Muñiz García

Tutor: Antonio Jesús Sierra Collado

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis amigos

A mis maestros

Agradecimientos

Muchas son las personas que me han inspirado y apoyado a lo largo de mi estancia en la universidad, y podría escribir otro documento de la misma extensión nombrándolos a todos, y aún así se que no les haría suficiente justicia.

He tenido la fortuna de encontrarme con unos compañeros de carrera, como José, Fran, Juan, Natalia, Edu, Nacho, Carmen, Andrés, Ferchu, o Mendoza, por decir tan sólo unos pocos, con los que he compartido muchas experiencias y que me han demostrado continuamente su integridad y profesionalidad, y ojalá nuestros caminos se crucen el día de mañana laboralmente.

He tenido la suerte de tener a mi lado a mis amigos, a mi Consejo de Sabios, con los que compartí cada triunfo y cada derrota, académica y personalmente, a los que a día de hoy les estaré eternamente agradecidos por todo su apoyo y afecto.

Por supuesto, he tenido la presencia y el cariño de mi familia, cuyo amor incondicional y fe en mi han sido inestimables para haber llegado hasta aquí.

Pero, además, hay una persona a la que me gustaría agradecérselo todo: a mi madre. Tú has sido la piedra angular sobre la que he edificado la persona que soy a día de hoy. Has sido mi inspiración, mi refugio en la tormenta, la que me dio la vida y la que dio su último aliento por el bienestar de sus hijos. Nada me hubiera gustado más que haber compartido este momento contigo, y todos los que aún están por llegar.

Descansa en paz, donde quieras que estés.

Luis Muñiz García

Sevilla, 2021

Resumen

Cada año, la Dirección General de Tráfico emprende campañas de concienciación para mitigar el número de accidentes en vías urbanas e interurbanas. Algunos de estos siniestros son fruto de la imprudencia al volante, o sencillamente de no prestar suficiente atención a la hora de conducir. Teniendo esto en mente, los fabricantes de coches están introduciendo las Inteligencias Artificiales en sus vehículos para asistir en la conducción al piloto, tomando el control parcial o total sobre los actuadores electromecánicos e incluso avisando verbalmente al conductor sobre alguna posible infracción, con objeto de aportar un plus de seguridad y calidad como servicios adicionales en el producto final. Por supuesto, estas Inteligencias Artificiales se comercializan como un sistema embebido dentro del propio coche, por lo que se plantea la posibilidad de elaborar un sistema modular utilizando ordenadores de placa reducida para lograr este fin, y así poder aportar esa seguridad a vehículos que anteriormente no disponían de este servicio. Con todo esto presente, en este proyecto se abordan todos los aspectos y fundamentos teóricos sobre las Inteligencias Artificiales aplicadas a la visión artificial, dando un enfoque ascendente y explicando en profundidad su naturaleza y funcionamiento.

También se ha desarrollado una aplicación en Python para la detección de imágenes mediante el uso de redes neuronales convolucionales enfocadas a la visión artificial. Dicha aplicación consta de un módulo para de un asistente de voz que enuncia los resultados de la detección utilizando YOLOv5, ofreciendo una interfaz gráfica para la interacción con el usuario.

Además, se exponen las características esenciales del sistema de código abierto de YOLOv5 para la detección de imágenes mediante el uso de redes neuronales convolucionales enfocadas a la visión artificial, explicando de forma somera el funcionamiento y la utilidad de sus dependencias.

Se han planteado dos entornos de entrenamiento (la nube de Google Colab y la propia Raspberry Pi 4B) para tener un contraste a la hora de analizar su viabilidad en términos de duración, calidad del modelo de inferencia y privacidad de los datos. Ambos entornos utilizarán el mismo conjunto de imágenes para realizar su entrenamiento, el cuál ha sido generado utilizando el *framework* de roboflow para la clasificación y preprocesamiento de las imágenes.

En última instancia, se exponen las conclusiones en base a los resultados obtenidos en cada fase de entrenamiento, analizando las métricas, exponiendo las fortalezas y debilidades de cada entorno escogido, y se proponen soluciones ante los problemas encontrados en ambos entornos, así como una posible línea de continuación implementando físicamente el dispositivo de asistencia a la conducción.

Abstract

Every year, the spanish Directorate General of Traffic (Dirección General de Tráfico) undertakes awareness campaigns to mitigate the number of accidents on urban and interurban roads. Some of these accidents are the result of reckless driving, or simply not paying enough attention while driving. With this in mind, car manufacturers are introducing Artificial Intelligences in their vehicles to assist the driver in driving, taking partial or total control over the electromechanical actuators and even verbally warning the driver about a possible infraction, in order to provide extra safety and quality as additional services in the final product. Of course, these Artificial Intelligences are marketed as an embedded system within the car itself, so the possibility of developing a modular system using small-board computers to achieve this end is being considered, and thus be able to provide this safety to vehicles that previously did not have this service. With all this in mind, this project addresses all the aspects and theoretical foundations of Artificial Intelligences applied to artificial vision, taking a bottom-up approach and explaining in depth their nature and how they work.

A Python application has also been developed for image detection using convolutional neural networks focused on artificial vision. This application consists of a module for a voice assistant that announces the results of the detection using YOLOv5, offering a graphical interface for interaction with the user.

In addition, the essential features of the YOLOv5 open source system for image detection using convolutional neural networks focused on computer vision are presented, explaining briefly the operation and usefulness of its dependencies.

Two training environments (the Google Colab cloud and the Raspberry Pi 4B itself) have been proposed in order to have a contrast when analysing their viability in terms of duration, quality of the inference model and data privacy. Both environments will use the same set of images for training, which has been generated using the roboflow framework for image classification and preprocessing.

Finally, conclusions are drawn based on the results obtained in each training phase, analysing the metrics, exposing the strengths and weaknesses of each chosen environment, and proposing solutions to the problems encountered in both environments, as well as a possible line of continuation by physically implementing the driving assistant device.

Índice

Agradecimientos ix
Resumen xi
Abstract xiii
Índice xiv
Índice de Tablas xvi
Índice de Ilustraciones xvi
Notación xviii
1 Introducción 1
1.1 Prefacio *1*
1.2 Motivación *2*
1.3 Antecedentes *3*
1.4 Objetivos *4*
1.5 Estructura *4*
2 Fundamento Teórico 7
2.1 El proceso de aprendizaje *7*
2.1.1 Analogía biológica 7
2.1.2 Redes Neuronales 11
2.1.3 Entrenamiento 14
3 Dependencias de YOLOv5 19
3.1 Dependencias *19*
3.1.1 COCO Datasets 19
3.1.2 Matplotlib 19
3.1.3 Numpy 20
3.1.4 OpenCV-Python 20
3.1.5 Pillow 20
3.1.6 PyYAML 20
3.1.7 SciPy 20
3.1.8 Torch 20
3.1.9 Torch-vision 20
3.1.10 Tqdm 20
3.1.11 Tensor Board 21
3.1.12 Seaborn 21
3.1.13 Pandas 21

3.1.14 Thop 21
3.1.15 Pycocotools 21
3.2 Relación y arquitectura de los componentes *21*
4 Aplicación 22
4.1 Etimología *23*
4.2 Estructura y componentes *23*
4.3 Clases y relaciones *24*
4.4 Ejemplo de ejecución *26*
5 Entrenamiento, Pruebas y Validación 30
5.1 Entrenamiento en la nube (Colaboratory-Google) *32*
5.1.1 Entorno y herramientas 33
5.1.2 Colección de imágenes de muestras 34
5.1.3 Entrenamiento del modelo 36
5.1.4 Inferencia 38
5.1.5 Resultados 39
5.2 Entrenamiento en Raspberry Pi 4B *41*
5.2.1 Entorno y herramientas 41
5.2.2 Colección de imágenes de muestras 41
5.2.3 Entrenamiento del modelo 41
5.2.4 Inferencia 43
5.2.5 Resultados 44
5.3 Comparación de los entrenamientos en ambos entornos. *45*
6 Conclusiones 11
Anexo A: Pasos para la instalación de Yolov5 en Raspberry Pi 4 B 13
Anexo B: Pasos para la configuración y el entrenamiento de la nube de Google Colab 13
Anexo C: Estructura de una anotación en formato XML 17
Anexo D: Estructura de una anotación en formato TXT 18
Anexo E: Código fuente: interfaz.py 19
Anexo F: Código fuente: ventana.py 23
Anexo G: Código fuente: detecta_yolov.py 28
Anexo H: Código fuente: assistant.py 29
Referencias 30

ÍNDICE DE ECUACIONES

Ecuación 1: Salida de un perceptrón 10
Ecuación 2: Variación de pesos en supervisión por corrección de error 13
Ecuación 3: Variación de los pesos en aprendizaje supervisado por refuerzo 14
Ecuación 4: Variación de los pesos en aprendizaje Hebbiano 15
Ecuación 5: Precisión del modelo 32
Ecuación 6: Exhaustividad del modelo 32

ÍNDICE DE TABLAS

Tabla 1: Prestaciones de los arquetipos de YOLOv5 31
Tabla 2 : Ensayo-Error con el entrenamiento por mini-batches 42
Tabla 3: Comparación de eficiencia de ambos entornos 45
Tabla 4: Comparativa de métricas en ambos entornos 46

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Ejemplo de reconocimiento de objetos usando visión artificial 2
Ilustración 2: Histograma con la clasificación de la mortandad 2018-2019 3
Ilustración 3: Esquema básico de una neurona humana 7
Ilustración 4: Modelo de caja negra de una neurona 8
Ilustración 5: Sinapsis entre dos neuronas. 8
Ilustración 6: Esquema conceptual del Perceptrón 9
Ilustración 7: Función Escalón Unitario 10
Ilustración 8: Diagrama de Estados función ReLU 10
Ilustración 9: Gráfica función ReLU 11
Ilustración 10: Tipos de arquitecturas de Redes Neuronales 11
Ilustración 11: Ejemplo de Red Neuronal Multicapa 12
Ilustración 12: Ejemplo de Arquitectura Radial 12
Ilustración 13: Ejemplo de arquitectura neuronal recurrente 13
Ilustración 14: Ejemplo de Red Neuronal Convolucional 13
Ilustración 15: Ejemplo de entrenamiento a nivel teórico 14
Ilustración 16: Ejemplo de Realimentación Negativa 15
Ilustración 17: Esquema concepetual de un sistema de corrección de error 15
Ilustración 18: Red Neuronal basada en la máquina de Boltzmann. 16
Ilustración 19: Ejemplo de red monocapa con aprendizaje Hebbiano 17
Ilustración 20: Tipos de aprendizaje 18
Ilustración 21: Ejemplo de barra de carga de tqdm 20
Ilustración 22: Gráficas generadas en la fase de entrenamiento usando seaborn 21
Ilustración 23: Esquema piramidal de la relación entre los componentes 22
Ilustración 24: Representación de Argos Panoptes, de Bernardino di Betto di Biagio. 23

Ilustración 25: Diagrama de Componentes de A.R.G.O.S. 24
Ilustración 26: Diagrama Clases de A.R.G.O.S. 25
Ilustración 27: Diagrama de Caso de Uso 26
Ilustración 28: Ventana principal de A.R.G.O.S. 27
Ilustración 29: Diálogo con el reacondicionamiento del entorno 28
Ilustración 30: intento de reacondicionamiento ya consolidado 28
Ilustración 31: Ventana del test de detección 29
Ilustración 32: Proceso de inferencia. Perspectiva del símbolo del sistema 29
Ilustración 33: Imagen de prueba. Paso de peatones. 30
Ilustración 34: Contraste entre prueba y detección 30
Ilustración 35: Detección de semáforo 30
Ilustración 36: Detección de límite de velocidad. 30
Ilustración 37: Arquetipos de YOLOv5 31
Ilustración 38: Escenario y componentes de Google Colab 32
Ilustración 39: Interfaz web de *Google Colab.* 33
Ilustración 40: Ratio de distribución de las imágenes del *dataset* 34
Ilustración 41: Interfaz web de *roboflow* 34
Ilustración 42: Dataset para la generación del modelo. 35
Ilustración 43: Clasificación de imágenes en roboflow 35
Ilustración 44: Exportación del *dataset* vía web 36
Ilustración 45: Resultados del entrenamiento y tiempo necesario (*Google Colab*) 37
Ilustración 46: Imagen *val_batch0_pred.jpg* 38
Ilustración 47: Imagen de prueba para la inferencia (original) 39
Ilustración 48: Imagen de prueba para la inferencia (detectado) 39
Ilustración 49: mAP 0.5 (*Google Colab*) 40
Ilustración 50: mAP 0.5:0.95 (*Google Colab*) 40
Ilustración 51: Precisión (*Google Colab*) 40
Ilustración 52: Exhaustividad (*Google Colab*) 40
Ilustración 53: Estrechamiento del recuadro de captura en la predicción (*Google Colab*) 40
Ilustración 54: Error en la detección de clases (*Google Colab*) 40
Ilustración 55: Error en la detección de objetos (Google Colab) 40
Ilustración 56: Finalización del entrenamiento (Raspberry Pi 4B) 42
Ilustración 57: Terminal de la detección en Raspberry pi 4B 43
Ilustración 58: Detección de señal de STOP (Raspberry Pi 4B) 43
Ilustración 59: Detección de límite de velocidad (Raspberry Pi 4B) 43
Ilustración 60: Detección de paso de peatones (Raspberry Pi 4B) 43
Ilustración 61: Detección múltiple (Raspberry Pi 4B) 43
Ilustración 62: mAP 0.5 (Raspberry Pi 4B) 44
Ilustración 63: mAP 0.5:0.95 (Raspberry Pi 4B) 44

Ilustración 64: Precisión (Raspberry Pi 4B) 44
Ilustración 65: Exhaustividad (Raspberry Pi 4B) 44
Ilustración 66: Estrechamiento del recuadro de captura en la predicción (Raspberry Pi 4B) 44
Ilustración 67: Error en la detección de clases (Raspberry Pi 4B) 44
Ilustración 68: Error en la detección de objetos (Raspberry Pi 4B) 44
Ilustración 69: Métricas de Raspberry Pi 4B 46
Ilustración 70: Métricas de *Google Colab* 46
Ilustración 71: Fuente de alimentación UPS ininterrumpible Innovateking-EU Raspberry Pi con baterías 18650 12
Ilustración 72: Archivo *road3.xml* 18
Ilustración 73: Etiquetado automático de *road3.png* en *roboflow* 18
Ilustración 74: Archivo *road3.txt* 19
Ilustración 75: Imagen *road3.png* clasificada 19

Notación

X_i	Entrada 'i' de la red neuronal
P_i	Peso 'i' asociado a la entrada homónima
θ	Sesgo
Y	Salida de la red neuronal
$\Delta(pesos_{i,j})$	Corrección de los pesos i, j
y_i	Salida de la neurona i (entrada)
y_j	Salida obtenida de la neurona j (salida)
d_i	Salida deseada
β	Factor de aprendizaje
VP	Verdadero Positivo
FP	Falso Positivo
VN	Verdadero Negativo
FN	Falso Negativo

1 INTRODUCCIÓN

La mitad está hecha cuando tienen buen principio las cosas.

- Fernando de Rojas -

Esta sección sirve como antesala para presentar el proyecto *Viabilidad y rendimiento de Tiny Yolov5 en Raspberry Pi 4 modelo B.* Comenzaremos con un breve prefacio con el que daremos pie al estado del arte. Veremos todos los antecedentes tecnológicos y las posibilidades que nos brinda actualmente la tecnología.

1.1 Prefacio

Con permiso del lector, me atrevo a definir con mis propias palabras al ingeniero: (del latín «ingeniun») *aquella persona cualificada que pone al servicio de la sociedad sus conocimientos técnicos, en pos de mejorarla.* Sus labores pueden abarcar la resolución de problemas, o la anticipación de los mismos, sobrepasando las limitaciones físicas del ser humano y llevando a su especie al siguiente paso evolutivo, tecnológicamente hablando.

A día de hoy, pocos son los ámbitos que no contemplan el uso de la palabra "ingeniería" para describir un proceso metodológico y exhaustivo con el que ha buscado optimizar un asunto concreto. Así pues, no se nos hace extraño escuchar en los medios de comunicación que un determinado empresario o una determinada corporación ha hecho uso de la *"Ingeniería Fiscal"* para poder maximizar los beneficios. No obstante, a pesar de todas las ópticas y carices con los que podríamos contemplar esta palabra, nos referiremos de ahora en adelante al ámbito científico-tecnológico.

El ser humano siempre ha tenido la tendencia de buscar el reflejo de su propia existencia en el entorno que lo engloba: En el arte, la arquitectura, incluso en la astronomía… absolutamente todo gira entorno a nosotros.

Creamos las matemáticas y la física como herramientas lógicas con la que representar de forma simbólica los fenómenos que nos rodean, desembocando en la electrónica e inevitablemente en la robótica como disciplina. Aquí hicimos gala una vez más de nuestra creativa imaginación en un esfuerzo técnico por replicar las características dinámicas del ser humano, y por mejorarlas en los aspectos donde flaqueaban: robustez, tiempo de respuesta, y la capacidad de procesamiento. Aún ignorando el sempiterno misterio de la consciencia y, por ende, de la inteligencia, nos lanzamos sin miedo a la replicación de la misma utilizando las ciencias de la computación moderna.

Así pues, en 1956 John McCarthy acuñó por primera vez el término "Inteligencia Artificial" para referirse a una serie de programas y máquinas con capacidad de cómputo avanzada (para más información, véase la referencia [33]). Ciertamente, la capacidad de diseñar dichas máquinas depende en gran medida de cómo evolucionen dos de las variables de las que depende proporcionalmente: el *Hardware* y el *Software*. De la misma forma que no hay ética sin estética, no puede haber grandes progresos en el software sin que avance de forma pareja el hardware. Irónicamente, los avances del hardware también dependen en gran medida de que haya software de diseño de circuitos lo suficientemente potentes con los que optimizar su desarrollo y producción.

Así pues, encontramos la necesidad de tener ordenadores con gran capacidad de computación. A día de hoy, por fortuna, disponemos de ambos requisitos, y podemos llevar a cabo aplicaciones de la Inteligencia Artifical en multitud de campos de investigación: conducción autónoma, ciencia de datos, traducción e interpretación de idiomas, y por supuesto, visión artificial. Podríamos definir la visión artificial como la disciplina técnica dentro del campo de la Inteligencia Artificial que hace uso de métodos para el procesamiento, análisis y elaboración de modelos de datos con los que categorizar, clasificar e identificar con cierto grado de verosimilitud las imágenes del mundo real.

Existen multitud de estándares con los que podemos trabajar para la elaboración de aplicaciones de visión artificial, y una de ellas es YOLOv5. YOLO, cuyas siglas en inglés son "*You Only Look Once*" (Sólo Miras Una Vez, referencia [34]) es un *framework* de código abierto usado para la detección de objetos en tiempo real. Podríamos encontrar un ejemplo visual de lo que es el reconocimiento de objetos en la siguiente ilustración (para más información, véase la referencia [23]).

Ilustración 1: Ejemplo de reconocimiento de objetos usando visión artificial

El objetivo de este proyecto será analizar el uso de este software dentro de lo que se conoce como una SBC (del inglés, *Single-Board Computer*) en este caso, el de una Raspberry Pi 4 modelo B, con objeto de analizar su rendimiento en términos de duración, calidad, viabilidad y utilidad real para un Caso de Uso sobre seguridad vial en el que entrenemos una red neuronal para el reconocimiento de señales de tráfico.

1.2 Motivación

En el 2020 se registró un total de 797 accidentes mortales, en el cuál fallecieron un total de 870 personas.

Muchos de estos siniestros fueron producidos debido a negligencias por parte de los conductores, o por simple fallo humano.

Si bien el pasado año alcanzamos una cifra récord en lo que a fallecimientos en carretera se refiere a la baja, aún queda un largo camino hasta minimizar drásticamente estos sucesos. Allí donde las campañas de concienciación de seguridad vial no triunfan, lo puede hacer la Inteligencia Artificial.

Otros años hemos obtenido un balance de los tipos de accidentes como se muestra en la siguiente ilustración (fuente en la referencia [22]).

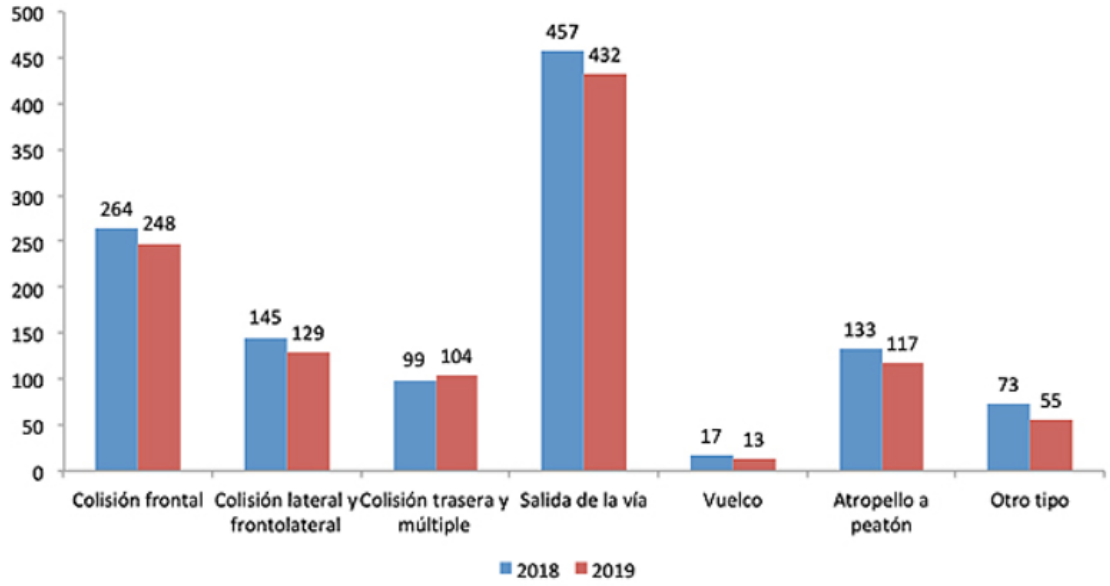


Ilustración 2: Histograma con la clasificación de la mortandad 2018-2019

La explicación puede residir en multitud de factores, entre los cuales podemos incluir la falta de visualización de las señales de tráfico, una conducción temeraria, o un error en la valoración de preferencias en un cruce.

La capacidad de procesamiento de la información cuando circulamos a alta velocidad no está exenta de pertenecer a la categoría de aspectos mejorables del ser humano.

Se propone que, como aplicación de la Inteligencia Artificial, podríamos reducir la mortandad si añadiésemos una capa adicional de procesamiento que apoyase al conductor en su desempeño al volante, reconociendo señales de tráfico mediante algún dispositivo integrado dentro del vehículo.

1.3 Antecedentes

Este proyecto es la continuación de los trabajos de final de carrera de dos compañeros: El de Ángel Moreno Prieto, con "*Detección de Objetos con TinyYOLOv3 sobre Raspberry Pi 3*" (disponible en la referencia [41]), y el de Lucía Reina López, "*Aplicación Android y Servicio Web Spring para la detección y registro de señales de tráfico de velocidad usando Deep Learning con tiny-yolov3 y OpenCV*" (disponible en la referencia [40]). Ambos proyectos utilizan una misma versión de YOLO, YOLOv3.

Partimos del trabajo de Ángel, cuyas conclusiones finales fueron que no era satisfactorio el rendimiento de YOLOv3 en la Raspberry Pi 3. En esta ocasión, probamos con *Hardware* nuevo (usando un modelo superior de Raspberry) y con un *Software* nuevo (la versión 5 de YOLO). Del trabajo de Lucía extraemos el Caso de Uso, el cuál lo llevaremos a cabo en nuestro SBC, además de la fase del entrenamiento en la nube.

Con todo ello, se ha desarrollado una aplicación que hace uso de YOLOv5, usando los modelos entrenados en la nube de Google Colab, y en la Raspberry Pi 4B, creando un asistente de voz que enuncie de viva voz las

señales de tráfico detectadas.

1.4 Objetivos

Para poder elaborar un sistema modular que asista al conductor, y así aportar ese plus de seguridad del que hablábamos en el resumen, nos valdremos de los ordenadores de placa única del fabricante Raspberry Pi, que abogan por un término medio entre sus competidores directos (Arduino y Nvidia) en relación calidad-precio.

El objetivo del presente documento será, por tanto, analizar la viabilidad y el rendimiento (en términos de tiempo de entrenamiento, privacidad y complejidad del entorno) en una Raspberry Pi 4B para el entrenamiento de una red neuronal convolucional aplicada a la visión artificial. Para ello, efectuaremos el entrenamiento en dos entornos:

- En la nube de Google (utilizando *Google Colab*), ya que es un entorno comúnmente usado. Será nuestro sesgo a la hora de comparar.
- En la susodicha Raspberry Pi 4B con la distribución de *Raspbian Buster*, midiendo sus métricas y algunos parámetros de interés como el tiempo de convergencia o el tamaño de Batch.

En adición a todo esto, se pretende realizar una aplicación de escritorio desarrollada en Python que aborde un caso de uso sobre seguridad vial, reconociendo hasta 4 entidades distintas (semáforos, pasos de peatones, señales de STOP y límites de velocidad) implicadas en las travesías urbanas e interurbanas, utilizando para ello el software de YOLOv5 junto con los modelos entrenados en ambos entornos para validar la calidad del entrenamiento.

Para ello, abordaremos en un enfoque ascendente todos los fundamentos teóricos, matemáticos e históricos hasta llegar a las prestaciones técnicas de YOLOv5.

Si bien es cierto que no se abordarán aspectos físicos de implementación, también buscaremos plantear todos los elementos necesarios para instalar un servicio de visión artificial con éxito en dicha placa.

Por último, buscaremos extraer las conclusiones en base a los resultados obtenidos, sopesando su verosimilitud, aportando mejoras ante los posibles inconvenientes encontrados y proponiendo posibles continuaciones en la línea de investigación relativa a este trabajo de fin de grado.

En el siguiente apartado comentaremos la estructura de este documento.

1.5 Estructura

Tras esta introducción, el contenido que el lector encontrará:

- El Fundamento Teórico: Donde se abarcarán los fundamentos más elementales sobre la inteligencia artificial hastallegar a las redes neuronales convolucionales en las que se basa YOLOv5.
- Las Dependencias de YOLOv5: Aquí encontraremos todas las librerías de Python que utiliza YOLOv5 para ofrecer el servicio de Visión Artificial.
- Aplicación: Para la puesta en práctica de las prestaciones del software de YOLOv5, así como justificación del rendimiento en la Raspberry Pi 4 B, se ha desarrollado una aplicación en Python. Se explicará la arquitectura, así como las técnicas de ingeniería del software para llevar a acabo el desarrollo, junto con su funcionamiento paso a paso partiendo de un modelo entrenado en uno de los dos entornos propuestos.
- Entrenamiento, pruebas y validación del modelo entrenado en la nube y el modelo entrenado en la Raspberry Pi 4B, con objeto de contrastar las prestaciones, las fortalezas y las debilidades de ambos entornos de entrenamiento.
- Y finalmente las conclusiones, reuniendo todos los resultados obtenidos durante los capítulos anteriores para enunciar un veredicto sobre la viabilidad del entorno.

Así pues, damos paso a la introducción teórica, donde explicaremos en qué está basado YOLO mediante un enfoque ascendente de las redes neuronales convolucionales.

2 FUNDAMENTO TEÓRICO

El cerebro es el órgano específico de la acción: conoce, delibera, valora y decide.

