E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Estimación de la densidad del tráfico mediante técnicas de Deep Learning

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor: Alejandro Lafuente Cacho
Directores: Mikel Galar Idoate y Mikel Sesma Sara
Pamplona, 30/01/2020

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

RESUMEN

En el presente documento se va a detallar todo el proceso de realización del Trabajo de Fin de Grado (TFG), el cual se ha realizado con el fin de buscar una manera efectiva y autónoma para realizar el conteo de vehículos por medio de sistemas dotados con Inteligencia Artificial.

Partimos del problema de realizar conteos de vehículos en un determinado lugar, como podría ser en la entrada de un parking o en un punto kilométrico de una autovía, con el fin de estimar un flujo medio de circulación por la misma.

De este tipo de situaciones, nace la necesidad de desarrollar un sistema que nos permita realizar esta tarea de forma autónoma, de manera que se realice de forma más rápida y eficaz.

El desarrollo del sistema se ha realizado usando un modelo de detección de objetos basado en técnicas de Deep Learning junto con un algoritmo de tracking, lo que nos ha permitido obtener un sistema capaz de realizar un conteo de vehículos para un tiempo y un lugar determinado.

PALABRAS CLAVE: Inteligencia Artificial; Deep Learning; Detección de Objetos; YOLO; Tracking.

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICAS

1. INTRODUCCIÓN

1.1 Descripción general

La evolución de la tecnología con el fin de mejorar nuestras vidas es algo cada vez más palpable, y la informática como parte importante de nuestro día a día es algo que también se ve afectada notablemente por esta evolución.

Dentro de este desarrollo tecnológico, la Inteligencia Artificial como parte de la informática ha recibido un gran impulso durante los últimos años. Conocemos como Inteligencia Artificial aquel campo de la informática el cual se basa en el desarrollo de programas y mecanismos que permitan a las máquinas realizar acciones que consideramos inteligentes, es decir, que de alguna manera puedan realizar tareas que son llevabas a cabo por humanos [1].

Una rama importante de la Inteligencia Artificial en la cual nos basamos en este trabajo, es el Machine Learning (Aprendizaje Automático) [2]. Esta rama tiene como objetivo desarrollar algoritmos que permitan a las computadoras aprender de forma autónoma, es decir, son técnicas las cuales se basan en la asimilación de representación de datos con el fin de que el sistema aprenda a tomar decisiones por sí solo.

Centrándonos más en Machine Learning, en este trabajo de fin de grado hacemos uso de un conjunto concreto de algoritmos de esta rama, a los cuales se les conoce con el nombre de Deep Learning (Aprendizaje Profundo) [3]. Este conjunto de algoritmos tiene la peculiaridad de que usan una arquitectura en forma de red neuronal, es decir están formados por neuronas distribuidas por un determinado número de niveles o capas jerárquicas.

El Deep Learning tiene una gran variedad de usos, pero uno de los campos en donde más se utiliza es en la clasificación de imágenes y visión por computador. En este tipo de algoritmos, cada neurona funciona como una unidad de procesamiento encargada de una tarea específica. Por ejemplo, cuando nos encontramos en un nivel inicial de la jerarquía, la red aprende algo simple de las imágenes (colores, bordes…) y a continuación envía hacia los siguientes niveles o capas de la red. En las siguientes capas se combina la información recibida de capas anteriores y aprende a discriminar entre elementos más complejos de las imágenes. Finalmente, en capas más profundas de la red, esta es capaz de diferenciar objetos complejos, como podrían ser personas, vehículos, animales…

Aunque el Machine Learning y Deep Learning se pueden ver como términos muy similares, su principal diferencia es que la extracción de características en muchos algoritmos convencionales de Machine Learning debe ser un proceso previo realizado por humanos, mientras que en Deep Learning este paso se realiza de forma automática por medio de la red neuronal (Fig. 1) [4].

