Proyecto Fin de Carrera

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Metodología para la detección de objetos en imágenes basada en la librería YOLO con aplicación a la detección de carros.

Autor: David Horcajada Jiménez

Tutor: Jesús Iván Maza Alcañiz

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Carrera
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Metodología para la detección de objetos en imágenes basada en la librería YOLO con aplicación a la detección de carros.

Autor:
David Horcajada Jiménez

Tutor:
Jesús Iván Maza Alcañiz
Profesor titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Proyecto Fin de Carrera: Metodología para la detección de objetos en imágenes basada en la librería YOLO con aplicación a la detección de carros.

Autor: David Horcajada Jiménez

Tutor: Jesús Iván Maza Alcañiz

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 maestros

Agradecimientos

En primer lugar, gracias infinitas a mi padre, mi madre y mi hermana, creo que nunca les podré agradecer lo suficiente por darme la oportunidad de realizar una carrera y ofrecerme el apoyo y cariño siempre que ha sido necesario. Los cuatro sabemos han sido años con muchos altibajos y nada fáciles, pero juntos hemos llegado hasta aquí.

También agradecer a todas aquellas personas que me han acompañado durante estos cuatro años, incluyendo siempre a mi familia, tanto los que están como los que ya no están, a mis amigos y amigas por el apoyo y los momentos necesarios de desconexión, y a toda persona que ha ido apareciendo y convirtiéndose en importante en mi vida durante este tiempo. Gracias de corazón.

A su vez dar gracias a todas las personas que han contribuido a mi formación educativa, desde las profesoras y profesores del colegio e instituto Adolfo Suárez de Las Mesas, mi pueblo, pasando por el instituto IES Fray Luis de León de Las Pedroñeras hasta la ETSI de la Universidad de Sevilla. De verdad, gracias.

Por último, agradecer a los miembros de la empresa 4i Intelligent Insights y a mi tutor de TFG Iván por ofrecerme la posibilidad de llevar a cabo este trabajo y guiarme durante el mismo.

David Horcajada JImínez

Sevilla, 2021

Resumen

En este trabajo se llevará a cabo un proceso de detección de carros en imágenes extraídas de diferentes videos. Para ello se utilizan técnicas de Machine Learning, en concreto la detección de objetos en imágenes mediante clasificación, lo cual consiste en un proceso con tres etapas: anotación, entrenamiento y test. En esta memoria se explicará con detalle tanto el procedimiento seguido en cada una de estas etapas como las herramientas utilizadas.

Abstract

In this project, a car detection process will be carried out in images extracted from different videos. For this, Machine Learning techniques are used, specifically the detection of objects in images through classification, which consists of a process with three stages: annotation, training and test. In this memory, both the procedure followed in each of these stages and the tools used will be explained in detail.

Índice

ÍNDICE DE CÓDIGOS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En esta primera parte de la memoria se realizará una breve introducción al proyecto junto con una serie de razones por la cual se ha decidido abordarlo. Por otro lado, se indicarán los objetivos a conseguir al final del mismo junto con una indicación de la estructura que seguiremos durante el desarrollo de la memoria para la explicación del proceso completo.

1.1 Motivación y contexto.

La detección de objetos es un tema fundamental pero desafiante en el campo del análisis de imágenes, desempeñando un papel importante en una amplia gama de aplicaciones y recibiendo especial atención en los últimos años. Esta detección es, a su vez, una combinación de clasificación de imágenes con localización precisa precisa de objetos que proporciona una adecuada y completa comprensión de la imagen.[1]

La detección en imágenes mediante clasificación a través de algoritmos de Machine Learning implica el uso de una red neuronal convolucional, la cual se define como el algoritmo capaz de proporcionar al computador del sentido de la vista, que detecte una cantidad limitada o específica de objetos. Este tipo de problema de detección presenta está caracterizado por:

- **Bounding Box:** es un rectángulo "imaginario" que sirve como punto de referencia para la detección de objetos. Estos rectángulos son dibujados por los anotadores sobre las imágenes, extrayendo el objeto de interés de las imágenes mediante la definición de sus coordenadas X e Y. Además, es una de las técnicas más populares en la anotación de imágenes en Deep learning, debido a que comparado con otros métodos el uso de BoundingBoxes nos reduce costes y aumenta la eficiencia en la anotación.

- **Clases:** diferenciación de los distintos tipos de objetos que se pretenderán detectar.

- **Confidence**: probabilidad de que una detección realizada contenga un objeto. [2]

Partiendo de esta base, se decidió abordar un proyecto en el cual, haciendo uso de una metodología apropiada para ello, se lograse disponer de un algoritmo capaz de realizar la detección de un tipo concreto de objetos, que en este caso serán carritos.

Este proyecto ha sido llevado de manera coordinada con la empresa **4i Intelligent Insights,** la cual se encargaría de proporcionarnos un conjunto de videos procedentes de diferentes supermercados, en los cuales se pueden apreciar distintos tipos de carros, lo que hará posible el desarrollo de la idea planteada.

1.2 Objetivos.

Los objetivos marcados para ser cumplidos durante el proyecto son los siguientes:

- Estudio del problema de detección, así como las diferentes etapas que lo conforman.
- Para cada una de las etapas, hacer un estudio de las posibles herramientas a emplear, eligiendo una para cada etapa.
- Estudio de la red neuronal YOLO, empleada para el proceso de detección de imágenes y de la cual se hará uso.
- Obtención de diferentes modelos de detección, haciendo uso de un conjunto de imágenes más o menos reducido y viendo las diferencias entre ellos.
- Planteamiento de posibles avances o mejoras a realiza en un futuro, partiendo de este mismo proyecto.

1.3 Estructura del documento.

Es así que se planteará una división del contenido durante la memoria, disponiendo de la siguiente distribución:

- Capítulo 2: En este capítulo se explicará el estado del arte, presentando y explicando de manera más detallada la red neuronal YOLO, la cual será empleada para el proceso de detección, así como su tercera versión YOLOv3.
- Capítulo 3: Presentación y explicación detallada de la red neuronal YOLO, la cual será empleada para el proceso de detección, así como su tercera versión YOLOv3.
- Capítulo 4: Metodología empleada para la resolución del problema y resultados obtenidos.
- Capítulo 5: Conclusión del proyecto y posibles mejoras a realizar en un futuro.
- Referencias: Se referenciará todo aquel documento, libro, etc, del que se haya extraído información.

2 ESTADO DEL ARTE

2.1 Introducción

En este estado se explicará es estado del arte en la detección de objetos usando técnicas de Deep Learning.

La detección de objetos es una combinación de clasificación de imágenes con localización precisa de objetos que proporciona una completa comprensión de la imagen. Antiguamente, la extracción manual de características acompañada por arquitecturas sin mucha profundidad para el entrenamiento eran utilizadas en este problema de detección. No obstante, con a llegada de las herramientas de Deep Learning se han conseguido superar muchas de estas limtaciones de las técnicas tradicionales para la detección de objetos. La detección genérica de objetos ha sido llevada un punto más lejos, dividiéndose en categorías como la detección de caras, detección peatonal, etc. Esto es un proceso fundamental en el proceso de visión por computador el cual proporciona información semántica sobre imágenes y videos.

Esto tiene diferentes aplicaciones en varios campos de la vida, como puede ser el reconocimiento de caras, diagnósticos médicos, clasificación de imágenes, etc. Por tanto, este campo ha sid centro de estudio de muchosinvestigadores recientemente.

La detección de objetos incluye dos operaciones, la localización del objeto en la imagen y su clasificación a una determinada clase a la que pertenece y la cual definirá el tipo de objeto del que se trata.

Reciéntemente se han publicado un gran número de estudios impresionantes en cuanto a la detección de objetos se refiere, tal y como se muestra en la figura 2.1. Estos estudios han sido llevados a cabo sobre aplicaciones que son capaces de detectar objetos particuares como podría ser la detección de texto, caras, peatonal o de vehículos. Otros estudios recientes se centran en los problemas en la detección en lugar de en su principio básico de funcionamiento.Es aquí donde entra el juego el Deep Learning.

El Deep Learning ha traído avances significativos en diferentes áreas como la detección de objetos, procesamiento de lenguaje natural, reconocimiento de voz, imágenes médicas o reconocimiento visual, entre otras, a través de un proceso de aprendizaje de representaciones complejas, abstractas y sutiles por parte de los sistemas.

Los avances actuales en cuanto a la detección de objetos necesitan ser puestos en común, especialmente para nuevos investigadores que quieren realizar investigaciones acerca de la visión artificial.

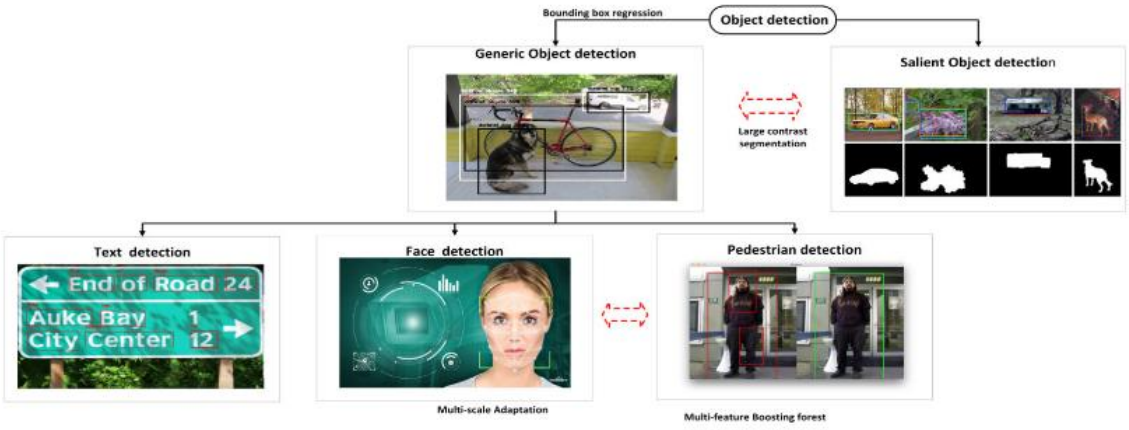


Figura 2.1: Estudios recientes realizados sobre la dtección de objetos.

Antes de entrar de lleno en los detalles de la detección de objetos mediante técnicas de Deep Learning, es importante explorar los beneficios que proporciona la arquitectura base del Deep Learning. Una red neuronal con una arquitectura profunda es conocida como un 'deep model'. La era de las redes neuronales comienza en 1940; la idea detrás de esto fue la resolución de un problema común de aprendizaje a través de la imitación del cerebro humano. La popularidad del Deep Learning aumenta en los 80s y 90s con el desarrollo de un algoritmo de 'back-propagation' propuesto por Hinton.

A principios de los 2000, la popularidad del Deep Learning comenzó a decaer debido a la falta de 'big data', los altos requerimientos de potencia computacional y el rendimiento insignificante en comparación a otras herramientas de Machine Learning. No obstante, en el año 2006 se produjo un ascenso en cuanto a popularidad del Deep Learning dado sus fantásticos resultados en el reconocimiento del habla. Alguno de los factores que contribuyeron a esta recuperación fueron los siguientes:

- En primer lugar, la principal razón fue la disponibilidad de grandes datasets de entrenamiento con anotaciones como en ImageNet.
- La invención de sistemas computacionales en paralelo de alto rendimiento, como las GPUs.
- Avances significativos en la arquitectura del modelo de Deep Learning y en las estrategias de entrenamiento.Auto-Encoder y Restricted Boltzmann Machine proporcionan un buen comienzo sin supervisión y estrategias de pre-entrenamiento por capas. El problema de overfitting durante el proceso de entrenamiento pueden ser resueltos usando aumento de datos y eliminación de regularización. Además, se lleva a cabo una normalización de Batches para la optimización del tiempo. La era de alto rendimiento comienza con los avances en la arquitectura de la red, como en AlexNet, GoogleNet, ResNet, etc.

La base principal de los modelos de Deep Learning es un modelo de red neuronal convolucional.El mapa caracterizado es unnombre adicional para el modelo de capas de la red neuronal y su capa de entrada es una matriz tridimensional para la intensidad de píxel de los tres canales de color.

Cada neurona se adjunta a una neurona adyacente hasta la capa posterior. Filtrando y agrupando se pueden crear especificaciones de características más robustas.

Algunas de las ventajas en el uso de una red neuronal convolucional sobre los métodos tradicionales son las siguientes:

- La habilidad de una 'deep neural network' para emitir es mucho más alta que en los métodos convencionales
- Es capaz de aprendr la jerarquía de las características automáticamente de manera directa desde los datos usando unaestructura multi-fase que supone a su vez una representación multi-nivel desde los píxels hasta las características semánticas de alto nivel.
- La arquitectura de la red neuronal convolucional puede proporcionar mejoras en ciertas tareas como la regresión en Boundng Boxes y clasificación multi-tarea de manera similar a la usada en Fast R-CNN.

La detección genérica de objetos, por su parte, es el proceso de localización de u objeto a través del uso de Bounding Boxes para indicar la confianza en la detección así como para clasificarlo en una clase concreta.

En la figura 2.2 se puede ver el estado del arte actual en la detección genérica de objetos, de manera que explicaremos algunos de ellos.

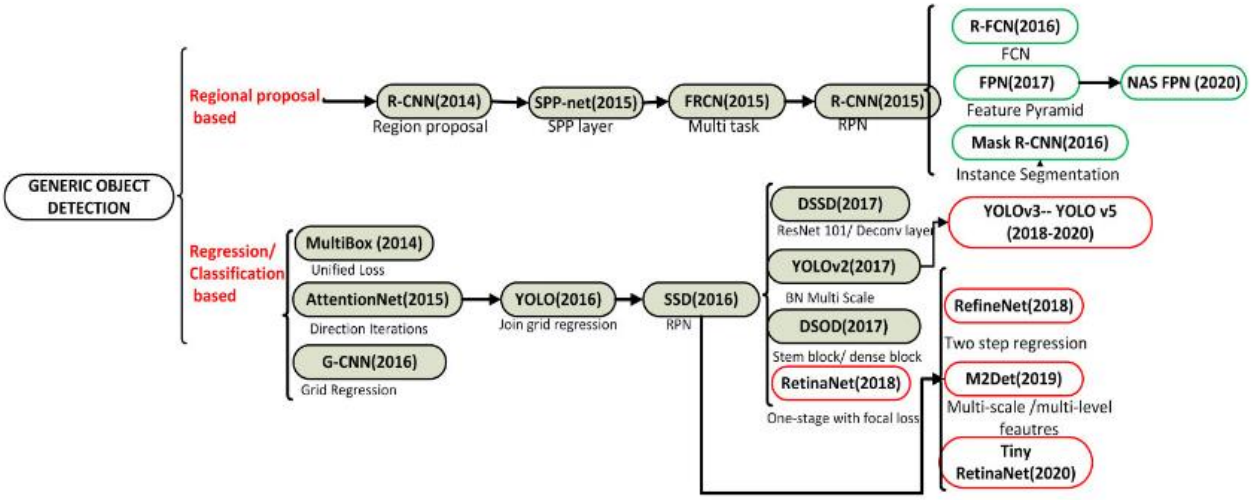


Figura 2.2: Estado del arte de la detección genérica de objetos.

2.2 Region Proposal Object Detection

Esta estructura imita la capacidad de atención del cerebro humano. En primer lugar, se realiza un escaneo de todp el escenario para posteriormente centrarse en la región de interés. De entre todas OVERRRFEAT presentea el rendimiento más prometedor. Fue la primera vez que nua red neuronal convolucional fue introducida en modo de ventana corrediza, la cual se encargaba de predecir Bounding Boxes directamente.

- *R-CNN*

El modelo R-CNN incluye tres módulos, tales como propuesta de regiones, extracción de características con base en 'deep' redes neruronales convolucionales y un módulo de clasificación/localización (figura 2.3)

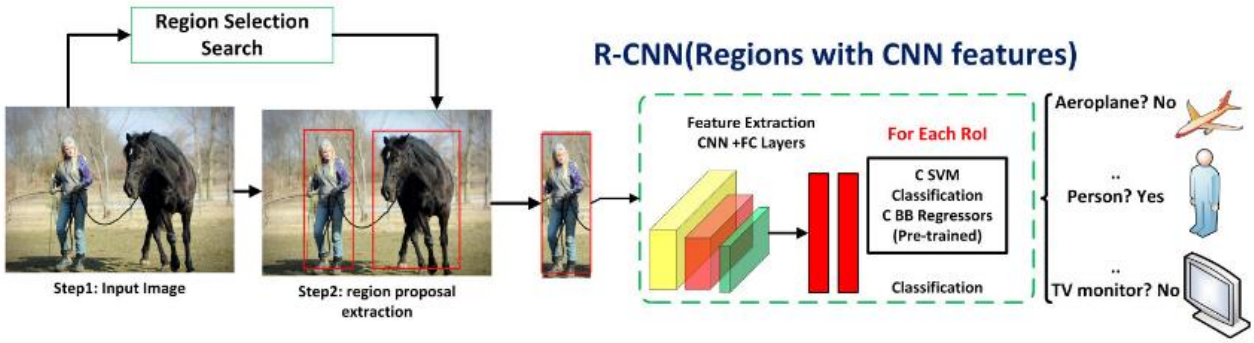


Figura 2.3: Estructura R-CNN.

En cuanto a la propuesta de regiones, R-CNN hace uso de una búsqueda de selección y genera 2000 regiones

propuestas a partir de una simple imagen. La indicación prominente y la agrupación de abajo hacia arriba han sido usadas para proporcionar una más rápida selección del tamaño de Bounding Boxes candidatos y reducir la búsqueda de espacio para la detección de objetos.

En el segundo módulo, consistente en la red neuronal convolucional, se realiza una redimensión de la imagen para obtener alrededor de 4096 características dimensionales.

En el tercer módulo, la localización final del objeto es respaldada por un ajuste y filtrado del resultado en las regiones a través de regresión de Bounding Boxes y supresión no máxima. Normalmente, los modelos pre-entrenados son usados para evitar el problema de insuficiencia de datos.