- Fernando Savater -

En este capítulo abordaremos los aspectos teóricos que sientan la base de las redes neuronales, dando un enfoque ascendente a nivel de componentes, comenzando con una breve analogía biológica y siguiendo con los conceptos más elementales relativos a la inteligencia artificial, para finalmente concluir con la fase del entrenamiento.

2.1 El proceso de aprendizaje

2.1.1 Analogía biológica

Como aventurábamos en la introducción, dentro del proceso de inventiva del ser humano se encuentra inevitablemente el análisis de su entorno, la interiorización de la causa-efecto de los fenómenos observados, la categorización y, si procede, la replicación e implementación de dichas ideas. Este proceso se ha aplicado en el desarrollo tecnológico a lo largo de toda su historia, teniendo un claro ejemplo en el diseño de los aeroplanos, inspirados en las capacidades anatómicas de las aves para alzar el vuelo, o en la invención del automóvil, inspirados en la motricidad de los equinos que durante tantos siglos han acompañado a nuestra especie para facilitar su desplazamiento.

Así mismo, la Inteligencia Artificial debe su base a un intento de replicar la dinámica y estructura celular del cerebro.

Podríamos tomar como unidad mínima cerebral las neuronas.

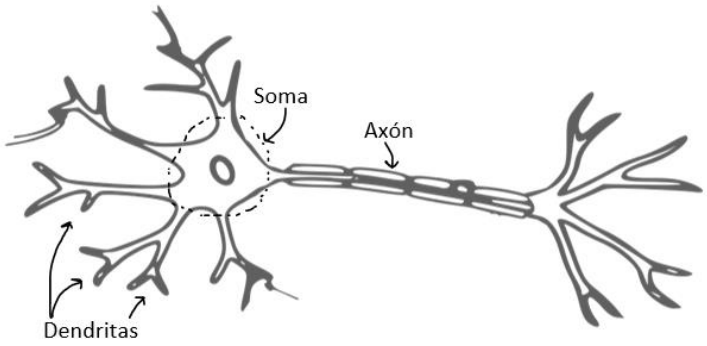


Ilustración 3: Esquema básico de una neurona humana

La imagen anterior está disponible en la referencia [24].

Las neuronas son las entidades encargadas de recibir, procesar y transmitir la información que le llega al cerebro, valiéndose de impulsos eléctricos y mecanismos bioquímicos. A grandes rasgos, posee tres componentes fácilmente diferenciables:

El Soma, núcleo de la célula que contiene todos los orgánulos y componentes necesarios que dan soporte vital

a la célula, además de contener el ADN. Se podría decir que es el motor que pone en funcionamiento la neurona, el cuál posee un disco duro en el que se almacena la información que le proporciona otra célula.

Por otro lado, tendríamos las dendritas, que son las ramificaciones encargadas de recibir los impulsos electroquímicos de las otras células. Aquí podríamos hacer una equivalencia en *hardware* con periféricos de entrada, sensores, por ejemplo.

Por último, tendríamos el axón, que se encargaría de transmitir la información recibida tras ser procesada dentro de la neurona, contribuyendo nuevamente al ciclo de comunicación dentro de la estructura del cerebro.

Así pues, podríamos contemplar una neurona con el siguiente modelo de caja negra.

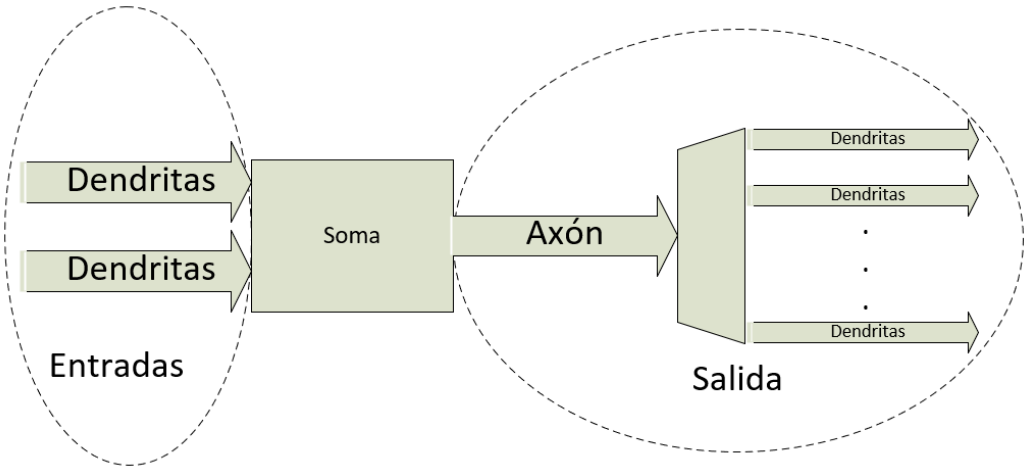


Ilustración 4: Modelo de caja negra de una neurona

La comunicación entre dos o más neuronas se produce en el fenómeno denominado sinapsis, en el cuál, una neurona transmite neurotransmisores a través de su axón a una segunda neurona. Esta última utiliza unos receptores ubicados en el extremo de la dendrita para alojar estas biomoléculas.

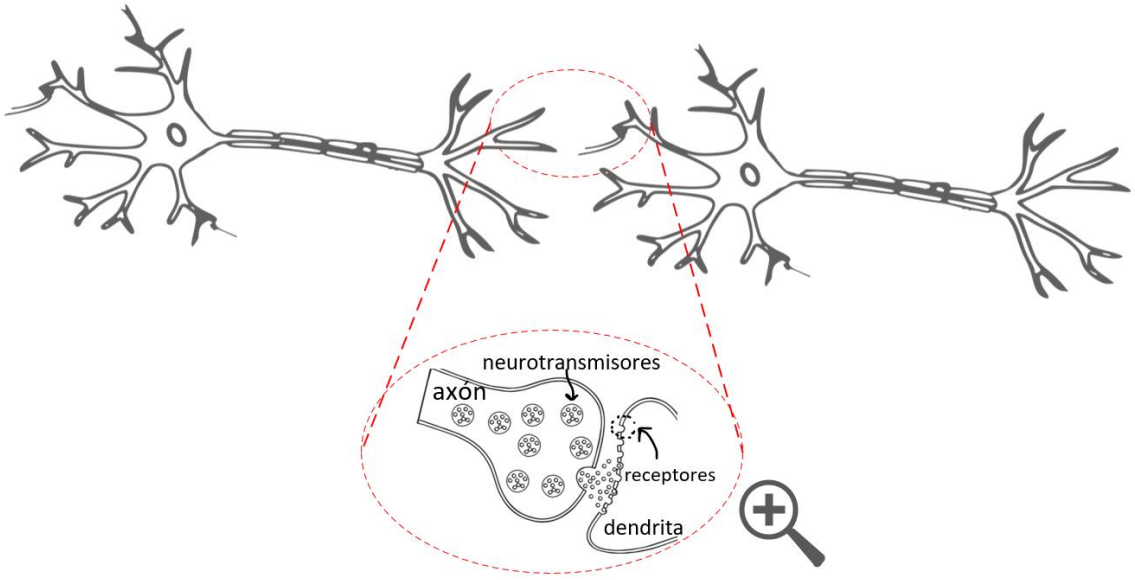


Ilustración 5: Sinapsis entre dos neuronas.

La imagen de la sinapsis está disponible en la referencia [25]. Podríamos hacer nuevamente una analogía con todo lo visto hasta ahora y definirlo dentro del contexto que nos atañe. Así pues, en 1960, el psicólogo Frank

Rosenblatt acuñó el término Perceptrón para referirse al modelo conceptual de una neurona artificial (para más información, véase la referencia [35]).

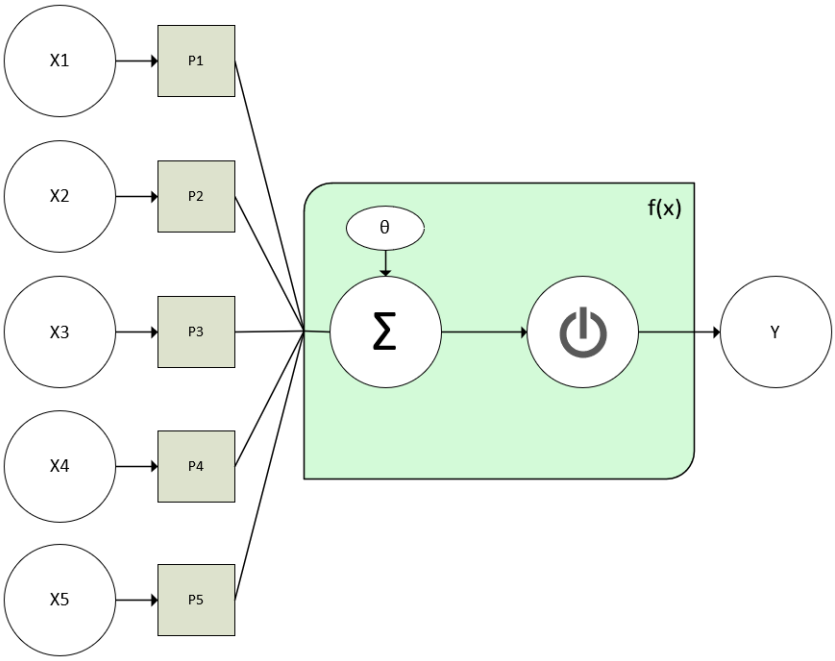


Ilustración 6: Esquema conceptual del Perceptrón

En el diagrama de la ilustración 6 podemos ver todas las componentes del perceptrón: en primer lugar, habrá una serie de estímulos externos que conformarán la entrada de nuestra neurona artificial. En este caso, hemos representado hasta cinco estímulos externos, denotados por x_i , los cuáles compondrán lo que denominaremos de ahora en adelante como el vector de entradas. Cada entrada, tendrá un peso asociado (P_i), que será un parámetro con el que dotaremos de una dependencia lineal a la relevancia de una determinada entrada.

Para entender esto, pongamos un ejemplo simplificado: supongamos que dispusiéramos de un detector de metales y tuviéramos que desempeñar una labor de búsqueda arqueológica en una playa. El detector nos avisará mediante una señal sonora conforme estemos sondeando en una dirección que nos acerque a un objeto de composición metálica, haciéndose más intenso el pitido en la medida en la que acortemos distancias con dicho objeto, y atenuándose conforme nos alejemos. Podríamos decir que el peso en este caso es la relación de cercanía con el elemento metálico.

Volviendo al campo de estudio que nos atañe, el vector de pesos será el artificio lógico-matemático con el que modelaremos la relevancia de una determinada entrada (una imagen) para determinar de forma unívoca y minimizando la probabilidad de error la correspondencia de dicha imagen con los resultados esperados.

Continuando con la disección de esta neurona artificial, nos encontraríamos con el bloque ‘f(x)’ el cuál podríamos identificar como una función de transferencia que nos devolverá una determinada salida en función delvector de entradas y del procesamiento de las mismas. Dicho procesamiento lo efectuarán la función de propagación y la función de activación (denotado por Σ y el símbolo de encendido respectivamente).

La función de propagación se encarga de sumar y ponderar todas las entradas con sus pesos, a lo cuál se le añade un sesgo, que no es más que un *offset* con el que desplazaremos el valor de la suma ponderada de partida, ajustando el resultado final. Podríamos decir que es un factor de corrección.

La función de activación en síntesis es un interruptor (por eso se ha elegido dicho símbolo) que determinará si el perceptrón está “encendido” o “apagado”, es decir, con esto determinaremos si el vector de entradas inicial ha producido un estímulo en nuestra neurona, ocasionando su activación.

En último lugar, tendríamos la salida, Y, la cuál se puede expresar matemáticamente atendiendo al diagrama de bloques como:

$$Y = \theta + \sum_{i=1}^{N} Xi \cdot Pi \tag{2-1}$$

Ecuación 1: Salida de un perceptrón

Todo lo visto hasta ahora sobre el perceptrón se puede enfocar como un sensor electroóptico que captará la luz y la convertirá en impulsos eléctricos, o visto desde un punto de vista más binario, en '0' y '1'.

En lo referente a la función de activación, cabe mencionar que existen distintos tipos, cuyo uso preferente varía en función de la aplicación. En una primera aproximación, hemos descrito su comportamiento como la función Escalón Unitario de Heaviside, pero lo cierto es que esto nos limitaría a trabajar con matrices binarias.

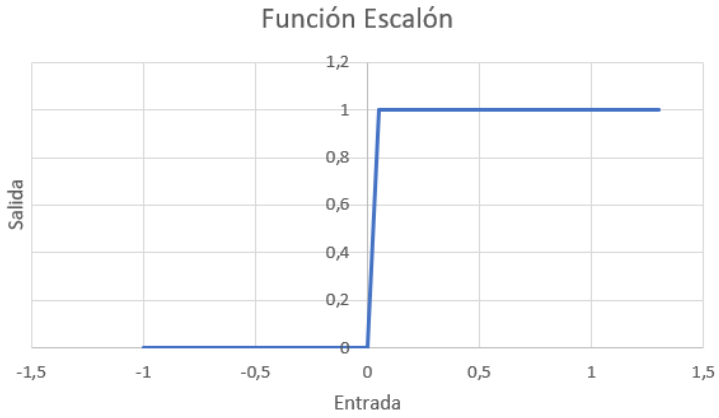


Ilustración 7: Función Escalón Unitario

Por ello, la función de activación más popularizada es la ReLU (de sus siglas en inglés, **Re**ctified **L**inear **U**nit), cuyo diagrama de estados y gráficas son los que se muestran a continuación:

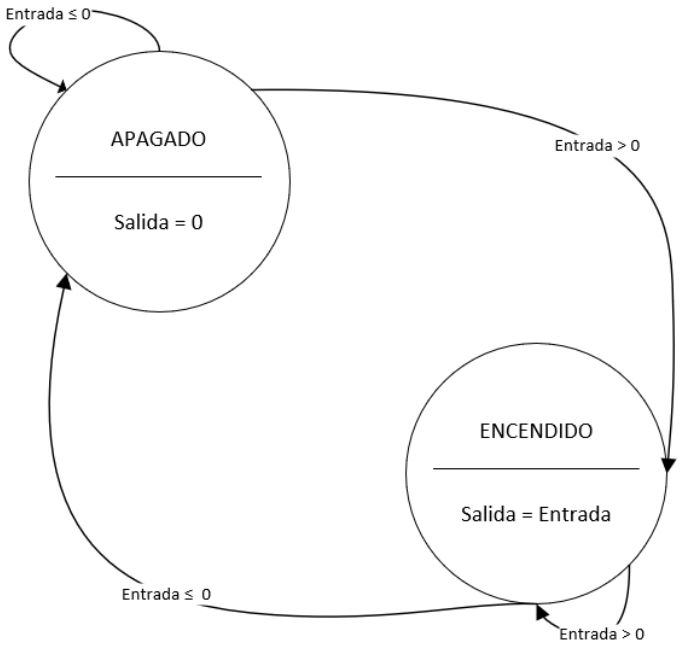


Ilustración 8: Diagrama de Estados función ReLU

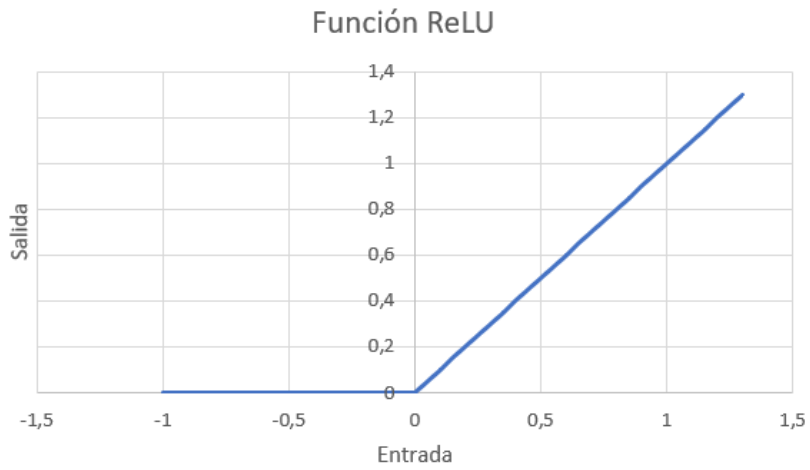


Ilustración 9: Gráfica función ReLU

2.1.2 Redes Neuronales

Una vez vista la unidad mínima, podemos hablar del conjunto a nivel operativo. Se entiende por red neuronal la agrupación de neuronas artificiales que contribuyen a un propósito común, comunicándose entre sí por medio de señales. Se pueden hacer distintas categorizaciones según la topología de red y según el método de aprendizaje. Para comenzar, veremos en el siguiente esquema los tipos de redes neuronales según la topología de red (fuente en la referencia [3]).

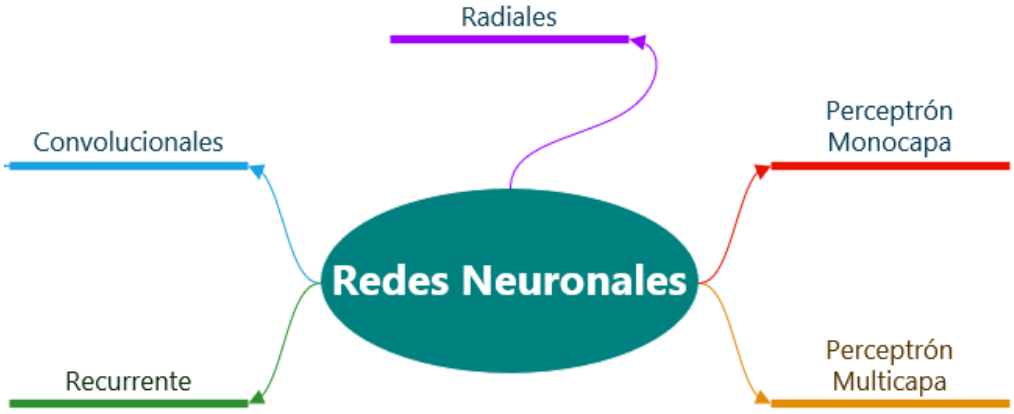


Ilustración 10: Tipos de arquitecturas de Redes Neuronales

Hasta ahora, hemos visto la unidad básica neuronal, que es el perceptrón. Ciertamente, un único perceptrón puede constituir una red neuronal en sí misma. Su ámbito de uso es la regeneración de los vectores de entrada dentro de redes neuronales donde haya distorsión de información o directamente esté incompleta la información en la entrada. Podríamos identificarlo como un regenerador de la señal en los enlaces de comunicación por fibra óptica, para mitigar las deformaciones de la señal producidas por la atenuación y el ruido presentes en el canal.

Seguidamente, podemos agrupar los perceptrones para formar una red multicapa. En esta red, los distintos bloques con las funciones de agregación y de activación conformarán lo que denominaremos capas ocultas. Dependiendo del conexionado, encontraremos un mallado parcial o total.

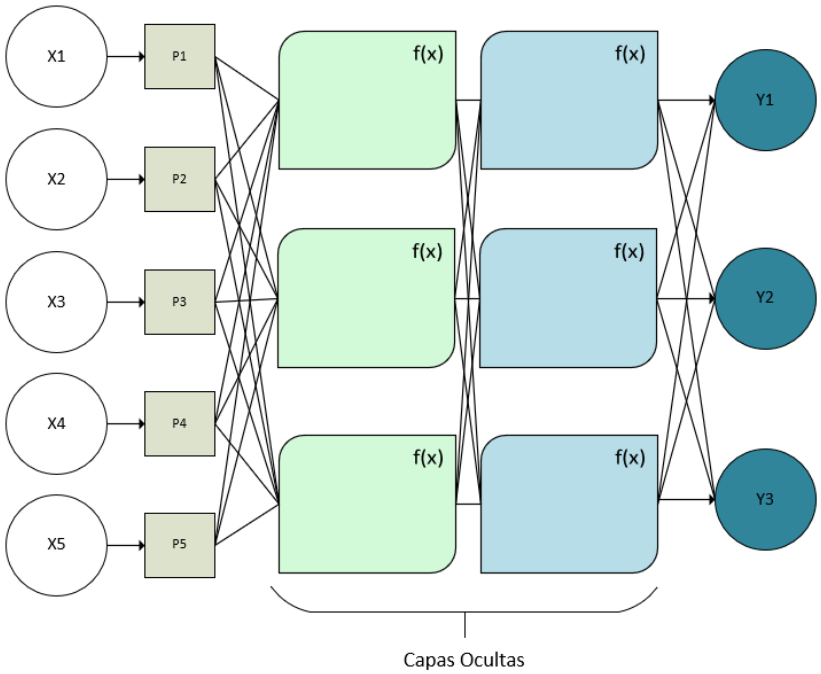


Ilustración 11: Ejemplo de Red Neuronal Multicapa

Con esta agrupación podemos conseguir una mayor escalabilidad, además de permitir la capacidad de especializar a las neuronas artificales.

Seguidamente, están las redes radiales, cuya peculiaridad respecto a las redes anteriores es la existencia de una única capa oculta denominada “centro”, y que la determinación de la salida se hace en función de la distancia hasta dicho centro (lo cual constituye una métrica de coste). La salida es una combinación lineal de las funciones de activación de las neuronas artificiales que la componen. En la siguiente imagen podemos observar un ejemplo de arquitectura de red radial.

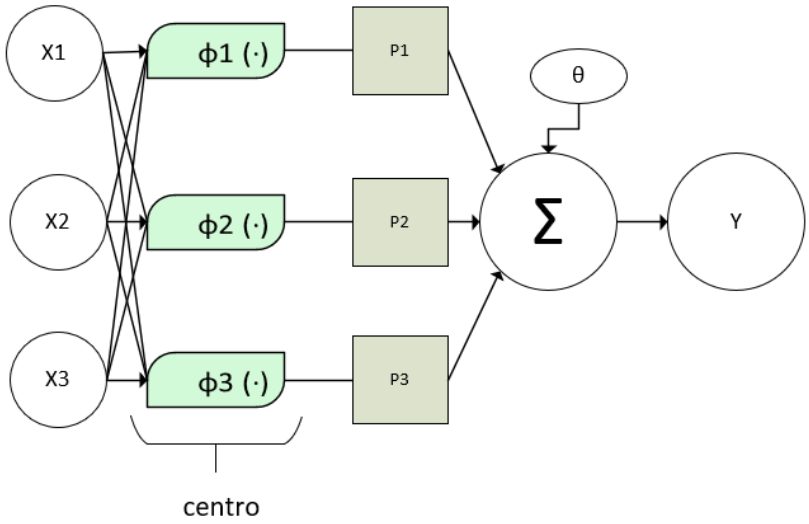


Ilustración 12: Ejemplo de Arquitectura Radial

A continuación, tendríamos las redes neuronales recurrentes, cuya morfología se caracteriza por la asuencia de estructura de capas. En su lugar, las neuronas se pueden interconectar entre sí de forma arbitraria, creando ciclos y dotando a la red de memoria (esto es, que la red determine que las salidas en un instante de tiempo son entradas del sistema en un instante posterior).

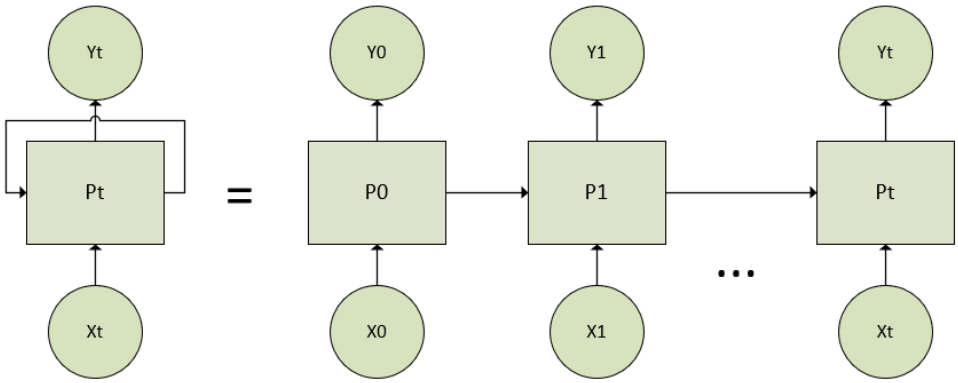


Ilustración 13: Ejemplo de arquitectura neuronal recurrente

En último lugar, tenemos las redes neuronales convolucionales. A diferencia de las redes multicapa de perceptrones, el interconexionado no es de mallado total con cada una de sus capas, sino que se produce suscripciones a determinadas capas. Con esto disminuye el uso de las neuronas artificiales implicadas en la red, además de disminuir el coste computacional. El enfoque geométrico que se le da a esta arquitectura es el de una matriz bidimensional sobre la que se opera. Este enfoque permite efectuar labores de segmentación y clasificación, las cuales son fundamentales para la visión artificial.

Reciben su nombre por la operación de convolución que realiza con las entradas con cada conjunto de neuronas artificiales (entendiéndose por convolución en este contexto el producto escalar matricial).

En la siguiente ilustración se muestra una arquitectura neuronal convolucional (fuente de la imagen en la referencia [21]).

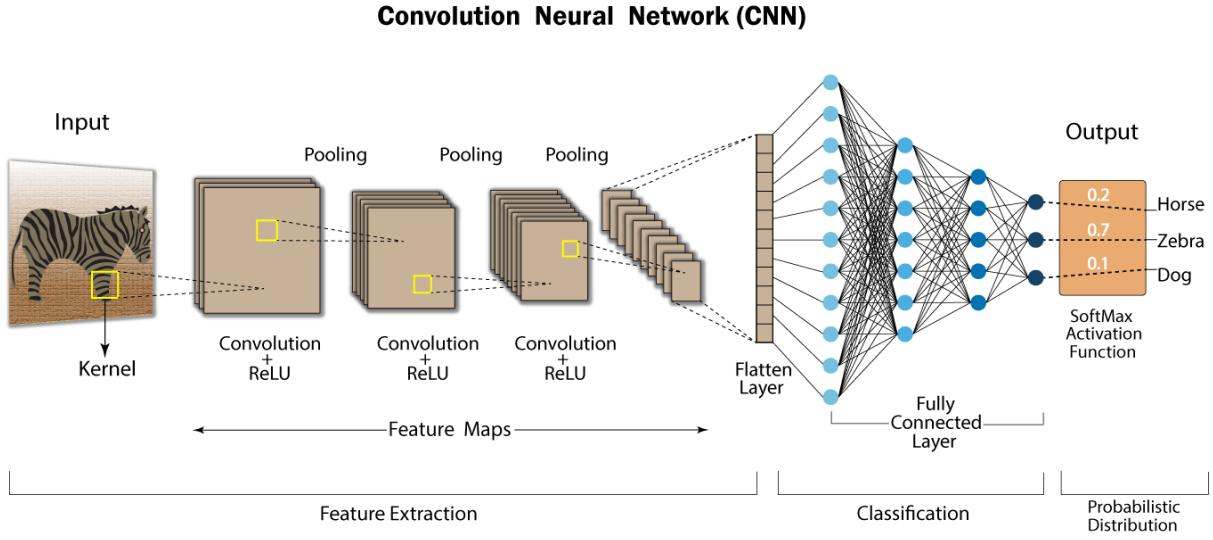


Ilustración 14: Ejemplo de Red Neuronal Convolucional

Se puede apreciar que la entrada de la red es una imagen, de la cuál sustraemos una región inicial a la que denominaremos *kernel*. A continuación, comenzaremos un proceso de análisis a un nivel elemental de la región seleccionada (tanteando, por ejemplo, las intensidades de color dentro de la matriz RGB). Seguidamente, volvemos a hacer un muestreo a partir de un fragmento de la matriz con la que estamos trabajando, aplicando el producto escalar con sus pesos correspondientes, resultando en una matriz de menor dimensión, y a dicho resultado lo cotejamos con la función de activación, que, en este caso, es la ReLU. A este proceso lo denominaremos sondeo de ahora en adelante, que no es más que la aplicación sistemática de la convolución junto con la función de activación.