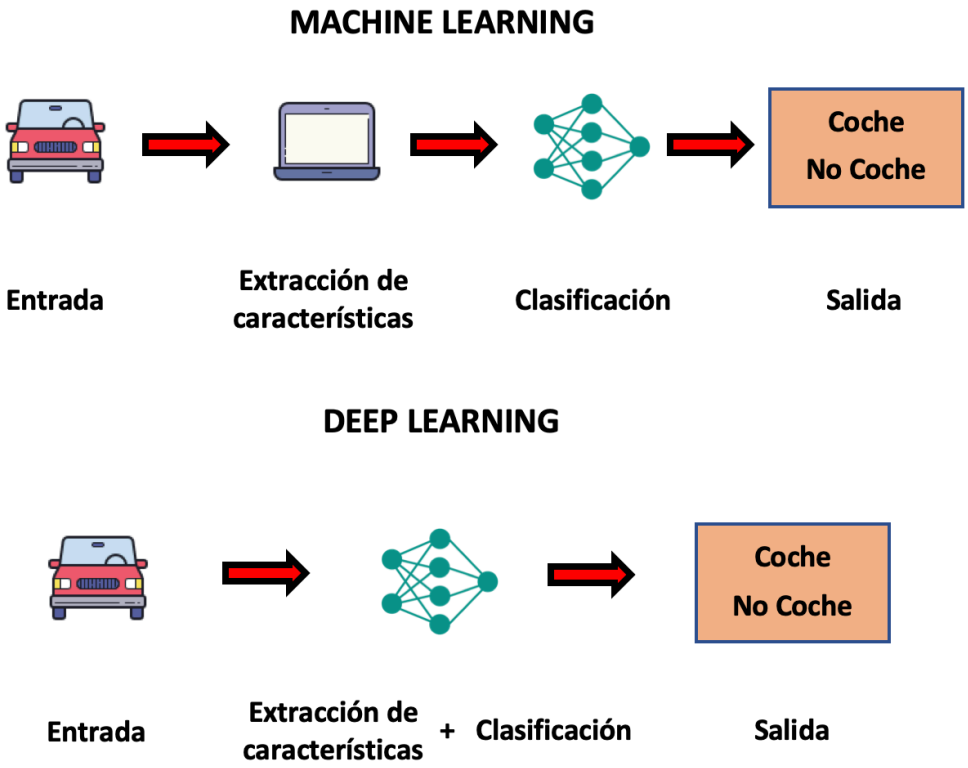


Figura 1. Machine Learning vs Deep Learning

Este tipo de técnicas tienen un gran abanico de aplicaciones, como podrían ser el área de diagnósticos médicos, mercados financieros, asistentes virtuales, detección de objetos, reconocimiento inteligente de defectos… [5].

En este trabajo nos vamos a centrar en la detección de objetos, más explícitamente en la detección de vehículos en situaciones de tráfico, con el objetivo de estimar la densidad de tráfico en un punto determinado. Este tipo de detección ha cogido un gran impulso durante los últimos años debido al auge del coche autónomo, en el cual hay muchos sistemas incluidos que utilizan este tipo de técnicas, como podrían ser la detección de peatones, la detección de señales de tráfico, detectores de líneas de la calzada…

1.2 Motivación y Justificación

Cada día que pasa nos encontramos con una sociedad más avanzada, en la que la tecnología ya es parte fundamental de nuestras vidas. Tanto en ámbito personal como empresarial, el uso de herramientas informáticas nos permite ser más productivos y rápidos a la hora de realizar nuestros cometidos.

Es una realidad que avanzamos hacia una industria 4.0, en la que la obra de mano humana está siendo paulatinamente reemplazada por máquinas o sistemas capaces de acometer esas tareas repetitivas de una forma más eficaz y eficiente.

Todos nos podemos imaginar lo tedioso, e incluso en algunos casos imposible, que puede llegar a ser el realizar un conteo de vehículos que pasan por un determinado punto. Es por ello que surge la necesidad de crear un sistema que sea capaz de detectar vehículos de forma autónoma y llevar un conteo de los mismos, de forma que nos ahorre tiempo y dinero a la hora de realizar una tarea tan repetitiva.

La realización de un sistema de estas características se puede llevar a cabo mediante el uso de técnicas de Deep Learning. Para ello, debemos realizar un análisis de todos los tipos de técnicas que lo componen y quedarnos con aquella que nos permita obtener unos mejores resultados. Debemos tener en cuenta el ámbito hacia el cual va destinado su uso, en nuestro caso, nos centraremos en un uso destinado al procesamiento de imágenes y detección de vehículos, por lo que las redes neuronales convolucionales (CNN) nos parecen la mejor opción [6].

El uso de un dataset con las propiedades idóneas para el entrenamiento de nuestro modelo nos puede permitir aumentar de manera significativa el rendimiento del clasificador. Para ello, nos debemos centrar en algunas características de los mismos como pueden ser el tamaño, situaciones meteorológicas de las imágenes, variedad de tipos de escenas, resolución de las imágenes, tipo de etiquetado…

Existen modelos implementados con un alto porcentaje de precisión, por lo que podemos tomar como base un detector de objetos más general y adaptarlo de forma que al final del proyecto sea capaz de detectar vehículos con una precisión aceptable.