- *FPN(Featured Pyramid Network)*

La invarianza de escala en los sistemas de detección de objetos pueden ser abordados a través de la construcción de pirámides de características en las pirámides de la imagen , dando lugar a pirámides de imágenes caracterizadas. Sin embargo, esto incrementa rápidamente el consumo de memoria y tiempo de entrenamiento. En muchas de las técnicas, una sola escala de entrada es usada para representar semánticas de alto nivel. En contraposición a esto, una variación de escala puede llevarnos hacia un modelo más robusto (figura 2.4) e inconsistencia entre el aumento de tiempo de entrenamiento y test debido a la construcción de la imagen durante el proceso de test.

Sin embargo, la arquitectura de FPN en un camino de arriba abajo y de abajo a arriba. Esto se traduce en una combinación de baja resolución y características robustas semánticamente con resolución de alto nivel usando severas conexiones laterales, tal y como se muestra en el apartado d) de la figura 2.4. Con el paso de 2, el 'sown-sampling' del mapa de características produce una jerarquía de características en el camino de abajo hacia arriba de la red troncal de reenvío ConvNet. Mientras, en el camino de arriba a abajo se crea un conjunto de referenciade los mapas de características a través de la selección de la última capa de cada etapa de la red, la cual es un conjunto de mapas de salida de cada capa de tamaño fijo. Los mapas de características de las capas superiores no son muestreados y se mejoran usando una conexión auténtica del mismo tamaño espacial desde abajo hacia arriba y desde arriba hacia abajo construyendo un camino de arriba a abajo.

Por su parte, las dimensiones del canal han sido reducidas, agregando una capa convolucional de 1x1 al mapa no sampleado mientras que la adición de elamentes se usa simplemente en caso de emergencia. El mapa de características final se genera a través de la adición de una convolución 3x3 para cada mapa combinado reduciendo el efecto de aliasing.El mapa de resolución más excepcional se consigue a través de múltiples iteraciones.

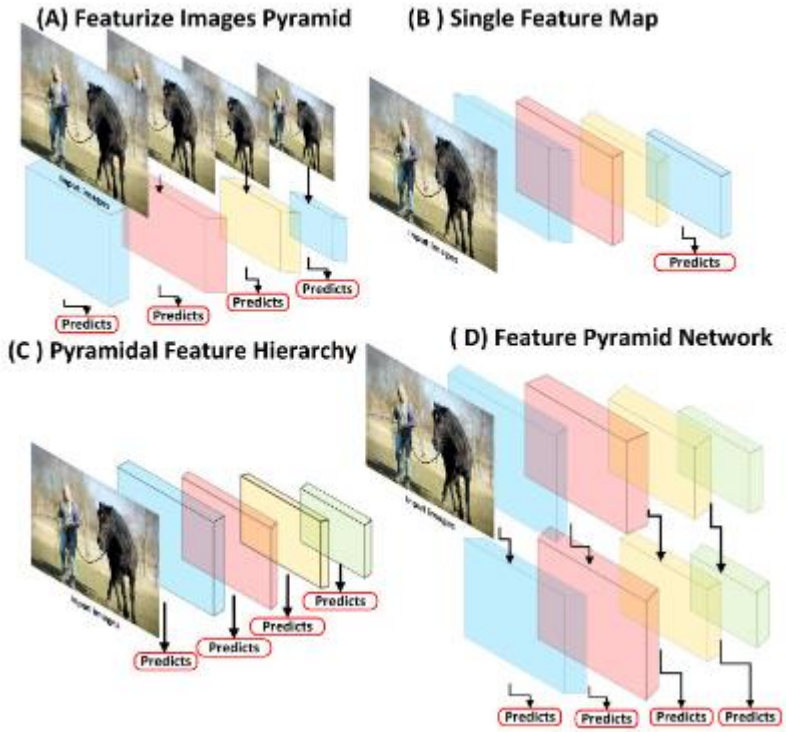


Figura 2.4: Concepto central de red FPN

Dentro de este tipo de Region Proposal Object Detection podemos encontrar otras redes, como son Mask R-CNN, Fast R.CNN. Faster R-CNN o R-FCN.

2.3 Regression/Clasification Object Detection

Esta arquitectura incluye varias fases correlacionadas, tales como generación de propuestas de regiones, extracción de características mediante el uso de una red neuronal convolucional, regresión de Bounding Box y clasificación, la cual entrena por separado.Un entrenamiento alternativo requiere del dearrollo de parámetros de convolución compartidos entre la red de detección y RPN, usado en módulo final-a-final del Faster R-CNN. En aplicaciones de tiempo real, el tiempo requerido para las tareas de detección de objetos se ve reducido con la invención de arquietecturas de una sola etapa basadas en probabilidad de clases, mapeadas directamente del pixel de la imagen a las coordenadas del Bounding Box, y clasificación/regresión global.

Esfuerzos significativos han sido ejercidos para mejoras los modelos de detección como tareas de clasificación/regresión. D.Erhan ha usado regresión basada en una red neuronal convolucional para detectar objetos mediante el desarrollo de máscaras binaria de test para imágenes e inferencia de Bounding Boxes para los objetos extraídos. Aún así, localizar la superposición de objetos y usar sobremuestreo ara producir un Bounding Box es una tares complicada. Para ello el autor propone un modelo de red neuronal convolucional basado en dos ramas paralelas, la primera para generar la máscara de segmentación de clases y la segunda para detectar el centro del objeto. El rendimiento del modelo es eficiente debido a que la segmentación y el resultado o marca de clase son obtenidos en el mismo modelo, el cual contiene en su mayoría operaciones ligadas a la red neuronal convolucional. [1]

Dentro de este conjunto aparece YOLO, el cual se explicará a continuación con una mayor profundidad dado

que será el utilizado para la realización de nuestro proyecto, tal y como se explicará en el tercer apartado de esta memoria.

Además de YOLO aparecen otros como DSOD o RetinaNet.

- YOLO

YOLO (You Only Look Once) es una red de detección de objetos en tiempo real. YOLO apareció como un nuevo enfoque para la detección de objetos, puesto que, si anteriormente a esta red se reutilizaban los clasificadores para la detección de formas, ahora la detección se orienta a un problema de regresión en el que aparecen cuadros delimitadores separados espacialmente y con probabilidades de clase asociada. Así,una sola red neuronal predice cuadros delimitadores y probabilidades de clase directamente desde imágenes completas a través de una sola evaluación.Desde que todo el pipeline de detección se compone de una sola red, este puede ser optimizado extremo a extremo directamente en cuanto a rendimiento.

La arquitectura de YOLO es extremadamente rápida, procesando en tiempo real a 45 frames por segundo. A su vez, existe una versión más rápida de la red, Fast YOLO, que procesa 155 frames por segundo manteniendo alrededor del doble de precisión en sus detecciones en comparación a otros detectores en tiempo real.

En contraposición, YOLO comete por norma general más errores de localización, pero es menos propenso a predecir falsos positivos, esto es, marcarcomo detección un objeto que realmente no debería ser detectado. YOLO aprende representaciones generales de los objetos y supera a otros métodos de detección como pueden ser R-CNN o DPM.

❖ Arquitectura y funcionamiento de la red.

Procesar imágenes con YOLO es simple y directo, realizando uns serie de pasos:

1. Cambiar tamaño de la imagen de entrada a 488x488.
2. Ejecutar la red convolucional.
3. Umbralizar las detecciones resultantes a través de la ‘confidence’ del modelo.

Esto puede visualizarse en la figura 3.1.

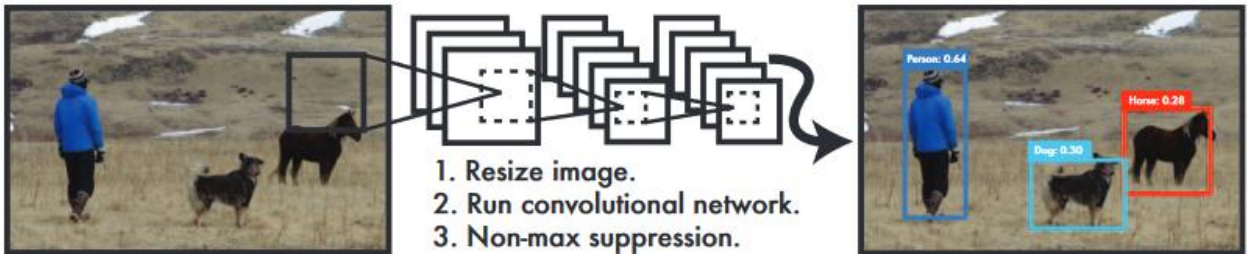


Figura 2.5: Estructura sistema de detección de YOLO.

Para evitar los problemas de rapidez que aparecían en los métodos que se nombraron anteriormente, YOLO replantea la detección de objetos como un único problema de regresión desde, directamente desde los píxeles de la imagen hasta las coordenadas del Bounding Box y la probabilidad de pertenencia a una clase determinada. Con este planteamiento, solo miramos una vez a la imagen para predecir qué objetos hay en ella y dónde se encuentran situados.

YOLO es relativamente simple: una sola red convolucional predice simultáneamente múltiples Bounding Boxes y probabilidades de clases para cada uno de ellos. YOLO entrena directamente sobre la imagen completa y optimiza el rendimiento de detección.

El hecho de que YOLO sea extremadamente rápido se debe a la simplificación de su pipieline. Simplemente se ejecuta la red sobre la imagen. Dado que como se dijo anteriormente la red funciona a 45 frames por segundo, esta será capaz de procesar la transmisión de video en tiempo real con menos de 25 milisegundos de latencia, con más del doble de precisión que el resto de sistemas de detección en tiempo real.

En segundo lugar, al visualizar la imagen completa YOLO codifica implícitamente información contextual sobre las clases y su apariencia. Esto le hace cometer menos de la mitad de errores en comparación con otros sistemas como Fast R-CNN.

En tercer lugar, YOLO aprende representaciones generales de los objetos, siendo menos probable a "romperse" cuando se aplicaa nuevos dominios o inputs inexperados.

Es importante tener en cuenta, tal y como se indicó anteriormente, que YOLO todavía se encuentra por detrás de otros sistemas de detección en cuanto a precisión se refiere, ya que si bien los detecta rápidamente esta detección puede llegar a ser bastante imprecisa en cuanto a localización, especialmente en objetos pequeños.

YOLO dispone de código abierto.

En cuanto a la detección, YOLO la realiza de forma unificada, es decir, unificando todos los elementos de detección en una sola red neuronal. Así, utiliza características de toda la imagen para cada Bounding Box, predeciendo a que clase pertenece cada uno simultáneamente.

El sistema divide la imagen de entrada en una cuadrícula de SxS.Si el centro de un objeto cae en una de las celdas de esta cuadrícula, esa cuadrícula es la responsable de detectar este objeto.

Cada celda de la cuadrícula predice B cuadros y proporciona un resultado de confianza para cada Bounding Box. Esta marca de confianza indica la precisión con la que piensa que el Bounding Box ha sido capaz de detectar el objeto que hay contenido dentro de él. Esta confianza se define como la convolución entre la probabilidadad de pertenencia a una determinada clase y la intersección entre el Bounding Box de anotación y el de predicción (Intersect of Union,IoU). Si no existe objeto, este resultado ha de ser cero. De lo contrario, queremos que el resultado de confianza sea igual al del IoU.

Cada Bounding Box consta de 5 predicciones (*x,y,w,h and confidence*). Las coordenadas *x* ,*y* representan el centro del Bounding Box en relación a los límites de la celda de la cuadrícula. El ancho y el alto son predecidos en relación a la imagen completa. Finalmente la confianza o *confidence* representa el IoU entre el Boundng Box de predicción y el cuadro de anotación.

Cada celda predice a su vez una probabilidad *C* condicional de clases Pr(Classi|Object),la cual está condicionada a la celda que contiene al objeto. Solo se predicen un conjunto de probabilidades por celda de la cuadrícula, independiente del número de Bounding Boxes *B*.

Durante la estapa de test, se multiplica la probabilidad condicional de clase con la confianza del Bounding Box individual de predicción, lo cual nos da un resultado un resultado de pertenencia específica a cada clase de cada "caja". (figura 3.2)

$$Pr(Classi|Object) * Pr(Object) * IOUtruthpred = Pr(\ Classi) * IOUtruth$$

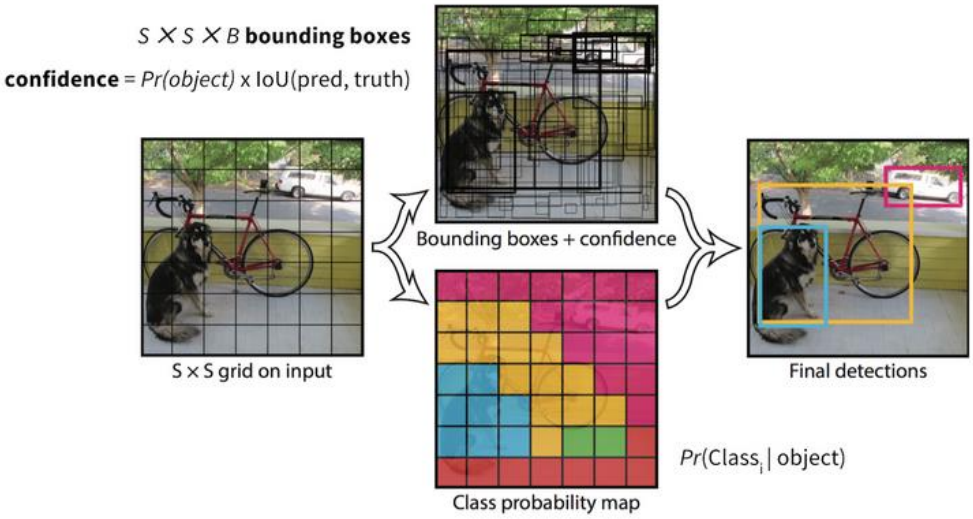


Figura 2.6 Funcionamiento interno de YOLO.

La red de YOLO es una red convolucional.

Las primeras capas convolucionales extraen características de la imagen mientras que el conjunto de capas conectadas predicen las probabilidades y coordenadas de salida.

Esta red neuronal está inspirada por el modelo de GoogleNet para la clasificación de imágenes. La red de YOLO está constituida por 24 capas convolucionales seguidas por 2 capas totalmente conectadas. Simplemente se usan 1x1 capas de reducción seguidas de 3x3 capas convolucionales. (figura 3.3)

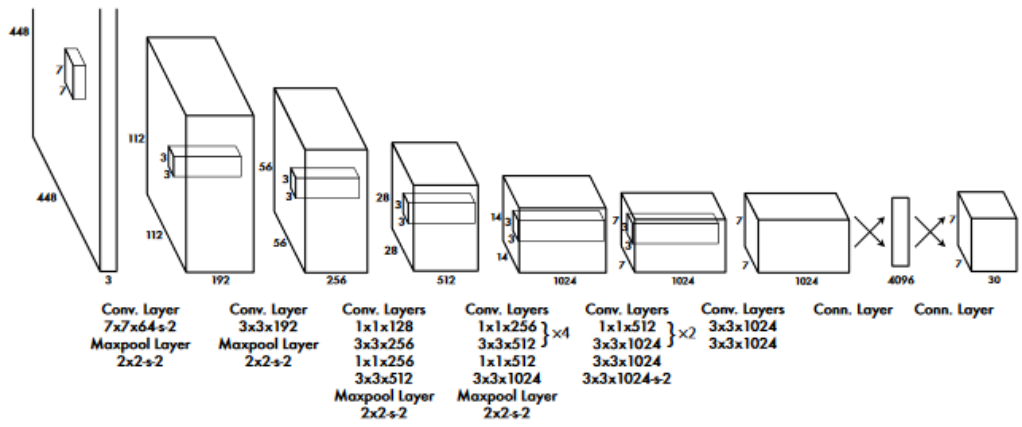


Figura 2.7: Arquietectura de la red neuronal de YOLO

Para llevar a cabo el entrenamiento en YOLO, se debe realizar previamente un entrenamiento, Una posibilidad sería pre-entrenar las 20 primeras capas convolucionales seguidas de una capa de agrupación-media y una capa completamente conectada.

Una vez preentrenado, convertimos el modelo para llevar a cabo la detección. Siguiendo el ejemplo proporcionado por Ren et al. Se aladen 4 capas convolucionales y dos completamente conectadas con unos pesos inicializados de manera aleatoria. Posteriormente, se incrementa la resolución de entrada de la red a 488x488.

Por su lado, la última capa predice la probabilidad de clase y las coordenadas del Bounding Box. *x,y,w,h* son expresadas como 0 y 1, ya que *w,h* se normalizan respecto al tamaño de la imagen real,y *x,y* se parametrizan para ser un offset de una celda en particular.

Se usa una función de activación lineal para la última capa y todas las demás usan la activación linela rectificada.

Para la optimización a la salida de nuestro modelo a través de un eroor de suma de cuadrados debido a la facilidad que esto supone para la optimización, aunque no corresponda con nuestro objetivo de maximización de la precisión media.

Para remediar esto, auentamos la pérdida en las predicciones de las coordenadas de los Bounding Boxes y decrementamos la perdida de confianza en las predicciones para los Bounding Boxes que no contienen objetos. Para ello se utilizan dos parámetros, λcoord y λnoobj, ambos con un valor de 5.

La suma de cuadrados también equilibra el error en los pesos en cajas grandes y pequeñas. Nuestra métrica de error debería reflejar una menor importancia en desviaciones en cajas grandes que en cajas pequeñas. Para ello, predecimos la raíz cuadrada de *w,h* en lugar de hacerlo directamente.

YOLO preice múltiples Bounding Boxes por cada celda, mientras que en el entrenamiento solo se busca la predicción de un Bounding Box para cada objeto. Para ello asignamos a un “predictor” como responsable de predecir el objeto en función de cual disponer de una mayor IoU. Esto hace que cada uno de estos “predictores” mejore la predicción de ciertos tamaños, relaciones de aspecto o clases de objetos.

No obstante, YOLO consta de una serie de limitaciones. Al imponer fuertes restricciones en cuanto a la predicción de Bounding Boxes pues cada celda solo puede predecir dos cajas y solo puede tener una clase. Esta limitación espacial reduce el número de objetos cercanos que nuestro modelo es capaz de predecir, siendo bastante impreciso en la detección de pequelos objetos que aparecen en grupo.