En términos geométricos, estamos proyectando la matriz con la que operamos, reduciendo su dimensión a su

mínima expresión. Tras finalizar la etapa de mapeo (todos los sucesivos sondeos), habríamos concluido con la extracción de características, dando paso a la etapa de clasificación. Aquí utilizaríamos una red neuronal de perceptrones multicapa completamente mallada que culminaría con una distribución probabilística con todos los posibles resultados esperados.

2.1.3 Entrenamiento

El entrenamiento consiste en adaptar los distintos pesos a la entrada de la red de forma que en las capas de salida se ajuste el resultado a los datos de los modelos proporcionados. Todos los conceptos que veremos a continuación pueden verse en la referencia [3].

Con el aprendizaje lo que buscamos es dotar a la red neuronal de un conocimiento para discernir si el resultado de sus operaciones es correcto o no. Con un ejemplo aplicado a la visión artificial entenderemos lo que buscamos exactamente.

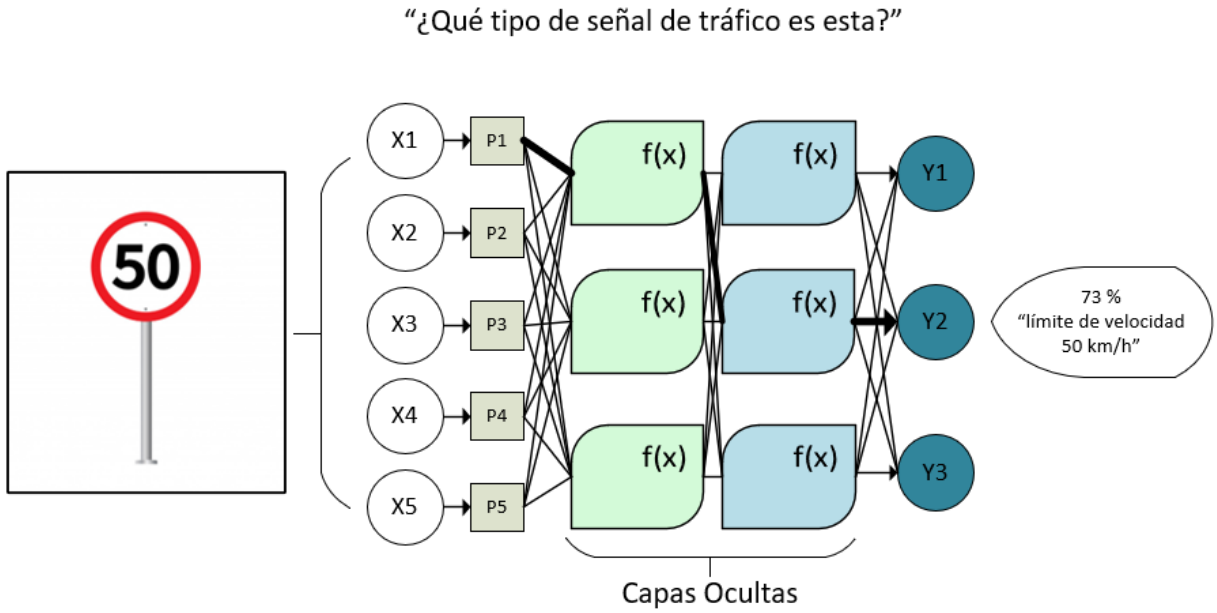


Ilustración 15: Ejemplo de entrenamiento a nivel teórico

Al ajustar el valor de los pesos, la salida queda mejor definida (se ha puesto con un trazo más grueso para plasmar este aumento en el peso 1). Cualquier posible resultado de la fase de entrenamiento irá con una probabilidad adjunta sobre la certeza del resultado. En este caso, nuestra red neuronal está segura al 73 % de que se corresponde con un límite de velocidad de 50 km/h, quedando un 27 % de probabilidad de haber errado e identificándolo con cualquier otro elemento reconocible en su repertorio de clases.

Este ejemplo sería una de las muchas iteraciones necesarias para que una red neuronal fuera capaz de identificar con precisión la señal de tráfico mostrada. Cuanto mayor sea el muestreo de imágenes en la fase de entrenamiento (es decir, cuanto más amplio sea el *dataset*) menor será la probabilidad de errar en un reconocimiento.

El entrenamiento se puede categorizar según su dinámica o según la supervisión de la propia red. Según su dinámica, existen varias formas de aplicar el proceso de aprendizaje en función de los pesos. Nuestra red puede aprender en base a un entrenamiento previo a su puesta en marcha, utilizando un conjunto de datos de entrenamiento y un conjunto de datos de pruebas. A esto se le denomina un entrenamiento *OFF LINE*.

Por otra parte, nuestra red puede aprender durante su funcionamiento, conforme se le presenta una nueva información al sistema. A esto se le conoce como entrenamiento *ON LINE*. En aras de visualizar esto mejor, se propone el siguiente esquema recopilatorio con los tipos de entrenamientos disponibles:

Según su criterio para la modificación de pesos, tenemos:

2.1.3.1 Redes Supervisadas

Son aquellas en las que se monitoriza el aprendizaje mediante una entidad supervisora externa que determina la salida en base a una determinada entrada.

Dentro de este tipo de aprendizaje, podemos distinguir los siguientes tipos:

- **Supervisado por corrección de error**: Consiste en reajustar los pesos en el conexionado de la red para reajustar en función de la salida deseada. Esta dinámica está presente en muchos ámbitos de la ingeniería, concretamente en el ámbito del control automático. En el siguiente diagrama podemos ver las componentes de una realimentación negativa.

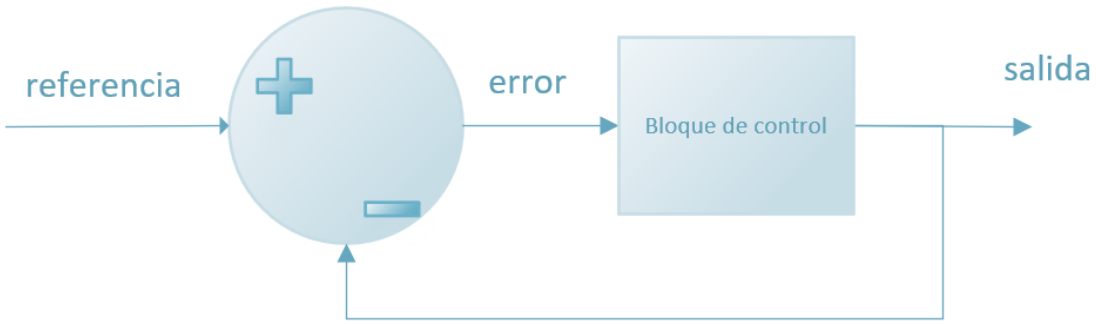


Ilustración 16: Ejemplo de Realimentación Negativa

Así pues, la expresión que rige la corrección de los pesos viene dada por la siguiente fórmula:

$$\Delta(pesos_{i,j}) = (y_j - d_i) \cdot y_i \cdot \beta \tag{2-2}$$

Ecuación 2: variación de pesos en supervisión por corrección de error

La cual, se puede deducir de la siguiente imagen:

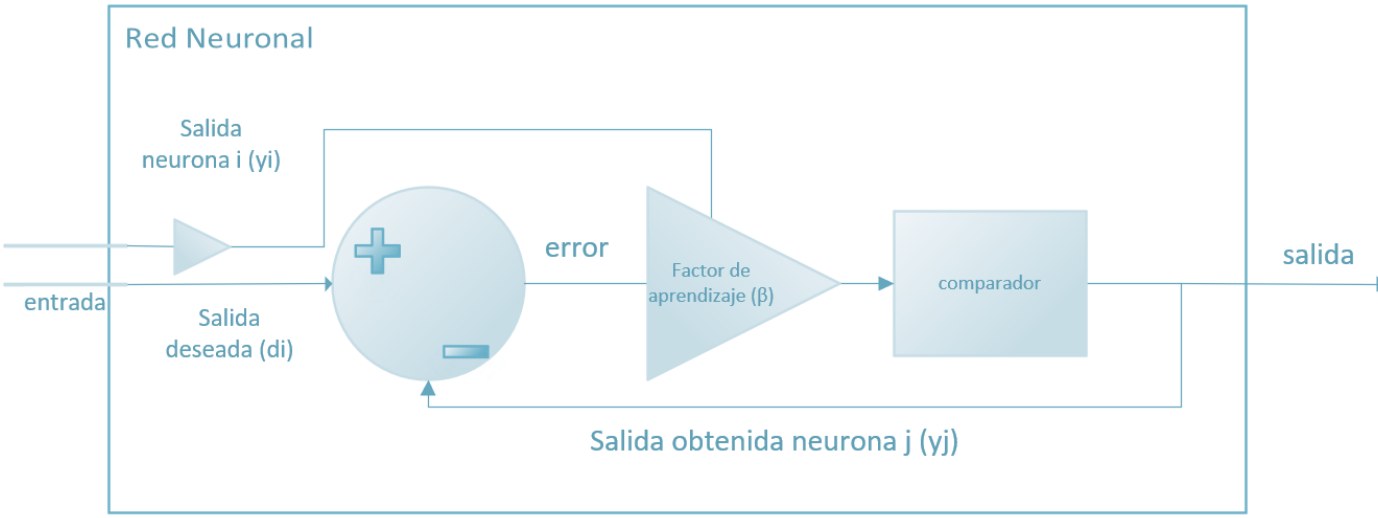


Ilustración 17: Esquema concepetual de un sistema de corrección de error

- **Supervisado por refuerzo**: En este tipo de aprendizaje supervisado, en vez de tener un ejemplo concreto de la salida que se busca como en el caso anterior, la función de supervisión la efectúa una señal de refuerzo que pondera la salida con un +1 si se ajusta al resultado deseado, o con un -1 en caso contrario. Por ello, es un proceso mucho más lento. La expresión que rige los pesos de los enlaces neuronales sería la siguiente:

$$\Delta(pesos_{i,j}) = (y_j \pm 1) \cdot y_i \cdot \beta \quad (2\text{-}3)$$

Ecuación 3: Variación de los pesos en aprendizaje supervisado por refuerzo

- **Supervisado Estocástico**: Aquí invertimos la forma de proceder. En vez de cambiar los pesos partiendo de una entrada determinada, lo que se hace es cambiar aleatoriamente el valor de los pesos y se observa si se cumple con la salida deseada. Un ejemplo aplicado de este método de aprendizaje es una red basada en la máquina de Boltzmann (fuente en la referencia [20]), como en la que se muestra a continuación.

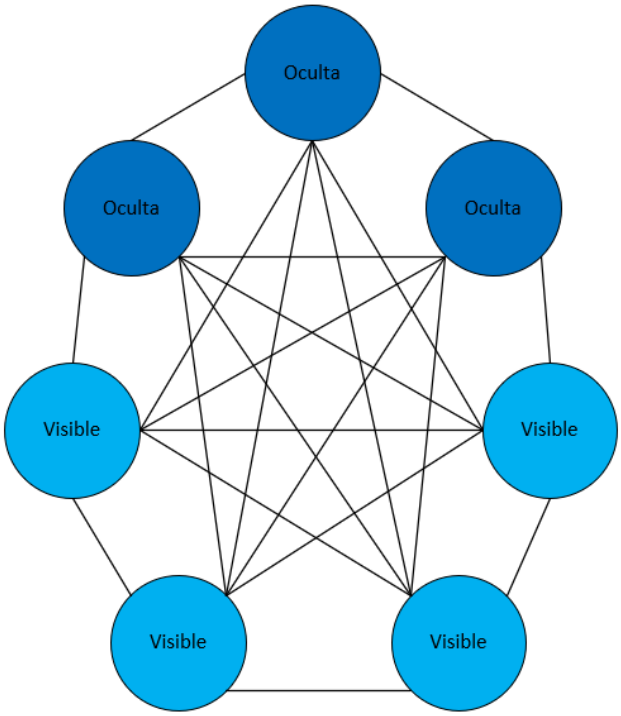


Ilustración 18: Red Neuronal basada en la máquina de Boltzmann.

2.1.3.2 Redes No Supervisadas

Son aquellas que no requieren de un agente externo para modificar el peso de las conexiones entre sus neuronas. El funcionamiento de estas redes se basa en la búsqueda de:

- Características.
- Regularidades.
- Correlaciones.
- Categorías.

Y las posibles interpretaciones que hay de sus salidas son:

- El grado de similitud entre la entrada actual y las entradas previas.
- La categoría de la entrada recibida (*clusterización*).
- Mapeo de características
- La codificación de la entrada recibida, generando a la salida una versión codificada de la entrada con menos bits sin perder información crítica.

Esta última interpretación es el fundamento en el que se basan las operaciones de convolución dentro de las redes neuronales convolucionales.

Dentro de este tipo de aprendizaje podemos discernir entre:

- **Aprendizaje Hebbiano:** busca extrapolar las características y crear una relación de semejanza a partir de los datos de entrada. Para ello, se ajusta el valor de los pesos en las conexiones a partir de la correlación de las salidas de las neuronas conectadas (neuronas 'i 'y 'j' respectivamente). Podemos visualizar este comportamiento en la representación de la siguiente red monocapa:

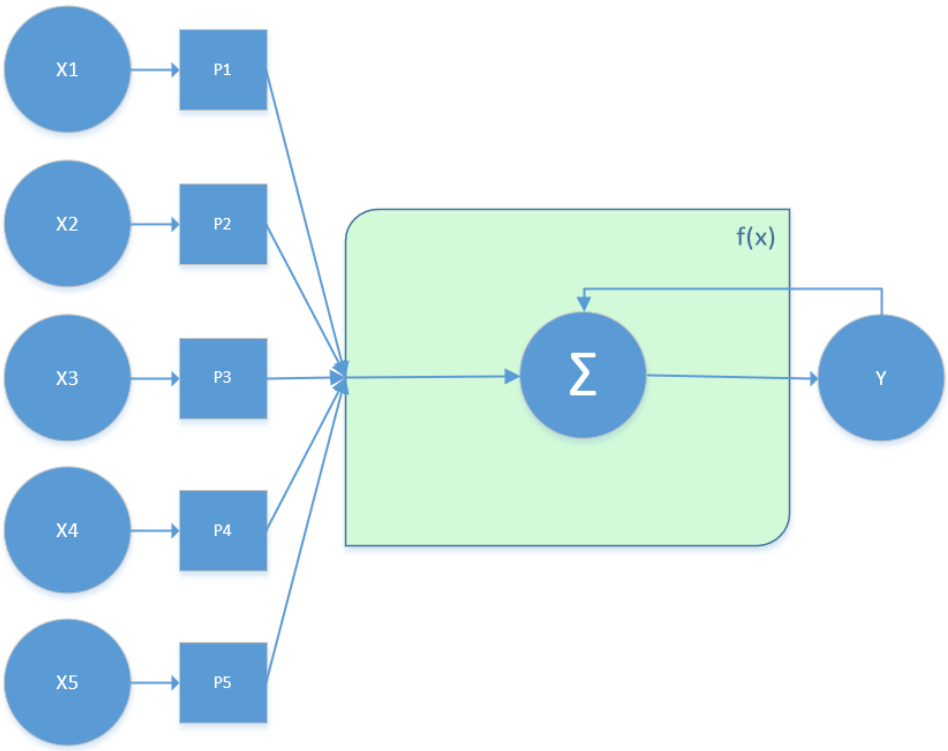


Ilustración 19: Ejemplo de red monocapa con aprendizaje Hebbiano

En todas las redes con aprendizaje Hebbiano, la variación de los pesos se puede expresar como:

$$\Delta(pesos_{i,j}) = y_j \cdot y_i \tag{2-4}$$

Ecuación 4: Variación de los pesos en aprendizaje Hebbiano

- **Aprendizaje competitivo y colaborativo:** Este tipo de aprendizaje busca la proximidad a un patrón de entrada, fomentando la competitividad de las neuronas. La neurona que más se acerque a dicho patrón, ajustará su peso (será la única en activarse).

Así pues, se propone una organización de las diferentes metodologías de aprendizajes en base a todo lo anteriormente explicado:

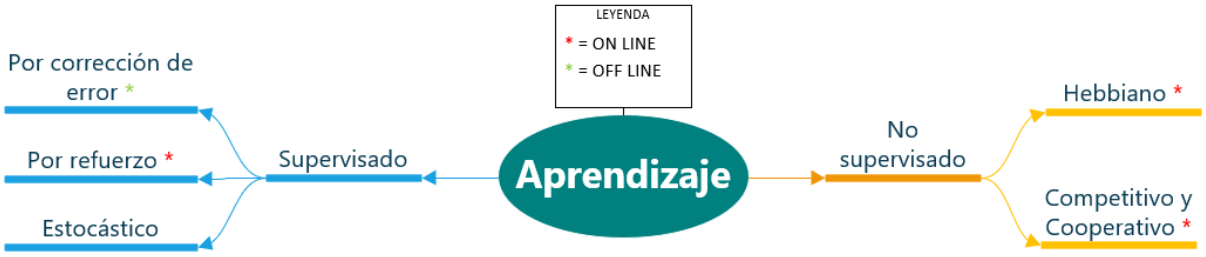


Ilustración 20: Tipos de aprendizaje

Y con todo esto finalizaríamos este capítulo sobre el fundamento teórico necesario. A continuación, analizaremos y comentaremos brevemente las dependencias que utiliza YOLOv5 junto con su función.

3 DEPENDENCIAS DE YOLOV5

Comenzar bien no es poco, pero tampoco es mucho.

- Sócrates -

En este capítulo abordaremos todos los aspectos técnicos necesarios de los que depende el software de Yolov 5. Se comenzará describiendo las prestaciones técnicas y el por qué de su elección frente a las distintas alternativas existentes en el mercado en la actualidad. Los pasos concretos de la instalación del software se relatan en el Anexo A de este documento.

3.1 Dependencias

En este apartado describiremos las dependencias de YOLOv5, así como su utilidad dentro del propio *framework*.

3.1.1 COCO Datasets

COCO (de sus siglas en inglés, *Common Objects in Context,*) es un conjunto de datos de referencia para la detección de objetos. Con COCO tendremos acceso a las siguientes funcionalidades:

1. Segmentación de objetos.
2. Reconocimiento del contexto.
3. Segmentación de entidades mediante *superpixel.*
4. Más de 300000 imágenes (de las cuáles, 220000 están categorizadas).
5. 1.5 millones de instancias de objetos.
6. 80 categorías de objetos.
7. 91 clasisficaciones de entidades.
8. 5 posibles títulos por imagen.
9. 250000 personas con puntos de interés.

En síntesis, COCO será el componente con el conjunto de datos entrenados previamente para disminuir la latencia en la detección de objetos. Para más información, véase la referencia [6].

3.1.2 Matplotlib

Este modulo de Python contiene todas las librerias necesarias para crear visualizaciones estáticas o animadas en Python. Se usará principalmente para representar en las gráficas el rendimiento de la red neuronal según su etapa. Para más información, acuda a la referencia [7].

3.1.3 Numpy

Es una librería de uso extendido que da soporte para todas las herramientas matemáticas, en especial, para operaciones matriciales, con las que trabajarán nuestras redes neuronales convolucionales. Puede encontrar más información en la referencia [10].

3.1.4 OpenCV-Python

Este modulo contiene todas las librerías con las que trabajar en Python con este software de vision artificial; será el modulo encargado de generarnos los recuadros de colores con los que capturamos los objetos en una imagen o *frame* de video con los que se efectúen la inferencia para la clasificación acorde al modelo entrenado. En capítulos posteriores veremos varias imágenes de ejemplo que ilustran esta funcionalidad. Para más información, véase la referencia [11].

3.1.5 Pillow

Esta librería multiplataforma contiene todas las herramientas para la edición y manipulación de imágenes en Python. Trabaja como una dependencia directa de OpenCV, además de servir como *driver* para almacenar las imágenes generadas en la fase de inferencia dentro de la memoria interna del dispositivo que estemos usando. Se puede encontrar más información sobre Pillow en la referencia [12].

3.1.6 PyYAML

Este modulo contiene todas las librerías para aplicar el formato de serialización de datos de una forma más sencilla. De hecho, será una librería indispensable cuando utilicemos nuestro *dataset* para la fase de entrenamiento, ya que le proporcionará un manifesto del conjunto de imágenes utilizado. Para más información, véase la referecia [36]

3.1.7 SciPy

Este conjunto de librerías aporta un conjunto de herramientas matemáticas y algorítmicas sobre las que se apoyará nuestra red neuronal. Para más información, véase la referencia [13].

3.1.8 Torch

Esta biblioteca de código abierto la usaremos para las tareas de aprendizaje automático. Es el modulo central sobre el que se construye la arquitectura de YOLOv5. Sustituye a *darknet* en versions anteriores de YOLO. Para más información, acuda a la referencia [14].

3.1.9 Torch-vision

Complementando al modulo anterior, este conjunto de librerías aportarán un refuerzo en el aprendizaje automático orientado a la vision artificial. Puede encontrarse más información en la referencia [15].

3.1.10 Tqdm

Este modulo es un apoyo visual que muestra una barra de progreso al utilizar bucles. Se usará cuando se efectúe la fase de entrenamiento del modelo, mostrando las etapas, junto con información relevante como el tiempo esperado hasta la finalización, el número de batch, y la memoria interna necesaria empleada. En la siguiente ilustración vemos un ejemplo de uso de este modulo, disponible en la referencia [8].

Ilustración 21: Ejemplo de barra de carga de tqdm

3.1.11 Tensor Board

Conjunto de librerías con las que manendremos un seguimiento del aprendizaje automático, visualizando grafos, histogramas con los pesos o las métricas de nuestra red neuronal. En secciones posteriores del documento veremos ilustraciones sobre la interfaz web empleada por este modulo, junto con los resultados de la fase de entrenamiento. Para más información, véase la referencia [16].

3.1.12 Seaborn

Este modulo basado en Matplotlib proporciona una interfaz de alto nivel para la presentación de la información de una forma más atractiva para el usuario. En la siguiente ilustración podemos ver un ejemplo (disponible en la referencia [9]) donde se aprecia esa capa adicional de estilos durante la representación de las gráficas con las métricas de un entrenamiento en YOLOv5.

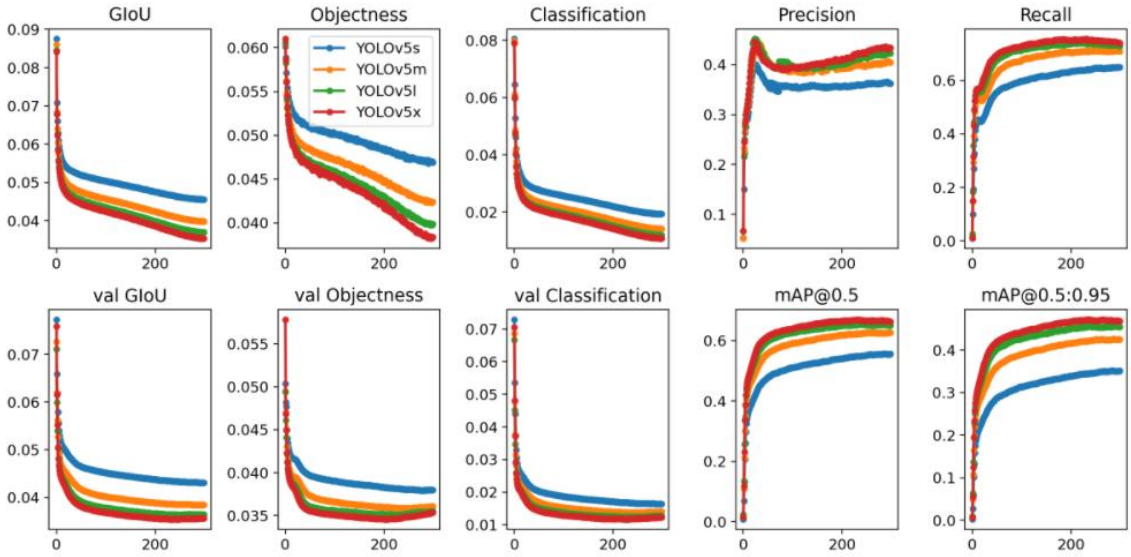


Ilustración 22: Gráficas generadas en la fase de entrenamiento usando seaborn

3.1.13 Pandas

Esta librería que se forma como extension de NumPy sirve para el análisis y la manipulación de datos en Python. Tiene un uso extendido para la gestión de archivos CSV. Para más información, véase la referencia [37].

3.1.14 Thop

Este modulo nos proporcionará información sobre el número de operaciones en coma flotante (FLOPs) que empleará nuestro modelo durante su entrenamiento y durante la fase de inferencia. Podemos encontrar más información en la referencia [38].

3.1.15 Pycocotools

Este conjunto de librerías sirve como interfaz lógica para el mapeo de los modelos de datos de COCO en YAML. Para más información, véase la referencia [39].

3.2 Relación y arquitectura de los componentes

En el siguiente esquema se puede apreciar visualmente la jerarquización en la relevancia de los distintos módulos anteriormente presentados en una estructura piramidal, quedando en la base aquellas librerías que son más esenciales y que se centran en la parte computacional de la red neuronal, y subiendo en altura para la capa

de presentación de la información.

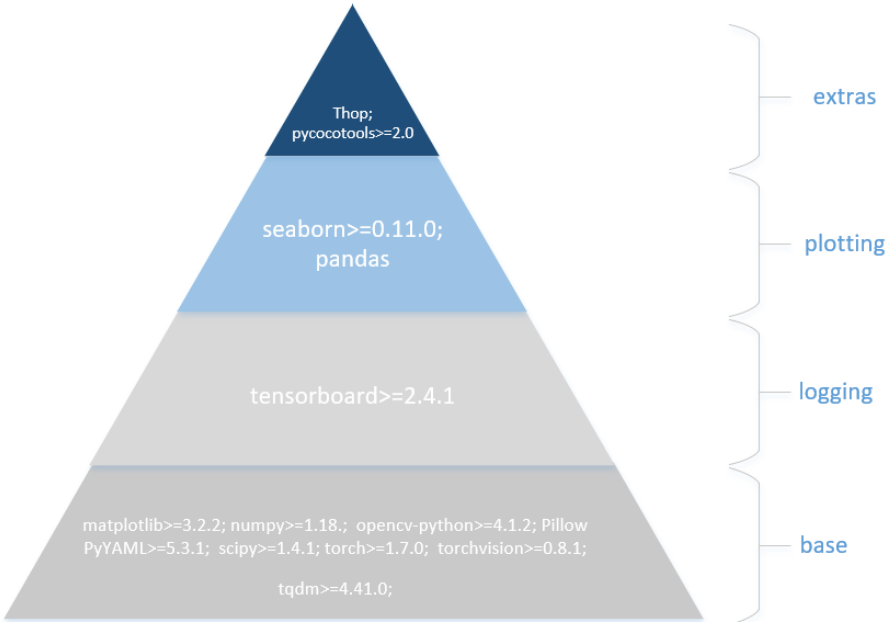


Ilustración 23: Esquema piramidal de la relación entre los componentes

Las dependencias anteriormente descritas poseen una relevancia relativa en cuanto a su uso. Algunas son la base de las operaciones centrales de todo el proceso de convergencia de una red neuronal convolucional, y otras son interfaces gráficas para la obtención e interpretación de los resultados respecto a los parámetros y estadísticos extraídos durante la ejecución.

Con todo esto, YOLOv5 consigue proporcionar el servicio de Visión Artificial.

4 APLICACIÓN

La función de un buen software es hacer que lo complejo aparente ser simple.