Como algoritmo de detección, uno de los principales es YOLO (You Only Look Once) [7]. Para su puesta en marcha debemos de realizar algunos cambios en su implementación, ya que este modelo se basa en una detección de objetos más generalista, mientras que nosotros lo queremos especializar en vehículos.

Para validar el sistema podremos emplear pruebas de testeo con imágenes que formen parte del dataset pero que no hayan sido empleadas durante la fase de entrenamiento, de manera que se obtenga un valor de precisión en cuanto a la detección de vehículos en imágenes nuevas para el modelo. Además, nuestro objetivo es utilizar algunas imágenes que no formen parte del dataset, de manera que se pueda ver su desempeño con otro tipo de imágenes.

Además, en este proceso de automatización de las tareas, en nuestro caso de conteo de vehículos, la utilización de algoritmos de tracking nos permite realizar un seguimiento de cada uno de los vehículos detectados de manera automática, haciendo efectiva la estimación de la densidad del tráfico.

El uso de técnicas de Deep Learning es uno de los temas con más relevancia y presencia en la actualidad. La aplicación de este tipo de técnicas es todavía novedosa y en la mayoría de los casos son muchos los problemas que surgen durante su implementación, por lo tanto, la realización de este trabajo de fin de grado puede aportar algo de conocimiento respecto a la resolución de problemas surgidos y al desarrollo de estas.

1.3 Objetivos

El objetivo principal de este trabajo es:

- Diseñar un sistema capaz de realizar una estimación de tráfico de vehículos en un punto determinado, haciendo uso de técnicas de Deep Learning.

Para llegar a alcanzar nuestro objetivo principal debemos realizar una serie de pasos, los cuales se convierten en nuestros objetivos específicos y los cuales detallaremos a continuación:

- Estudiar los diferentes tipos de técnicas de Deep Learning con el fin de seleccionar la que mejor se adapte a nuestras necesidades del proyecto.
- Realizar un análisis de los dataset en función de sus características, con el fin de utilizar el óptimo para el entrenamiento del modelo.
- Poner en marcha del algoritmo de detección YOLO, modificando sus parámetros para que se adapte a nuestro dataset de entrenamiento.
- Entrenar el modelo, mediante el uso del dataset creado.
- Validar el modelo, analizando la capacidad de detección adquirida en el entrenamiento con nuevas imágenes.
- Implementar un algoritmo de tracking que nos permita realizar un seguimiento de cada uno de los objetos detectados mediante nuestro modelo de detección YOLO.

2. PRELIMINARES

2.1 Machine Learning

Conocemos el Machine Learning como aquella disciplina que nos permite dotar a los sistemas de la capacidad de aprender de forma autónoma. Todo este proceso de aprendizaje autónomo está basado en la capacidad del sistema para reconocer patrones de comportamiento de determinados parámetros en un conjunto de datos [8].

La capacidad de aprendizaje de este tipo de sistemas viene marcada por la cantidad de datos con la que somos capaces nutrir al sistema. Generalmente, a un mayor número de datos, mejor rendimiento nos ofrecerá el sistema al desarrollar su tarea. Sin embargo, si la cantidad de datos es muy pequeña, puede que el sistema no sea capaz de adquirir las capacidades necesarias para suministrarnos respuestas realmente válidas. Por ello, podemos afirmar que la cantidad de datos de la que disponemos para poder suministrar al sistema es una parte esencial a la hora de obtener un sistema dotado de Machine Learning que nos resulte realmente útil.

Dentro de esta disciplina, podemos encontrar tres tipos principales de formas de aprender:

- ***Aprendizaje supervisado:*** Este tipo de aprendizaje se basa en suministrar al sistema datos junto con sus resultados esperados, que pueden ser numéricos, en el caso de la regresión, o etiquetas en el de la clasificación. Por ejemplo, en un problema de clasificar imágenes de perros y gatos, se le suministran imágenes de perros y gatos en donde cada imagen viene bien definida sobre la clase a la que pertenece. Gracias a estas imágenes y sus respectivas etiquetas, el sistema aprende a registrar patrones que le indiquen si una imagen corresponde a un tipo o a otro de forma que cuando se introduzcan nuevas imágenes sin etiquetas, este pueda predecir de manera eficaz a qué tipo pertenece la nueva imagen.

- ***Aprendizaje no supervisado:*** Este tipo de aprendizaje se basa en la capacidad del sistema de comprender y obtener patrones directamente de las entradas que suministramos al sistema, sin que estas dispongan de ningún tipo de salida esperada. Es un proceso más complejo que el aprendizaje supervisado y se asemeja más a la forma en la que los humanos somos capaces de procesar la información. Como ejemplo de este tipo de aprendizaje, pondríamos el problema de agrupar clientes de un supermercado según su comportamiento, es decir, en función del tipo de productos que compran. De esta forma podríamos predecir comportamientos de

compra de determinados tipos de clientes y saber qué productos son más propensos a consumir.