Además, dado que nuestro modelo aprende a predecir Bounding Boxes a partir de datos, se “pierde”al generalizar objetos que presentan relaciones de aspecto nuevas o inusuales, así como nuevas o inusuales configuraciones. El modelo utiliza por tanto características relativamente generales para predecir Bounding Boxes dado que su arquitectura dispone de múltiples capas de submuestreo desde la imagen de entrada.

Finalmente, mientras entrenamos, nuestra función de error trata de la misma manera errores en Bounding Boxes grandes y pequeños, mientras que un error pequeño en un Bounding Box grande no tiene demasiada importancia al contrario que un error pequeño en un Bounding Box pequeño, ya que este tendría un efecto mucho mayor en el valor del IoU.

Desde su aparición se han desarrollado diferentes versiones, siendo la última de ellas la cuarta (YOLOv4), aunque en ese proyecto se hará uso de la tercera (YOLOv3). [3](doc YOLO)

- *YOLOv3*

En esta tercera versión de YOLO, el sistema predice los Bounding Boxes usando grupos de dimensiones como "anchor boxes" . Así la red predice 4 coordenadas por cada Bounding Box, *tx,ty,tw,th*. Si la celda está desplazada (offset) desde la esquina superior izquierda por (*cx,cy)* y el Bounding Box anterior tiene anchura y altura *pw,ph*, la predición corresponde con (figura 3.4):

$$bx = \sigma(tx) + cx$$

$$by = \sigma(ty) + cy$$

$$bw = pwe^{tw}$$

$$bh = phe^{th}$$

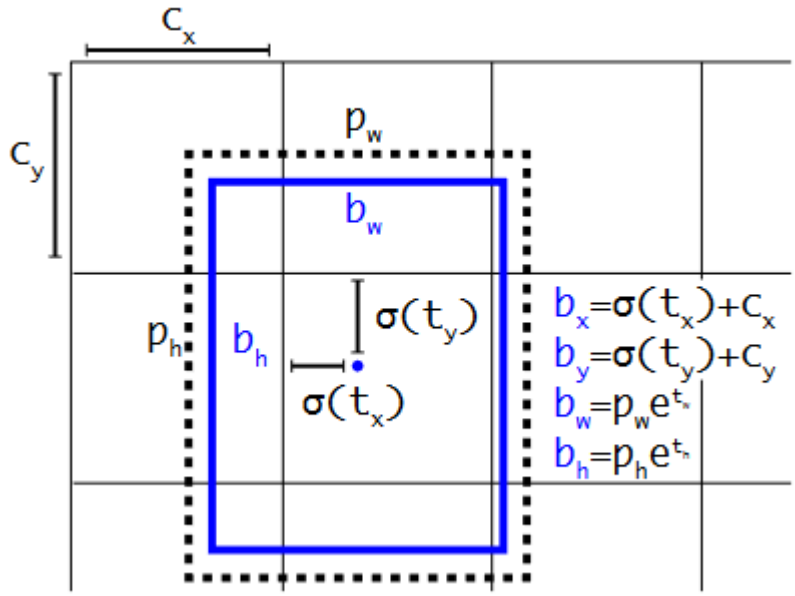


Figura 2.8: Predicción Bounding Box en YOLOv3

Durante el entrenamiento se hace uso de la suma de pérdida del error al cuadrado.

YOLOv3 predice un resultado objetivo para cada Bounding Box usando regresión logística. Esto debería realizarse si el Bounding Box anterior sobrepasa la "tierra de verdad" del objeto en mayor medida que cualquier Boundnig Box anterior. Si este Bounding Box anterior no es el mejor pero sobrepasa la tierra de verdad del objeto en una cantidad superior a cierto umbral, se ignora la predicción. Este umbral es .5i un Bounding Box anterior no está asignado a ninguna tierra de verdad de ningún objeto no incurre en la pérdida de coordenadas o predicción de clase.

Cada cuadro o caja predice la clase que el Bounding Box puede contener a través de un clasificador multietiqueta. Para lograr un buen rendimiento, se usa logística clasificadora independiente. Durante el entrenamiento se usa la pérdida de entropía binaria cruzada (binary cross-entropy los) para las predicciones de clase.

El clasificador multietiqueta modela mejor los datos en caso de disponer de datassets en los que se produce una superposición de etiquetas.

YOLOv3 predice Bounding Boxes en 3 diferentes escalas. Nuestro sistema extrae características desde estas escalas usando un concepto similar para así presentar una red piramidal. A partir del extractor base de características se añaden una serie de capas convolucionales, de las cuales las últimas predicen un tensor en 3-d que codifica el Bounding Box, la objetividad y la predicción de clases.

Se hace uso a s vez de una nueva red para llevar a cabo la extracción de características. Esta nueva red es un híbrido entre la red que constituía la versión anterior (YOLOv2), conocida como Darknet-19 y una nueva red residual. Nuestra red usa sucesivas capas convolucionales de 3x3 y 1x1 disponiendo de más conexiones a la vez que es significativamente mayor. En concreto dispone de 53 capas conolucionales por lo que recibirá el nombre de Darknet-53.(Figura3.5).

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

Figura 2.9: Estructura de la red Darknet-53.

Esta nueva red es mucho más potente que Darknet-19 y sigue siendo más eficiente que ResNet-101 o ResNet-152 (Figura 3.6)

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	**171**
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Figura 2.10: Comparativa entre diferentes redes.

Atendiendo a los resultados de la figura 3.6, vemos que Darknet-53 es mejor que ResNet-101 y 1.5 veces más

rápida, mientras que tiene un rendimiento similar a ResNet-152 pero es 2 veces más rápida.

Además, Darknet-53 logra la más alta medida de operaciones de punto flotante por segundo, lo cual significa que la red aprovecha mejor la GPU, haciendo su uso más eficiente a la vez que más rápido.

Por su parte, el entrenamiento se sigue realizando sobre la imagen completa Se hace uso de entrenamiento multi-escala, normalización de batches y todo aquello que yase realizaba previamente. Se hace uso de la estructura de Darknet-53 tanto para entrenamiento como para testado.

Es importante destacar, que con las nuevas predicciones multiescala, se logra revertir la situación de los problemas causados en la detección de objetos pequeños, aunque en comparativa funciona algo peor para medianos y grandes objetos. [4](Doc YOLOv3)

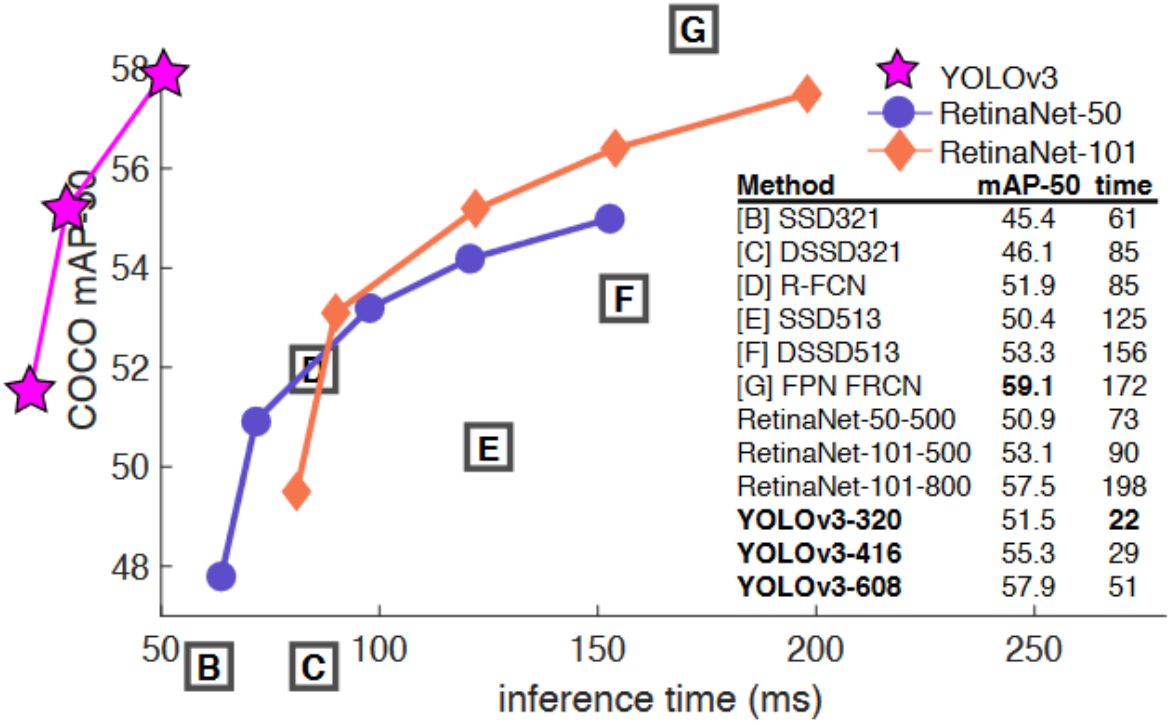


Figura 2.11: Rendimiento Yolov3 en comparación a RetinaNet-50 y RetinaNet-101

3 METODOLOGÍA E IMPLEMENTACIÓN

En coordinación con la empresa **4i Intelligent Insights,** se procederá a la realización de un problema de detección de carros en imágenes, el cual ha sido propuesto por la misma empresa.

Así, valiéndonos de distintos acuerdos de la empresa con una serie de supermercados se obtendrán una serie de videos de grabación continua a través de cámaras de seguridad, tanto frontales como cenitales, que nos servirán para la realización del problema.

3.1.Problema a resolver

Aplicando la detección de objetos mediante técnicas de Deep Learning, se procede a realizar un modelo que sea capaz de detectar carros, tal y como se explicó anteriormente. Para ello se hará uso de YOLOv3, así como de las herramientas necesarias para abordar el proyecto en su totalidad. Una vez desarrollado, el modelo deberá ser capaz de realizar detecciones y clasificar las mismas en las siguientes tres clases:

0. ***Shopping_carts*** : clase correspondiente a los carros de la compra.
1. ***Baby_carts*** : clase asociada a carritos de bebés.
2. ***Others*** : clase que abarca desde carritos de mano hasta varios carros en fila.

El problema constará por tanto de tres partes distintas, las cuales serán desarrolladas en los siguientes apartados:

- **Anotación.**
- **Entrenamiento.**
- **Testado.**

3.2. Anotación

La anotación puede ser vista como el proceso en el cual dotamos al modelo de visión artificial de información acerca de lo que aparece en la imagen. Así, una vez definidas las tres clases que vamos a tratar de detectar, estas son anotadas a través de Bounding Boxes.

Existen distintas herramientas para llevar a cabo la notación de imágenes. Para nuestro proyecto, haremos uso de **CVAT (Computer Vision Annotation Tool).**

CVAT es una herramienta de anotación gratuita, la cual nos ofrece la posibilidad de anotar imágenes de manera interactiva con un gran número de propiedades. Esta plataforma ofrece dos versiones, una de ellas una demo online, limitada a un máximo de 10 tareas con 500Mb para datos.

Por otro lado, la versión completa, cuya guía de instalación encontramos en la siguiente dirección: https://openvinotoolkit.github.io/cvat/docs/administration/basics/installation/

Resulta conveniente destacar dentro del proceso de instalación el uso de Docker, el cual es un contenedor de aplicaciones, permitiendo ejecutarlas, compilarlas, depurarlas y probarlas, aunque no será aquí donde expliquemos su uso debido a que todo lo necesario aparece en las instrucciones del enlace adjuntado en el párrafo anterior.

Una vez realizada la instalación, siempre que iniciemos el equipo debemos de asegurarnos de haber iniciado

correctamente el contenedor en caso de querer dar uso a la aplicación. Podemos tener certeza de ello sin necesidad de abrir la aplicación si en la barra de herramientas el símbolo del contenedor presenta el aspecto de la figura 3.1:

Figura 3.1 Símbolo Docker activo.

El símbolo de advertencia puede o no puede aparecer en caso de haber actualizaciones disponibles.

Tras haber inicializado correctamente el contenedor, pasamos al uso de la aplicación. Una vez creado nuestro superuser de manera local, algo que se explica en la guía de instalación, y abierta la herramienta, nos encontraremos con la pantalla de la figura 3.2 en la cual deberemos iniciar sesión:

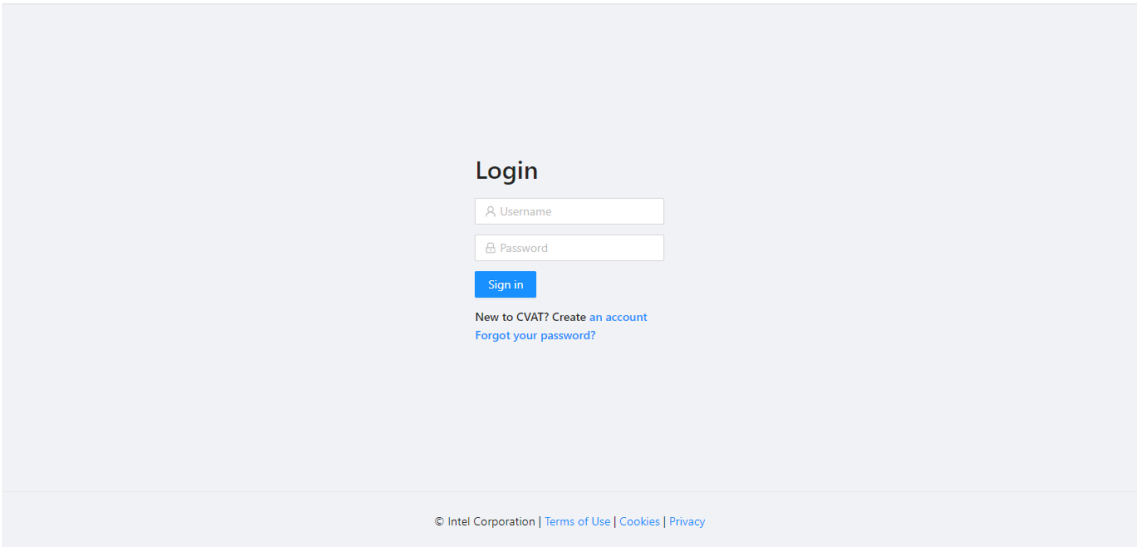


Figura 3.2 Pestaña de identificación en CVAT.

En el momento en que hayamos iniciado sesión correctamente, nos encontraremos con la pantalla de la figura 3.3:

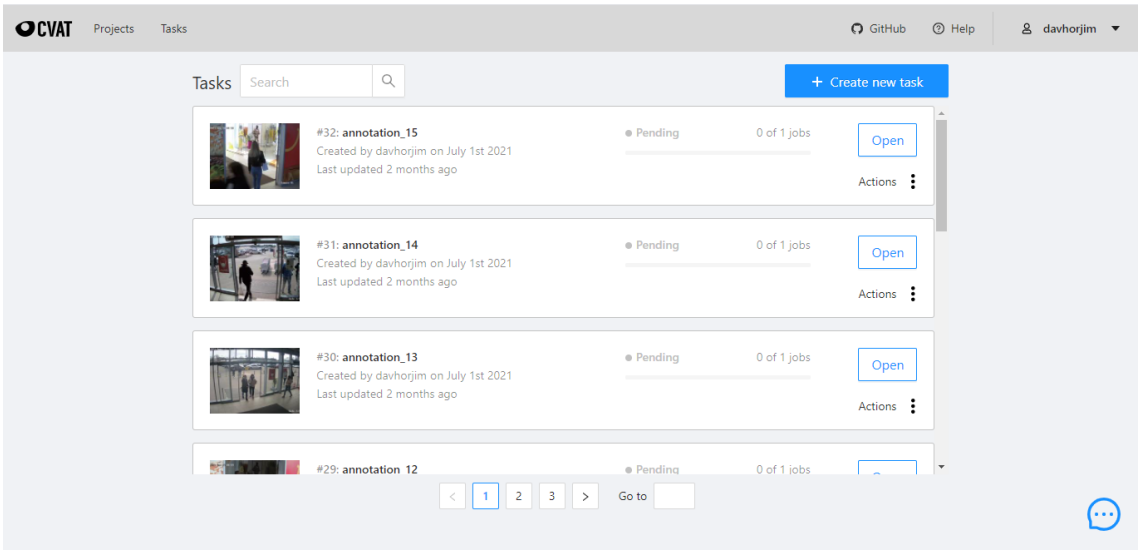


Figura 3.3 Página principal de CVAT.

Es aquí cuando podremos empezar a trabajar. Para trabajar en CVAT debemos tener claros los conceptos de proyecto (***Project***) y tarea (***Task***):

- ***Projects:*** Los proyectos, de manera general, sirven para definir el proceso, ya que son en estos en los que quedan definidas las clases que posteriormente serán utilizadas en las tareas para llevar a cabo la anotación, como veremos más adelante.

  A la hora de crear un nuevo proyecto, nos encontraremos con la pantalla de la figura 3.4:

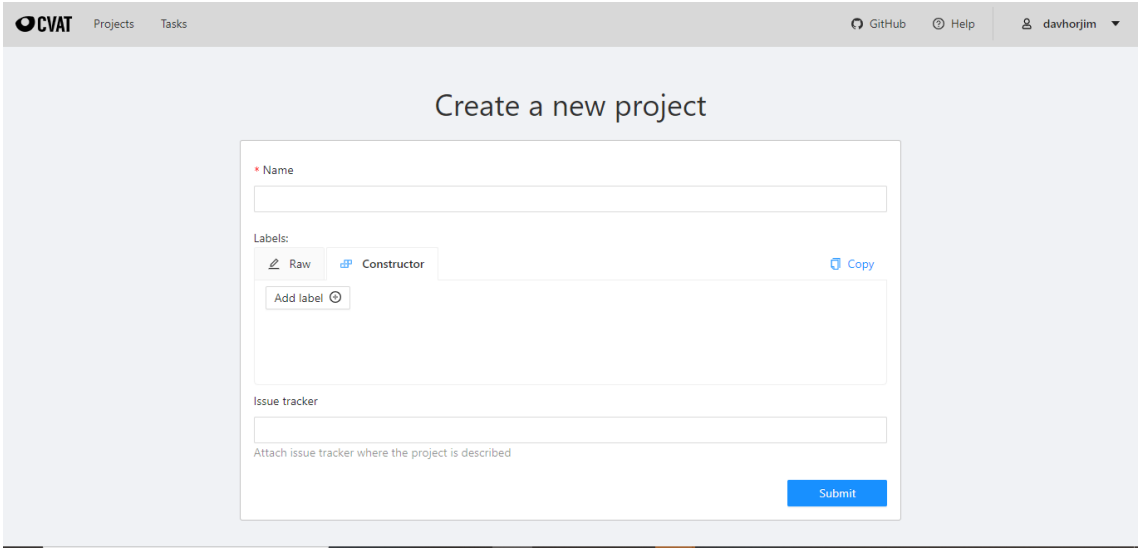


Figura 3.4 Crear proyecto en CVAT.