- Grady Booch -

En este capítulo veremos un ejemplo de aplicación utilizando las dependencias de YOLOv5 para la detección de imágenes, empleando además interfaces gráficas en PyQT5 y asistentes de voz. El objetivo de la aplicación es ofrecer una interfaz al usuario con la que interactúe con el software de detección de imágenes, posibilitando la opción de cargar imágenes desde el ordenador y efectuando una inferencia con una certeza calculada en tiempo de ejecución.

En el siguiente apartado, introduciremos al asistente de voz desarrollado para desempeñar esta tarea: A.R.G.O.S. (*Assistant for RecoGnitiOn Services*).

4.1 Etimología

La elección del nombre para el asistente no es casual. En la mitología griega, Argos Panoptes (del griego, Άργος Πανοπτης «que todo lo ve, de todos los ojos») era un gigante de cien ojos al servicio de la diosa Hera. Su propósito era vigilar a Ío, que había sido transfigurada en res por la diosa debido a la infidelidad de Zeus con la ninfa.

Hera mandó a Argos a custodiar en Nemea a Ío junto a un olivo, y mantenía siempre su guardia gracias a que alternaba el descanso de sus ojos: mientras algunos estaban cerrados, otros permanecían abiertos. Finalmente, Argos fue decapitado por el dios Hermes, que fue enviado por Zeus para recuperar a Ío.

La siguiente imagen, disponible en la referencia [31], representa a Argos custidando a Ío bajo un olivo.

Ilustración 24: Representación de Argos Panoptes, de Bernardino di Betto di Biagio.

Tras este breve repaso histórico, concluimos que es acertada la elección de las siglas para el asistente de voz. En los siguientes apartados veremos en más detalle cómo está implementado A.R.G.O.S.

4.2 Estructura y componentes

En esta sección veremos todos los elementos que conforman el asistente, utlizando para ello las representaciones usadas en ingeniería del software.

A.R.G.O.S. está desarrollado en Python, al igual que el software de YOLOv5. Esto facilita la integración con la arquitectura de YOLOv5, y por ende, la comunicación interna y las labores de Desarrollo al no tener que utilizar un *middleware.* En la siguiente ilustración veremos un diagrama de componentes que resume los principals módulos y librerías usados para la implementación:

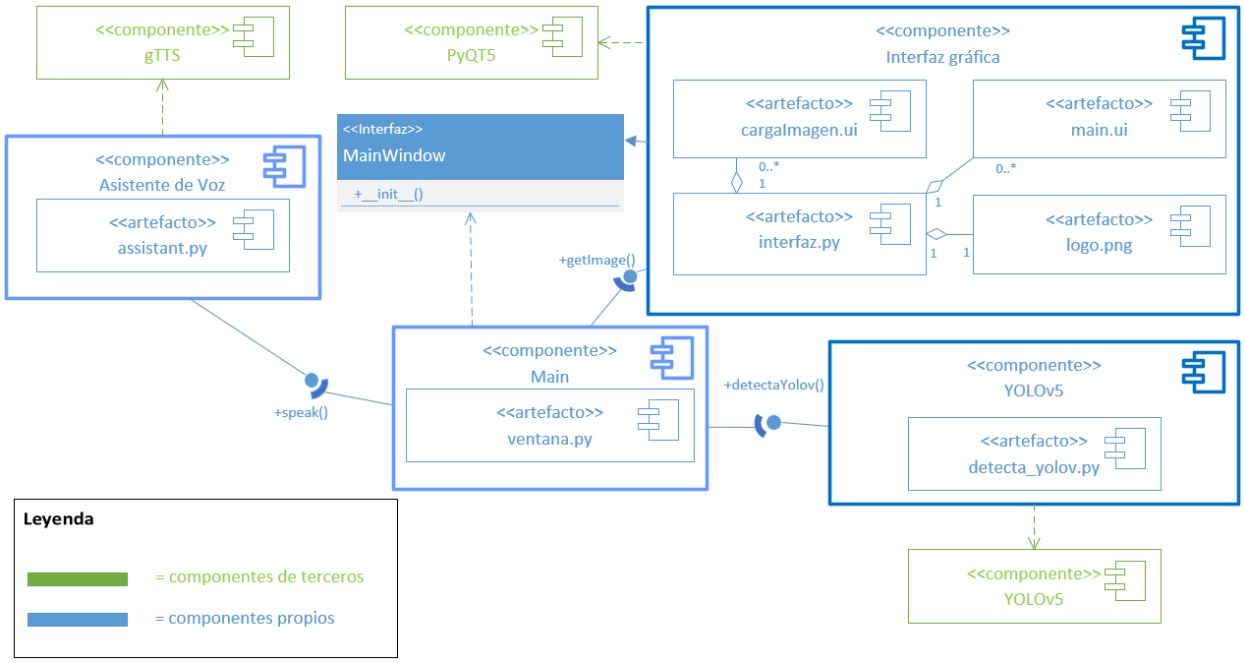


Ilustración 25: Diagrama de Componentes de A.R.G.O.S.

La componente central de la aplicación la conforma el Main, el cuál solicita los servicios del asistente de voz, de inferencia y de interfaz gráfica a las componentes correspondientes. Éstas se valen de sus artefactos y de las clases desarrolladas dentro de estos archivos para llevar a cabo su implementación, utilizando los módulos de terceros a modo de interfaz de programación de aplicaciones.

4.3 Clases y relaciones

Para el desarrollo de la aplicación, se ha seguido el paradigma de la Programación Orientada a Objetos, en la cuál, el Código se estructura por clases. Esto facilita enormemente la escalabilidad y el diseño de la propia aplicación, ya que se pueden utilizar patrones de diseños y de comportamiento para lograr una implementación eficiente.

Por ejemplo, para utilizar debidamente las distintas APIs de las que están presents en el programa, utilizamos el patron de fachada, de forma que se obtenga una interfaz entre las clases principales y los métodos de los módulos de terceros.

Así mismo, para la realización de determinadas tareas (como el refresco del display de la forma de onda, que se verá más adelante), usamos el patrón de delegación, con el que la clase principal escinde un hilo para la ejecución de dicha subrutina sin comprometer el acceso a los recursos centrales de la aplicación.

Teniendo en cuenta la diversidad de las referencias empleadas para la documentación y el desarrollo de la aplicación, se ha decidido mantener las nomenclaturas de los autores originales. Por ello, Podemos ver cómo algunas clases poseen los nombres de los atributos y los métodos en inglés, y las que han sido desarrolladas específicamente para este proyecto estarán en castellano.

En el siguiente diagrama de clases podemos ver cómo se interrelacionan las distintas clases y los objetos desde una perspectiva de bajo nivel de implementación.

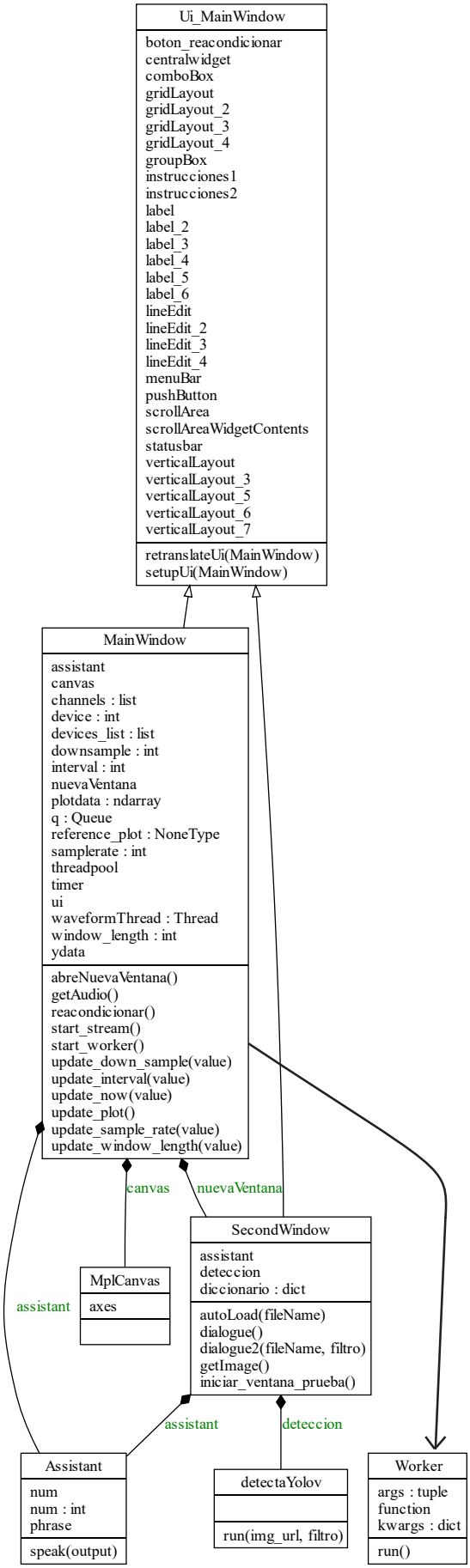


Ilustración 26: Diagrama Clases de A.R.G.O.S.

A continuación, explicaremos de forma somera las distintas relaciones entre las clases

Las clases "Window" (*MainWindow* y *SecondWindow*) proporcionarán la interfaz necesaria para la interacción del usuario con el programa. Éstas heredan sus propiedades de *Ui_MainWindow*, que proporciona todos los métodos, clases y objetos de la API de PyQT5 para desarrollar interfaces gráficas de usuarios. La segunda ventana desempeñará las labors de inferencia, para lo cuál, necesitará hacer uso del asistente de voz y de la interfaz de YOLOv5. Por contra, la ventana principal servirá a modo de menu inicial desde el cuál lanzar la aplicación. Además, dispone de un entorno gráfico para visualizer la forma de onda recogida por el periférico de entrada seleccionado (en nuestro caso, el micrófono predeterminado del sistema). Con esto podremos ver cuándo habla el asistente (además de percibirlo acústicamente).

En el siguiente Diagrama de Caso de Uso vemos plasmado este procedimiento.

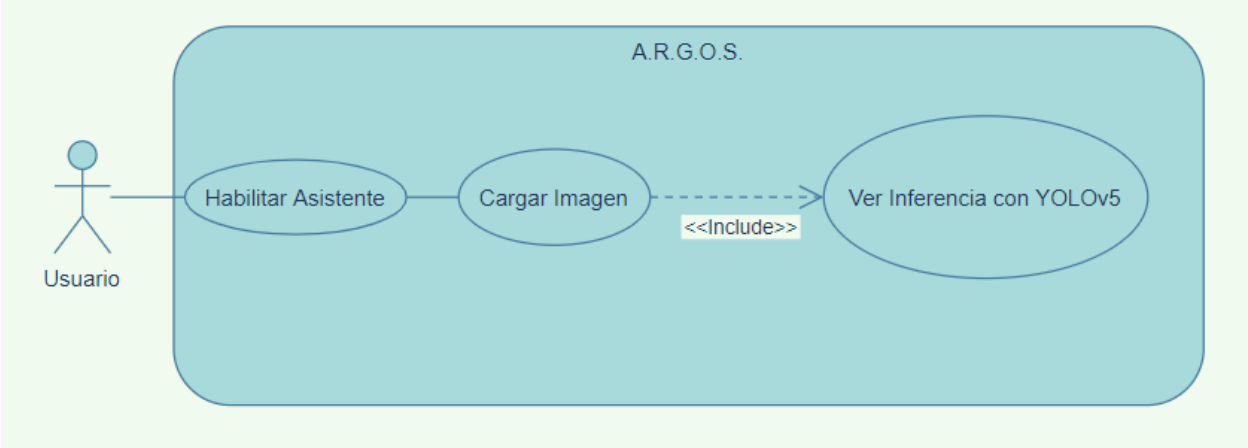


Ilustración 27: Diagrama de Caso de Uso

Con todo esto concluimos con la introducción esquemática del asistente. En los anexos correspondientes (E,F,G y H) se puede ver el código fuente de la aplicación.

En el siguiente epígrafe veremos un ejemplo de ejecución paso a paso, para entender mejor la comunicación entre las distintas clases mencionadas.

4.4 Ejemplo de ejecución

En este epígrafe ilustraremos paso a paso cada etapa de la ejecución de la aplicación desarrollada.

En primer lugar, nos situamos en el directorio de trabajo del asistente.

```
C:\Users\usuario>cd Desktop/argos
```

Y a continuación, ejecutamos el siguiente comando:

```
C:\Users\usuario\Desktop\argos Python ventana.py
```

El resultado se muestra en la siguiente ilustración.

Ilustración 28: Ventana principal de A.R.G.O.S.

Se puede observar que la interfaz posee varias componentes: por un lado, se ha llevado a cabo el desarrollo de un *display* para representar la forma de onda recogida por el micrófono. Lo utilizaremos para representar visualmente la interacción del asistente. Esto, en adición con la respuesta en formato vocal, mejoran la experiencia del usuario.

Los parámetros de audio que aparecen abajo están bloqueados y sólo se pueden editar desde el código fuente. Esto está diseñado así para evitar errores en la configuración que produzcan inconsistencias en la programación por hilos.

Seguidamente, encontramos 2 botones:

- "Habilitar Asistente" que activará a A.R.G.O.S. para iniciar una interacción humano-máquina con el usuario. Al pulsarlo, A.R.G.O.S. hablará con su frase inicial: "¡Hola! Mi nombre es ARGOS. ¿En qué puedo ayudarte?" y en paralelo, se lanzará una ventana emergente para habilitar el entorno de pruebas.
- "Reacondicionar entorno para otra prueba". Esto elimina los subdirectorios que resultan de la ejecución del software para una inferencia, de forma que pueda accederse de forma unívoca a la ruta hasta el resultado de la detección, simplificando las labores de desarrollo y la correcta gestión del entorno de pruebas.

Una buena práctica es pulsar este último botón después de arrancar la aplicación, por si hubiera residuos de ejecuciones anteriores. Cuando dichos subdirectorios residuales estaban presentes y han sido eliminados, aparece un diálogo subrayado en blanco en el tercio inferior de la interfaz, tal y como se muestra continuación:

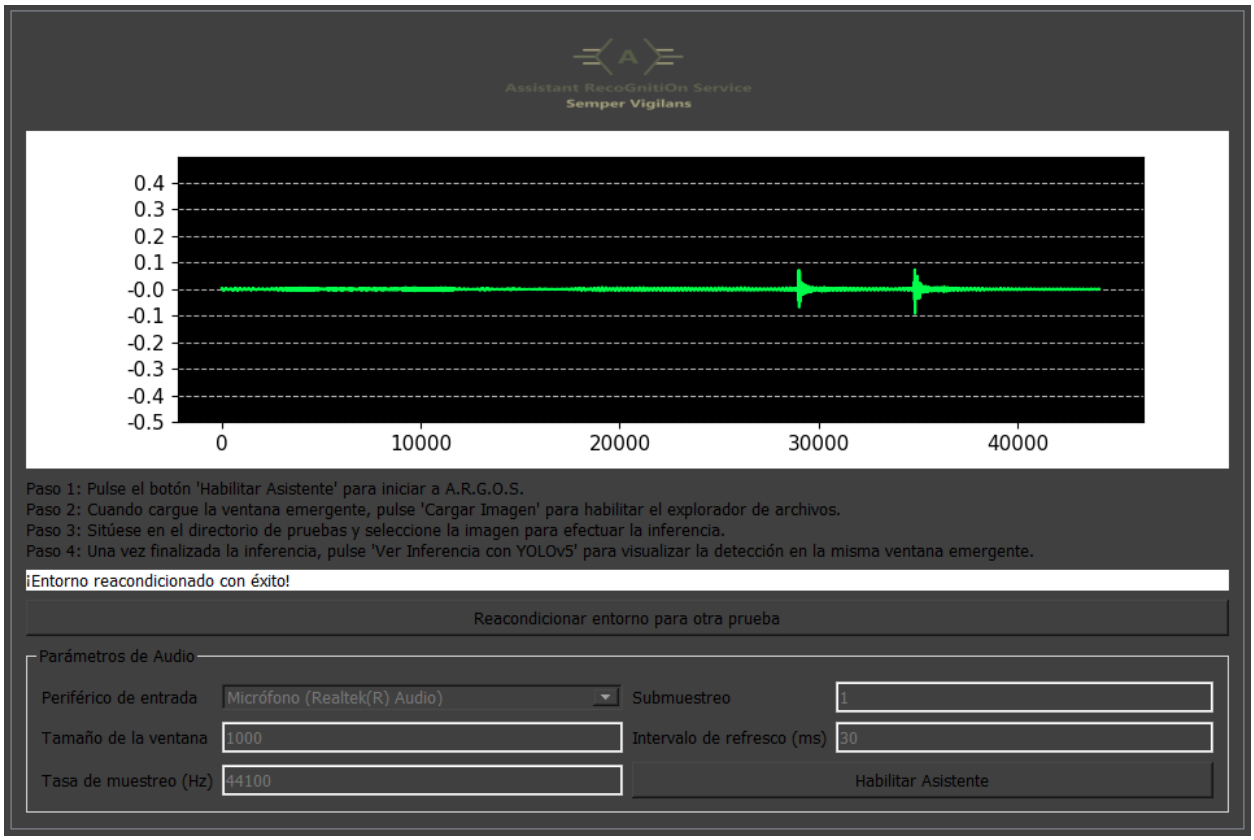


Ilustración 29: Diálogo con el reacondicionamiento del entorno

Pero, ¿y qué ocurre si no había subdirecotorios anteriores? Aparecerá un mensaje de error inocuo para la ejecución informando de la inexistencia de dicho directorio:

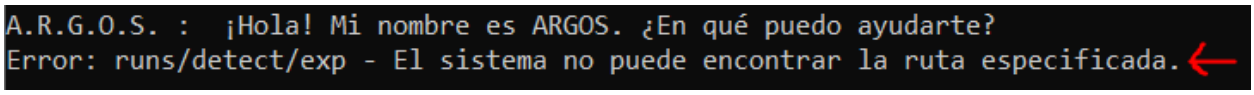


Ilustración 30: intento de reacondicionamiento ya consolidado

Una vez preparado el entorno para efectuar una prueba de inferencia, nos dirigimos a la nueva ventana que ha emergido, y observamos que ésta consta de dos botones y de dos recuadros de texto:

- "<Aquí se cargará tu imagen de prueba>" que no es más que una etiqueta de PyQT5 en la que cargaremos la imagen con la que queremos detectar uno de los siguientes elementos:
  - Límites de velocidad
  - Señales de STOP
  - Semáforos
  - Pasos de peatones.
- "<Aquí se cargará la inferencia de la imagen anterior>", que, análogamente con la etiqueta anterior, será el espacio donde cargaremos el producto de la detección, que no será más que la imagen de prueba con una edición en la que habrá un recuadro de color, con el elemento y el procentaje de certeza (en tanto por uno). El asistente se encargará de convertir ese porcentaje a tanto por cien.
- "Cargar Imagen" que abrirá el explorador de archivos en una ventana emergente para que selecciones gráficamente la ruta hasta la imagen. Se recomienda por simplicidad disponer de la imagen en el mismo directorio donde se está ejecutando la aplicación principal.
- "Ver Inferencia con YOLOv5" que cargará el resultado de la detección desde el directorio correspondiente.

En la siguiente ilustración, podemos ver la apariencia de la ventana del entorno de pruebas para la detección.

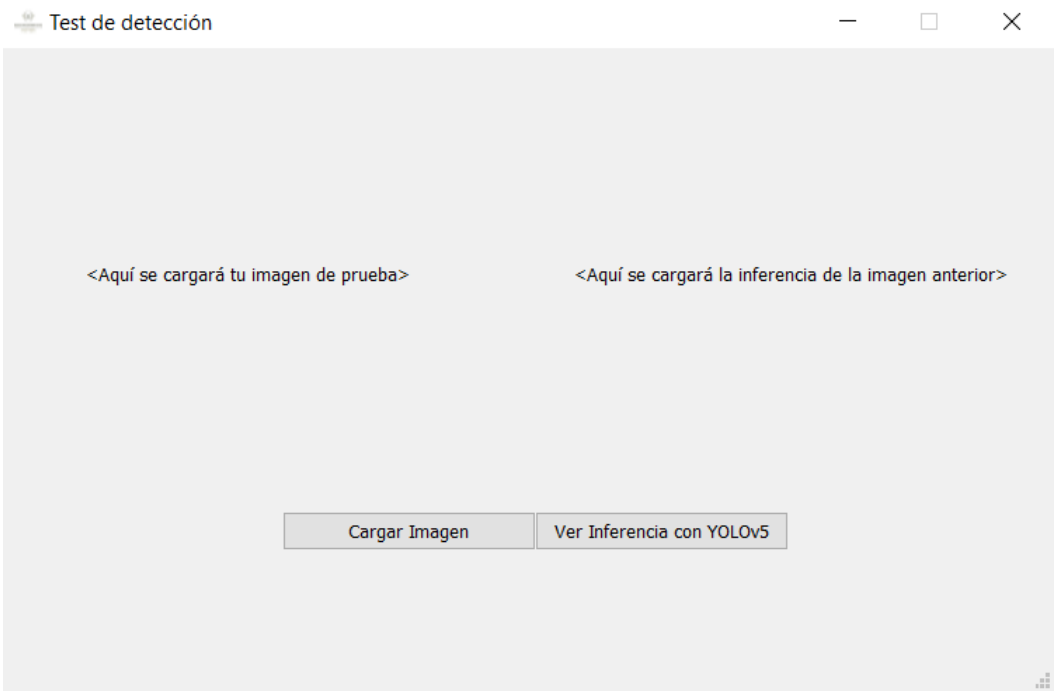


Ilustración 31: Ventana del test de detección

Una vez que pulsamos en "Cargar Imagen" y eleccionamos la imagen de prueba, comenzará el proceso de inferencia. A.R.G.O.S. utilizará el software de YOLOv5 con nuestro modelo entrenado y creará la imagen editada anteriormente mencionada. Cuando se inicia este proceso, dirá: "Iniciando inferencia: Espere unos segundos." Acto seguido, utilizará la API de YOLOv5 para efectuar la detección. Al terminar, si ha habido una detección válida, dirá qué elemento ha detectado en la imagen junto con su certeza. En la siguiente imagen podemos ver un registro de la ejecución en el prompt en el que se está detectando un paso de peatones en la imagen de pruebas.

```
A.R.G.O.S. :  Iniciando inferencia: Espere unos segundos.
peaton
YOLOv5  2021-7-18 torch 1.9.0+cpu CPU

Fusing layers...
C:\Python39\lib\site-packages\torch\nn\functional.py:718: UserWarning: Named tensors and all their associated APIs are an
experimental feature and subject to change. Please do not use them for anything important until they are released as stabl
e. (Triggered internally at  ..\c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
Model Summary: 283 layers, 7263185 parameters, 0 gradients, 16.8 GFLOPs
WARNING: --img-size 200 must be multiple of max stride 32, updating to 224
image 1/1 C:\Users\usuario\Desktop\argos\peaton.png: 192x224 1 crosswalk, Done. (0.091s)
Results saved to runs\detect\exp
1 labels saved to runs\detect\exp\labels
Done. (0.140s)
A.R.G.O.S. :  paso de peatones detectado. Certeza: 66.02 %
```

Ilustración 32: Proceso de inferencia. Perspectiva del símbolo del sistema

Al terminar la inferencia, se cargará primero en la ventana emergente la imagen que hemos seleccionado. Para ver los resultados de la detección, pulsaremos en "Ver Inferencia con YOLOv5".

En ocasiones, el proceso de inferencia puede efectuar una edición que no se vea del todo bien. Esto puede deberse a la configuración del ancho de pixel (configurable desde la API de YOLOv5) que utiliza el asistente para dibujar el recuadro. Por ello, el asistente dicta la certeza de la detección, además de quedar registrado en la ventana de símbolo del sistema.

A.R.G.O.S. averigua la certeza de la detección gracias al archivo de texto con el que YOLOv5 anota las dimensiones del recuadro y el porcentaje en tanto por uno. Se lee línea a línea, y se extrae el último parámetro con dicha certeza, y luego se modifica dicha información para que sea interpretada vocalmente.

En las siguientes ilustraciones podemos ver cómo queda la ventana emergente tras finalizar la inferencia y cargar el resultado.

Ilustración 33: Imagen de prueba. Paso de peatones.

Ilustración 34: Contraste entre prueba y detección

Y a continuación, mostramos otros ejemplos de detección con un semáforo y con un límite de velocidad de 80 km/h.

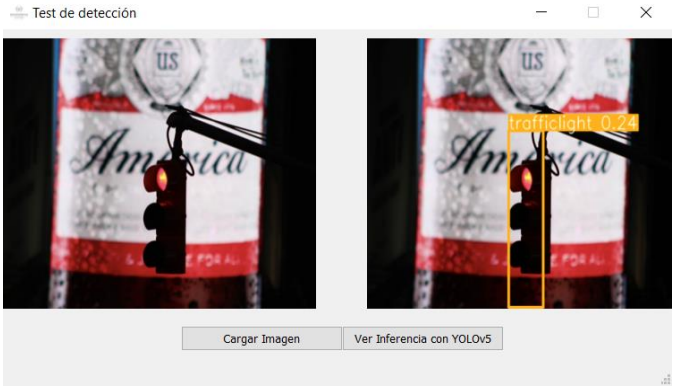


Ilustración 35: Detección de semáforo

Ilustración 36: Detección de límite de velocidad.

Una vez vista la aplicación, cabe preguntarse: ¿cómo determina el asistente qué se está viendo en cada imagen? ¿y cómo estima la certeza? En el siguiente capítulo, trataremos en profundidad todos los aspectos relativos al aprendizaje de nuestro modelo, junto con las pruebas necesarias y su validación.

5 ENTRENAMIENTO, PRUEBAS Y VALIDACIÓN

En este capítulo veremos los procedimientos para entrenar nuestro detector de imágenes, definiendo un modelo de datos a partir de un conjunto de imágenes. Esta parte es esencial para la detección de imágenes, ya que la calidad de la inferencia está supeditada a la elaboración de un conjunto de datos lo suficientemente grande. A su vez, el tiempo de convergencia del entrenamiento y la calidad de la inferencia dependerán también del arquetipo de entrenamiento de YOLO escogido. En la siguiente figura se muestran estos

arquetipos y sus características (imagen disponible en la referencia [5]).

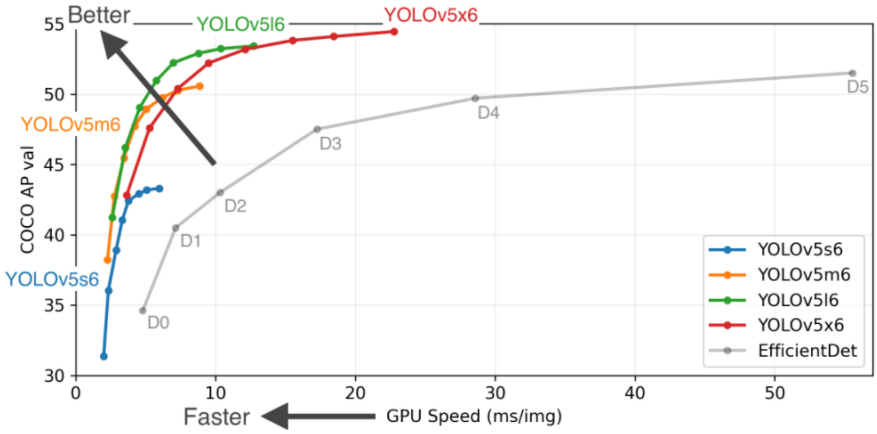


Ilustración 37: Arquetipos de YOLOv5

Y en la siguiente tabla, un resumen de sus prestaciones.