- ***Aprendizaje por refuerzo:*** Este tipo de aprendizaje se basa en la experiencia que van obteniendo los sistemas con sus decisiones. Cuando el sistema toma una decisión errónea o un resultado incorrecto, se le penaliza dentro de un sistema de registro de valores. De manera contraria, cuando el sistema acierta, se le premia. De esta forma se obliga al sistema a que tienda a tomar decisiones premiadas, así como a no escoger decisiones que han sido penalizadas previamente. Es una técnica basada en la prueba y error, de forma que una de sus ventajas es que el sistema no debe disponer de una gran cantidad de información inicial. Un claro ejemplo de este tipo de aprendizaje son los robots [9] capaces de realizar tareas cotidianas como colgar perchas o cerrar botellas, los cuales para ser capaces de realizar este tipo de tareas aprenden mediante prueba/error. Por ejemplo, en caso de no ser capaz de colgar la percha se penaliza en su sistema de registro de valores, de manera que en próximas situaciones similares el robot habrá aprendido a realizar la tarea correctamente.

Tipo de aprendizaje	Categoría	Aplicación
Supervisado	Regresión	Predicción del crecimiento poblacional
		Previsión de mercados
		Previsión del clima
	Clasificación	Detección de fraude de identidad
		Clasificación de imágenes
		Diagnósticos médicos
No Supervisado	Clustering	Segmentación de clientes
		Marketing dirigido
		Sistemas de recomendación
	Reducción de la dimensionalidad	Visualización de Big Data
		Descubrimiento de estructuras
		Compresión significativa
Reforzado	Navegación de robots	
	Juegos con inteligencia artificial	
	Decisiones en tiempo real	

Tabla 1. Tipos de Machine Learning

2.2 Deep Learning

Podemos empezar a hablar de Deep Learning como un conjunto de técnicas de Machine Learning en la cual los sistemas adquieren conocimiento a partir de ejemplos mediante el uso de redes neuronales.

El Deep Learning tiene sus orígenes allá en los años 80, donde la falta de volumen de datos y el hardware de la época hacía que su implementación fuera inviable [10]. De esta manera, este campo fue olvidado hasta estos últimos años donde ha vuelto a coger un gran impulso. En la actualidad, el imparable aumento de generación de datos y una mayor potencia de cálculo gracias a las GPU nos permiten crear sistemas capaces de ser entrenados en unas pocas horas y con un alto rendimiento.

Los modelos empleados contienen una arquitectura en forma de red neuronal (Fig. 2) y son entrenados con grandes conjuntos de datos etiquetados. El uso de un buen conjunto de datos y una buena arquitectura permite desarrollar sistemas que en algunos casos desempeñan sus tareas con tal precisión que superan el rendimiento humano [11].

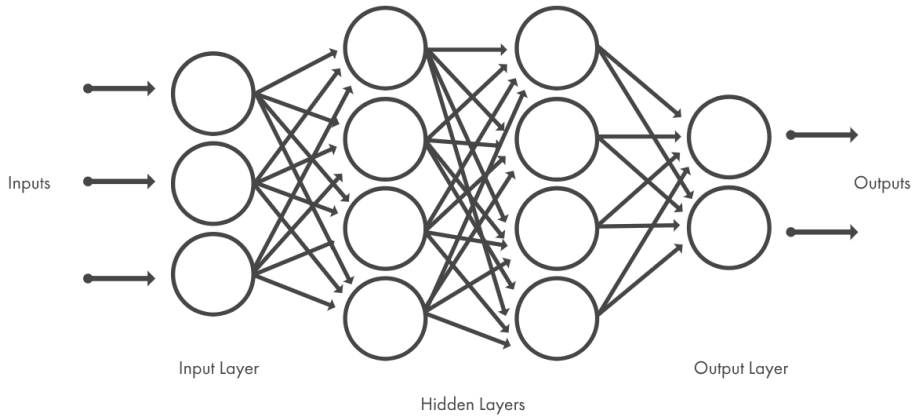


Figura 2. Arquitectura Red Neuronal

A menudo el uso de este tipo de arquitectura hace que a los modelos de Deep Learning se les conozca también como redes neuronales profundas. El termino profundo hace referencia al número de capas ocultas empleadas en la red neuronal. Una red neuronal tradicional contiene alrededor de 2-3 capas ocultas, mientras que las redes profundas pueden contener hasta 150 capas ocultas [12].