Como se puede intuir, debemos dotar a cada proyecto de un nombre, de manera que a través de él podremos asociar las tareas. Además, podemos crear las clases o etiquetas en este caso haciendo click izquierdo en Add label, y esto puede ser mediante dos formatos distintos :

- ***Raw:*** para usuarios más avanzados. Representa la información de las diferentes etiquetas en lenguaje *json* con opción de editar y copiar etiquetas a través de texto.[5]

- ***Constructor:*** Formato en el que se definen de manera interactiva. En él únicamente será necesario definir el nombre de la etiqueta o clase y darle un color determinado. Esto último es

importante a la hora de anotar para ser capaces de diferenciar los Bounding Boxes de las diferentes clases en caso de aparecer varias de ellas en una misma imagen.

- ***Tasks:*** Es en ellas en las que se llevará a cabo el proceso de anotación. A la hora de crear una tarea, nos encontraremos con la pantalla de la figura 3.5:

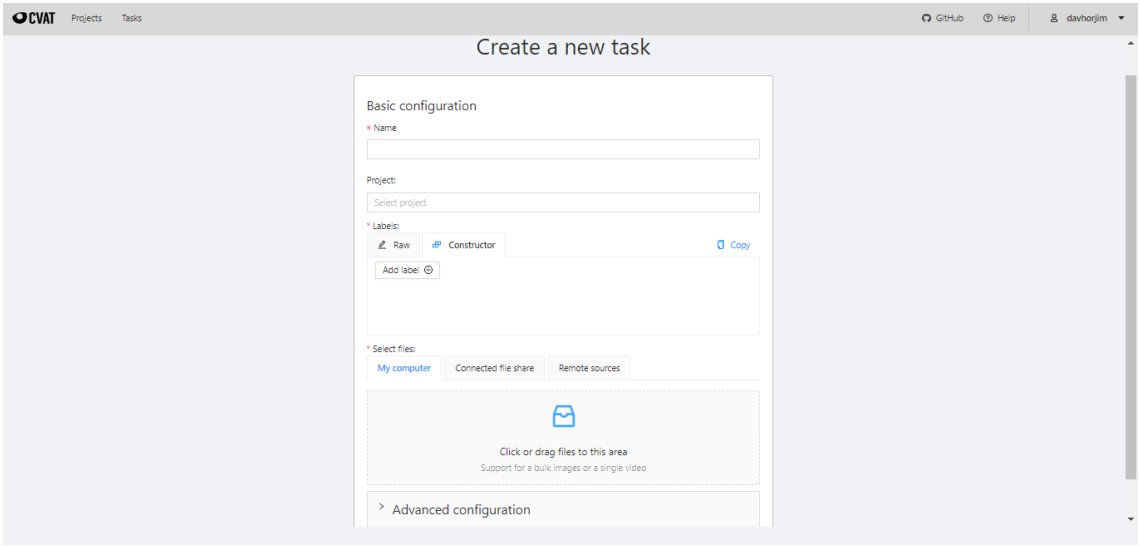


Figura 3.5 Crear tarea en CVAT.

Como se puede observar, debemos asignarle un nombre a la tarea, tras lo cual debemos asociarla a un proyecto. Esto se realiza simplemente indicando el nombre del proyecto al que queramos asociar la tarea en el campo *Project*. Esasí que las clases o etiquetas que fuesen creadas en el proyecto en cuestión estarán disponibles en la nueva tarea.

Tras haber realizado ambas cosas, simplemente con arrastrar las imágenes o videos que vayan a ser anotados a *select file* tendremos la tarea lista para comenzar a trabajar en ella.

Es así como para nuestro proyecto, a la hora de crear el proyecto se crearán 3 clases o etiquetas a la hora de crear un nuevo proyecto, una con el nombre de ***shoppingcarts***, otra con el nombre de ***baby_carts*** y otra que será ***others.*** Así, una vez abramos el proyecto aparecerán como en la figura 3.6:

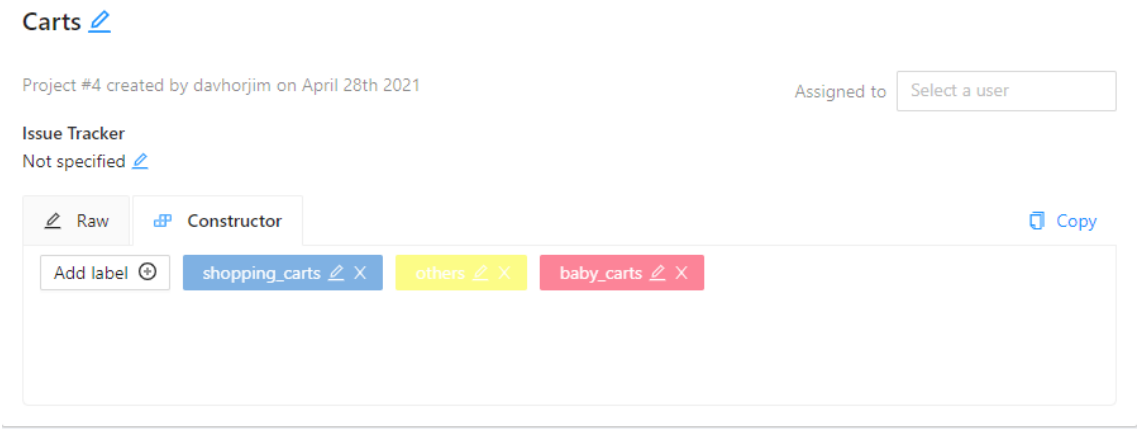


Figura 3.6 CVAT labels.

Una vez dentro del proyecto procedemos a abrir una de sus tareas. En concreto, para este trabajo se abrirán 10 tareas, de manera que serán anotados 10 videos.

Si abrimos una de las tareas creadas nos encontramos una interfaz como en la figura 3.7:

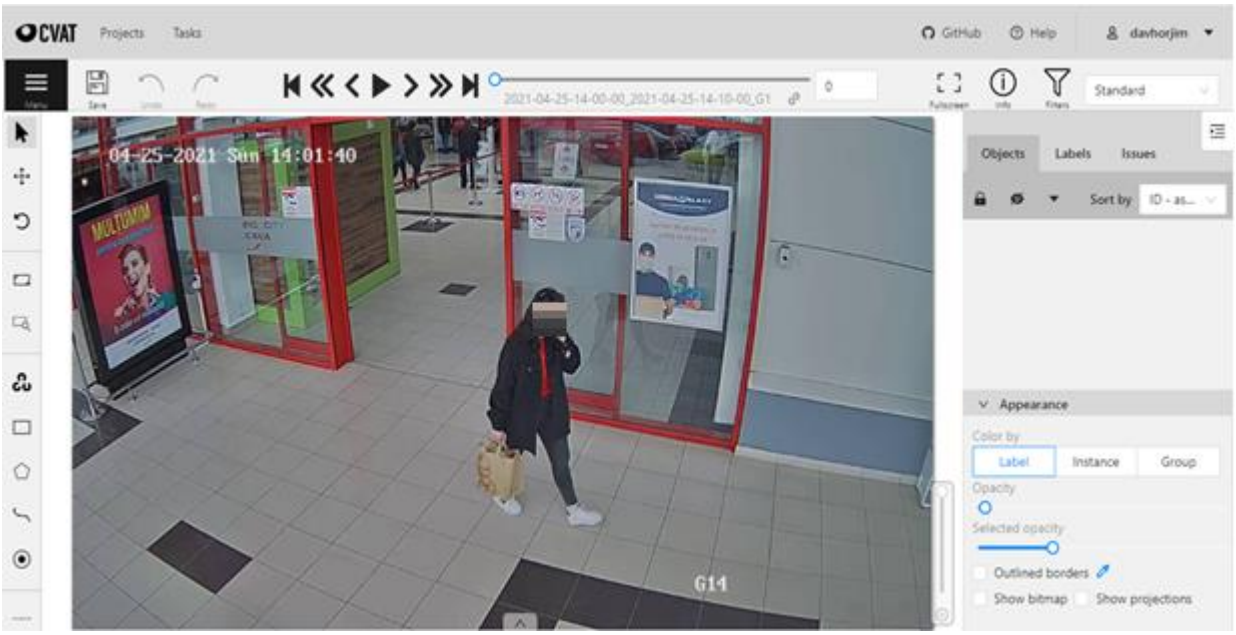


Figura 3.7 CVAT interface.

Esta es la interfaz de trabajo de CVAT, la cual se irá explicando paso a paso.

En primer lugar, si nos fijamos aparece un número, en este caso 0, en la parte superior. Esto se corresponde con el frame del video en el que estamos trabajando, ya que CVAT descompone el video en frames de manera automática, como parece lógico para poder anotar.

Para pasar de un frame a otro disponemos de tres opciones, podemos avanzar un frame haciendo click izquierdo en la flecha simple, avanzar 10 frames haciendo click en las dos flechas o avanzar hasta el último o primer frame si clickamos en la flecha limitada por una barra. Además se puede saltar a cualquier frame que deseemos escribiendo su número en la casilla donde aparece el cero y haciendo intro. En nuestro caso, se hará uso del avance de 10 frames para llevar a cabo la anotación.

Una vez vayamos avanzando y nos encontremos un carrito perteneciente a cualquiera de las tres clases definidas, debemos proceder a anotar manualmente. Esto se realiza dibujando el Bounding Box que delimita el objeto dentro de la imagen, teniendo en cuenta siempre la clase del objeto.

Para ello, dentro de la herramienta nos vamos a icono de la figura 3.8, en la cual aparecerá una pestaña con diferentes opciones:

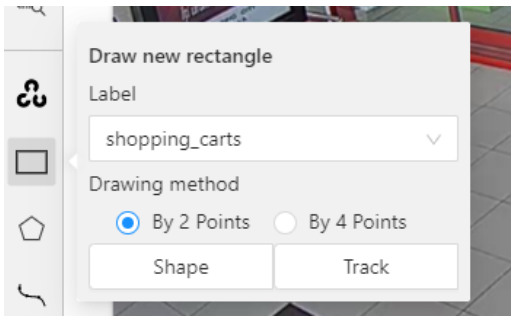


Figura 3.8 Configuración de un nuevo Bounding Box en CVAT .

Dentro de las opciones a la hora de configurar la siguiente anotación a realizar a través de Bounding Box nos encontramos el nombre de la etiqueta, el cual deberá corresponderse con el tipo de carro que vaya a ser anotado, así como la opción de dibujar la "caja" a través de dos o cuatro puntos. Además, podemos comprobar que aparecen dos opciones, las cuales son:

- ***Shape:*** Se utiliza en casos en los que un objeto aparece para ser anotado en un único frame, de manera que una vez llevada a cabo su anotación, cuando avancemos cualquier número de frames su Bounding Box "desaparecerá" estando únicamente presente en en el frame en el que fue utilizado.
- ***Track:*** Sirve para llevar a cabo la interpolación, es decir, cuando un objeto que está en movimiento aparece en varios frames, el Bounding Box ha de seguir a este objeto adaptándose a su nueva posición, orientación e incluso luminosidad en la imagen. Es así que esta opción mantiene la "caja" entre frames, siempre que estos sean posteriores al frame en el que se produjo su aparición, y seremos nosotros los encargados de hacerla desaparecer en el momento en que el objeto quede fuera de la imagen y por tanto no deba ser anotado. En las figuras 3.9, 3.10, 3.11 se puede comprobar de manera visual un ejemplo de interpolación:

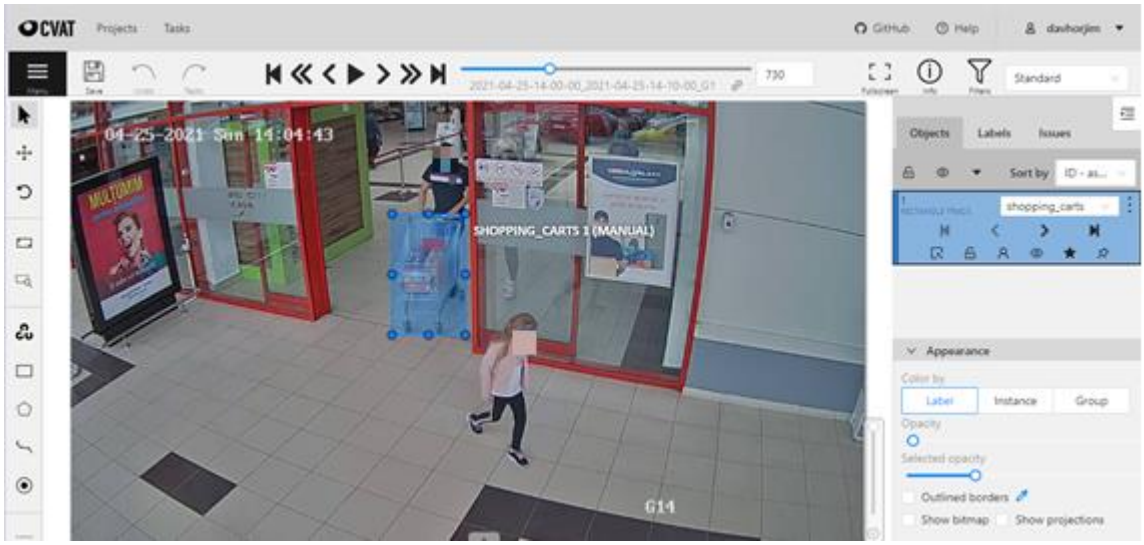


Figura 3.9 Interpolación CVAT 1/3.

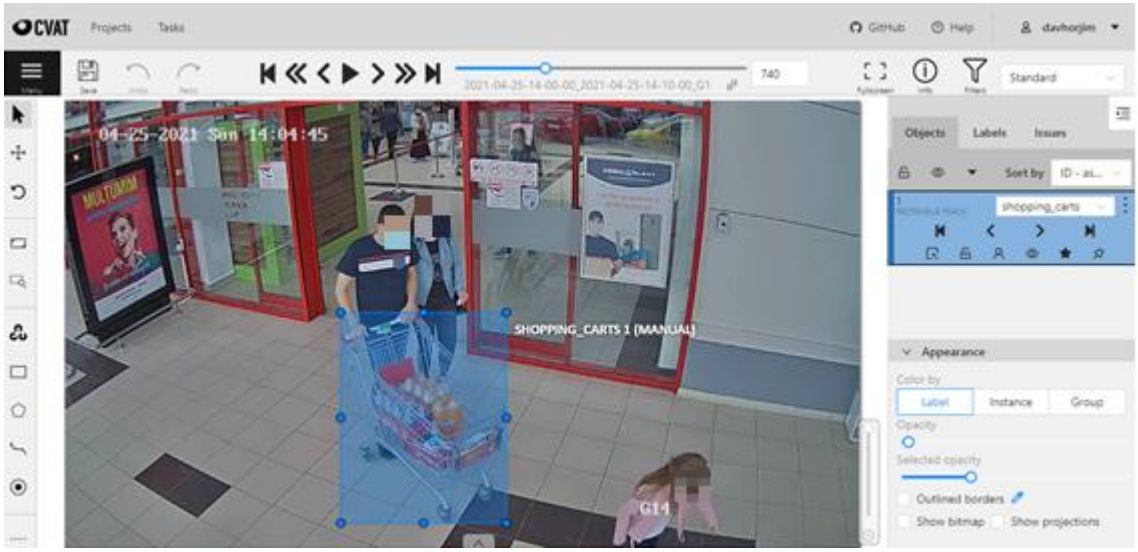


Figura 3.10 Interpolación CVAT 2/3.

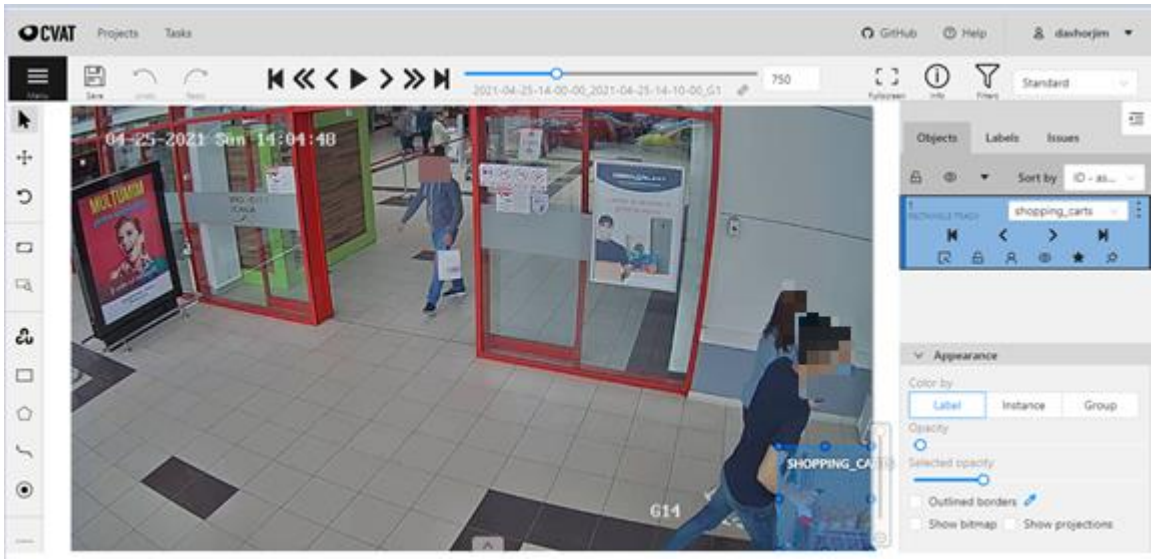


Figura 3.11 Interpolación CVAT 3/3.

Una vez haya desaparecido el objeto de la imagen, para indicárselo al programa debemos de marcar la siguiente opción dentro de la pestaña que aparece en la figura 3.12:

Figura 3.12 Finalizar interpolación en CVAT.