Modelo	Tamaño (pixels)	mAP^{val} 0.5:0.95	mAP^{test} 0.5:0.95	mAP^{val} 0.5:0.95	Velocidad V100 (ms)	Parámetros (M)	Flops (Operaciones en coma flotante por segundo)
YOLOv5s6	1280	43.3	43.3	61.9	4.3	12.7	17.4
YOLOv5m6	1280	50.5	50.5	68.7	8.4	35.9	52.4
YOLOv5l6	1280	53.4	53.4	71.1	12.3	77.2	117.7
YOLOv5x6	1280	54.4	54.4	72.0	22.4	141.8	229.9

Tabla 1: Prestaciones de los arquetipos de YOLOv5

Hay tres columnas con la métrica "mAP" (mean Average Precision) que no es más que una forma de cuantificar la precisión de un detector. Para explicar este concepto, veremos primero los conceptos de Falso Positivo, Falso Negativo, Verdadero Positivo, y Verdadero Negativo.

- Un **Verdadero Positivo** es el resultado de una predicción en la que se determina que es válida porque se coincide con la referencia de la que parte el modelo, y, efectivamente, se cumple.
- Un **Falso Positivo** por contra es la determinación de que una predicción es válida cuand no lo es.
- Un **Verdadero Negativo** es el resultado de una predicción en la que se determina que no es válida porque no coincide con la referencia de la que parte el modelo, acertando en su criterio.
- Un **Falso Negativo** es el resultado de descartar una muestra al determinarse que no casa con el modelo, cuando realmente sí coincide.

Seguidamente, modelaremos y definiremos la Precisión y Exhaustividad:

- La Precisión de define matemáticamente como el cociente entre los Verdaderos Positivos y la suma de los verdaderos positivos y los falsos positivos. Es decir, es una forma de medir cuántas veces se ha hecho una predicción correcta. Cuantos más falsos positivos haya, mayor será el divisor, hacienda que el valor oscile entre 0 y 1.

$$Precisión = \frac{VP}{VP + FP} \tag{5-1}$$

Ecuación 5: Precisión del modelo

- La Exhaustividad (también denominada en algunos libros *Recall*) es un parámetro más estricto que el anterior, que cuantifica la bondad de una predicción contando también los falsos negativos:

$$Exhaustividad = \frac{VP}{VP + FN} \tag{5-2}$$

Ecuación 6: Exhaustividad del modelo

Luego, para determinar la cuantía del valor de mAP durante el entrenamiento, se promedia la exhaustividad, y se acumula el estadístico para finalmente asignarle un valor entre 0 y 100. Si volvemos a mirar la tabla, podemos ver que hay una correlación entre la velocidad de un arquetipo, la carga computacional del mismo y la precisión del modelo final. Para las pruebas que se abordarán en este documento utilizaremos la version más rápida del software, que utiliza el arquetipo YOLOv5s6 (abrevidado normalemente en términos de implementación como YOLOv5s).

Abordadas las prestaciones, haremos uso del del software de YOLOv5 en dos dispositivos: un ordenador portátil con sistema operativo Windows 10, utilizando la nube de Google Colaboratory junto el módulo de Python de YOLOv5, y en la Raspberry Pi 4B.

También se pondrá de manifiesto la necesidad de utilizar la nube como medio para entrenar nuestro modelo, exponiendo los tiempos de convergencia para el entrenamiento en cada dispositivo.

5.1 Entrenamiento en la nube (Colaboratory-Google)

En este apartado trataremos todos los aspectos relacionados con la nube de Google Colaboratory. Para comenzar, mostramos el siguiente diagrama que explica conceptualmente cómo está desplegada la infraestructura de software desde la que nos ofrecen el servicio de entrenamiento en la nube.

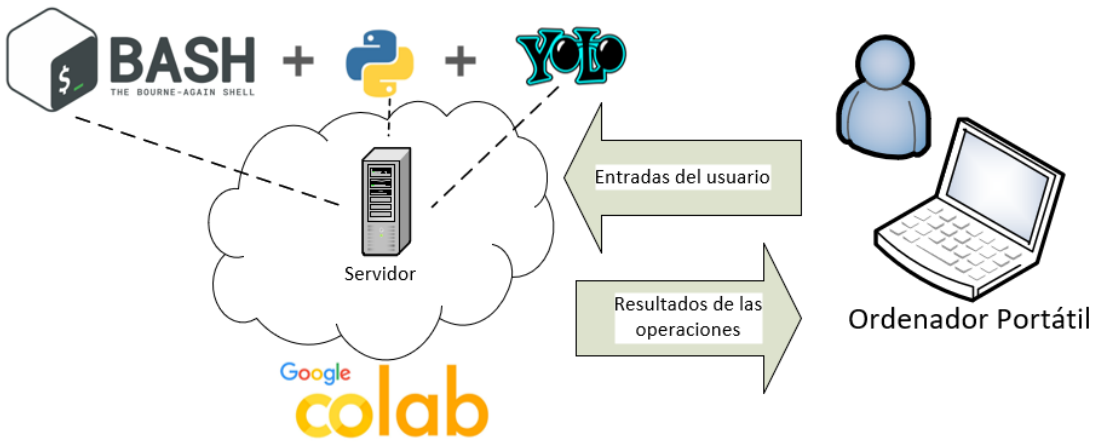


Ilustración 38: Escenario y componentes de Google Colab

Así pues, un usuario puede acceder (autenticándose con su cuenta de Google) a la plantilla del proyecto de YOLOv5 y simplemente cambiar los parámetros de entradas para entrenar el modelo.

5.1.1 Entorno y herramientas

En este epígrafe veremos dos herramientas fundamentales para poder realizar el entrenamiento en la nube: Por un lado, la nube de *Google Colab*, y por otro lado, la aplicación web de *roboflow* para poder categorizar y ordenar nuestros datasets.

5.1.1.1 Nube de Colab-Google

Google Colaboratory es un servicio en la nube basado en *Notebook* (como *Jupyter*) que permite utilizar unidades de procesamiento tensorial (*TPUs*) y unidades gráficas de procesamiento (*GPUs*) que permiten el desarrollo de software de código abierto, estándares abiertos y servicios computacionales.

En la siguiente ilustración Podemos ver la interfaz web de Google Colab con la plantilla para instalar las dependencias de YOLOv5, así como sus resultados tras la ejecución (imagen disponible en la referencia [26]).

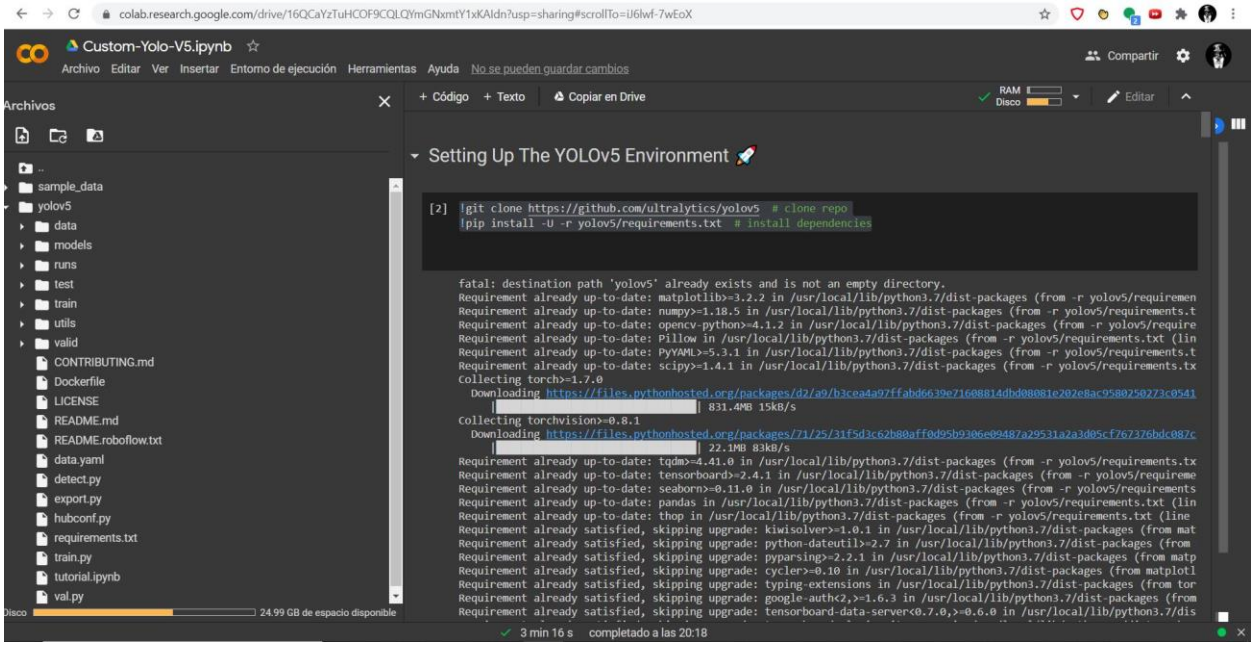


Ilustración 39: Interfaz web de *Google Colab*.

Para poder utilizar esta interfaz parametrizándola con nuestro dataset, necesitaremos utilizar *roboflow*, el cuál inyectará desde un repositorio previamente acondicionado por el usuario, el conjunto de imágenes sobre las que realizar el entrenamiento.

5.1.1.2 Roboflow

Roboflow es un entorno de desarrollo para vision artificial que ofrece un servicio de recopilación y clasificación de datos, así como de entrenamiento de los modelos, utilizando técnicas de preprocesamiento. Puede encontrarse más información en la referencia [32].

Tras subir un dataset, podemos clasificar cada imagen con una herramienta gráfica, aplicar transformaciones a cada imagen para ampliar nuestro modelo de datos y dar mayor variedad y riqueza a la inferencia. Cuanto más complejo y elaborado sea el proceso de transformación de las imágenes, más tardará en converger el modelo. Finalmente, nos permite acondicionar el dataset para ser recibido or un entorno de entrenamiento como YOLOv5, en el cuál necesitaremos 3 tipos de directorios de imágenes:

- *Training*
- *Valid*
- *Testing*

La elección del nombre de estos directorios no es casual: *roboflow* trabaja internamente con la misma

nomenclatura que utiliza YOLOv5, luego, éstos deben ser los nombres para que haya una correspondencia unívoca al ejecutar los comandos pertinentes durante el muestreo de imágenes. Los ratios en los que se suelen distribuir las imágenes se puedne apreciar en la siguiente imagen.

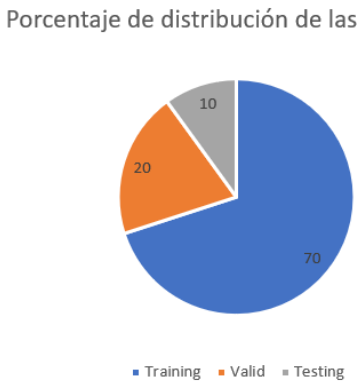


Ilustración 40: Ratio de distribución de las imágenes del *dataset*

En la siguiente ilustración podemos ver cómo es el entorno de trabajo de *roboflow*, junto con el dataset utilizado para generar el modelo de datos para la detección de señales de tráfico.

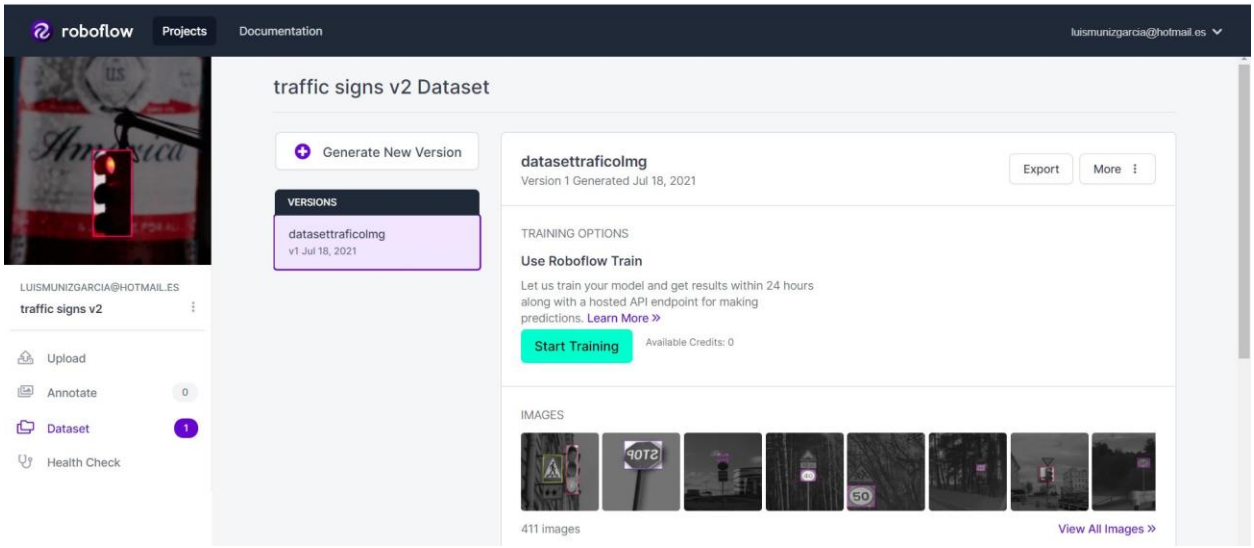


Ilustración 41: Interfaz web de *roboflow*

En el siguiente epígrafe veremos una muestra del dataset utilizado para el entrenamiento.

5.1.2 Colección de imágenes de muestras

Como ya hemos razonado en apartados anteriores, es importante elaborar un buen conjunto de muestras de imágenes para fortalecer la inferencia de nuestro modelo de detección de imágenes. Hemos acudido al repositiorio de la referencia [27] del cuál hemos extraido las muestras. A continuación, hemos subido a *roboflow* un total de 175 imágenes, junto con sus etiquetas. En los anexos C y D se pueden apreciar dos ejemplos de cómo están parametrizadas las clasificaciones de las imágenes.

En la siguiente ilustración se puede apreciar un muestreo del dataset.

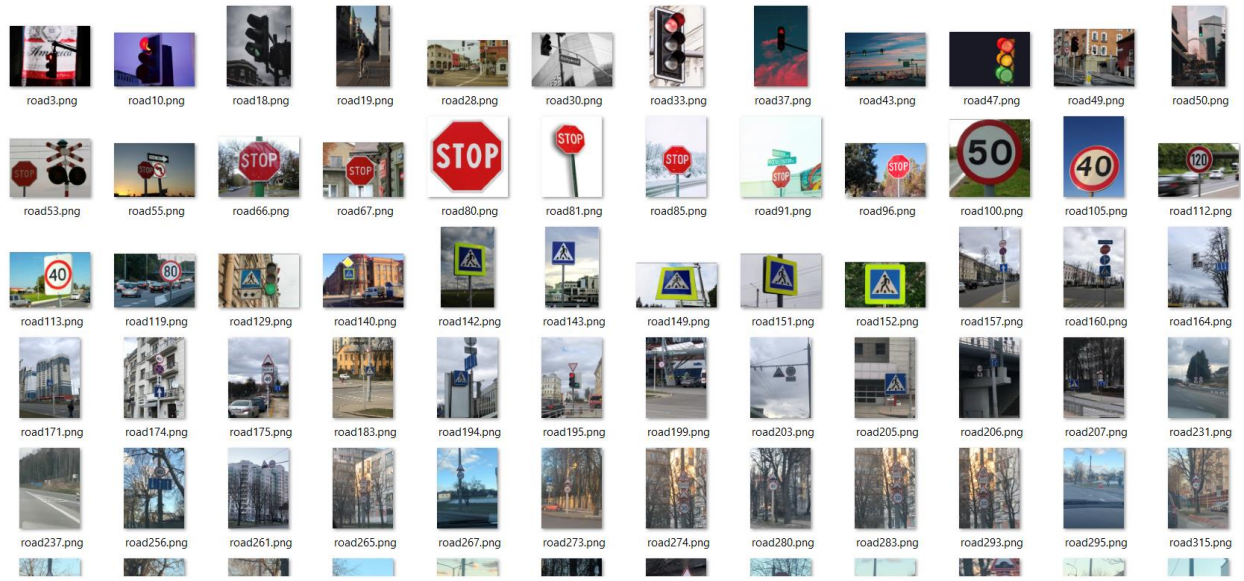


Ilustración 42: Dataset para la generación del modelo.

Y en la siguiente podemos ver un ejemplo de clasificación.

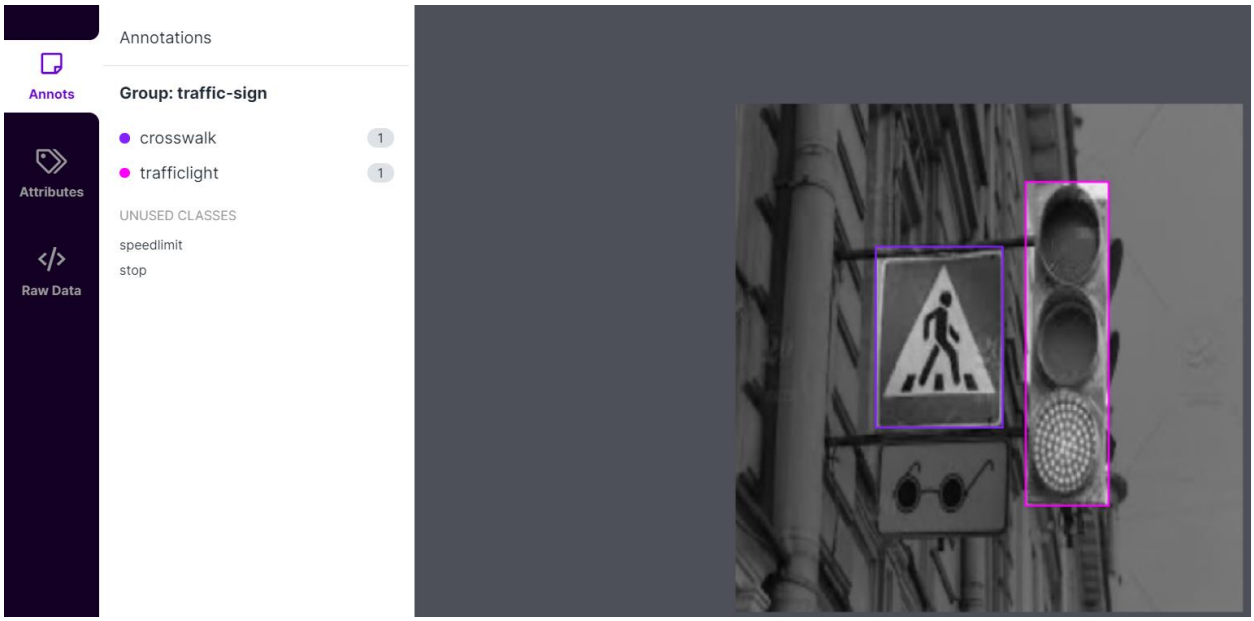


Ilustración 43: Clasificación de imágenes en roboflow

Aplicaremos este procedimiento a nuestro conjunto de imágenes, para finalmente exportar el dataset con un enlace para incrustar en nuestra plantilla de Google Colab.

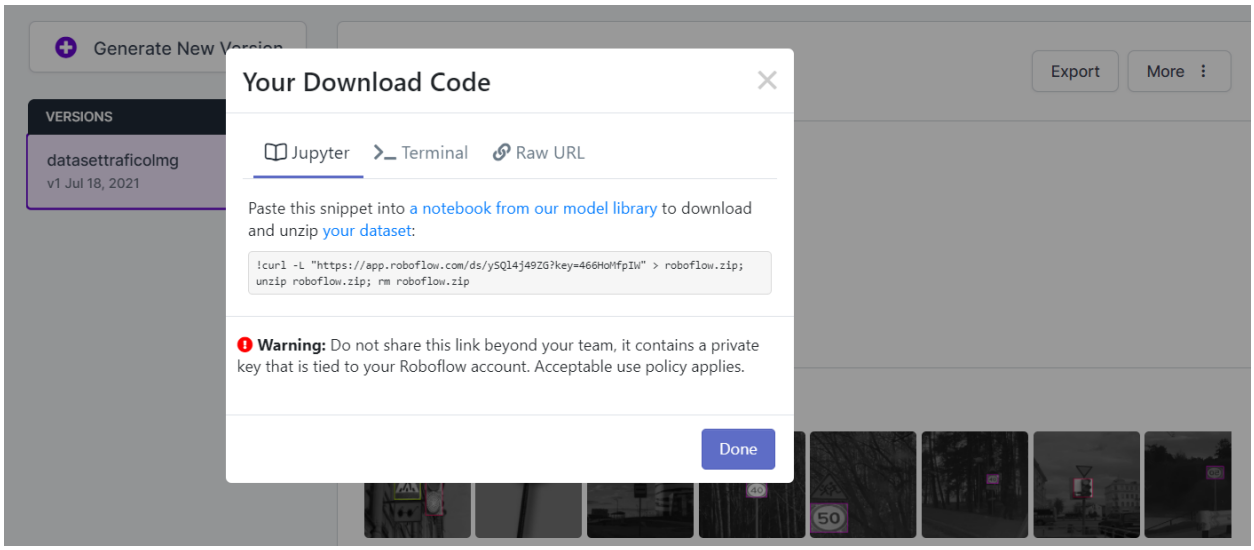


Ilustración 44: Exportación del *dataset* vía web

Con todo esto, ya estamos listos para proceder al entrenamiento. El enlace proporcionado contiene un comando para descargar contenido web utilizando el protocolo HTTPS. Dicho contenido será un zip con el dataset y un archivo de preprocesamiento en de extensión YAML, que es lo que necesita YOLOv5 para iniciar el entrenamiento. Dicho archivo se denominará "data.yaml".

5.1.3 Entrenamiento del modelo

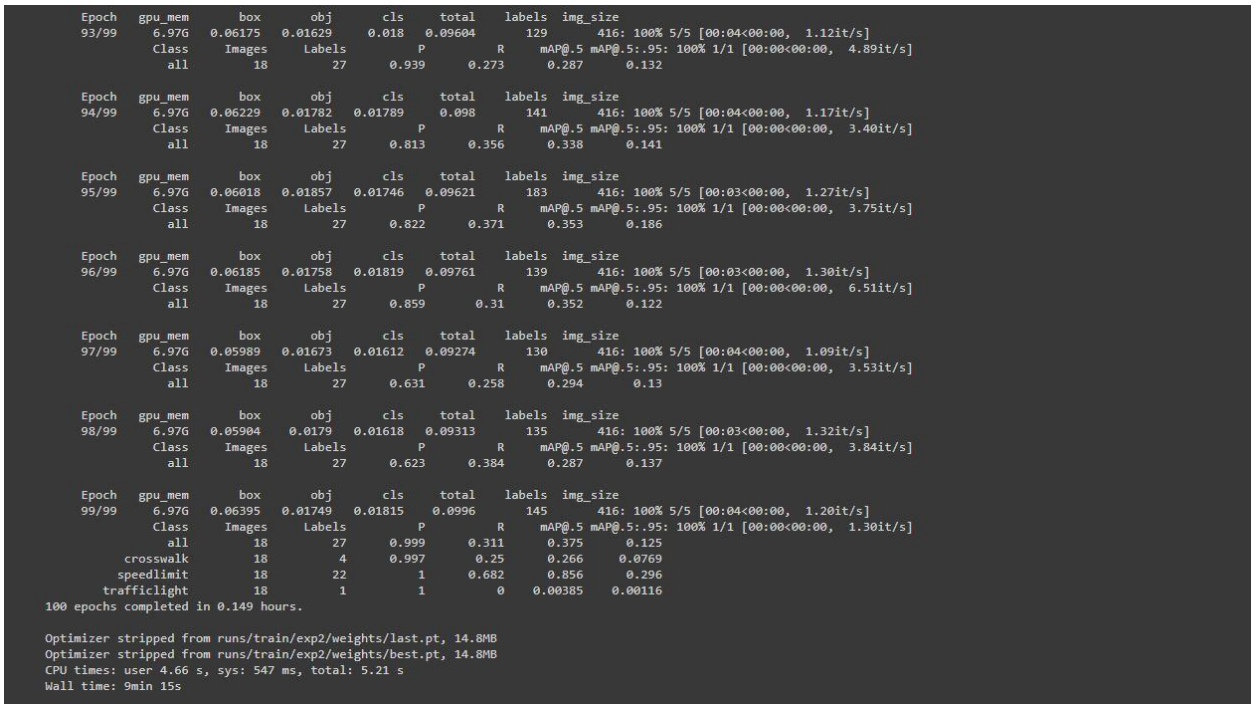
Tras acondicionar nuestra plantilla descargando todas las dependencias necesarias y ejecutando las operaciones pertinentes en el terminal, procedemos al entrenamiento de nuestro modelo. Para ello, desde la celda correspondiente del *Google Colab*, introducimos los siguientes comandos:

```
%%time
```

```
%cd /content/yolov5/
```

```
!Python train.py --img 416 --batch 80 --epochs 100 --data './data.yaml' --cfg ./models/custom_yolov5s.yaml --weights ''
```

Con esto damos paso al proceso de entrenamiento en el que se calcularán las etapas necesarias para la convergencia dle modelo, el uso del espacio del disco duro así como la memoria utilizada. En la siguiente imagen podemos ver la finalización del entrenamiento y su salida por pantalla.



```
      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      93/99     6.97G   0.06175   0.01629     0.018   0.09604       129       416: 100% 5/5 [00:04<00:00,  1.12it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  4.89it/s]
                   all         18         27      0.939      0.273      0.287      0.132

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      94/99     6.97G   0.06229   0.01782   0.01789     0.098       141       416: 100% 5/5 [00:04<00:00,  1.17it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.40it/s]
                   all         18         27      0.813      0.356      0.338      0.141

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      95/99     6.97G   0.06018   0.01857   0.01746   0.09621       183       416: 100% 5/5 [00:03<00:00,  1.27it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.75it/s]
                   all         18         27      0.822      0.371      0.353      0.186

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      96/99     6.97G   0.06185   0.01758   0.01819   0.09761       139       416: 100% 5/5 [00:03<00:00,  1.30it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  6.51it/s]
                   all         18         27      0.859       0.31      0.352      0.122

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      97/99     6.97G   0.05989   0.01673   0.01612   0.09274       130       416: 100% 5/5 [00:04<00:00,  1.09it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.53it/s]
                   all         18         27      0.631      0.258      0.294       0.13

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      98/99     6.97G   0.05904    0.0179   0.01618   0.09313       135       416: 100% 5/5 [00:03<00:00,  1.32it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  3.84it/s]
                   all         18         27      0.623      0.384      0.287      0.137

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      99/99     6.97G   0.06395   0.01749   0.01815    0.0996       145       416: 100% 5/5 [00:04<00:00,  1.20it/s]
                 Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:00<00:00,  1.30it/s]
                   all         18         27      0.999      0.311      0.375      0.125
             crosswalk         18          4      0.997       0.25      0.266     0.0769
            speedlimit         18         22          1      0.682      0.856      0.296
           trafficlight         18          1          1          0    0.00385    0.00116
100 epochs completed in 0.149 hours.

Optimizer stripped from runs/train/exp2/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/exp2/weights/best.pt, 14.8MB
CPU times: user 4.66 s, sys: 547 ms, total: 5.21 s
Wall time: 9min 15s
```

Ilustración 45: Resultados del entrenamiento y tiempo necesario (*Google Colab*)

Es decir, para el dataset proporcionado, el servidor de Google ha necesitado **9 minutos y 15 segundos** para elaborar nuestro modelo de inferencia. Los archivos generados tras esta operación son los pesos que necesitará YOLOv5 para la inferencia: *best.pt* y *last.pt*

- *Best.pt* son los mejores pesos recolectados durante el entrenamiento
- *Last.pt* es el peso de la última etapa del entrenamiento.