El Deep Learning es usado en multitud de campos como podrían ser el tratamiento de radiografías para detección de cáncer [13], asistentes digitales para la mejora de toma de decisiones en tiempo real como podría ser CORTI [14], traductores de idiomas como Google Translate [15], sistemas de reconocimiento de voz como Deep Speech [16] …

2.3 Redes Neuronales Convolucionales (CNN)

Cuando hablamos de redes neuronales convolucionales nos referimos a uno de los tipos de red neuronal profunda más populares en lo que a tratamiento y clasificación de imágenes se refiere [6].

Una CNN se diferencia de una red neuronal convencional en que realiza la convolución entre las características aprendidas y los datos de entrada, haciendo uso de capas convolucionales 2D. Esto hace que este tipo de redes sean idóneas para el tratamiento de datos en 2D como son las imágenes.

La red dispone desde un principio de las características relevantes, mientras que estas se aprenden con el entrenamiento de la red con un conjunto de imágenes. Este conjunto de imágenes es conocido como dataset, y de él la red extraerá patrones que le enseñarán a realizar la clasificación o detección de objetos.

Las CNN aprenden a detectar diferentes características en las imágenes gracias al uso de cientos de capas ocultas de su arquitectura. Cada nivel de profundidad en las capas ocultas aumenta la complejidad de las características de la imagen aprendidas.

Las primeras capas de la red se encargan de reconocer patrones simples como puede ser la detección de bordes, mientras que las capas más profundas se encargan de tareas más complejas como podrían ser detectar formas concretas de objetos.

Las últimas capas de la red deben llevar a un número determinado de salidas, en función por ejemplo del número de clases de objetos que queremos clasificar.

2.3.1 Convolución

Para entender el funcionamiento de una red neuronal convolucional (CNN), debemos entender el concepto de convolución [17], parte esencial del funcionamiento de este tipo de redes neuronales.

En primer lugar, debemos saber que las imágenes son tratadas como matrices de unas determinadas dimensiones (nº píxeles X nº píxeles X nº canales de color). Cada píxel que compone la imagen es un valor numérico al cual se asocia un determinado color, de manera que el conjunto de todos estos píxeles de diferentes colores es lo que conforma nuestra imagen.

Una imagen está compuesta por varias matrices en función del número de canales de color que componen la imagen, por lo que la convolución en imágenes a color se debe realizar sobre cada una de las matrices de los canales de color.

La convolución es un proceso que involucra a dos matrices. Por un lado, tenemos nuestra imagen original, la cual queremos tratar, y por otro lado tenemos una matriz de menores dimensiones llamada *Kernel.*

El proceso de convolución simplemente consiste en desplazar el *Kernel* por cada uno de los elementos de la imagen original y realizar la suma de los productos de cada uno de los elementos de ambas matrices, como podemos observar en la siguiente figura (Fig. 3). De esta manera el valor obtenido de esta suma será colocado en la misma posición que en la imagen original, pero en una nueva imagen que llamaremos "Imagen Convolucionada", la cual al termino del proceso tendrá el mismo tamaño que la imagen original.

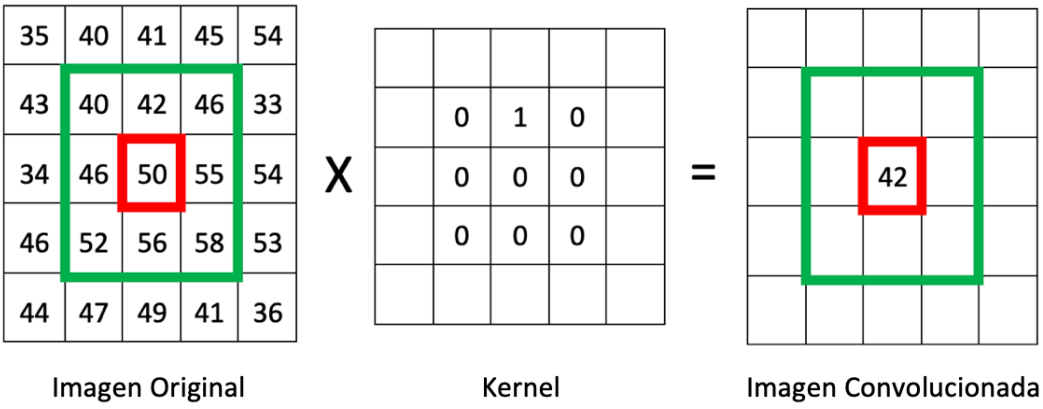


Figura 3. Convolución

El *Kernel* funciona como un filtro para las imágenes que nos permite obtener propiedades de las mismas, como detectar bordes, líneas verticales y horizontales, enfocar o desenfocar una imagen…

En el caso de que la imagen sea a color, es decir disponga de canales RGB, tendríamos que usar un *Kernel* de tamaño (n x n x nº canales), de forma que usaríamos ese *Kernel* para cada matriz del canal correspondiente. De esta forma obtendríamos los valores de los N *Kernel*, los cuales se sumarán de manera que conformarán una única salida, es decir como si fueran un único canal.