Este es el procedimiento básico para llevar a cabo la anotación. Una vez hayamos acabado de anotar, debemos de extraer los datos. Para ello disponemos de dos opciones con diferentes formatos cada una de ellas:

- ***Dump annotations:*** Sirve para extraer simplemente las anotaciones, el cualquiera de los formatos que nos ofrece la plataforma, entre los que podemos encontrar CVAT for images, CVAT for video, COCO 1.0, YOLO 1.1, etc.
- ***Export as a dataset:*** En este caso, lo que exportamos del programa es un dataset, el cual se compone tanto de anotaciones como de imágenes, de manera que se descargan todos los frames como imágenes individuales mas los ficheros de anotación correspondientes.

Por otro lado, podemos "subir" anotaciones previamente extraídas mediante la opción ***Upload annotations***. (figura 3.13)

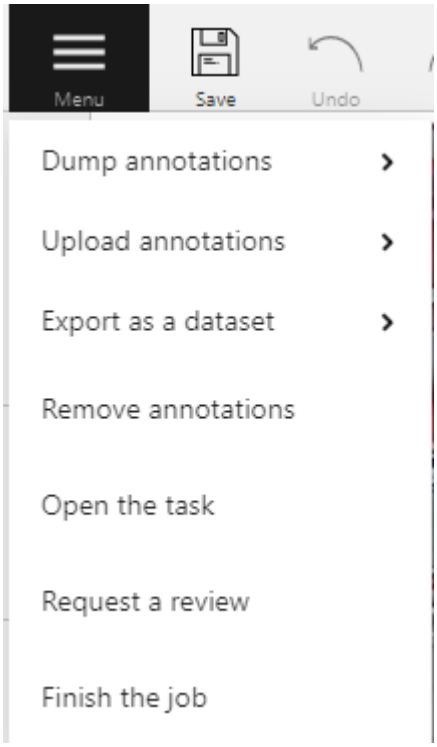


Figura 3.13 Pestaña de CVAT para exportar en importar anotaciones.

Es así como se efectuarán las anotaciones a lo largo de los diferentes videos, los cuales como se indicó anteriormente vendrán grabados por cámaras frontales de entrada y salida, así como por cámaras cenitales de dentro del supermecado. (figuras 3.14, 3.15, 3.16)

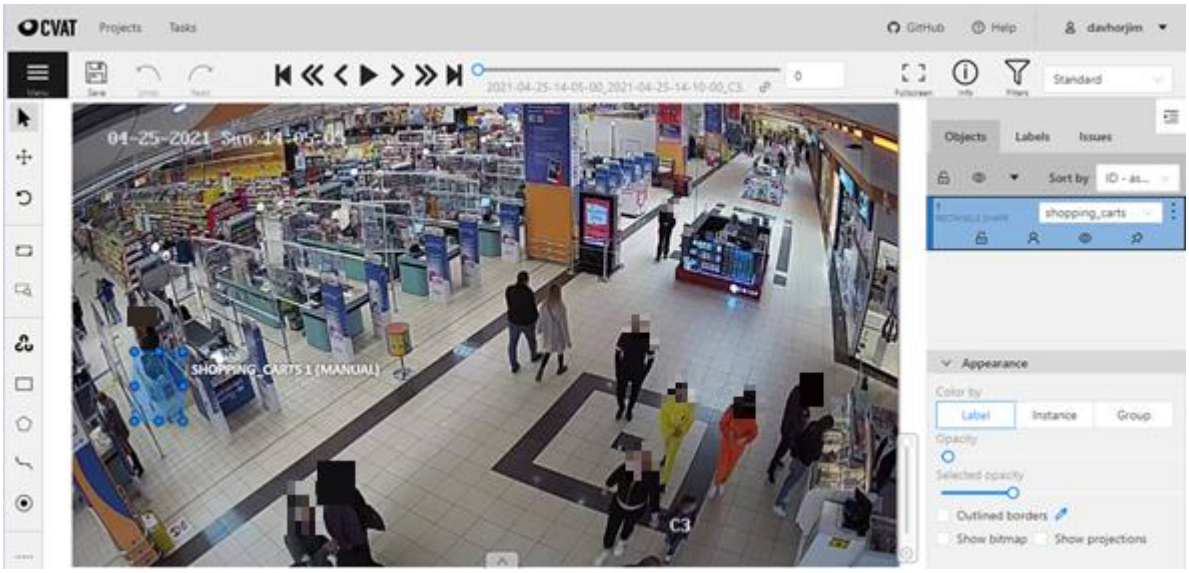


Figura 3.14 Ejemplo imagen cámara cenital.

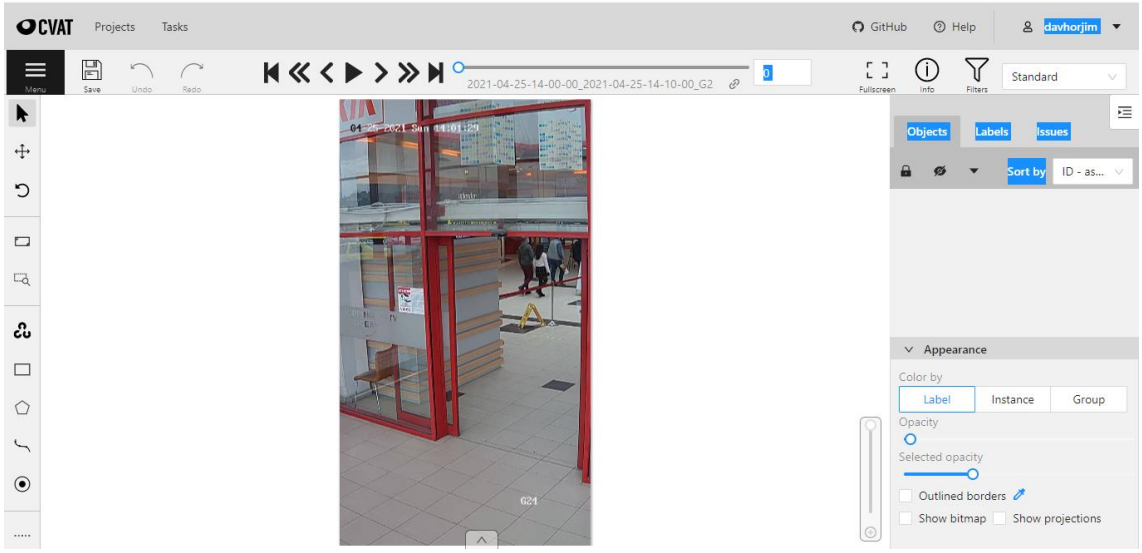


Figura 3.15 Ejemplo imagen cámara frontal de entrada.

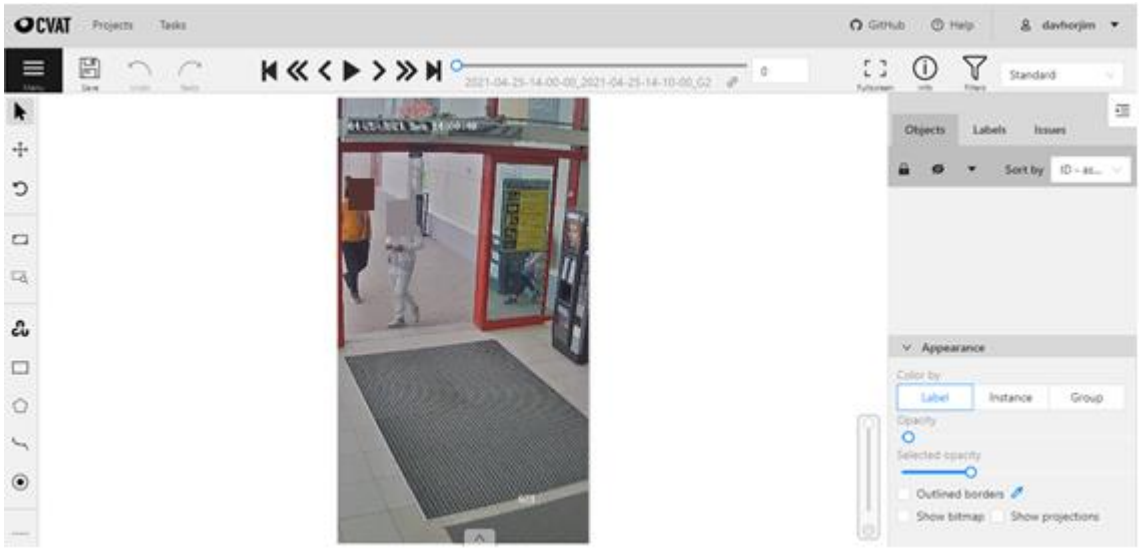


Figura 3.16 Ejemplo imagen cámara frontal de salida.

Estos serían por tanto los diferentes puntos de vista sobre los que se realizarán las anotaciones.

Además, para intentar llevar a cabo un entrenamiento lo más completo posible, no solamente en cuanto a la orientación de los carros en las imágenes si no también en aspectos de luminosidad, se efectuará una anotación de videos que fueron grabados a distintas horas del día.

3.3. Entrenamiento.

Una vez terminada la anotación de los diferentes videos que vayan a ser incluidos en el set de entrenamiento, pasamos a la ejecución del mismo.

Para poder llevar a cabo el entrenamiento, tal y como se comentó anteriormente se hará uso de YOLOv3. Esta red se caracteriza por establecerse a través de lo que se conoce como 'YOLO Pipeline', el cual deberemos "montar" en un entorno de programación adecuado. En concreto, en lo conocido como ***Jupyter Notebooks***.

Jupyter Notebook es una interfaz web de código abierto que permite la inclusión de texto, video, audio e imágenes, así como la ejecución de código a través del navegador en diferentes lenguajes. Esta ejecución se realiza mediante la comunicación con un núcleo (kernel) de cálculo.[7]

Para la ejecución de este tipo de interfaz, existen diferentes herramientas. Hemos de tener en cuenta que, a la hora de realizar el entrenamiento será conveniente el uso de una GPU que permita reducir la carga de trabajo del procesador central.

Se hará uso durante el proyecto de dos herramientas, la primera que será utilizada será ***Google Colab.***

Una vez dentro de la aplicación, se realiza en ella una definición de sí misma(figura 3.17)

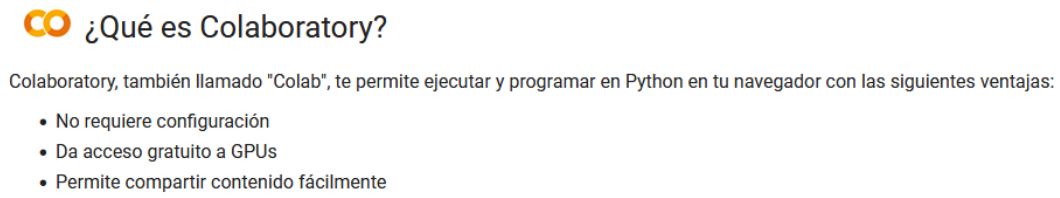


Figura 3.17 Explicación Colab.

Para empezar a trabajar en Colab, debemos crear un nuevo cuaderno. Para ello, disponemos de la pestaña de la figura 3.18.

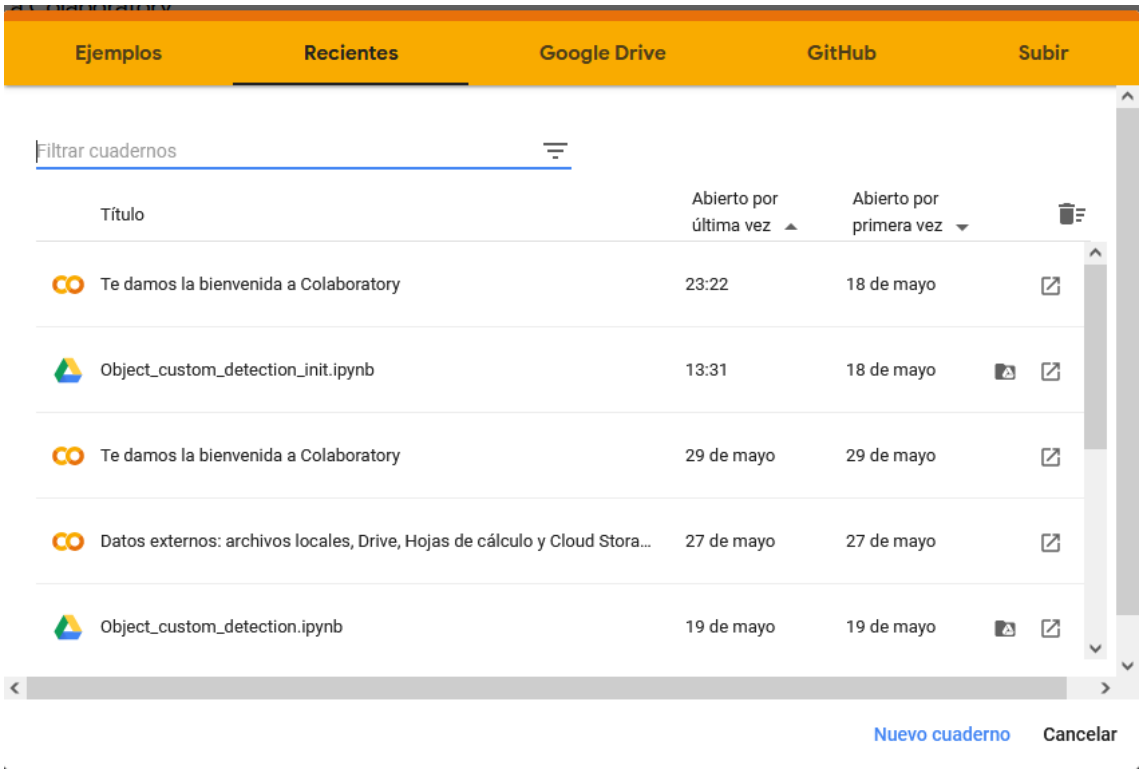
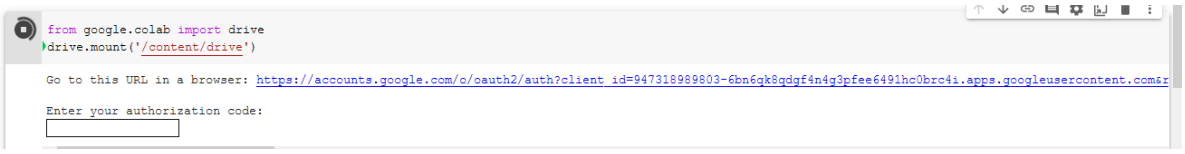


Figura 3.18 Pestaña de inicio en Colab.

Así, haciendo click izquierdo sobre *Nuevo Cuaderno*, crearemos el cuaderno en el que se implementará el Pipeline de YOLOv3, el cual se detallará paso a paso:

- En primer lugar, aprovechándonos de las facilidades de Google para conectar sus propias herramientas, "asociaremos" una cuenta de Drive a Colab, de manera que cualquier archivo generado en Colab en el directorio correspondiente al Drive quedará almacenado de manera permanente en este último. Hay que tener en cuenta que el almacenamiento en Colab es volátil, es decir, una vez desconectados de la GPU actual el contenido generado hasta el momento de la desconexión desaparece del almacenamiento local de Colab, al contrario que en el Drive. Por tanto, la primera acción a realizar será la aparente en el código 3.1.



Código 3.1 "Montar" Google Drive en Google Colab.

Como vemos en el código 3.1, nos aparecerá un enlace al que deberemos acceder para elegir la cuenta de Google cuyo Drive queremos asociar a Colab. Una vez elegida, nos aparecerá un código que debemos copiar y pegar en el cuadro de texto que aparece en blanco. (Figura 3.19, código 3.2)

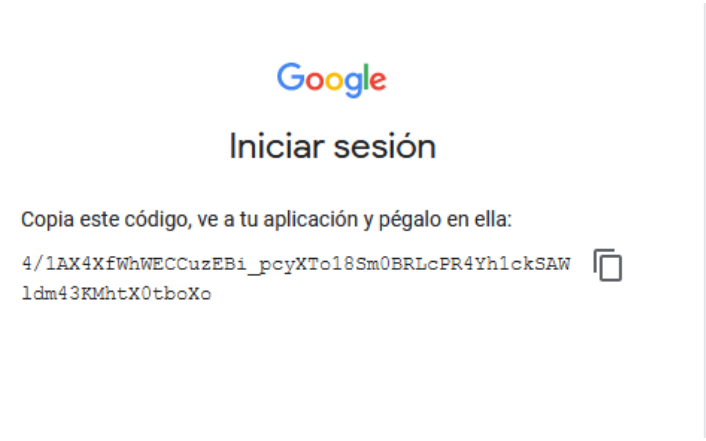


Figura 3.19 Clave de autorización para Drive en Colab.

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

```
Mounted at /content/drive
```

Código 3.2 Drive "montado" en Colab.

Es ahora cuando ya tendríamos asociada la cuenta de Drive deseada con nuestro a cuaderno de Colab.

- Tras asociar ambas herramientas, nos "moveremos" hasta el directorio de drive en el cual dispondremos e una carpeta específica para el contenido de YOLO, creada manualmente. Una vez dentro, clonaremos en el mismo el repositorio de git en el cual se encuentra la arquitectura de la red (código 3.3)

```
[ ] !ls '/content/drive/My Drive/yolo_custom_model_Training'
```

```
!git clone 'https://github.com/AlexeyAB/darknet' '/content/drive/My Drive/yolo_custom_model_Training/darknet'  #Clonamos el repositorio
```

Código 3.3 Directorio de entorno de YOLO y clonar repositorio.

Una vez hayamos clonado el repositorio al completo, en Drive dispondremos de la carpeta correspondiente a la red, con el nombre de darknet. Tras esto, pasaremos a crear un entorno, en Drive, para poder llevar a cabo el entrenamiento al completo. Así, este entorno estará compuesto por 5 carpetas:

- ***Darknet:*** La carpeta indicada anteriormente, contiene todos los archivos o ficheros que componen la red neuronal. Posteriormente se indicarán los ficheros a modificar dentro de la misma.

- ***Obj_train_data:*** Carpeta que dispondrá de los datasets de entrenamiento, con todos los ficheros necesarios para disponer de los archivos que nos permitan realizar el propio entrenamiento, los cuales se indicarán más adelante.