En la práctica, sólo necesitaremos *best.pt* para efectuar una inferencia. Podemos visualizar cómo ha ido el entrenamiento viendo una imagen que resume el proceso de inferencia, *val_batch0_pred.jpg*.

Ilustración 46: Imagen *val_batch0_pred.jpg*

Vemos que en el muestreo hay una gran tasa de acierto, si bien el grado de verosimilitud en la predicción del modelo oscila entre el 30 % y el 80 % de certeza.

En el siguiente apartado veremos un ejemplo de inferencia con una imagen que no pertenece al dataset, así como el comando necesario para efectuar dicha operación.

5.1.4 Inferencia

Una vez obtenido el modelo, procederemos con una prueba para ver que funciona. En la misma interfaz web, ejecutamos la celda que contiene el siguiente commando:

```
%cd /content/yolov5/
!Python detect.py --weights/content/yolov5/runs/train/exp/weights/best.pt
--img 416 --conf 0.4 --source ./test/images
```

Es decir, le especificamos la ruta hasta los pesos, y la ruta hasta las imágenes de pruebas sobre las que

queremos hacer la inferencia. En nuestro caso, utilizamos la siguiente imagen.

Ilustración 47: Imagen de prueba para la inferencia (original)

Y tras la operación de inferencia, obtenemos la siguiente imagen.

Ilustración 48: Imagen de prueba para la inferencia (detectado)

Es decir, tras muestrear con una señal de límite de velocidad a 70 km/h, hemos obtenido dos detecciones:

1. Detección parcial del límite de velocidad en la mitad izquierda de la señal, capturando el dígito '7' del '70', con una certeza en la predicción del 48%.
2. Detección global de la señal de tráfico en la cuál se capturan ambos dígitos, y las tres circunferencias concéntricas con el patrón de color blanco-rojo-blanco, con una certeza en la predicción del 28 %.

Lo que nos queda por plantear es si la inferencia ha sido buena o no, y argumentar el por qué. Todo esto lo veremos en el siguiente epígrafe en el que extraeremos la conclusión en base a los resultados obtenidos.

5.1.5 Resultados

Valiéndonos de los módulos de *tensor board*, podemos analizar las siguientes métricas.

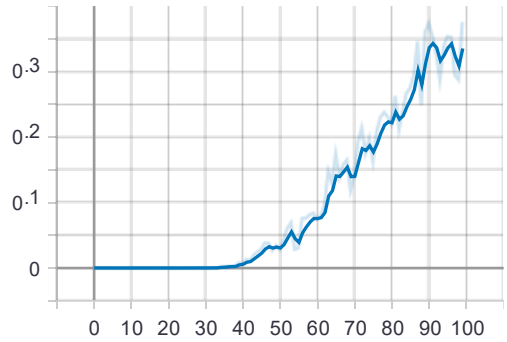


Ilustración 49: mAP 0.5 (*Google Colab*)

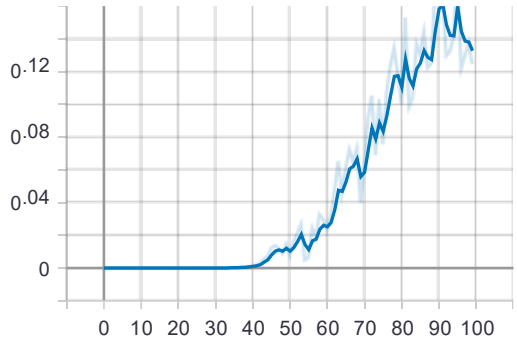


Ilustración 50: mAP 0.5:0.95 (*Google Colab*)

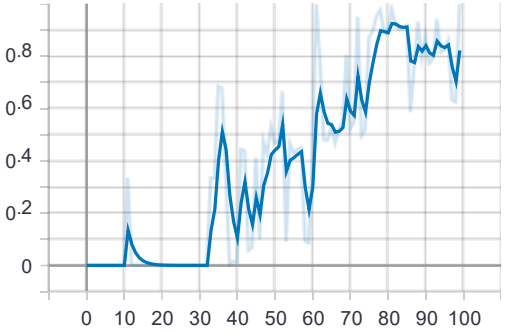


Ilustración 51: Precisión (*Google Colab*)

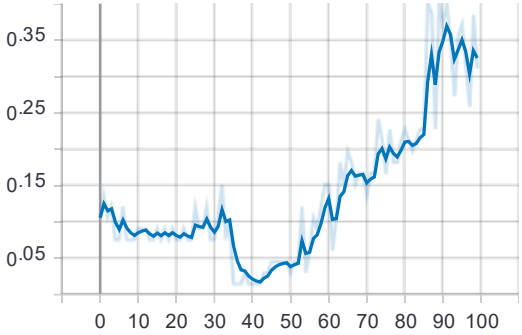


Ilustración 52: Exhaustividad (*Google Colab*)

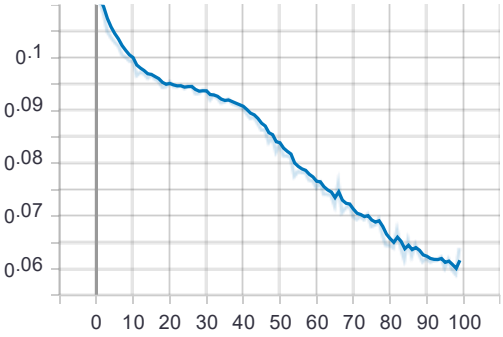


Ilustración 53: Estrechamiento del recuadro de captura en la predicción (*Google Colab*)

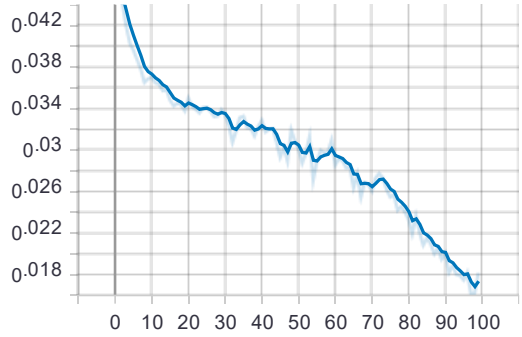


Ilustración 54: Error en la detección de clases (*Google Colab*)

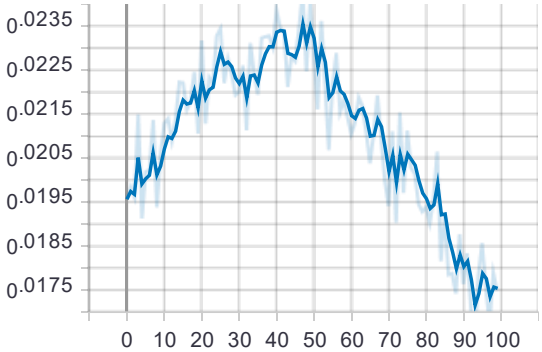


Ilustración 55: Error en la detección de objetos (Google Colab)

Es decir, nuestro modelo ha ido cobrando precisión en el transcurso de las etapas, disminuyendo los errores en la predicción y aumentando también la exhaustividad, y modificando acorde las métricas que dependen de estos factores. También podemos apreciar oscilaciones en las etapas intermedias. Esto puede deberse al muestreo de imágenes utilizado (175 imágenes).

Un *dataset* se considera que es lo suficientemente extenso para proporcionar una buena inferencia cuando su tamaño ronda las 1000 imágenes, ocasionando a su vez un mayor retardo en la convergencia.

Luego, queda patente que el modelo es aceptable y que detecta las imágenes proporcionadas con suficiente certeza. Si quisieramos mejorar el modelo, bastaría con aumentar el dataset y esperar a la conclusión del entrenamiento en la nube.

5.2 Entrenamiento en Raspberry Pi 4B

En este epígrafe veremos todos los procedimientos para entrenar, efectuar detecciones y validar nuestro modelo en una Raspberry Pi 4B.

5.2.1 Entorno y herramientas

Para el entrenamiento en nuestra Raspberry Pi 4B necesitaremos tener instalado el sistema operativo Raspbian con la distribución de Buster, la cuál está basada en Debian 10.

Además, necesitaremos descargarnos el código fuente del repositorio de YOLOv5 en Github. Por ultimo, usaremos la línea de comandos como interfaz de control del software.

5.2.2 Colección de imágenes de muestras

En aras de comparar ambos entornos, utilizaremos el mismo dataset: 'datasettraficolmg', el cuál pudimos ver en el apartado anterior de Google Colab.

5.2.3 Entrenamiento del modelo

Para la realización del entrenamiento hay que modificar ligeramente el comando.

En vez de:

```
Python train.py --img 416 --batch 80 --epochs 100 --data './data.yaml' --
cfg ./models/custom_yolov5s.yaml --weights ''
```

usaremos:

```
Python3 train.py --img 416 --batch 4 --epochs 100 --data './data.yaml' --
cfg ./models/custom_yolov5s.yaml --weights ''
```

Ahora, lo natural sería preguntarnos: ¿por qué?

Cuando usamos la nube de Google Colab, especificamos que el servidor usara un batch de 80 (es decir, que tomase un muestreo de 80 imágenes por cada etapa). Gracias a esto, se consiguió una rápida convergencia. Esto es posible gracias a los recursos de memoria del propio servidor. En cambio, las prestaciones de un SBC son más reducidas (acordes a su coste). Si intentásemos entrenar con un batch de 80 en la Raspberry Pi, nos saltaría un error de memoria insuficiente.

Para paliar este hecho, haremos uso de una de las técnicas de entrenamiento ya vistas en la introducción teórica: el entrenamiento online, también conocido como entrenamiento por mini lotes.

La forma de averiguar cuál es el máximo batch que permite el SBC es utilizando métodos de tanteo. A continuación, se muestra una tabla con las iteraciones probadas en orden cronológico:

Tamaño de Batch	¿funciona?
80	No
40	No
10	No
1	Sí
2	Sí
4	Sí

Tabla 2 : Ensayo-Error con el entrenamiento por mini-batches

Así pues, el tamaño final de muestreo escogido para el entrenamiento fue de 4 imágenes por etapa. Esto aumenta consierablemente el tiempo de espera (93.861 horas) que es aproximadamente unas 630 veces más que el tiempo de convergencia en la nube, pero es la única forma de efectuar esta fase en nuestra Raspberry Pi.

En la siguiente ilustración Podemos ver una captura de pantalla con el resultado final del entrenamiento:

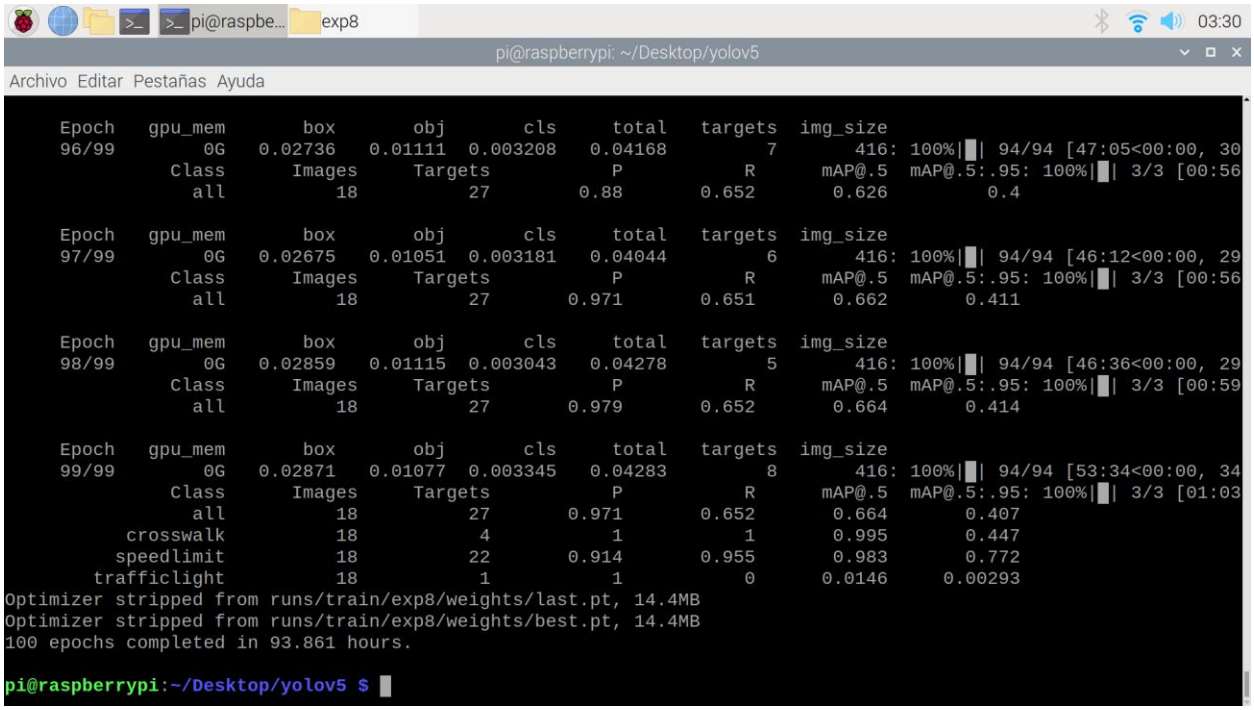


Ilustración 56: Finalización del entrenamiento (Raspberry Pi 4B)

Una vez más, se han generado los archivos ‘best.pt’ y ‘last.pt’, con el resultado del entrenamiento de nuestro modelo. En el siguiente epígrafe probaremos la calidad de la inferencia con imágenes de muestra.

5.2.4 Inferencia

Una vez generado nuestro modelo tras la fase de entrenamiento, ejecutamos el siguiente comando en el terminal:

```
Python detect.py --weights runs/train/exp/weights/best.pt
--img 416 --conf 0.4 --source ../test/images
```

Tras lo cual, se creará un directorio "detect" ubicado al mismo nivel que el directorio "train", y dentro se ubicará un subdirectorio "exp" (de *experiment*). El resultado de la ejecución se puede observar en la siguiente ilustración.

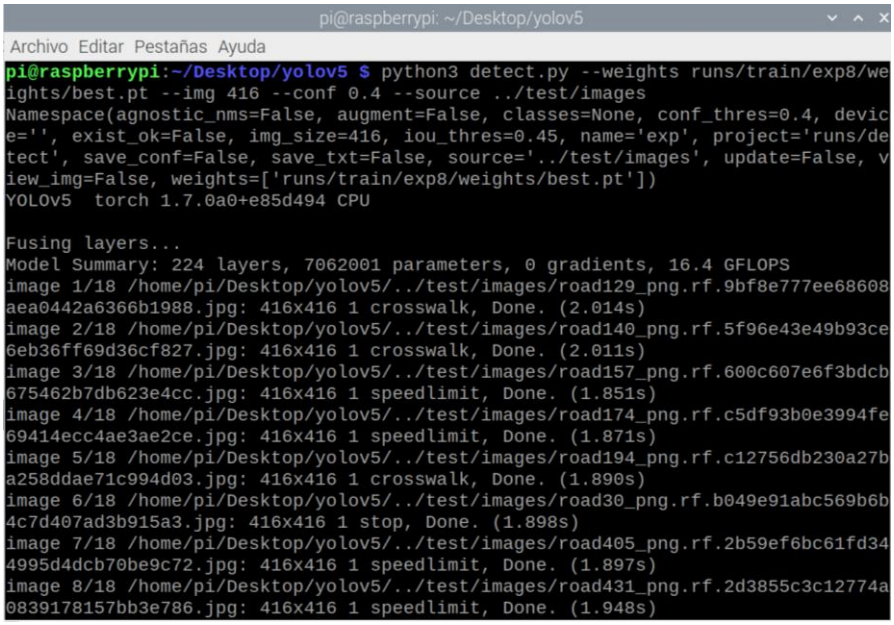


Ilustración 57: Terminal de la detección en Raspberry pi 4B

Se nos indica las el directorio donde se está realizando la detección, el tipo de clase detectada, el tiempo que ha tardado en detectar y los parámetros de operación del modelo. Si accedemos al directorio `runs/detect/exp/`, encontraremos el resultado de la detección extraído del directorio de prueba (*test*). A continuación, se muestra una selección de imágenes donde se aprecia la detección de las 4 clases, además de los dos modos de detección (con y sin certeza en la predicción).

Ilustración 58: Detección de señal de STOP (Raspberry Pi 4B)

Ilustración 59: Detección de límite de velocidad (Raspberry Pi 4B)

Ilustración 60: Detección de paso de peatones (Raspberry Pi 4B)

Ilustración 61: Detección múltiple (Raspberry Pi 4B)

A priori, parece que el rendimiento del entrenamiento ha sido satisfactorio. En el siguiente epígrafe comentaremos la validez de estos resultados.

5.2.5 Resultados

Una vez más, comentaremos las métricas extraídas gracias a tensorboard.

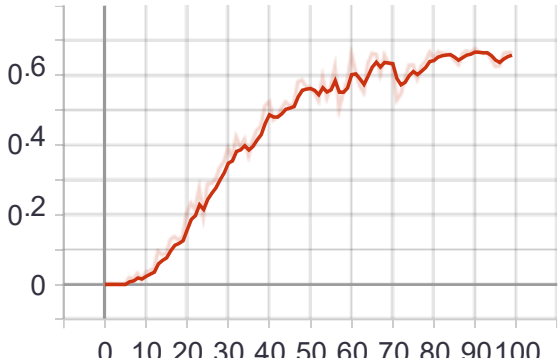


Ilustración 62: mAP 0.5 (Raspberry Pi 4B)

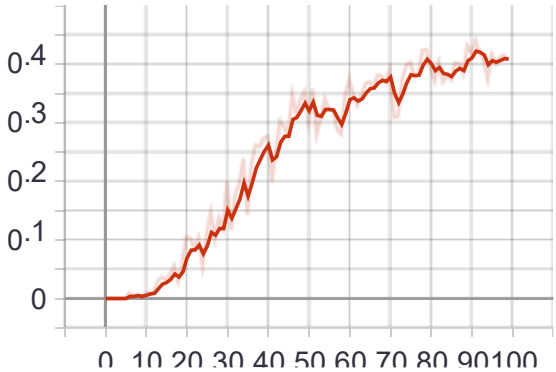


Ilustración 63: mAP 0.5:0.95 (Raspberry Pi 4B)

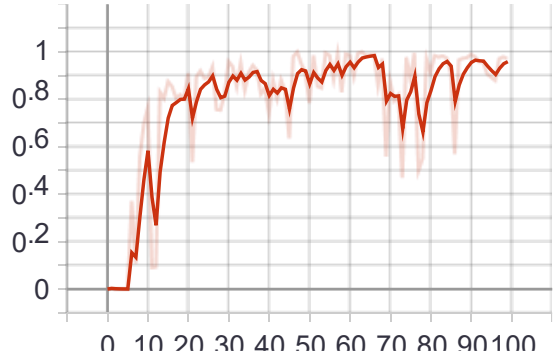


Ilustración 64: Precisión (Raspberry Pi 4B)

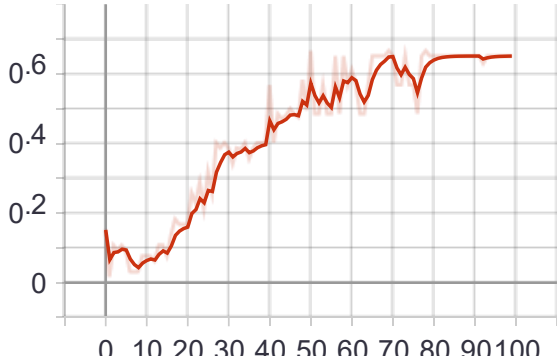


Ilustración 65: Exhaustividad (Raspberry Pi 4B)

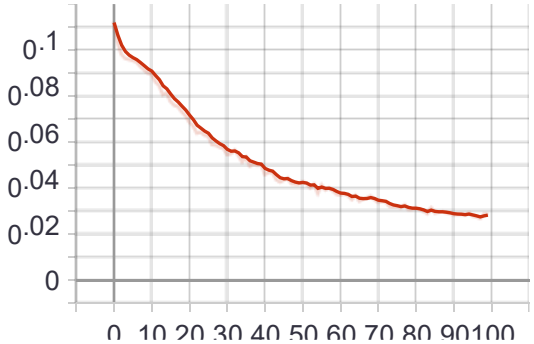


Ilustración 66: Estrechamiento del recuadro de captura en la predicción (Raspberry Pi 4B)

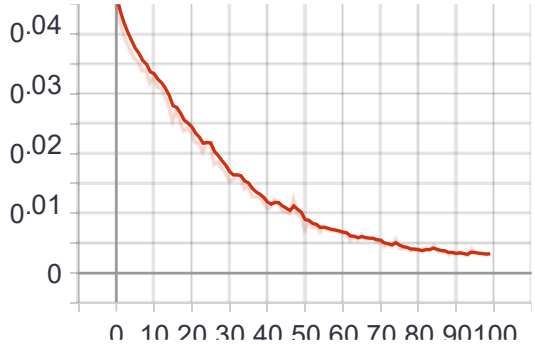


Ilustración 67: Error en la detección de clases (Raspberry Pi 4B)

Ilustración 68: Error en la detección de objetos (Raspberry Pi 4B)

Una vez más, observamos que la precision sigue un comportamiento exponencial, mientras que el error en la clasificación sigue una exponencial decreciente conforme avanzan las etapas en el proceso de aprendizaje. Con esto se logra un modelo de alta calidad, lo cuál queda reflejado por los porcentajes de certeza en la predicción (cercanos al 90 % de certeza). Cabe destacar la ausencia de sobreoscilaciones en contraste con el procedimiento homólogo en la nube de Google.

Con esto terminamos la extracción de resultados del entrenamiento en la nube. En el siguiente apartado veremos una comparativa en base a los resultados de ambos entornos.

5.3 Comparación de los entrenamientos en ambos entornos.

En este apartado veremos uno de los aspectos centrales de este trabajo, y es analizar cuán viable es realizar un entrenamiento en una Raspberry Pi 4 B, comparándolo con su alternativa en la nube.

En la siguiente tabla comparative, vemos sendos modelos de detección y sus métricas para ambos entornos de entrenamiento.

Entorno	Arquetipo utilizado	Tiempo de convergencia	dataset	Resultado de la inferencia
Raspberry Pi 4B	YOLOv5s.pt	93 horas, 51 minutos y 39 segundos	‘datasettraficolmg’	Muy Bueno.
Google Colab	YOLOv5s.pt	9 minutos y 15 segundos	‘datasettraficolmg’	Aceptable

Tabla 3: Comparación de eficiencia de ambos entornos

Es bastante significativa la diferencia entre los tiempos de convergencia: Ambos modelos parten del mismo dataset, sin embargo, la precisión en la detección y la disminución del error en la identificación varían notablemente en función del entorno: en la nube de Google hay más oscilaciones en la fase de entrenamiento.

Esto puede deberse a la diferencia del número de batch en cada entorno. YOLOv5 posee un tamaño de batch agnóstico (es decir, que ante un diferente tamaño de batch, proporciona un resultado similar en la calidad del entrenamiento) y aún así hay diferencias en la certeza de la detección.

En la siguiente tabla comparativa podemos observar todas las métricas de ambos entornos de entrenamiento.

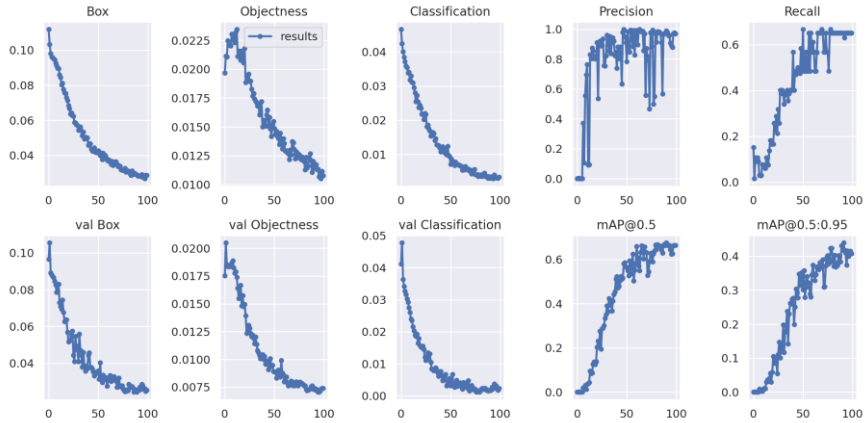
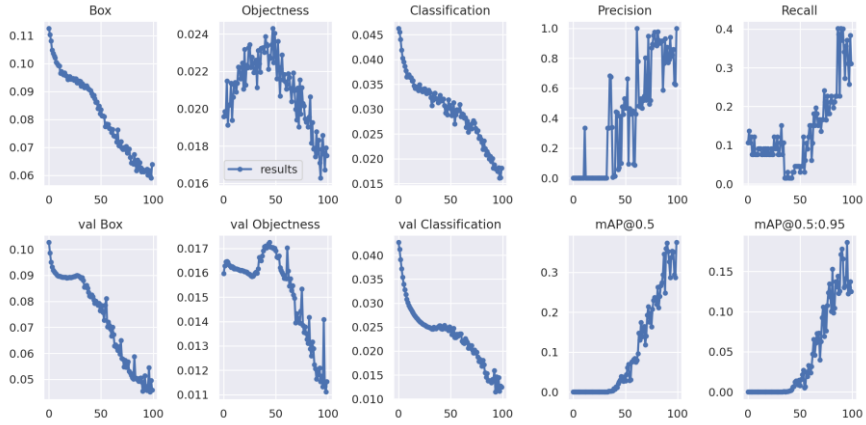
Entorno	Métricas
Raspberry Pi 4B	 Ilustración 69: Métricas de Raspberry Pi 4B
Google Colab	 Ilustración 70: Métricas de *Google Colab*

Tabla 4: Comparativa de métricas en ambos entornos

Con todo esto presente, deducimos que usando un entrenamiento online se obtiene una calidad superior en el modelo resultante.

En el siguiente capítulo extraeremos las conclusiones en base a los resultados obtenidos.

6 CONCLUSIONES

Hemos visto cómo entrenar nuestro modelo en dos entornos, cada uno con sus ventajas y sus inconvenientes.

Luego, cabe plantearse las siguientes preguntas:

- ¿Qué entorno es mejor?
- ¿Es viable realizar un entrenamiento en la Raspberry Pi 4B?
- ¿Qué soluciones hay para los problemas encontrados en el entrenamiento de ambos entornos?

Para responder a la primera pregunta, hace falta plantear varios factores de interés para el usuario.

El primero de ellos es evidentemente el tiempo. Dependiendo del tiempo disponible para realizar el entrenamiento, se escogerá un entorno u otro en función de lo restrictivo de este factor. Así pues, con poco tiempo se recomienda utilizar el entrenamiento en la nube, ya que (con el arquetipo más simple de YOLOv5) converge en menos de 10 minutos, y si se dispone de un intervalo de 7 días, la solución del entrenamiento local en la SBC es factible.

El segundo factor relevante es la privacidad: al hacerse un entrenamiento en la nube utilizando una plantilla como la que hemos visto, se le avisa al usuario de que el Notebook es público. Si bien el servidor borra la caché utilizada tras un periodo de inactividad por parte del usuario (o incluso al finalizar sesión) es cierto que durante su utilización se dejan expuestos los resultados del entrenamiento. Esto es crítico para escenarios en los que una empresa pretende evitar que se vulnere su propiedad intelectual, ya sea la del dataset utilizado o del modelo de inferencia generado.

El tercer factor es la complejidad en la preparación del entorno: Configurar la Raspberry Pi instalando las dependencias es notoriamente más tedioso que cargar un Notebook con una plantilla en la que hay que hacer modificaciones menores. Muchas veces hay fallos con las dependencias y problemas de compatibilidad que el usuario deberá paliar, lo cuál requiere también un tiempo considerable, que puede chocar frontalmente con el primer factor.