2.3.2 Pooling

La técnica de Pooling [18], está basada en reducir las dimensiones de nuestra imagen original y de las resultantes de las capas ocultas. Esto se realiza con el fin de resumir características de una región específica de la imagen. De esta manera nos quedaremos con las características más importantes que detecta cada filtro.

Su funcionamiento consiste en definir un tamaño de ventana (n x n), de manera que esta ventana sea aplicada por la imagen original. De esta forma, de los píxeles que formen parte de esa ventana en cada pasada, nos quedaremos con aquel que cumpla las condiciones del filtro que le estemos aplicando.

Como resultado a esta operación, el tamaño de los datos se reduce por un factor igual al tamaño de la ventana de muestra.

En nuestro caso vamos a utilizar Max Pooling, en el cual al realizar el proceso de Pooling nos quedaremos con el píxel que tenga un mayor valor como podemos observar en la siguiente figura (Fig. 4).

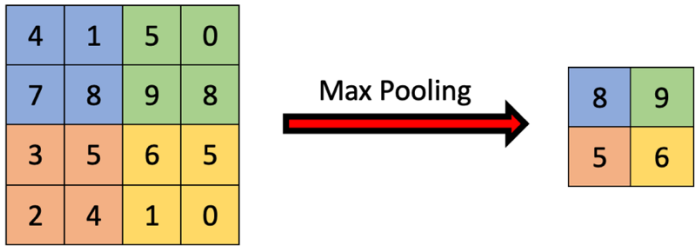


Figura 4. Max Pooling

No solo contamos con Max Pooling como tipo de Pooling [19] que podemos aplicar a las imágenes, también podemos usar Min Pooling, cuyo objetivo es quedarnos con el píxel que tenga un menor valor (Fig. 5).

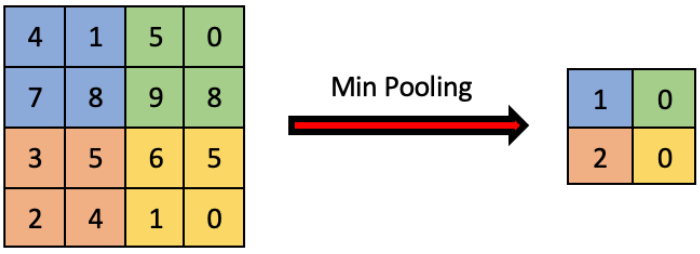


Figura 5. Min Pooling

Así mismo, disponemos de Average Pooling, mediante el cual nos quedamos con el valor medio de todos los pixeles en la región seleccionada (Fig. 6).

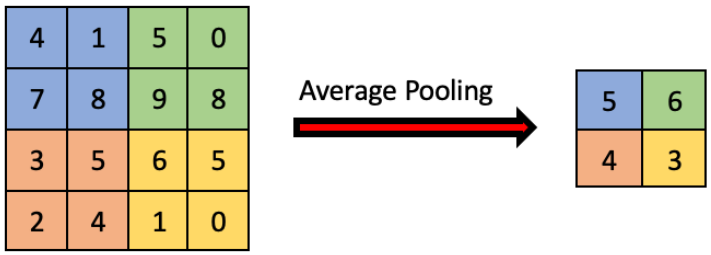


Figura 6. Average Pooling

En la siguiente figura (Fig. 7), podemos ver el resultado de aplicar cada uno de los tipos de Pooling explicados anteriormente. De manera que cada uno de ellos nos ofrece unas determinadas características, Min Pooling oscurece la imagen, Max Pooling aclara la imagen, mientras que Average Pooling difumina la imagen.