- ***Obj_test_data:*** Idéntica a obj_train_data, con la diferencia de que dispondrá de los datasets que serán utilizados para el test.

- ***Custom_weight:*** En esta carpeta guardaremos un modelo pre-entrenado, necesario para llevar a cabo el entrenamiento.

- ***Backup:*** Carpeta en la cual se guardarán los modelos generados en el entrenamiento.

Tras haber dejado creado el entorno, pasaremos a la modificación de los ficheros necesarios para el correcto funcionamiento, en concreto se modificarán:

- ***Makefile****:* En la carpeta darknet. Ya que en Colab haremos uso de una GPU remota, debemos activar las opciones de GPU Y CUDNN, poniéndolas a '1' tal y como se nos indica en el propio fichero. Por otro lado, debemos activar la opción de OPENCV, la cual es una biblioteca de visión artificial desarrollada por Intel.(código 3.4)

```
GPU=1
CUDNN=1
CUDNN_HALF=0
OPENCV=1
AVX=0
OPENMP=0
LIBSO=0
ZED_CAMERA=0
ZED_CAMERA_v2_8=0

# set GPU=1 and CUDNN=1 to speedup on GPU
# set CUDNN_HALF=1 to further speedup 3 x times (Mixed-precision on Tensor Cores) GPU: Volta, Xavier, Turing and higher
# set AVX=1 and OPENMP=1 to speedup on CPU (if error occurs then set AVX=0)
# set ZED_CAMERA=1 to enable ZED SDK 3.0 and above
# set ZED_CAMERA_v2_8=1 to enable ZED SDK 2.X
```

Código 3.4 Makefile.

- ***creating-files-data-and-name.py*** y ***creating-train-and-test-txt-files.py****:* En estos ficheros escritos en Python, tal y como indica su extensión, realizaremos una modificación muy simple, ya que en ambos documentos se indicará la ruta en la cual se encuentran las imágenes sobre las que este fichero deberá actuar, indicando las carpetas obj_train_data u obj_test_data, dependiendo de si nos encontramos en el proceso de entrenamiento o test, respectivamente.(código 3.5)

```
full_path_to_images = 'obj_train_data'
```

Código 3.5 Ruta donde encontrar imágenes en ficheros .py para el entrenamiento.

- ***Yolov3_custom.cfg***: En la carpeta cfg, dentro de darknet. En este fichero, se "configura" el entrenamiento o el test, ya que se definen el número de batches y subdivisiones para ambos casos, comentando y descomentando la parte correspondiente según el proceso a realizar. Además, se definen otros aspectos importantes como anchura y alturao número de canales, que deberá ser igual al número de clases.
  Por otro lado, en este fichero se definen el número de iteraciones a realizar en el entrenamiento,

siendo el número de pasos el 80% y 90% de las iteracciones totales.

Como última acción a realizar, en las partes correspondientes a YOLO se definirán las máscaras y el número de clases, así como los anchors fijos, entre otras cosas.(código 3.6)

```
[net]
# Testing
batch=1
subdivisions=1
#Training
#batch=64
#subdivisions=16
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 2000
policy=steps
steps=1600,1800
scales=.1,.1
```

```
[yolo]
mask = 0,1,2
anchors = 10,13,  16,30,  33,23,  30,61,  62,45,  59,119,  116,90,  156,198,  373,326
classes=3
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

Código 3.6 Configurar yolov3_custom.

- Tras tener todos los ficheros necesarios modificados, pasaremos a la siguiente parte dentro del proceso. En ella, realizaremos la compilación de la red al completo a través del ejecutable *darknet*, el cual se encuentra dentro de la carpeta con su mismo nombre.(código 3.7)

```
[ ]  %cd /content/drive/My Drive/yolo_custom_model_Training/darknet

[ ]  #NOTA: ANTES DE COMPILAR, VER SI ES ENTRENAMIENTO O TEST PARA CAMBIAR yolov3_custom.cfg

     !make clean

[ ]  !make
```

Código 3.7 Compilación de la red Darknet.

Como podemos apreciar, lo que debemos hacer en primer lugar es “movernos” hacia el directorio de Darknet. Una vez en el mismo, con !make clean lo que haremos será “limpiar” la compilación para asegurarnos de que no quedan residuos de compilaciones anteriores, para posteriormente compilar a través del comando !make. Es importante destacar que antes de realizar la compilación debemos asegurarnos de tener el fichero *yolov3_custom.cfg* programado de acuerdo al proceso a realizar, tal y como se indica en el comentario. Tras terminar la compilación, si ejecutamos el comando del código 3.8:

```
[ ]  !darknet/darknet
```

Código 3.8 Comprobación de correcta compilación.

Nos debe aparecer un mensaje por la consola en la cual se nos indique que darknet es una función. En caso de ser así, la compilación se ha realizado de manera correcta.

- Tras compilar la red, pasaremos a ejecutar los ficheros de Python *creating-files-data-and-name.py* y *creating-train-and-test-txt-files.py.* Para ello, debemos encontrarnos en el directorio del entorno de YOLO, ejecutando el código 3.9.

```
[ ]  !python obj_train_data/creating-files-data-and-name.py # o !python obj_test_data/creating-files-data-and-name.py
!python obj_train_data/creating-train-and-test-txt-files.py # o !python obj_train_data/creating-train-and-test-txt-files.py
```

Código 3.9 Ejecutar ficheros .py en set de imágenes para generar archivos *.txt, .data* y *.names*.

Estos ficheros se encargan de generar los ficheros *train.txt, test.txt, labelled_data.data* y *classes.names.* Los ficheros de texto (.txt) incluyen el nombre de todas las imágenes presentes en la carpeta, necesario para que a la hora de entrenar o testar el programa sea capaz de hallar las mismas.

Por su lado, el fichero .data incluye información importante, como es el número de clases, la ruta hacia *train.txt* o *test.txt*, así como el directorio donde guardar los modelos generados en el entrenamiento y la ruta hacia el fichero .names, el cual indica las clases que aparecen en el proyecto.

- Una vez dispongamos de todo lo anterior, pasaremos a realizar el entrenamiento. Antes de esto, debemos asegurarnos de disponer de un modelo pre-entrenado como se indicó anteriormente. En nuestro caso será darknet53.conv.74.

Así podemos pasar a entrenar. Para ello ejecutamos el código 3.10.

```
!darknet/darknet detector train obj_train_data/labelled_data.data darknet/cfg/yolov3_custom.cfg custom_weight/darknet53.conv.74 -dont_show #train
```

Código 3.10 Comando de entrenamiento a partir de modelo pre-entrenado.

En este comando ejecutamos la red a través de la función Darknet, indicando que se va a realizar un entrenamiento mediante el argumento *train* y haciendo referencia a su vez a los directorios donde se encuentran tanto el archivo .data, cuyo contenido se explicó anteriormente, así como la ruta tanto al fichero *yolov3_custom* como al modelo pre-entrenado del que nos serviremos para entrenar.

Hemos de tener en cuenta que a medida que va avanzando el entrenamiento, se irán guardando los modelos cada 1000 iteraciones, además de guardarse un modelo correspondiente a la última iteración con el nombre yolov3_custom_last.weights, de manera que si queremos continuar un entrenamiento previamente iniciado pero no terminado, lo que haremos será ejecutar el código 3.10 cambiando el modelo pre-entrenado por este generado por el entrenamiento, dando lugar al código 3.11.