Luego, no hay una respuesta tajante y contundente, hay un conjunto de circunstancias y matices que harán preferible una solución frente a otra.

Para la segunda pregunta, la respuesta es “depende”. Es viable en la medida en la que se ha podido realizar, pero hay que considerar que se ha tardado casi 100 horas. Si no tenemos conectado nuestra SBC a un suministro de alimentación ininterrumpido, corremos el peligro de que un apagón se lleve por delante todas las horas de entrenamiento, teniendo que volver a empezar desde cero. En relación a los factores anteriormente mencionados, el tiempo del que se disponga también condiciona en gran medida la viabilidad del entrenamiento en la Raspberry Pi 4B.

Por último, para la tercera pregunta planteamos las siguientes consideraciones para paliar algunos de los inconvenientes anteriormente mencionados:

- Para la privacidad, se puede copiar la estructura del Notebook de Google Colab importándola a la propia nube. También se puede cambiar la configuración de la privacidad, de forma que el proyecto no sea accesible públicamente. Todo esto hace que se alcancen unos mínimos aceptables en lo relativo al factor de la privacidad.
- Para la calidad en el modelo de inferencia en la nube, teniendo en cuenta las capacidades del servidor de Google, propondría cambiar el arquetipo de YOLOv5 utilizado (en vez de YOLOv5s, utilizar YOLOv5m o cualquier versión superior). También se puede cambiar la parametrización del Batch, reduciéndola y logrando un aprendizaje online. Esto incrementará el tiempo de convergencia, pero logrará unas métricas que se ajusten mejor al modelo teórico, disminuyendo las sobreoscilaciones en las curvas de error.

- Por último, para los problemas derivados del suministro eléctrico de la Raspberry Pi, existen varios dispositivos modulares que pueden acomplarse a nuestra SBC, dotándole de un suministro auxiliar accionado por software mediante microcontroladores, los cuales monitorizan que la tensión de alimentación nunca esté por debajo de cierto umbral operativo.

Ilustración 71: Fuente de alimentación UPS ininterrumpible Innovateking-EU Raspberry Pi con baterías 18650

Con todas estas consideraciones, podemos esocoger cualquiera de los dos entornos para realizar el entrenamiento, atendiendo únicamente a los factores de interés.

En mi opinión, la solución de entrenamiento en la nube es más interesante y atractiva, haciendo innecesario entrenar en una Raspberry Pi. De hecho, YOLOv5 tiene sus propias librerías en Python, de forma que podemos entrenar y hacer inferencias en nuestro ordenador, que normalmente dispondrá de unas prestaciones de software y hardware que disminuirán el tiempo de convergencia considerablemente.

Esta etapa es realmente la más conflictiva, porque para efectuar la inferencia necesitaremos los archivos de extensión ".pt", con independencia de dónde se hayan generado. Luego, para un entrenamiento usaría un dispositivo mejor capacitado en cuanto a prestaciones técnicas, y para efectuar la detección usaría el medio que requeriese el caso de uso.

Si estamos implementando un sistema de detección en tiempo real de señales de tráfico, se podría utilizar una Raspberry pi 4B con un sistema de alimentación ininterrumpida dentro del vehículo, habilitando una cámara mediante las interfaces disponibles de la SBC y acoplándola en el salpicadero.

Sería una propuesta interesante para un trabajo de fin de carrera, partiendo de todo lo explicado en este proyecto.

ANEXO A: PASOS PARA LA INSTALACIÓN DE YOLOV5 EN RASPBERRY PI 4 B

Para descargar con éxito el software de YOLOv5, hemos seguido los pasos que hay en un tutorial [42] en el que nos indican cómo instalarlo en una Raspberry Pi 4 modelo B debidamente preparada y configurada. En nuestro caso, hemos escogido una versión del sistema operativo de 32 bits pese a que el Hardware del dispositivo permita una instalación de un sistema operativo de 64 bits. Cabe mencionar, que al tratar con un procesador ARM, muchas de las dependencias de PyPi no podremos instalarlas usando el repositorio remoto de módulos, y tendremos que buscar individualmente los *wheels* necesarios para instalar las dependencias. En algunos casos como se comentará más adelante hemos tenido que hacer un downgrade a los requisitos técnicos ubicados en *yolov5/requirements.txt*.

1. Descargar de github el repositorio original: https://github.com/ultralytics/yolov5
2. Descomprimir el archivo comprimido con el proyecto. La rama del proyecto sería la *master*, que es la última versión estable del mismo. Le cambiamos el nombre al directorio padre a "yolov5".
3. Abrimos un terminal y escribimos por línea de comandos: "`pip3 install numpy`". Con esto descargamos el módulo de numpy, que es fundamental para el cálculo de la IA.
4. Seguidamente, instalamos las librerías de matpotlib con "`pip3 install matplotlib`".
5. Descargar el wheel necesario para la versión de torch: https://github.com/Kashu7100/pytorch-armv7l/blob/main/torch-1.7.0a0-cp37-cp37m-linux_armv7l.whl . Ésta es una versión no oficial de Torch, por lo que modificamos *requirements.txt* y cambiamos el requisito mínimo de Torch a >= 1.6.0. Con esto, nos debría coger sin problema nuestra versión, la 1.7.0a0.
6. Desde el terminal, introducimos el siguiente comando desde el directorio donde esté ubicado el Wheel: "`pip3 install torch-1.7.0a0-cp37-cp37m-linux_armv7l.whl`"
7. Descargamos el wheel de torchvision y hacemos un downgrade en el archivo de requisitos a la version 0.4: https://github.com/nmilosev/pytorch-arm-builds/blob/master/torchvision-0.4.0a0%2Bd31eafa-cp37-cp37m-linux_armv7l.whl
8. Abrimos nuevamente el terminal y tecleamos: "`pip3 install torchvision-0.4.0a0+d31eafa-cp37-cp37m-linux_armv7l.whl`"

Con todo esto, no deberíamos obtener ningún tipo de error por parte del instalador automático de requirements.txt. Finalmente, introducimos el siguiente comando: "`pip3 install -U -r yolov5/requirements.txt`" y esperamos hasta que termina de instalar las dependencias restantes.

ANEXO B: PASOS PARA LA CONFIGURACIÓN Y EL ENTRENAMIENTO DE LA NUBE DE GOOGLE COLAB

En este anexo se presentan todos los pasos necesarios tabulados para configruar y entrenar un modelo de detección de YOLOv5

Paso	comandos	Explicación

1	`!git clone https://github.com/ultralytics/yolov5 # clone repo`
	`!pip install -U -r yolov5/requirements.txt`
	`# install dependencies`
	`#installing for google colab GPU use`
	`!pip install torch==1.6.0+cu101 torchvision==0.7.0+cu101 -f https://download.pytorch.org/whl/torch_stable.html`
	`%cd /content/yolov5`
	`!ls`
		Clonamos el repositorio de github, instalamos las dependencias y hacemos especial hincapié de las más conflictivas: torch==1.6.0+cu101
		Torchvision==0.7.0+cu101 -f

2	`import torch`
	`from IPython.display import Image, clear_output # for displaying images`
	`from utils.google_utils import gdrive_download # for downloading models/datasets`
	`clear_output()`
	`print('Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))`
		Comprobamos la correcta instalación de las dependencias en el entorno de Python, comprobando la versión de las mismas

3	`!curl -L "https://app.roboflow.com/ds/ySQl4j49ZG?key=466HoMfpIW" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip` `%cat data.yaml`	Descargamos de roboflow el nuestro dataset, lo descomprimimos, extraemos el fichero "data.yaml" y eliminamos el archivo comprimido para economizar espacio en el disco duro del servidor

4	`import yaml`
	`with open("data.yaml", 'r') as stream:`
	`  num_classes = str(yaml.safe_load(stream)['nc'])`
	`#customize iPython writefile so we can write variables`
	`from IPython.core.magic import register_line_cell_magic`
	`@register_line_cell_magic`
	`def writetemplate(line, cell):`
	`  with open(line, 'w') as f:`
		Definimos el número de clases que contiene el archivo YAML.
		Este archivo contiene información sobre el número de clases y las etiquetas requeridas en el proyecto.

	`f.write(cell.format(**globals()))`	

5	`%%writetemplate /content/yolov5/data.yaml`
	
	`train: ./train/images`
	`val: ./valid/images`
	
	`nc: 4`
	`names: ['crosswalk', 'speedlimit', 'stop', 'trafficlight']`
	`#Let's check the data.yaml file for confirmation`
	
	`%cat data.yaml`
		Acondicionamos las rutas del proyecto para efectuar las operaciones en el modelo.

6	`with open(r'data.yaml') as file:`
	`  # The FullLoader parameter handles the conversion from YAML`
	`  # scalar values to Python the dictionary format`
	`  clear_output()`
	`  labels_list = yaml.load(file, Loader=yaml.FullLoader)`
	
	`  label_names = labels_list['names']`
	`print("Number of Classes are {}, whose labels are {} for this Object Detection project".format(num_classes,label_names))`
		Comprobamos que, efectivamente, el número de clases se corresponde con el del preprocesado realizado en roboflow

7	`%%writetemplate /content/yolov5/models/custom_yolov5s.yaml`
	
	`# parameters`
	`nc: {num_classes}  # number of classes  # CHANGED HERE`
	`depth_multiple: 0.33  # model depth multiple`
	`width_multiple: 0.50  # layer channel multiple`
	
	`# anchors`
	`anchors:`
	`  - [10,13, 16,30, 33,23]  # P3/8`
	`  - [30,61, 62,45, 59,119]  # P4/16`
	`  - [116,90, 156,198, 373,326]  # P5/32`
		Cambios en la configuración para hacer compatible el modelo con el número de clases requeridas en este proyecto.

```
# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]],  # 17 (P3/8-
small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, BottleneckCSP, [512, False]],  # 20 (P4/16-
medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]],  # 23 (P5/32-
large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  #
```

	Detect(P3, P4, P5)]	
8	import os os.chdir('/content/yolov5')	Cambiamos de directorio.
9	%%time %cd /content/yolov5/ !Python train.py --img 416 --batch 80 --epochs 100 --data './data.yaml' --cfg ./models/custom_yolov5s.yaml --weights ''	Entrenamos el modelo. Empleamos 100 etapas, usamos nuestro modelo de datos "*data.yaml*" y no especificamos ningún peso, para generar el *best.pt* y el *last.pt*
10	%load_ext tensorboard %tensorboard --logdir /content/yolov5/runs	Ejecutamos tensorboard para analizar las métricas.
11	%cd /content/yolov5/ !Python detect.py --weights /content/yolov5/runs/train/exp/weights/best.pt --img 416 --conf 0.4 --source ./test/images	Efectuamos una inferencia para comprobar la calidad del detector.

12	import glob
	from IPython.display import Image, display
	
	for imageName in glob.glob('/content/yolov5/runs/detect/exp/*.jpg'): #assuming JPG
	  display(Image(filename=imageName))
	  print("\n")
		Mostramos las imágenes generadas por la detección.

ANEXO C: ESTRUCTURA DE UNA ANOTACIÓN EN FORMATO XML

En este anexo veremos una alternativa de formato de etiquetación para nuestros datasets en roboflow: las etiquetas en XML. Contienen todos los parámetros estructurado según un lenguaje de etiquetas. En la siguiente imagen podemos ver la parametrización de la imagen "road3.png".

road3.xml | road3.txt | interfaz.py

```
<annotation>
    <folder>images</folder>
    <filename>road3.png</filename>
    <size>
        <width>400</width>
        <height>300</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>trafficlight</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>178</xmin>
            <ymin>134</ymin>
            <xmax>236</xmax>
            <ymax>261</ymax>
        </bndbox>
    </object>
</annotation>
```

Ilustración 72: Archivo *road3.xml*

La ventaja de usar este formato es la univocidad a la hora de catalogar las imágenes en roboflow: se carga el archivo de etiquetas y automáticamente aparece un cuadro con la etiqueta y la clase para el modelo. En la siguiente imagen podemos apreciar este fenómeno.

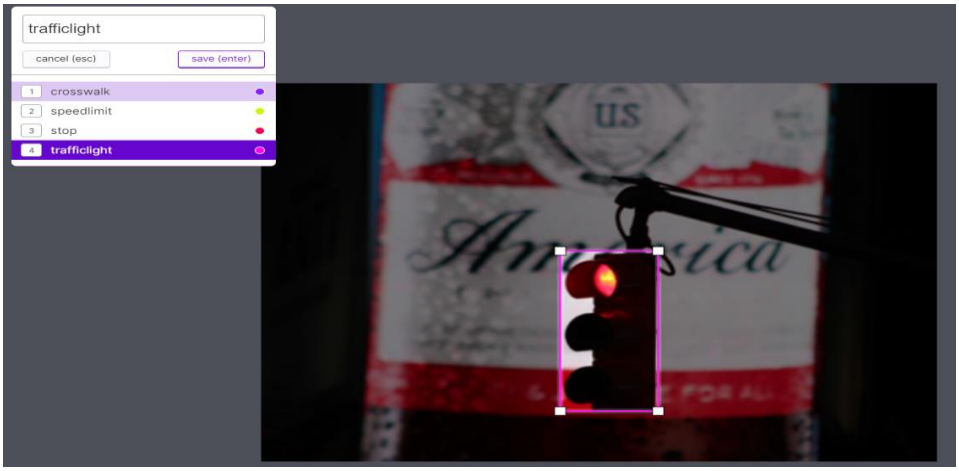


Ilustración 73: Etiquetado automático de *road3.png* en *roboflow*

Se usará este procedimiento para etiquetar todas las etiquetas del dataset mencionado en este documento: “datasettraficolmg”.

ANEXO D: ESTRUCTURA DE UNA ANOTACIÓN EN FORMATO TXT

En este anexo veremos cómo se organiza la información para que YOLOv5 pueda etiquetar las imágenes.

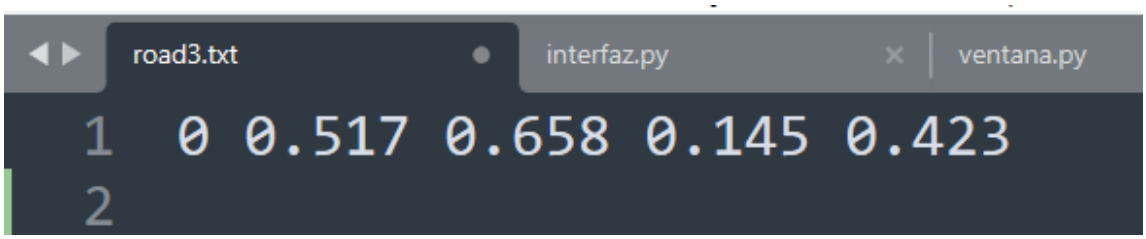


Ilustración 74: Archivo *road3.txt*

El primer valor corresponde con el identificador de la clase, los dos siguientes a las coordenadas X e Y en el eje cartesiano donde se sitúa el centro de la etiqueta de clasificación, y los dos últimos números se corresponden con el ancho y el alto del recuadro de clasificación medidos desde el centro. En la siguiente imagen podemos ver un ejemplo más visual de esta parametrización.

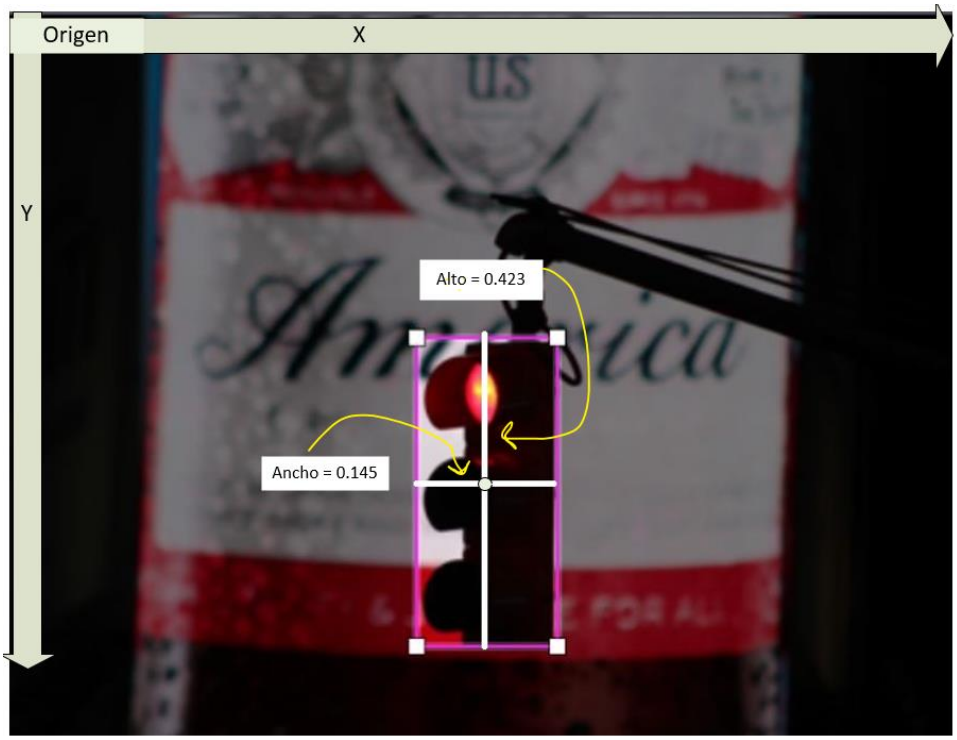


Ilustración 75: Imagen *road3.png* clasificada

De esta forma conseguimos parametrizar cada imagen con 5 sencillos campos.

ANEXO E: CÓDIGO FUENTE: INTERFAZ.PY

```
# -*- coding: utf-8 -*-

#    Form    implementation    generated    from    reading    ui    file
'C:\Users\usuario\Desktop\main.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5
is
```

```
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtGui import QPixmap

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1331, 875)
        MainWindow.setStyleSheet("background-color: #404040;")
        MainWindow.setDocumentMode(False)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.scrollArea = QtWidgets.QScrollArea(self.centralwidget)
        self.scrollArea.setGeometry(QtCore.QRect(230, 90, 951, 631))
        self.scrollArea.setMaximumSize(QtCore.QSize(1300, 711))
        self.scrollArea.setAutoFillBackground(True)

self.scrollArea.setSizeAdjustPolicy(QtWidgets.QAbstractScrollArea.AdjustT
oContents)
        self.scrollArea.setWidgetResizable(True)
        self.scrollArea.setAlignment(QtCore.Qt.AlignCenter)
        self.scrollArea.setObjectName("scrollArea")
        self.scrollAreaWidgetContents = QtWidgets.QWidget()
        self.scrollAreaWidgetContents.setGeometry(QtCore.QRect(0,      0,
1300, 629))

self.scrollAreaWidgetContents.setObjectName("scrollAreaWidgetContents")
        self.verticalLayout_5                                          =
QtWidgets.QVBoxLayout(self.scrollAreaWidgetContents)
        self.verticalLayout_5.setObjectName("verticalLayout_5")
        self.verticalLayout_6 = QtWidgets.QVBoxLayout()
        self.verticalLayout_6.setContentsMargins(369, 0, -1, -1)
        self.verticalLayout_6.setObjectName("verticalLayout_6")
        #vamos a cargar la imagen manualmente
        pixmap = QPixmap('logo.png')
        pixmap_image = QtGui.QPixmap(pixmap)
        self.label = QtWidgets.QLabel(self.scrollAreaWidgetContents)
        self.label.setMaximumSize(QtCore.QSize(190, 114))
        self.label.setText("")
        self.label.setPixmap(pixmap_image)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setScaledContents(True)
        self.label.setMinimumSize(1,1)
        self.label.show()
        #self.label.addPixmap(QtGui.QPixmap("logo.png"),
QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.label.setObjectName("label")
        self.verticalLayout_6.addWidget(self.label)
        self.verticalLayout_5.addLayout(self.verticalLayout_6)
        self.verticalLayout_3 = QtWidgets.QVBoxLayout()
        self.verticalLayout_3.setObjectName("verticalLayout_3")
        self.gridLayout_4 = QtWidgets.QGridLayout()
        self.gridLayout_4.setObjectName("gridLayout_4")
        self.verticalLayout_3.addLayout(self.gridLayout_4)
```

```
        self.verticalLayout_5.addLayout(self.verticalLayout_3)
        self.verticalLayout_7 = QtWidgets.QVBoxLayout()
        self.verticalLayout_7.setObjectName("verticalLayout_7")
        self.instrucciones1                                              =
QtWidgets.QLabel(self.scrollAreaWidgetContents)
        self.instrucciones1.setObjectName("instrucciones1")
        self.instrucciones1.setText("Paso 1: Pulse el botón 'Habilitar
Asistente' para iniciar a A.R.G.O.S.\nPaso 2: Cuando cargue la ventana
emergente, pulse 'Cargar Imagen' para habilitar el explorador de
archivos.\nPaso 3: Sitúese en el directorio de pruebas y seleccione la
imagen para efectuar la inferencia.\nPaso 4: Una vez finalizada la
inferencia, pulse 'Ver Inferencia con YOLOv5' para visualizar la
detección en la misma ventana emergente.")
        self.verticalLayout_7.addWidget(self.instrucciones1)
        self.instrucciones2                                              =
QtWidgets.QLabel(self.scrollAreaWidgetContents)
        self.instrucciones2.setText("Aviso: Utilice únicamente imágenes
con extensión PNG.")
        self.instrucciones2.setObjectName("instrucciones2")
        self.verticalLayout_7.addWidget(self.instrucciones2)
        self.boton_reacondicionar                                        =
QtWidgets.QPushButton(self.scrollAreaWidgetContents)
        self.boton_reacondicionar.setText("Reacondicionar entorno para
otra prueba")
        self.boton_reacondicionar.setObjectName("boton_reacondicionar")
        self.verticalLayout_7.addWidget(self.boton_reacondicionar)
        self.verticalLayout_5.addLayout(self.verticalLayout_7)
        self.groupBox                                                    =
QtWidgets.QGroupBox(self.scrollAreaWidgetContents)
        self.groupBox.setObjectName("groupBox")
        self.gridLayout_3 = QtWidgets.QGridLayout(self.groupBox)
        self.gridLayout_3.setObjectName("gridLayout_3")
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setObjectName("verticalLayout")
        self.gridLayout = QtWidgets.QGridLayout()
        self.gridLayout.setObjectName("gridLayout")
        self.label_5 = QtWidgets.QLabel(self.groupBox)
        self.label_5.setObjectName("label_5")
        self.gridLayout.addWidget(self.label_5, 0, 0, 1, 1)
        self.lineEdit_3 = QtWidgets.QLineEdit(self.groupBox)
        self.lineEdit_3.setObjectName("lineEdit_3")
        self.gridLayout.addWidget(self.lineEdit_3, 0, 1, 1, 1)
        self.label_6 = QtWidgets.QLabel(self.groupBox)
        self.label_6.setObjectName("label_6")
        self.gridLayout.addWidget(self.label_6, 1, 0, 1, 1)
        self.lineEdit_4 = QtWidgets.QLineEdit(self.groupBox)
        self.lineEdit_4.setObjectName("lineEdit_4")
        self.gridLayout.addWidget(self.lineEdit_4, 1, 1, 1, 1)
        self.verticalLayout.addLayout(self.gridLayout)
        self.pushButton = QtWidgets.QPushButton(self.groupBox)
        self.pushButton.setObjectName("pushButton")
        self.verticalLayout.addWidget(self.pushButton)
        self.gridLayout_3.addLayout(self.verticalLayout, 0, 1, 1, 1)
        self.gridLayout_2 = QtWidgets.QGridLayout()
        self.gridLayout_2.setObjectName("gridLayout_2")
        self.label_2 = QtWidgets.QLabel(self.groupBox)
        self.label_2.setObjectName("label_2")
```

```
        self.gridLayout_2.addWidget(self.label_2, 0, 0, 1, 1)
        self.comboBox = QtWidgets.QComboBox(self.groupBox)
        self.comboBox.setObjectName("comboBox")
        self.gridLayout_2.addWidget(self.comboBox, 0, 1, 1, 1)
        self.label_3 = QtWidgets.QLabel(self.groupBox)
        self.label_3.setObjectName("label_3")
        self.gridLayout_2.addWidget(self.label_3, 1, 0, 1, 1)
        self.lineEdit = QtWidgets.QLineEdit(self.groupBox)
        self.lineEdit.setObjectName("lineEdit")
        self.gridLayout_2.addWidget(self.lineEdit, 1, 1, 1, 1)
        self.label_4 = QtWidgets.QLabel(self.groupBox)
        self.label_4.setObjectName("label_4")
        self.gridLayout_2.addWidget(self.label_4, 2, 0, 1, 1)
        self.lineEdit_2 = QtWidgets.QLineEdit(self.groupBox)
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.gridLayout_2.addWidget(self.lineEdit_2, 2, 1, 1, 1)
        self.gridLayout_3.addLayout(self.gridLayout_2, 0, 0, 1, 1)
        self.verticalLayout_5.addWidget(self.groupBox)
        self.scrollArea.setWidget(self.scrollAreaWidgetContents)
        MainWindow.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.menuBar = QtWidgets.QMenuBar(MainWindow)
        self.menuBar.setGeometry(QtCore.QRect(0, 0, 1331, 26))
        self.menuBar.setObjectName("menuBar")
        MainWindow.setMenuBar(self.menuBar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "A.R.G.O.S."))
        self.radioButton_2.setText(_translate("MainWindow", "Iniciar Set
de Pruebas 1"))
        self.radioButton_3.setText(_translate("MainWindow", "Iniciar Set
de Pruebas 2"))
        self.radioButton.setText(_translate("MainWindow", "Iniciar Set de
Pruebas 3"))
        self.groupBox.setTitle(_translate("MainWindow",  "Parámetros  de
Audio"))
        self.label_5.setText(_translate("MainWindow", "Submuestreo"))
        self.lineEdit_3.setText(_translate("MainWindow", "1"))
        self.label_6.setText(_translate("MainWindow",    "Intervalo    de
refresco (ms)"))
        self.lineEdit_4.setText(_translate("MainWindow", "30"))
        self.pushButton.setText(_translate("MainWindow",       "Habilitar
Asistente"))
        self.label_2.setText(_translate("MainWindow",    "Periférico    de
entrada"))
        self.label_3.setText(_translate("MainWindow",   "Tamaño   de   la
ventana"))
        self.lineEdit.setText(_translate("MainWindow", "1000"))
        self.label_4.setText(_translate("MainWindow",  "Tasa  de  muestreo
(Hz)"))
```

```
        self.lineEdit_2.setText(_translate("MainWindow", "44100"))
#import logo_rc
```