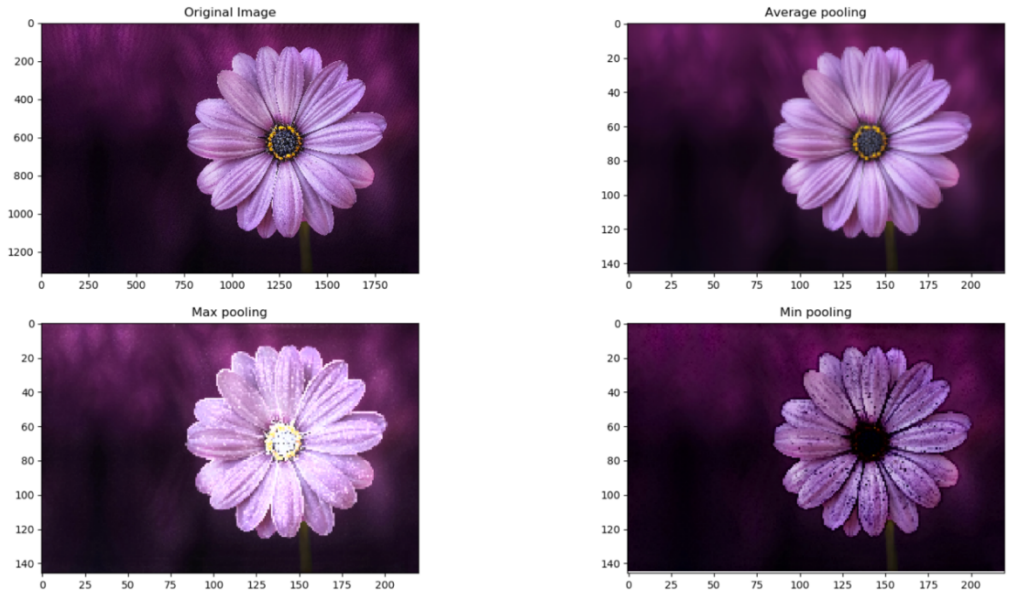


Figura 7. Aplicación de los distintos tipos de Pooling a una imagen.

2.3.3 Aprendizaje de características en CNN

Como el resto de redes neuronales, las Redes Neuronales Convolucionales (CNN) se componen de diversos tipos de capas, entre las que podemos encontrar una capa de entrada, una capa de salida y múltiples capas ocultas intermedias.

Todas estas capas realizan operaciones sobre los datos de entrada, con el fin de aprender características y obtener patrones de los mismos. En nuestra red neuronal convolucional disponemos de tres capas comunes que nos ayudan a realizar el aprendizaje de características:

- ***Capas de convolución:*** Por este tipo de capas se hacen pasar las imágenes de entrada. A cada imagen se le aplica varios filtros de convolución, de manera que nos quedaremos con unas determinadas características de la imagen, como hemos descrito sobre la convolución en apartados anteriores.

- ***Capas de activación (ReLU)*** [20] ***:*** Este tipo de capas nos permite poner todos los valores negativos a cero y mantener los valores positivos, de forma que el entrenamiento de la red sea más rápido y efectivo. A este tipo de capas se les conocen como capas de activación, ya que solo las características positivas siguen su camino hacia las siguientes capas.

- ***Capas de Pooling:*** Este tipo de capas simplifica la salida que obtenemos, ya que produce una disminución de la cantidad de datos que disponemos como hemos explicado anteriormente en el apartado de Pooling.

El uso de estas tres operaciones de manera iterativa nos permite obtener capas que aprenden e identifican características de las imágenes que introducimos en la red. De esta manera podemos decir que la red neuronal está aprendiendo.

2.3.4 Parámetros en CNN

A la hora realizar el entrenamiento de una red neuronal convolucional (CNN), disponemos de distintos parámetros los cuales nos ayudan a ajustar el entrenamiento a las características que deseamos. Aunque en función del modelo que usemos tenemos parámetros específicos del mismo, nos vamos en centrar en algunos parámetros globales para cualquier red neuronal que usemos como son:

- ***Pesos de los filtros (o Kernels):*** Es un valor numérico que se asocia a cada neurona con el fin de darle una determinada importancia. Este valor se irá actualizando en cada época de entrenamiento que realice la red neuronal. Con este parámetro de los pesos nos referimos a los valores de los *kernels*, es decir, en cada iteración se irán actualizando los *kernels* de manera que obtengamos un menor error al utilizar la red neuronal.

- ***Pesos de las capas finales (fully connected):*** En estas capas finales cada neurona de salida esta conectada completamente con todas las neuronas de la capa anterior, de ahí viene el nombre *fully connected*. En un principio los pesos de estas neuronas son inicializados de manera aleatoria, sin embargo, a medida que realizamos épocas en la red neuronal y obtenemos el error cometido, estos pesos se van actualizando con el fin de reducir ese error cometido y así obtener un mejor rendimiento.

Existen otro tipo de parámetros los cuales son usados para realizar la configuración del modelo, estos parámetros son conocidos como **hiperparámetros** y algunos de ellos son los siguientes:

- ***Tamaño de los filtros:*** Decidiremos el tamaño (nxn) de los filtros que usaremos para realizar Pooling. También es importante decidir el número de filtros que va a contener cada capa, ya que a mayor número de filtros, mayor número de matrices de salida y de neuronas obtendremos.