```
!darknet/darknet detector train obj_train_data/labelled_data.data darknet/cfg/yolov3_custom.cfg backup/yolov3_custom_last.weights -dont_show #train
```

Código 3.11 Comando de entrenamiento a partir de modelo de entrenamiento previo.

Es aquí cuando nos encontramos con un problema en el uso de Colab, ya que los entrenamientos dependiendo del número de iteraciones pueden tardar más o menos horas en completarse, aunque siempre son de larga duración. El problema radica en que Colab se desconecta de la GPU de forma automática después de llevar un tiempo sin estar realizando ninguna acción sobre el cuaderno, pudiendo perder el progreso si trabajamos fuera del directorio correspondiente a Drive. Además, hemos de tener en cuenta que el espacio de almacenamiento disponible en Drive es limitado, de manera que para entrenamientos de varios videos, cada uno de ellos con miles de frames, es muy probable quedarnos sin espacio. Es así que se buscará otra alternativa.

Tras barajar diferentes alternativas, finalmente se ha decidido utilizar una herramienta llamada ***Gradient Paperspace***. Esta plataforma ofrece, al igual que Colab, la posibilidad de desarrollar Jupyter Notebooks, con la diferencia de que ahora se nos permitirá elegir entre diferentes plantillas (templates) para posteriormente crear una instancia sobre ella, en la cual se pasará a programar en el cuaderno.(figura 3.20)

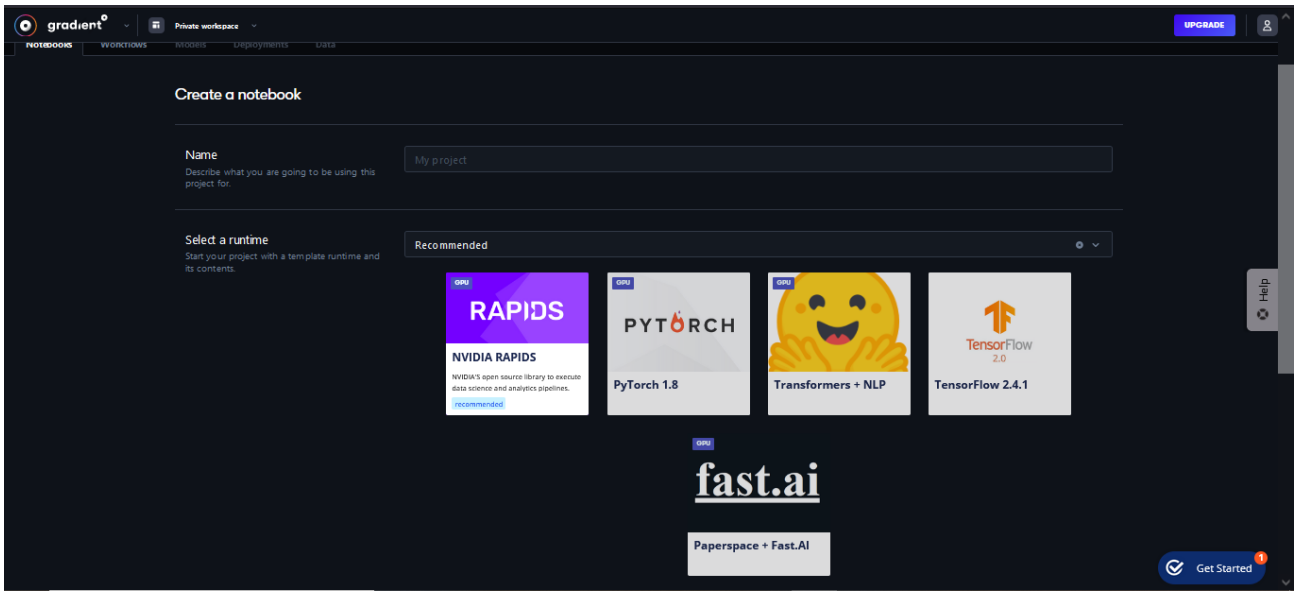


Figura 3.20 Crear cuaderno en Gradient.

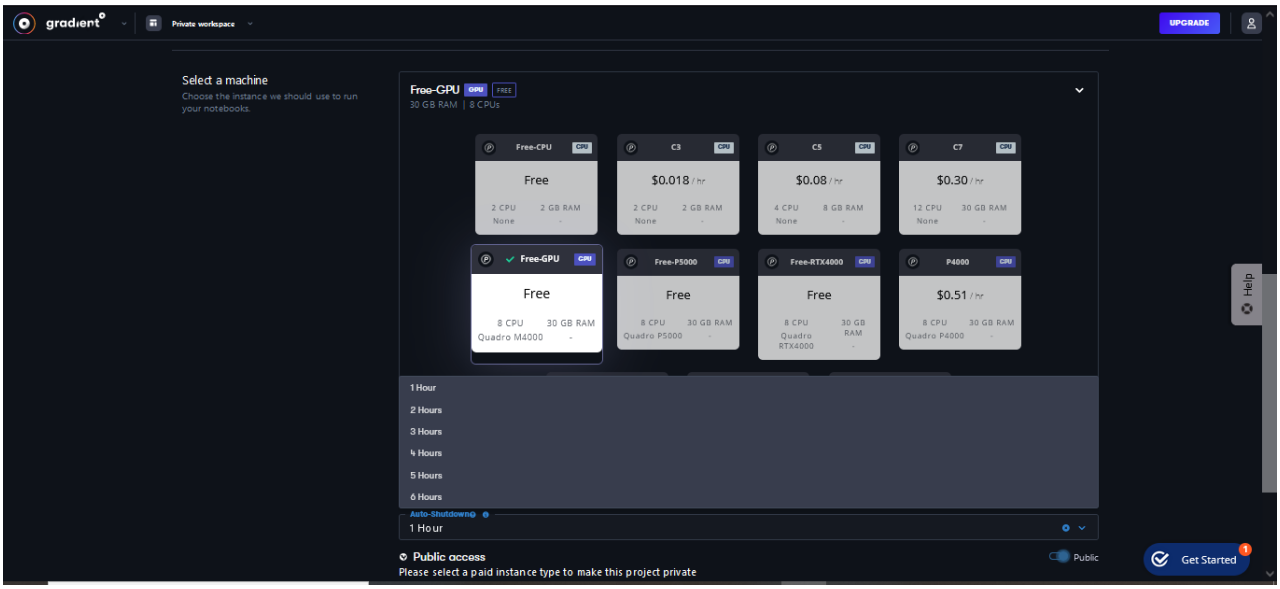


Figura 3.21 GPUs ofrecidas por Gradient.

Además, a diferencia de Colab, se nos permite elegir entre diferentes GPUs (figura 3.21) a las que poder conectarnos, siendo algunas de estas de pago por horas de uso, con tiempo de auto-shutdown en el cual finalizará la conexión. Esta es una de las ventajas que esta herramienta presenta frente a Colab, ya que hasta que este tiempo no haya transcurrido podremos tener la certeza de que permaneceremos conectados a la GPU sin desconexiones aleatorias.

Para nuestro proyecto, se hará uso de **PyTorch 1.8,** haciendo uso a su vez de una **GPU P5000.** Es importante comentar que cada vez que nos conectemos a la instancia podremos hacerlo desde una GPU distinta con un tiempo de auto-shutdown a elegir, de manera que no estaremos limitados a una sola configuración GPU/tiempo.

Una vez creado el cuaderno o Notebook, trataremos de entender de manera general el entorno que se nos ofrece:

- Por un lado, en la interfaz de la herramienta podemos visualizar todo el contenido presente en el directorio del cuaderno. Dado que se dispone de una memoria local, los archivos quedarán almacenados de manera permanente en este directorio. (figura 3.22)

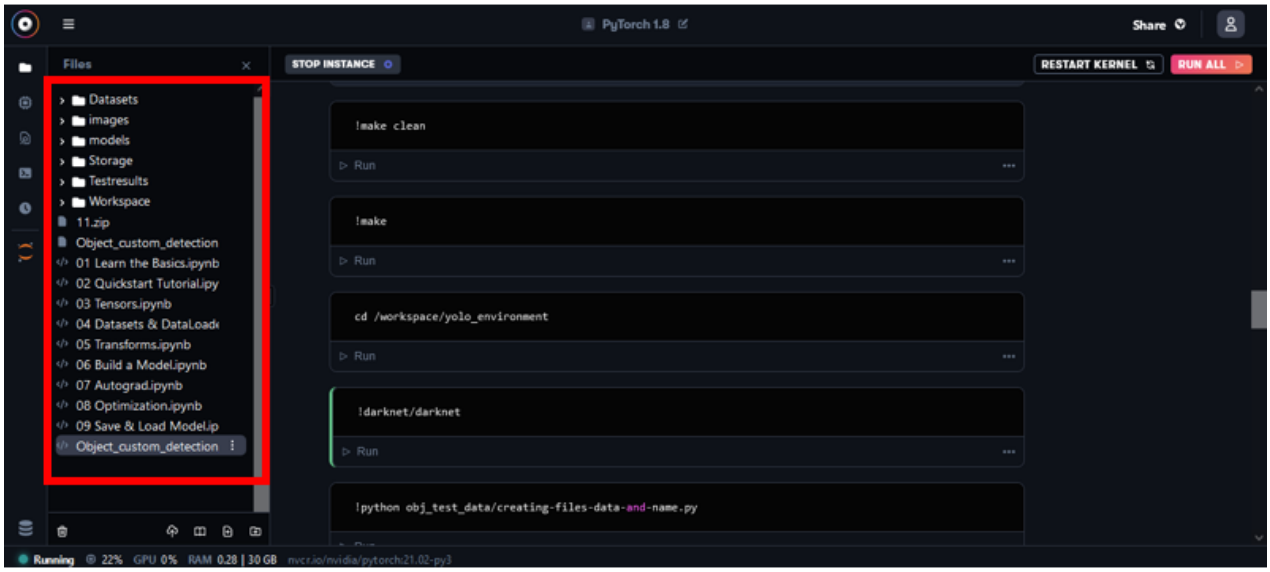


Figura 3.22 Interfaz cuaderno Gradient.

No obstante, si comprobamos mediante el comando !pwd el directorio en el que nos encontramos, vemos que nos aparece en la consola un mensaje con la ruta /notebooks, por lo que podemos intuir que existe un directorio superior y por tanto que la herramienta dispone de un espacio de direcciones dentro del cual se encuentra la del cuaderno. Así, pasaremos a comprobar cual es este espacio de direcciones, para lo que ejecutaremos el comando *cd ..* el cual nos llevará hacia el directorio superior. Una vez en él, haciendo !ls podremos visualizar este espacio de direcciones, en el cual aparecen los directorios que conforman la instancia (figura 3.23)

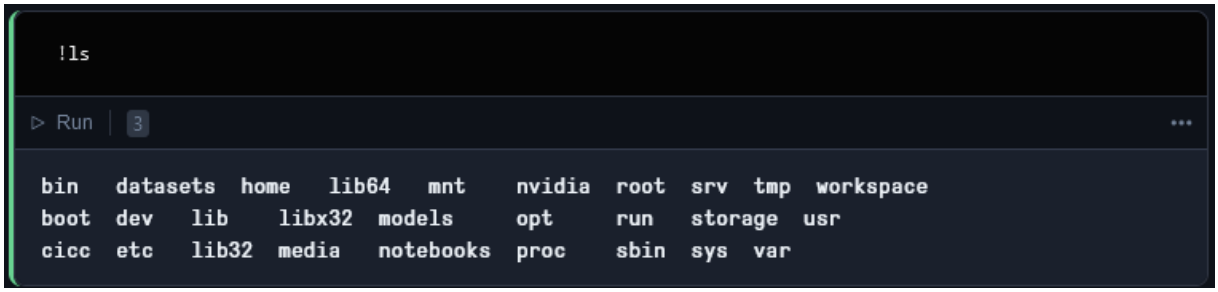


Figura 3.23 Espacio de directorios de Gradient.

Viendo la organización de los directorios, nos aprovecharemos del mismo para llevar a cabo nuestro programa, disponiendo de una mejor organización de los datos que van a componer el mismo. Así, se hará uso el directorio */workspace* para disponer del entorno de YOLO (yolo_environment) dentro del mismo, así como del directorio */media* para almacenar en él los sets de imágenes tanto de entrenamiento como de test que serán utilizados durante el desarrollo del proyecto.

Una vez planteada esta organización, el siguiente paso será "traer" todo el entorno de YOLO desde Drive hasta Gradient. Para ello, haremos uso del comando *!wget*. Este comando nos permitirá importar archivos de Drive, siendo a través de archivos comprimidos la manera más sencilla de hacerlo, por lo que todo aquello que queramos importar deberá ser previamente almacenado como archivo .zip en Drive.

Para poder importar los archivos de Drive, debemos disponer de la id del archivo. Esto se consigue haciendo el archivo público mediante su enlace y extrayendo su id del mismo, la cual dentro del enlace será el visible en la figura 3.24.

Figura 3.24 Identificador de archivo en Drive para poder ser compartido

Así, ejecutando el comando ¡wget, en el cual debemos copiar la id correspondiente del archivo a importar y nombrando al archivo junto con su extensión (nombre que tendrá dentro del entorno de Gradient) seremos capaces de "traernos" cualquier archivo comprimido de Drive. Siempre se almacenarán los archivos importados en el directorio en el que nos encontremos al ejecutar el comando

- Tras haber importado en el programa tanto el entorno de YOLO como el conjunto de sets de imágenes, en primer lugar de entrenamiento, y copiado los mismos en la carpeta correspondiente, seguiremos el mismo proceso que en Colab, es decir, compilaremos la red, ejecutaremos los archivos de Python y comenzaremos a entrenar.

En este proyecto se han realizado, en concreto, 3 entrenamientos diferentes:

- En primer lugar, un entrenamiento con un set de un único video grabado a partir de una cámara frontal, con el objetivo de familiarizarnos con el proceso y determinar la forma de extraer los modelos generados. Para este entrenamiento, se definirán 2000 como el número de iteraciones a ejecutar para completar el mismo. Una vez finalizado, en el directorio correspondiente al entorno de YOLO aparecerá una gráfica de la evolución seguida durante el entrenamiento, indicando el error a medida que avanzan las iteraciones(figura 3.25)

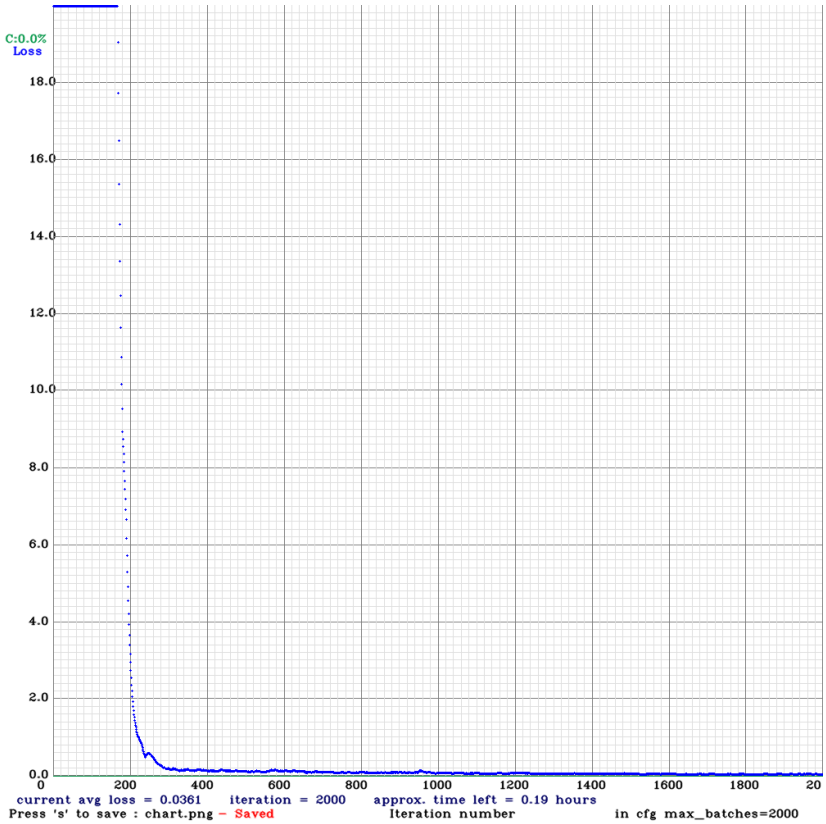


Figura 3.25 Gráfica de entrenamiento con set de 1 video.

Como se puede apreciar, el average loss decae de manera exponencial durante las primeras 400 iteraciones aproximadamente, manteniéndose más o menos constante hasta las 2000 iteraciones entorno al 0.04. No obstante, ni en este caso ni en los siguientes nos basaremos en este valor, simplemente se mencionará, ya que serán otras métricas las que determinene la validez o precisión de nuestros modelos.

Además, se generan 4 modelos con este entrenamiento, en concreto:

- *yolov3_custom_1000.weights*: Modelo a las 1000 iteraciones

- *yolov3_custom_2000.weights*: Modelo para 2000 iteraciones.

- *yolov3_custom_last.weights*:Modelo correspondiente a la última iteracción de entrenamiento, por tanto será igual que *yolov3_custom_2000.weights*

- *yolov3_custom_final.weights*: Modelo de mayor precisión dentro del entrenamiento.

Es importante indicar que para poder extraer cualquier archivo de Gradient, el procedimiento seguido se ha basado en aprovechar la posibilidad que ofrece la plataforma de abrir el cuaderno desde JupyterLab (figura 3.26).

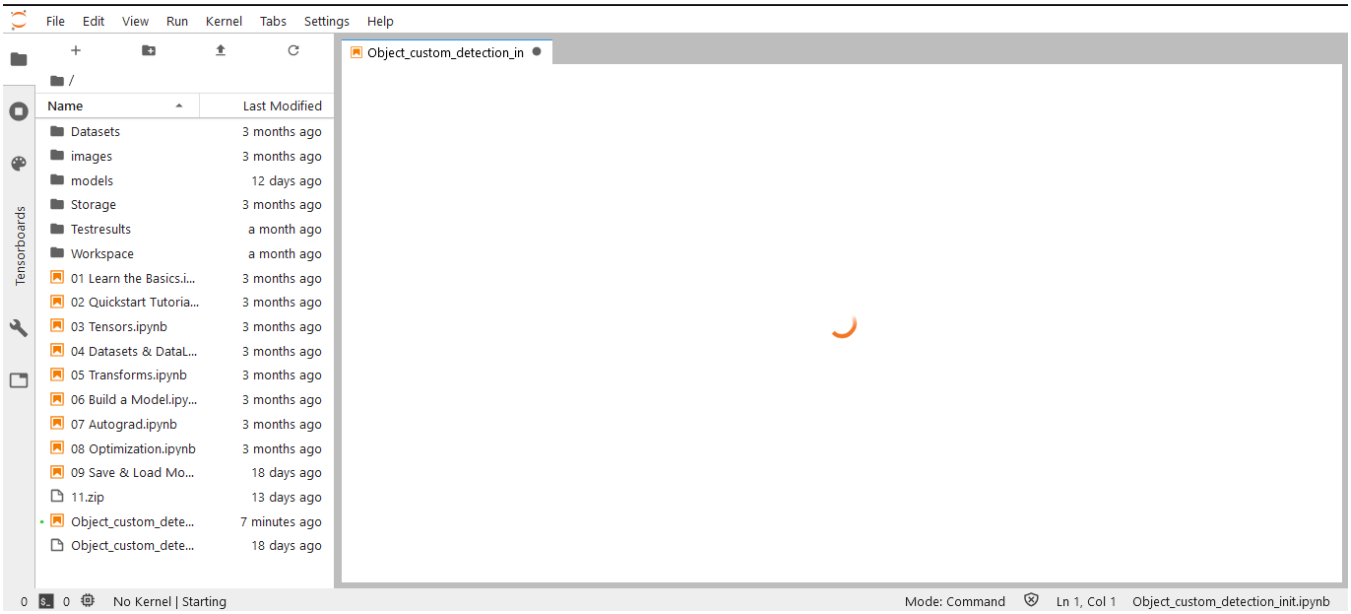


Figura 3.26 JupyterLab Interface.

En esta plataforma solo tendremos acceso al directorio del cuaderno, de manera que lo que haremos será, en Gradient, copiar los archivos necesarios de los directorios de trabajo y pegarlos en uno de los directorios del cuaderno. Es así que podremos descargarlos directamente haciendo click derecho sobre el archivo y click izquierdo sobre download, lo que descargará directamente el archivo.(figura 3.27)

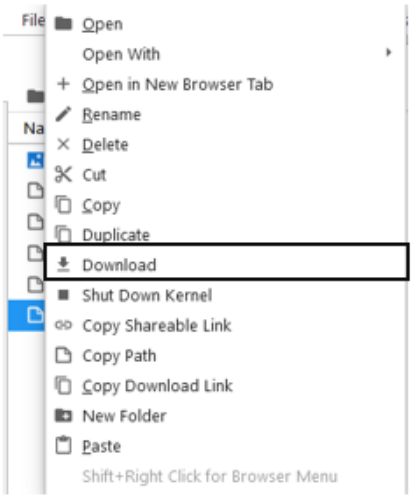


Figura 3.27 Descargar archivo en JupyterLab.

- Tras haber realizado el entrenamiento para el set de 1 video, pasaremos a realizar un entrenamiento para un set de 5 videos, todos de cámaras frontales. Más adelante se analizarán las métricas de cada uno de ellos, pero ahora simplemente nos limitaremos a ver las gráficas y mencionar los modelos generados. Así, una vez terminado el entrenamiento, el cual también se fijará a 2000 iteraciones, obtendremos la gráfica de evolución del error de la figura 3.28:

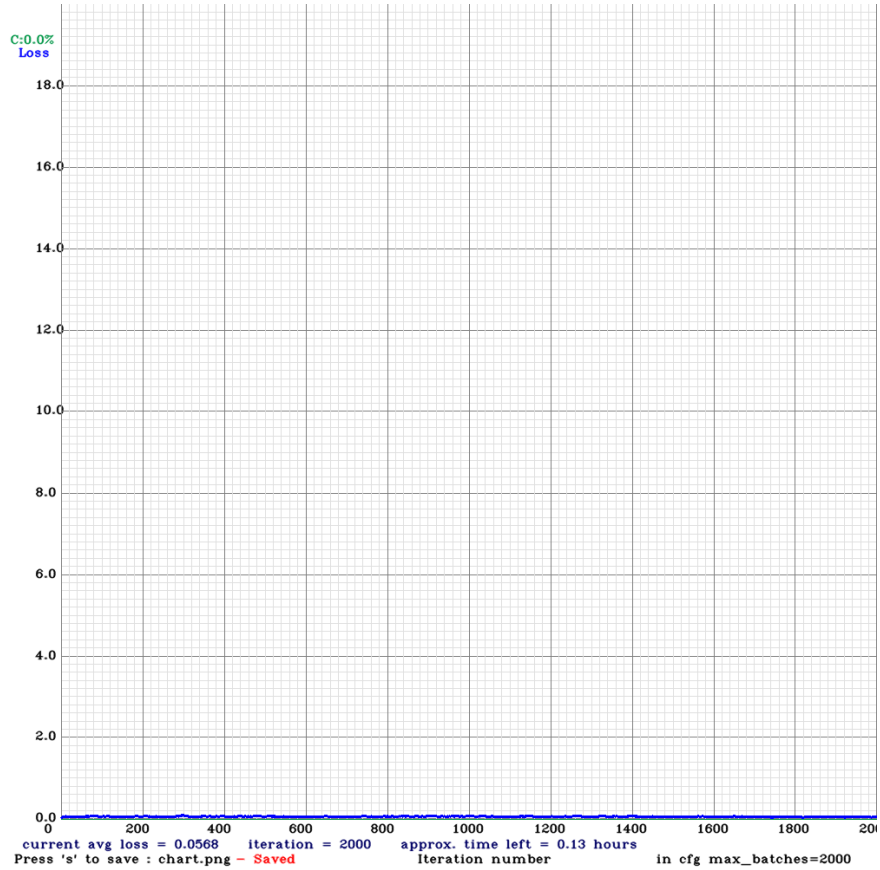


Figura 3.28 Gráfica de entrenamiento con set de 5 videos.

Tal y como se puede apreciar en la gráfica, para este entrenamiento no se produce una caída del average loss, si no que prácticamente se mantiene constante durante todo el entrenamiento, siendo ahora de unos 0.057 aproximadamente.

A su vez, tal y como se produjo en el entrenamiento del set de un único video, se generarán los modelos:

- *yolov3_custom_1000.weights*
- *yolov3_custom_2000.weights*
- *yolov3_custom_final.weights*
- *yolov3_custom_last.weights*

Una vez generadas gráficas y modelos se exportarán descargándolos en JupyterLab, tal y como se hizo anteriormente.

- Tras haber realizado el entrenamiento con este set de 5 videos, pasaremos a realizar el último entrenamiento, en este caso de un set de imágenes procedentes de 8 videos procedentes tanto de cámaras frontales como cenitales. En este entrenamiento, además, aumentaremos el número de iteraciones a 6000.

En este caso la gráfica aparecerá cortada a las últimas iteraciones. Esto se debe a que debido a la larga duración del entrenamiento se realizó en dos conexiones diferentes con la GPU, cumpliendo en una de ellas un tiempo de auto-shutdown de 6 horas, de manera que se sobreescribió únicamente la gráfica de evolución de esta segunda parte. Aún así, seremos capaces de ver el error medio una vez finalizado el entrenamiento, como se muestra en la figura 3.29.

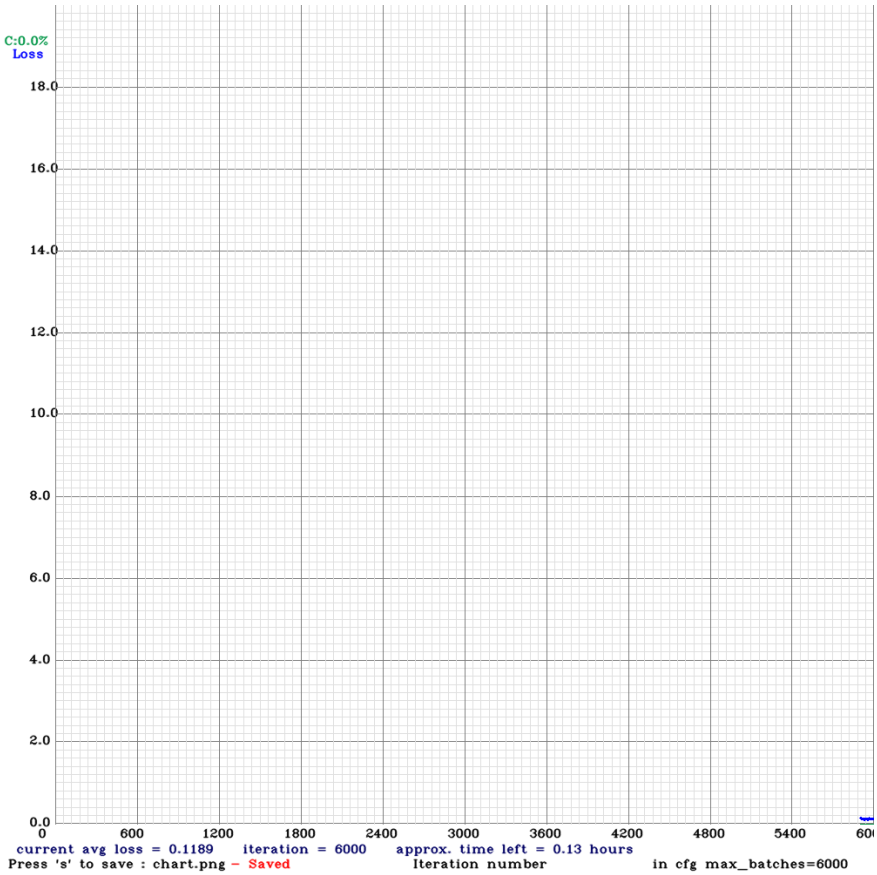


Figura 3.29 Gráfica de entrenamiento con set de 8 videos.

En este caso, como podemos comprobar se dispone de un average loss de un 0.1189. Además, se generan en este caso un mayor número de modelos al disponer a su vez de un mayor número de iteraciones para el entrenamiento, en concreto:

- *yolov3_custom_1000.weights*
- *yolov3_custom_2000.weights*
- *yolov3_custom_3000.weights*
- *yolov3_custom_4000.weights*
- *yolov3_custom_5000.weights*
- *yolov3_custom_6000.weights*
- *yolov3_custom_final.weights*
- *yolov3_custom_last.weights*

Llegados a este punto, podemos dar por concluida la parte correspondiente al entrenamiento, de manera que pasaremos a la tercera y última parte, correspondiente al test.

3.4. Test.

En esta tercera y última parte de nuestro proyecto, nos encargaremos de comprobar la validez de nuestros modelos previamente generados. Para ello, se hará uso de dos métricas distintas:

- ***mAP (mean-Average Precision)*** : Esta métrica sirve para comprobar tanto si la clase es correcta como si la posición del Bounding Box de detección respecto del de anotación es "buena", devolviendo un resultado en tanto por ciento.

- ***IoU(Intersect of Union):*** Métrica que indica el porcentaje de acierto del área de predicción frente al Bounding Box de anotación (real) que se pretende detectar. (figura 3.30)

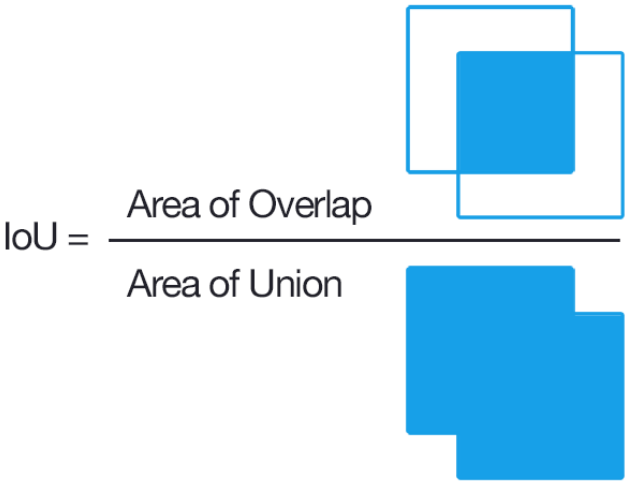


Figura 3.30 Explicación visual del Intersect of Union.

Una vez conocidas ambas métricas, pasaremos a calcularlas a través del propio programa. Es importante tener en cuenta que para hallar tanto el mAP como el IoU haremos uso del set de entrenamiento, mientras que posteriormente se realizará el test sobre el set generado específicamente para ello, compuesto tanto por un vídeo procedente de cámara frontal como uno de cámara cenital.

Es así que una vez generados los distintos modelos de entrenamiento seremos capaces de obtener tanto el IoU como el mAP a través del código 3.12.