ANEXO F: CÓDIGO FUENTE: VENTANA.PY

Este código se ha desarrollado partiendo del código fuente descrito en la referencia [29], haciendo las modificaciones pertinentes.

```
# modulos de terceros
import sys
import matplotlib
import threading
import time
import shutil
matplotlib.use('Qt5Agg')
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.figure import Figure
import matplotlib.ticker as ticker
import queue
import numpy as np
import sounddevice as sd
#modulos de pyqt5
from PyQt5 import QtCore, QtWidgets,QtGui
from PyQt5.QtWidgets import QApplication,QWidget, QVBoxLayout,
QPushButton, QFileDialog , QLabel, QTextEdit,QDialog
from PyQt5 import uic
from PyQt5.QtCore import pyqtSlot, Signal
from PyQt5.QtMultimedia import QAudioDeviceInfo,QAudio,QCameraInfo
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import QObject, QThread, pyqtSignal
input_audio_deviceInfos =
QAudioDeviceInfo.availableDevices(QAudio.AudioInput)
#imports propios
import assistant
from assistant import *
from interfaz import *
from detecta_yolov import *
class SecondWindow(QtWidgets.QMainWindow, Ui_MainWindow):
          def __init__(self):
                    #super(SecondWindow, self).__init__(parent)
                    self.diccionario = {
                    "semaforo.png" : 3,
                    "stop.png": 2,
                    "limite_velocidad.png": 1,
                    "peaton.png": 0
                    }
                    super(self.__class__, self).__init__()
                    self.setMaximumSize(750,900)
                    uic.loadUi('cargaImagen.ui', self) # Load the .ui
file
                    icon = QtGui.QIcon()
                    icon.addPixmap(QtGui.QPixmap("logo.png"),
```

```
QtGui.QIcon.Normal, QtGui.QIcon.Off)
                    self.setWindowIcon(icon)
                    self.setWindowTitle("Test de detección")

          self.botonCargaImagenPrueba.clicked.connect(self.getImage)
                    self.iniciar_ventana_prueba()
          def iniciar_ventana_prueba(self):
                    self.assistant = Assistant()
                    t2 =threading.Thread(target=self.dialogue)
                    t2.start()
          def dialogue(self):
                    self.assistant.speak("¡Hola! Mi nombre es ARGOS. ¿En
qué puedo ayudarte?")
          def dialogue2(self,fileName,filtro):
                    self.assistant.speak("Iniciando inferencia: Espere
unos segundos.")
                    self.deteccion.run(fileName,filtro)
          def getImage(self):
                    fname = QFileDialog.getOpenFileName(self, 'Open
file','c:/', "archivo de imágenes (*.jpg *.png)")
                    imagePath = fname[0]
                    splitted = imagePath.split("/")
                    fileName = splitted[-1] #para yolov5
                    pixmap = QPixmap(imagePath)
                    self.imagenPrueba.setPixmap(QPixmap(pixmap))
                    self.imagenPrueba.setScaledContents(True)
                    filtro = self.diccionario[fileName]
                    self.deteccion = detectaYolov()
                    t3 =
threading.Thread(target=self.dialogue2(fileName,filtro))
                    t3.start()
                    self.botonDetectar.clicked.connect(lambda:
self.autoLoad(fileName))
          def autoLoad(self,fileName):
                    imagePath = "runs/detect/exp/"+fileName
                    pixmap = QPixmap(imagePath)
                    self.imagenInferencia.setPixmap(QPixmap(pixmap))
                    self.imagenInferencia.setScaledContents(True)
class MplCanvas(FigureCanvas):
          def __init__(self, parent=None, width=5, height=4, dpi=100):
                    fig = Figure(figsize=(width, height), dpi=dpi)
                    self.axes = fig.add_subplot(111)
                    super(MplCanvas, self).__init__(fig)
                    fig.tight_layout()

class MainWindow(QtWidgets.QMainWindow, Ui_MainWindow):
    def __init__(self):
          super(self.__class__, self).__init__()

          self.assistant = Assistant()
          #self.t2 =threading.Thread(target=self.speaking)
          self.ui = uic.loadUi('main.ui',self)
          self.setupUi(self)
          self.resize(1200, 890)
          icon = QtGui.QIcon()
          icon.addPixmap(QtGui.QPixmap("logo.png"), QtGui.QIcon.Normal,
```

```
QtGui.QIcon.Off)
          self.setWindowIcon(icon)
          self.threadpool = QtCore.QThreadPool()
          self.waveformThread = threading.Thread(target =
self.start_worker)
          self.waveformThread.setDaemon(True)
          self.devices_list= []
          for device in input_audio_deviceInfos:
                    self.devices_list.append(device.deviceName())
          self.comboBox.addItems(self.devices_list)

          self.comboBox.currentIndexChanged['QString'].connect(self.upda
te_now)
          self.comboBox.setCurrentIndex(0)
          self.canvas = MplCanvas(self, width=4, height=3, dpi=100)
          self.ui.gridLayout_4.addWidget(self.canvas, 2, 1, 1, 1)
          self.reference_plot = None
          self.q = queue.Queue(maxsize=20)
          self.device = 0
          self.window_length = 1000
          self.downsample = 1
          self.channels = [1]
          self.interval = 30
          device_info =  sd.query_devices(self.device, 'input')
          self.samplerate = device_info['default_samplerate']
          length  =
int(self.window_length*self.samplerate/(1000*self.downsample))
          sd.default.samplerate = self.samplerate
          self.plotdata =  np.zeros((length,len(self.channels)))
          self.update_plot()
          self.timer = QtCore.QTimer()
          self.timer.setInterval(self.interval) #msec
          self.timer.timeout.connect(self.update_plot)
          self.timer.start()

          self.lineEdit.textChanged['QString'].connect(self.update_windo
w_length)

          self.lineEdit_2.textChanged['QString'].connect(self.update_sam
ple_rate)

          self.lineEdit_3.textChanged['QString'].connect(self.update_dow
n_sample)

          self.lineEdit_4.textChanged['QString'].connect(self.update_int
erval)
          self.waveformThread.start()
          self.nuevaVentana= SecondWindow()
          self.pushButton.clicked.connect(self.abreNuevaVentana)
          self.boton_reacondicionar.clicked.connect(self.reacondicionar)
    def reacondicionar(self):
          directorio = "runs/detect/exp"
          try:
                    shutil.rmtree(directorio)
                    self.instrucciones2.setText("¡Entorno
reacondicionado con éxito!")
                    self.instrucciones2.setStyleSheet("""
```

```
        QWidget {
            background-color: white;
            }
        """)
          except OSError as e:
                    print("Error: %s - %s." % (e.filename, e.strerror))
    def abreNuevaVentana(self):
          self.nuevaVentana.show()
    def getAudio(self):
          try:
                    def audio_callback(indata,frames,time,status):
                              self.q.put(indata[::self.downsample,[0]])
                    stream  = sd.InputStream( device = self.device,
channels = max(self.channels), samplerate =self.samplerate, callback  =
audio_callback)
                    with stream:
                              input()
          except Exception as e:
                    print("ERROR: ",e)
    def start_worker(self):
          worker = Worker(self.start_stream, )
          self.threadpool.start(worker)
          #workerThread= threading.Thread(target = self.start_stream)
          #workerThread.start()
          #assistantThread.start()

    def start_stream(self):
          self.lineEdit.setEnabled(False)
          self.lineEdit_2.setEnabled(False)
          self.lineEdit_3.setEnabled(False)
          self.lineEdit_4.setEnabled(False)
          self.comboBox.setEnabled(False)
          self.pushButton.setEnabled(True)
          self.getAudio()
    def update_now(self,value):
          self.device = self.devices_list.index(value)
          print('Device:',self.devices_list.index(value))
    def update_window_length(self,value):
          self.window_length = int(value)
          length  =
int(self.window_length*self.samplerate/(1000*self.downsample))
          self.plotdata =  np.zeros((length,len(self.channels)))
          self.update_plot()
    def update_sample_rate(self,value):
          self.samplerate = int(value)
          sd.default.samplerate = self.samplerate
          length  =
int(self.window_length*self.samplerate/(1000*self.downsample))
          self.plotdata =  np.zeros((length,len(self.channels)))
          self.update_plot()
    def update_down_sample(self,value):
          self.downsample = int(value)
          length  =
int(self.window_length*self.samplerate/(1000*self.downsample))
          self.plotdata =  np.zeros((length,len(self.channels)))
          self.update_plot()
```

```
    def update_interval(self,value):
          self.interval = int(value)
          self.timer.setInterval(self.interval) #msec
          self.timer.timeout.connect(self.update_plot)
          self.timer.start()
    def update_plot(self):
          try:
                    data=[0]
                    while True:
                              try:
                                        data = self.q.get_nowait()
                              except queue.Empty:
                                        break
                              shift = len(data)
                              self.plotdata = np.roll(self.plotdata, -
shift,axis = 0)
                              self.plotdata[-shift:,:] = data
                              self.ydata = self.plotdata[:]
                              self.canvas.axes.set_facecolor((0,0,0))
                              if self.reference_plot is None:
                                        plot_refs =
self.canvas.axes.plot( self.ydata, color=(0,1,0.29))
                                        self.reference_plot =
plot_refs[0]
                              else:

          self.reference_plot.set_ydata(self.ydata)
                    self.canvas.axes.yaxis.grid(True,linestyle='--')
                    start, end = self.canvas.axes.get_ylim()
                    self.canvas.axes.yaxis.set_ticks(np.arange(start,
end, 0.1))

          self.canvas.axes.yaxis.set_major_formatter(ticker.FormatStrFor
matter('%0.1f'))
                    self.canvas.axes.set_ylim( ymin=-0.5, ymax=0.5)

                    self.canvas.draw()
          except:
                    pass
class Worker(QtCore.QRunnable):

          def __init__(self, function, *args, **kwargs):
                    super(Worker, self).__init__()
                    self.function = function
                    self.args = args
                    self.kwargs = kwargs

          @pyqtSlot()
          def run(self):
                    self.function(*self.args, **self.kwargs)
if __name__ == "__main__":
          app = QtWidgets.QApplication(sys.argv)
          mainWindow = MainWindow()
          t1 = threading.Thread(target=mainWindow.show())
          t1.start()
          sys.exit(app.exec_())
```

ANEXO G: CÓDIGO FUENTE: DETECTA_YOLOV.PY

```
import yolov5
from yolov5 import train, test, detect, export
import sys
from assistant import *
class detectaYolov():
          def __init__(self):
                    super(self.__class__, self).__init__()
          def run(self,img_url,filtro):
                    file_name_without_extension =
img_url.replace(".png","")
                    print(file_name_without_extension)
                    traffic_dict = {
                    "0": "paso de peatones",
                    "1": "límite de velocidad",
                    "2": "stop",
                    "3": "semáforo"
                    }
                    if filtro == 0:
                              deteccion = detect.run(source=img_url,
weights="best.pt", conf_thres=0.65,line_thickness=2,max_det=1,
imgsz=200,classes = filtro,save_crop=True,save_conf = True,save_txt=True)
                    elif filtro == 1:
                              deteccion = detect.run(source=img_url,
weights="best.pt", conf_thres=0.68,line_thickness=2,max_det=1,
imgsz=200,classes = filtro,save_crop=True,save_conf = True,save_txt=True)
                    elif filtro == 2:
                              deteccion = detect.run(source=img_url,
weights="best.pt", conf_thres=0.0,line_thickness=2,max_det=5,
imgsz=200,classes = filtro,save_crop=True,save_conf = True,save_txt=True)
                    elif filtro == 3:
                              deteccion = detect.run(source=img_url,
weights="best.pt", conf_thres=0.23,line_thickness=2,max_det=1,
imgsz=200,classes = filtro,save_crop=True,save_conf = True,save_txt=True)
                    else:
                              print("Error, filtro no valido")
                              sys.exit()
                    text =
open("runs/detect/exp/labels/"+file_name_without_extension+".txt",'r')
                    asistente_aux = Assistant()
                    for x in text:
                              if x[0]==str(filtro):
                                        parametros = str(x).split(" ")
                                        prob = float(parametros[-1])
                                        prob_sobre_cien = prob*100

          #print(traffic_dict[str(filtro)]+" detectado. Certeza:
"+str(round(prob_sobre_cien,2))+" %")

          asistente_aux.speak(traffic_dict[str(filtro)]+" detectado.
```

```
Certeza: "+str(round(prob_sobre_cien,2))+" %")
                                else:
                                        print("sin filtro")
if __name__=="__main__":
          detecta = detectaYolov()
          detecta.run("imagenes_validas_test/limite_velocidad.png",1)
```

ANEXO H: CÓDIGO FUENTE: ASSISTANT.PY

El siguiente código se ha desarrollado partiendo del código fuente disponible en la referencia [30].

```
# importing speech recognition package from google api
import speech_recognition as sr
import playsound # to play saved mp3 file
from gtts import gTTS # google text to speech
import os
from PyQt5.QtCore import QObject, QThread, pyqtSignal, pyqtSlot
from PyQt5 import QtCore
class Assistant(QtCore.QObject):
          num = 0
          phrase = QtCore.pyqtSignal(str)
          @QtCore.pyqtSlot()
          def speak(self,output):
                    self.num += 1
                    print("A.R.G.O.S. : ", output)
                    toSpeak = gTTS(text = output, lang ='es', slow =
False)
                    # almacena el archivo de audio generado por gTTS
                    file = str(self.num)+".mp3"
                    toSpeak.save(file)
                    self.phrase.emit(output)
                    # reproducimos el archivo de sonido
                    playsound.playsound(file, True)
                    os.remove(file)
```

REFERENCIAS

[1] Mike Ohanu "YOLOv5 Tutorial". 2020 [En línea]. Disponible en: https://michaelohanu.medium.com/yolov5-tutorial-75207a19a3aa

[2] Albert Soto i Serrano "YOLO Object Detector for Onboard Driving Images". 2017 [En línea]. Disponible en: https://ddd.uab.cat/pub/tfg/2017/tfg_71066/paper.pdf

[3] Diego Calvo "Clasificación de las redes neuronales artificiales". 2017 [En línea]. Disponible en: https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[4] Marco Antonio Lopez Paredes "Línea de aprendizaje: Redes Neuronales. Red monocapa y multicapa". 2014 [En línea]. Disponible en: https://es.slideshare.net/levygt/redes-neuronales-multicapa-y-monocapa

[5] Ultralytics "YOLOv5" 2021 [En línea]. Disponible en: https://github.com/ultralytics/yolov5

[6] COCO Common Objects in Context. [En línea]. Disponible en: https://cocodataset.org/#home

[7] Matplotlib v3.4.3 Documentation. matplotlib.org. 2021 [En línea]. Disponible en: https://pypi.org/project/matplotlib/

[8] Casper da Costa-Luis tqdm v4.62.1 Documentation. 2021 [En línea]. Disponible en: https://tqdm.github.io/

[9] Seaborn v0.11.2 statistical data visualization. Documentation. [En línea]. Disponible en: https://seaborn.pydata.org/

[10] NumPy v1.21 Manual. numpy.org 2020 [en línea]. Disponible en: https://numpy.org/doc/stable/

[11] OpenCV Open Source Computer Vision v4.5.2. 2020. Documentation. [En línea]. Disponible en: https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html

[12] Pillow (PIL Fork) 7.1.2 documentation. Pillow.readthedocs.io. 2020 [En línea]. Disponible en: https://pillow.readthedocs.io/en/stable/

[13] SciPy v1.7.1. Documentation. docs.scipy.org. 2021. [En línea]. Disponible en: https://docs.scipy.org/doc/scipy/reference/

[14] Torch v.1.9.0. Documentation. pytorch.rg. 2021. [En línea]. Disponible en: https://pytorch.org/docs/stable/index.html

[15] torchvision v.1.8.0. Github 2021. [En línea]. Disponible en: https://github.com/pytorch/vision

[16] tensorboard el kit de herramientas de visualización de TensorFlow. 2021. [En línea]. Disponible en: https://www.tensorflow.org/tensorboard?hl=es-419

[17] Sociedad Andaluza de Educación Matemática Thales. 2000. "Características de las redes neuronales" . [En línea]. Disponible en: https://thales.cica.es/rd/Recursos/rd98/TecInfo/07/capitulo3.html

[18] Fernando Berzal "Entrenamiento de redes neuronales". 2018. 1st Edition. ISBN-13: 978-1731314338. Disponible en: https://www.amazon.es/Redes-Neuronales-Deep-Learning-Edici%C3%B3n/dp/1731314337

[19] Jose Luis Calderón O. "Aprendizaje Competitivo y Cooperativo". 2003. [En línea]. Disponible en: https://es.slideshare.net/mentelibre/redes-neuronales-aprendizaje-competitivo-cooperativo

[20] eswitha_reddy "Types of Boltzmann Machines". 2020. [En línea]. Disponible en: https://www.geeksforgeeks.org/types-of-boltzmann-machines/

[21] Swapna "Convolutional Neural Network (CNN)". 2020. Image. [En línea]. Disponible en: https://developersbreach.com/convolution-neural-network-deep-learning/

[22] La Moncloa. “Balance de Seguridad Vial. 2019 finaliza con 1.098 fallecidos, el mínimo histórico de víctimas mortales en carretera.” 2020. Imagen. [En línea]. Disponible en:

https://www.lamoncloa.gob.es/serviciosdeprensa/notasprensa/interior/Paginas/2020/020120-seguridad_vial.aspx

[23] Neeraj Menon "People Detection and Tracking (SSD + Kalman Filtering)". 2021. Imagen. [En línea]. Disponible en: https://devmesh.intel.com/projects/people-detection-and-tracking

[24] Fisiosaludable "Neurona". 2017. Imagen. [En línea]. Disponible en: https://fisiosaludable.com/publicaciones/conceptos/99-neurona-2

[25] Twinkl.com "Synapse Diagram Cells Science KS4 Bw RGB" . 2017. Imagen. [En línea]. Disponible en: https://www.twinkl.co.za/illustration/synapse-diagram-cells-science-ks4-bw-rgb

[26] Prabhat Kumar Sahu Google Colab Custom-yolo-v5.ipynb. 2021. Notebook de Google. [En línea]. Disponible en: https://colab.research.google.com/drive/16QCaYzTuHCOF9CQLQYmGNxmtY1xKAIdn?usp=sharing#scrollTo=dOrWg4mthaji

[27] MakeML "Road Sign Dataset". 2020. Dataset. [En línea]. Disponible en: https://makeml.app/datasets/road-signs

[28] Innovateking-EU " Fuente de alimentación UPS ininterrumpible Innovateking-EU Raspberry Pi con baterías 18650". 2019. [En línea]. Disponible en: https://www.amazon.es/Innovateking-EU-Raspberry-alimentaci%C3%B3n-ininterrumpible-Cargador/dp/B082KJJMP2/ref=sr_1_3?dchild=1&keywords=Innovateking-EU&qid=1629363447&sr=8-3

[29] PyShine "PytQt5 GUI design to plot Live audio data from Microphone". Tutorial. 2020. [En línea]. Disponible en: https://pyshine.com/How-to-make-a-real-time-voice-plot-in-PyQt5/

[30] GeeksforGeeks "Voice Assistant using Python". 2021. Tutorial. [En línea]. Disponible en: https://www.geeksforgeeks.org/voice-assistant-using-Python/

[31] Pinturicchio "Io, Argos y Mercurio". 1452-1513. Imagen . [En línea]. Disponible en: https://fr.gallerix.ru/storeroom/1713238383/N/571482244/

[32] Roboflow "Give your software the power to see objects in images and video" . 2021. Documentation. [En línea]. Disponible en: https://docs.roboflow.com/

[33] John McCarthy En wikipedia.org. 2021 [En línea]. Disponible en: https://es.wikipedia.org/wiki/John_McCarthy

[34] You Only Look Once en pjreddie.com. 2015. [En línea]. Disponible en: https://pjreddie.com/darknet/yolo/

[35] Frank Rosenblatt en wikipedia.org. 2020. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Frank_Rosenblatt

[36] PyYAML en pyyaml.org. v0.2.5 Documentation 2020. [En línea]. Disponible en: https://pyyaml.org/

[37] Pandas en pandaspydata.org. v1.3.2. Documentation 2021 [En línea]. Disponible en: https://pandas.pydata.org/

[38] Lyken17 "A tool to count the FLOPs of PyTorch model." v0.0.31-2005241907 . Project 2020 [En línea]. Disponible en: https://github.com/Lyken17/pytorch-OpCounter/

[39] tylin COCO API. v2.0. Project 2019 [En línea]. Disponible en: https://github.com/cocodataset/cocoapi

[40] Reina López, Lucía. "Aplicación Android y Servicio Web Spring para la detección y registro de señales de tráfico de velocidad usando Deep Learning con tiny-yolov3 y OpenCV". 2020. [En línea]. Disponible en: https://biblus.us.es/bibing/proyectos/abreproy/93242/fichero/TFG-3242+REINA+L%C3%93PEZ%2C+LUC%C3%8DA.pdf

[41] Moreno Prieto, Ángel. "Detección de Objetos con TinyYOLOv3 sobre Raspberry Pi 3". 2019. [En línea]. Disponible en: https://idus.us.es/bitstream/handle/11441/107053/TFG-3340-MORENO%20PRIETO.pdf?sequence=1&isAllowed=y

[1] Mike Ohanu "YOLOv5 Tutorial". 2020 [Online]. Available on: https://michaelohanu.medium.com/yolov5-

tutorial-75207a19a3aa

[2] Albert Soto i Serrano "YOLO Object Detector for Onboard Driving Images". 2017 [Online]. Available on: https://ddd.uab.cat/pub/tfg/2017/tfg_71066/paper.pdf

[3] Diego Calvo "Clasificación de las redes neuronales artificiales". 2017 [Online]. Available on: https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[4] Marco Antonio Lopez Paredes "Línea de aprendizaje: Redes Neuronales. Red monocapa y multicapa". 2014 [Online]. Available on: https://es.slideshare.net/levygt/redes-neuronales-multicapa-y-monocapa

[5] Ultralytics "YOLOv5" 2021 [Online]. Available on: https://github.com/ultralytics/yolov5

[6] COCO Common Objects in Context. [Online]. Available on: https://cocodataset.org/#home

[7] Matplotlib v3.4.3 Documentation. matplotlib.org. 2021 [Online]. Available on: https://pypi.org/project/matplotlib/

[8] Casper da Costa-Luis tqdm v4.62.1 Documentation. 2021 [Online]. Available on: https://tqdm.github.io/

[9] Seaborn v0.11.2 statistical data visualization. Documentation. [Online]. Available on: https://seaborn.pydata.org/

[10] NumPy v1.21 Manual. numpy.org 2020 [online]. Available on: https://numpy.org/doc/stable/

[11] OpenCV Open Source Computer Vision v4.5.2. 2020. Documentation. [Online]. Available on: https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html

[12] Pillow (PIL Fork) 7.1.2 documentation. Pillow.readthedocs.io. 2020 [Online]. Available on: https://pillow.readthedocs.io/en/stable/

[13] SciPy v1.7.1. Documentation. docs.scipy.org. 2021. [Online]. Available on: https://docs.scipy.org/doc/scipy/reference/

[14] Torch v.1.9.0. Documentation. pytorch.rg. 2021. [Online]. Available on: https://pytorch.org/docs/stable/index.html

[15] torchvision v.1.8.0. Github 2021. [Online]. Available on: https://github.com/pytorch/vision

[16] tensorboard el kit de herramientas de visualización de TensorFlow. 2021. [Online]. Available on: https://www.tensorflow.org/tensorboard?hl=es-419

[17] Sociedad Andaluza de Educación Matemática Thales. 2000. "Características de las redes neuronales" . [Online]. Available on: https://thales.cica.es/rd/Recursos/rd98/TecInfo/07/capitulo3.html

[18] Fernando Berzal "Entrenamiento de redes neuronales". 2018. 1st Edition. ISBN-13: 978-1731314338. Available on: https://www.amazon.es/Redes-Neuronales-Deep-Learning-Edici%C3%B3n/dp/1731314337

[19] Jose Luis Calderón O. "Aprendizaje Competitivo y Cooperativo". 2003. [Online]. Available on: https://es.slideshare.net/mentelibre/redes-neuronales-aprendizaje-competitivo-cooperativo

[20] eswitha_reddy "Types of Boltzmann Machines". 2020. [Online]. Available on: https://www.geeksforgeeks.org/types-of-boltzmann-machines/

[21] Swapna "Convolutional Neural Network (CNN)". 2020. Image. [Online]. Available on: https://developersbreach.com/convolution-neural-network-deep-learning/

[22] La Moncloa. “Balance de Seguridad Vial. 2019 finaliza con 1.098 fallecidos, el mínimo histórico de víctimas mortales en carretera.” 2020. Image. [Online]. Available on: https://www.lamoncloa.gob.es/serviciosdeprensa/notasprensa/interior/Paginas/2020/020120-seguridad_vial.aspx

[23] Neeraj Menon "People Detection and Tracking (SSD + Kalman Filtering)". 2021. Image. [Online]. Available on: https://devmesh.intel.com/projects/people-detection-and-tracking

[24] Fisiosaludable "Neurona". 2017. Image. [Online]. Available on: https://fisiosaludable.com/publicaciones/conceptos/99-neurona-2

[25] Twinkl.com "Synapse Diagram Cells Science KS4 Bw RGB" . 2017. Image. [Online]. Available on: https://www.twinkl.co.za/illustration/synapse-diagram-cells-science-ks4-bw-rgb

[26] Prabhat Kumar Sahu Google Colab Custom-yolo-v5.ipynb. 2021. Notebook de Google. [Online]. Available on: https://colab.research.google.com/drive/16QCaYzTuHCOF9CQLQYmGNxmtY1xKAIdn?usp=sharing#scrollTo=dOrWg4mthaji

[27] MakeML "Road Sign Dataset". 2020. Dataset. [Online]. Available on: https://makeml.app/datasets/road-signs

[28] Innovateking-EU " Innovateking-EU Raspberry Pi uninterruptible UPS power supply with 18650 batteries". 2019. [Online]. Available on: https://www.amazon.es/Innovateking-EU-Raspberry-alimentaci%C3%B3n-ininterrumpible-Cargador/dp/B082KJJMP2/ref=sr_1_3?dchild=1&keywords=Innovateking-EU&qid=1629363447&sr=8-3

[29] PyShine "PytQt5 GUI design to plot Live audio data from Microphone". Tutorial. 2020. [Online]. Available on: https://pyshine.com/How-to-make-a-real-time-voice-plot-in-PyQt5/

[30] GeeksforGeeks "Voice Assistant using Python". 2021. Tutorial. [Online]. Available on: https://www.geeksforgeeks.org/voice-assistant-using-Python/

[31] Pinturicchio "Io, Argus and Mercury". 1452-1513. Image . [Online]. Available on: https://fr.gallerix.ru/storeroom/1713238383/N/571482244/

[32] Roboflow "Give your software the power to see objects in images and video" . 2021. Documentation. [Online]. Available on: https://docs.roboflow.com/

[33] John McCarthy En wikipedia.org. 2021 [Online]. Available on: https://es.wikipedia.org/wiki/John_McCarthy

[34] You Only Look Once en pjreddie.com. 2015. [Online]. Available on: https://pjreddie.com/darknet/yolo/

[35] Frank Rosenblatt en wikipedia.org. 2020. [Online]. Available on: https://es.wikipedia.org/wiki/Frank_Rosenblatt

[36] PyYAML en pyyaml.org. v0.2.5 Documentation 2020. [Online]. Available on: https://pyyaml.org/

[37] Pandas en pandaspydata.org. v1.3.2. Documentation 2021 [Online]. Available on: https://pandas.pydata.org/

[38] Lyken17 "A tool to count the FLOPs of PyTorch model." v0.0.31-2005241907 . Project 2020 [Online]. Available on: https://github.com/Lyken17/pytorch-OpCounter/

[39] tylin COCO API. v2.0. Project 2019 [Online]. Available on: https://github.com/cocodataset/cocoapi

[40] Reina López, Lucía. "Aplicación Android y Servicio Web Spring para la detección y registro de señales de tráfico de velocidad usando Deep Learning con tiny-yolov3 y OpenCV". 2020. [Online]. Available on: https://biblus.us.es/bibing/proyectos/abreproy/93242/fichero/TFG-3242+REINA+L%C3%93PEZ%2C+LUC%C3%8DA.pdf

[41] Moreno Prieto, Ángel. "Detección de Objetos con TinyYOLOv3 sobre Raspberry Pi 3". 2019. [Online]. Available on: https://idus.us.es/bitstream/handle/11441/107053/TFG-3340-MORENO%20PRIETO.pdf?sequence=1&isAllowed=y

[42] DeepLearning " YOLOv5 object detection on Raspberry pi 4" . 2021.Tutorial [Online]. Available on: https://youtu.be/pIUFXGxR_-Q