```
!darknet/darknet detector map obj_train_data/labelled_data.data darknet/cfg/yolov3_custom.cfg backup/yolov3_custom_final.weights
```

Código 3.12 mAP e IoU.

En él, tal y como se puede apreciar, seguimos la estructura del comando ejecutado para el entrenamiento, solo que esta vez aparecerá el argumento *map* en lugar de *train,* indicando así que es esto lo que queremos hallar al ejecutarlo, mientras que por otro lado el modelo sobre el que queremos hallar las diferentes métricas debe ser, tal y como indica la lógica, uno de los modelos generados en el entrenamiento y no el modelo pre-entrenado, de manera que debemos indicar la ruta y el modelo en concreto sobre el que trabajar.

Se hallarán así por tanto las métricas para los modelos genrerados con 5 y 8 videos, obteniendo los siguientes resultados:

- **Set de 5 videos:**
  - *yolov3_custom_1000.weights*
    - ***mAP = 0.098519 (9.85 %)***
    - ***IoU = 27.63 %***
  - *yolov3_custom_2000.weights = yolov3_custom_final.weights = yolov3_custom_last.weights*
    - ***mAP = 0.295982 (29.60 %)***
    - ***IoU = 49.08 %***
- **Set de 8 videos:**
  - *yolov3_custom_1000.weights*
    - ***mAP = 0.068165 (6.82 %)***
    - ***IoU = 24.12 %***

- *yolov3_custom_2000.weights*
  - ***mAP = 0.401651 (40.17 %)***
  - ***IoU = 62.86%***

- *yolov3_custom_3000.weights*
  - ***mAP = 0.605599 (60.56 %)***
  - ***IoU = 61.05 %***

- *yolov3_custom_4000.weights*
  - ***mAP = 0.560977 (56.10 %)***
  - ***IoU = 52.97 %***

- *yolov3_custom_5000.weights*
  - ***mAP = 0.675226 (67.52 %)***
  - ***IoU = 67.17 %***

- *yolov3_custom_6000.weights = yolov3_custom_final.weights = yolov3_custom_last.weights*
  - ***mAP = 0.677878 (67.79 %)***
  - ***IoU = 65.20 %***

En vista de los resultados, podemos comprobar que tanto en el set de 5 como en el de 8 videos se produce un gran aumento en el valor de las métricas en el paso de 1000 a 2000 iteraciones. Dado que el set de 5 videos no dispone de más modelos aparte de estos 2, no podremos hallar conclusiones a partir del mismo, aunque sí lo podremos hacer de los modelos generados con el set de 8 videos.

De hecho, si observamos en el paso de 2000 a 3000 iteraciones, podemos comprobar que también se produce un aumento significativo en torno a un 20% en la métrica correspondiente al mAP, aunque el IoU se mantiene prácticamente constante al reduirse únicamente en un 1%.

Una de las particularidades de estos resultados se puede observar en el paso de 3000 a 4000 iteraciones, ya que se produce una disminución del mAP que ronda el 5%, así como una decaída del IoU de aproximadamente un 10%. Esto no ha de preocuparnos demasiado, ya que puede ocurrir que al ir iterando un conjunto tan elevado de imágenes se produzcan una serie de aumentos y disminuciones en las métricas. Sin ir más lejos, vemos como ya a las 5000 iteraciones se vuelve a producir una subida que ronda el 10% y 15% para mAP e IoU, respectivamente.

Por último, a las 6000 iteraciones el valor de las métricas es casi idéntico al de 5000, con una simple disminución de un 2% en el IoU.

Tras hallar estas métricas y ver la 'validez' de los diferentes modelos generados, pasaremos a implementar dos de ellos, uno del set de 5 videos y otro del set de 8 videos, sobre el set generado para el test. Así, seremos capaces de comprobar las distintas detecciones realizadas por cada uno de ellos, pudiendo a su vez realizar una comparativa.

Lo primero que debemos hacer, una vez generado el set de imágenes para el test a partir de dos videos, uno de cámara frontal y otro de cámara cenital, es modificar el fichero *yolov3_custom.cfg*. En él únicamente deberemos comentar la parte referente al número de *batch* y *subdivisions* de entrenamiento, descomentando a su vez los valores de los mismos para el testado, que serán 1 en ambos casos.

Tras todo esto, volvemos a compilar la red y pasamos a realizar el test. Para ello ejecutaremos el código 3.13:

```
!darknet/darknet detector test obj_train_data/labelled_data.data darknet/cfg/yolov3_custom.cfg
backup/yolov3_custom_final.weights -dont_show -ext_output < obj_test_data/train.txt > result.txt
```

Código 3.13 Test con *.txt* de salida.

Ambas imágenes conforman el comando completo. Como vemos tiene una estructura similar al de entrenamiento y métricas, con la diferencia de que en este caso se realizará el test, de ahí el argumento *test*. Además, debido a que ha resultado imposible para el testado extraer todas las imágenes con su Bounding Box de detección, se ha adoptado una solución consistente en extraer los resultados en un fichero de texto con el nombre *result.txt*, en el cual aparecerán las detecciones realizadas para todos y cada uno de los frames, con un tanto por ciento de probabilidad de pertenecer a la clase detectada. Presenta por tanto el aspecto de la figura 3.31:

```
obj_test_data/frame_001133.PNG: Predicted in 18.322000 milli-seconds.
shopping_carts: 28%     (left_x:  168   top_y:  214   width:   25   height:   38)
Enter Image Path:  Detection layer: 82 - type = 28
 Detection layer: 94 - type = 28
 Detection layer: 106 - type = 28
obj_test_data/frame_000216.PNG: Predicted in 18.354000 milli-seconds.
shopping_carts: 95%     (left_x:   96   top_y:  386   width:   73   height:  110)
shopping_carts: 57%     (left_x:  773   top_y:  245   width:   78   height:   81)
Enter Image Path:  Detection layer: 82 - type = 28
 Detection layer: 94 - type = 28
 Detection layer: 106 - type = 28
obj_test_data/frame_000297.PNG: Predicted in 18.391000 milli-seconds.
others: 28%     (left_x:   26   top_y:  247   width:   18   height:   26)
shopping_carts: 27%     (left_x:   94   top_y:  384   width:   50   height:  104)
shopping_carts: 40%     (left_x:  786   top_y:  243   width:   86   height:   80)
Enter Image Path:  Detection layer: 82 - type = 28
 Detection layer: 94 - type = 28
 Detection layer: 106 - type = 28
```

Figura 3.31 Ejemplo de resultados del test.

Esta imagen está extraída del *.txt* correspondiente al test del set de 8 videos. Como vemos, se indica para cada una de las imágenes el tiempo de detección, la clase detectada y la colocación del Bounding Box de detección dentro de la imagen a través de sus coordenadas (x,y) correspondientes al vértice superior izquierdo del cuadro, además del ancho y alto de cada uno de ellos.

Dado que el test sobre el set creado específicamente para ello se implementará para ambos modelos, a través de estos *.txt* de resultados podemos comprobar si realmente hay diferencias entre el uso de un modelo y de otro.

Si observamos la detección realizada sobre uno de los frames del video del testset grabado a partir de una cámara cenital (recordar que el set de entrenamiento de 8 videos incluía videos con este tipo de cámara, mientras que el de 5 no) podemos comprobar lo siguiente:

- **Modelo de 8 videos:**

```
obj_test_data/frame_000263.PNG: Predicted in 18.329000 milli-seconds.
shopping_carts: 37%     (left_x:  100   top_y:  392   width:   50   height:   87)
shopping_carts: 51%     (left_x:  731   top_y:  238   width:   72   height:   89)
shopping_carts: 62%     (left_x:  876   top_y:  390   width:   73   height:  128)
Enter Image Path:  Detection layer: 82 - type = 28
 Detection layer: 94 - type = 28
 Detection layer: 106 - type = 28
```

Figura 3.32 Ejemplo 1 resultado test modelo de 8 videos.

Como vemos en la figura 3.32, se han detectado 3 carros pertenecientes a la clase *shopping_carts* con probabilidades de 37%, 51% y 62% de pertenencia a esta clase.

- **Modelo de 5 videos:**

```
obj_test_data/frame_000263.PNG: Predicted in 18.290000 milli-seconds.
Enter Image Path:  Detection layer: 82 - type = 28
 Detection layer: 94 - type = 28
 Detection layer: 106 - type = 28
```

Figura 3.33 Ejemplo 1 resultado test modelo de 5 videos.

Por su lado, para el mismo frame, en la figura 3.33 vemos que el modelo de 5 videos no ha detectado carros de ninguna clase.

Si observamos ahora un ejemplo de un frame perteneciente a un video procedente de una cámara frontal, el cual también fue incluido en el testset, podemos comprobar lo siguiente:

- **Modelo de 8 videos:**

```
obj_test_data/_test_4736.PNG: Predicted in 18.373000 milli-seconds.
shopping_carts: 48% (left_x:   27   top_y:  618   width:  418   height:  466)
Enter Image Path:  Detection layer: 82 - type = 28
 Detection layer: 94 - type = 28
 Detection layer: 106 - type = 28
```

Figura 3.34 Ejemplo 2 resultado test modelo de 8 videos.

Con el modelo correspondiente al set de 8 videos (figura 3.34) se produce la detección de un carro perteneciente a la clase *shopping_carts* con una probabilidad de pertenencia de 48% como se puede apreciar en la imagen.

- **Modelo de 5 videos:**

```
obj_test_data/_test_4736.PNG: Predicted in 18.367000 milli-seconds.
Enter Image Path:  Detection layer: 82 - type = 28
 Detection layer: 94 - type = 28
 Detection layer: 106 - type = 28
```

Figura 3.35 Ejemplo 2 resultado test modelo de 8 videos.

Por su lado, con el modelo de 5 videos (figura 3.35) no se detecta nada para un mismo frame.

No obstante, nos gustaría poder apreciar estas diferencias en la detección también de forma visual. Tal y como se comentó anteriormente, no se ha conseguido extraer todas las imágenes con su Bounding Box de detección mediante el uso de un solo comando. Aunque, por otro lado, si existe la posibilidad de extraer estas imágenes frame a frame, de manera que observando los resultados de los *.txt* de salida generados en los test, podremos extraer un conjunto reducido de frames en el que se observen diferencias entre los modelos y visualizar las detecciones de ambos. El comando es el que aparece en el código 3.14:

```
!darknet/darknet detector test obj_train_data/labelled_data.data darknet/cfg/yolov3_custom.cfg
backup/yolov3_custom_last.weights obj_test_data/_test_4736.PNG -dont_show
```

Código 3.14 Test realizado sobre un frame, con imagen de resultado de predicción.

En conjunto, ambas líneas forman el comando con el que podemos extraer esta imagen de detección. Como se puede observar, el comando es similar al del test en el que extraíamos el *.txt* con los resultados, solo que en este caso aparece un argumento que indica el frame sobre el que queremos realizar la detección. Una vez ejecutado, se generará una imagen con el nombre de *predictions.jpg*, la cual podremos guardar en el espacio del cuaderno y descargar a través de JupyterLab.

A continuación, se mostrarán varios ejemplos (figuras 3.36, 3.37, 3.38, 3.39, 3.40, 3.41, 3.42, 3.43).

- **Modelo de 5 videos:**

Figura 3.36 Detección cámara cenital con modelo de 5 videos.

Para este frame de la figura 3.36 con el modelo de 5 videos, se produce la detección de un único carro. Además, vemos que el Bounding Box de detección no se corresponde con la superficie real del carro.

Figura 3.37 Carro con Bounding Box de detección con modelo de 5 videos.

Esto es lo que se conoce con un **verdadero positivo (true positive),** ya que se produce la detección de un carro como un carro.(Figura 3.37)

Por otro lado, si nos fijamos en la imagen completa, en la parte inferior izquierdo aparece lo que se conoce como un **falso negativo(false negative)** ya que no se produce la detección de un carro como un carro.

- **Modelo de 8 videos:**

Figura 3.38 Detección cámara cenital con modelo de 8 videos.

Como parece lógico, con el modelo de 8 videos (figura 3.38) se ha llevado a cabo una detección más ajustada a

lo que realmente buscábamos, ya que este modelo incluye videos con este tipo de cámara.Es así que se han detectado dos carros (**dos verdaderos positivos**), con unos Bounding Boxes de detección muy parecidos a lo que sería el Bounding Box de anotación (figura 3.39):

Figura 3.39 Carros con Bounding Boxes de detección con modelo de 8 videos.

Vamos a visualizar ahora un ejemplo procedente de uno de los frames del video del testset grabado con la cámara frontal:

- **Modelo de 5 videos:**

Figura 3.40 Detección cámara frontal con modelo de 5 videos.

Como se puede apreciar en la figura 3.40, en este caso aparece un **falso negativo**, ya que no se produce la detección del carro que aparece en la imagen.

- **Modelo de 8 videos:**

Figura 3.41 Detección cámara frontal con modelo de 8 videos.

En este caso (figura 3.41), por el contrario, si que se produce la detección del carro (**verdadero positivo**). No obstante, el Bounding Box de detección no se ajusta de una manera muy precisa al que sería de anotación, de manera que la detección no dispondrá de una probabilidad de pertenencia muy elevada a esta clase de carros.

Por último, se procederá a visualizar uno de los posibles casos muy recurrentes en los problemas de detección, el cual corresponde con los **falsos positivos (false positive).** Esto se produce cuando se realiza una detección errónea, como en las figuras 3.42 y 3.43.

Figura 3.42 Falso positivo 1.

Esta detección (figura 3.42) ha sido realizada por el modelo de 5 videos, y tal y como se puede apreciar se ha producido una detección errónea al considerar una región de la imagen como un carro, cuando en realidad no lo es.

Figura 3.43 Falso positivo 2.

En este ejemplo (figura 3.43), procedente de una detección realizada con el modelo de 8 videos, aparece otro falso positivo, dado que se considera como un carro una región de la imagen que en realidad no lo es.

4 CONCLUSIÓN Y MEJORAS FUTURAS

5.1 Conclusiones.

Este trabajo ha sido el inicio de un largo proyecto que será desarrollado por la empresa. Así, aun no llegando a haber generado un modelo lo suficientemente fiable como para poder empezar a pensar en una posible comercialización, si que se pueden apreciar ciertos indicios que marcan un desarrollo importante a medida que se han ido generando los distintos modelos. Además, el objetivo de la empresa a corto plazo es simplemente la detección de carros, en una especie de detección binaria (si hay carro/ no hay carro) lo cual es algo que con los modelos actuales, especialmente el de 8 videos, empieza a realizarse con un gran número de aciertos. Esto invita a pensar que con un modelo más completo en cuanto a número de videos se podría llegar a disponer de un modelo bastante apto en cuanto a nuestras pretensiones.

5.2 Mejoras futuras.

Aún quedan una serie de aspectos a mejorar como podrían ser:

- Conseguir un set de entrenamiento con un mayor número de videos con el que logremos obtener un modelo que logre obtener un valor cercano al 100% en las métricas de validez, especialmente en el mAP, reduciendo así el número de falsos positivos y falsos negativos.

- Lograr encontrar una plataforma que nos permita desarrollar un entrenamiento y testado sin tantas limitaciones como las que aparecen en Gradient y Colab. Ya que en Colab el tiempo de conexión a la GPU es reducido pero se producen desconexiones arbitrarias, mientras que en Gradient se dispone de un rango de tiempo en el que evitamos la desconexión de forma casi segura, pero por el contrario tarda un tiempo excesivo en preparar la instancia y conectar con la GPU. Algunas alternativas podrían ser las plataformas *Cloud* de Facebook o Amazon, entre otras.

- Formar un set de test más completo, añadiendo más videos procedentes de diferentes cámaras de manera que podamos determinar si nuestros modelos son realmente válidos viendo un mayor número de casos.

- Implementación de un modelo para la detección en tiempo real, ya que es como realmente se le dará uso al proyecto en caso de lograr comercializarse.

- Probar YOLOv4 y comparar con YOLOv3.

REFERENCIAS

[1] Exploring_Deep_Learning-Based_Architecture_Strategies_Applications_and_Current_Trends_in_Generic_Object_Detection_A_Comprehensive_Review.pdf

[2] https://keymakr.com/blog/what-are-bounding-boxes/

[3] You_Only_Look_Once_Unified_Real-Time_Object_Detection.pdf

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

[5] https://openvinotoolkit.github.io/cvat/docs/manual/basics/creating_an_annotation_task/

[6] https://eprints.ucm.es/id/eprint/48304/1/ManualJupyter.pdf

[7] https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/