- ***Número de capas de la red:*** Cuanto mayor sea el número de capas, mayor tiempo de cálculo y mayor número de datos de entrada vamos a necesitar.

- ***Número de épocas:*** Con este parámetro nos referimos a cada iteración que realizará la red por el dataset de entrenamiento, con el fin de mejorar el aprendizaje del mismo.

- ***Tasa de aprendizaje:*** Es un valor numérico usado en el entrenamiento de la red con el fin de regular la velocidad de aprendizaje. Si es una tasa muy grande, el algoritmo aprende más rápido, pero tendremos mayor imprecisión en el resultado. Si es demasiado pequeño, tardará mucho y podría no finalizar nunca. Por lo que es fundamental usar un valor adecuado para que la red aprenda a un ritmo rápido, pero sin tener imprecisiones en los resultados.

2.3.5 Clasificación en CNN

Una vez que nuestra red ha aprendido las características en sus numerosas capas, pasamos a la fase de clasificación.

Como paso final de nuestra red neuronal, disponemos de una capa totalmente conectada, la cual nos genera un vector con N dimensiones, siendo N el número de clases que la red es capaz de predecir. Este vector contendrá las probabilidades de cada clase para la imagen que estemos clasificando. De esta forma, la clase con una mayor probabilidad será la elegida como salida de la red neuronal.

Para obtener probabilidades con unos valores normalizados, haremos uso de funciones como puede ser Softmax [21], la cual nos permite que todas las probabilidades obtenidas para las N clases tengan un valor entre 0 y 1, de forma que la suma total de probabilidades de las N clases será 1.

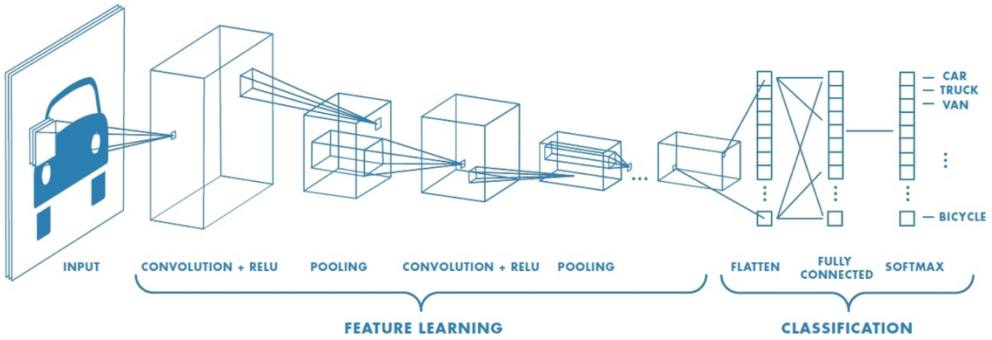


Figura 8. Ejemplo de Red Neuronal Convolucional

2.3.6 Medición del error cometido en CNN

Para medir el error cometido en el uso de CNN podemos realizarlo mediante el uso de varias fórmulas. Cada una de ellas nos ofrece ventajas e inconvenientes:

- ***Error Cuadrático Medio (RMSE):*** Se realiza la raíz cuadrada de la diferencia entre el valor previsto (correcto) y el valor obtenido:

$$RMSE = \sqrt[2]{\sum_{i=1}^{n} \frac{(Y_i - y_i)^2}{n}},$$

donde Y_i es el valor previsto, y_i es el valor obtenido y n es el número de ejemplos totales. Como ventaja penaliza mucho los valores que son muy grandes.

- ***Error Absoluto Medio (MAE):*** Se calcula como la suma de media de los valores absolutos de los errores:

$$MAE = \sum_{i=1}^{n} \frac{|Y_i - y_i|}{n} \ ,$$

donde Y_i es el valor previsto, y_i es el valor obtenido y n es el número de ejemplos totales. Como ventaja, es más fácil de interpretar que el error cuadrático medio. Penaliza menos los valores que son muy grandes.

- ***Error Absoluto Medio Escalado (MASE):*** Se calcula de la misma manera que el error absoluto medio, pero escalándolo:

$$MASE = \frac{\sum_{i=1}^{n}|Y_i - y_i|}{\frac{n}{n-1}\sum_{i=2}^{n}|Y_i - y_{i-1}|} \ ,$$

donde Y_i es el valor previsto, y_i es el valor obtenido y n es el número de ejemplos totales. Como ventaja, escala los valores y es simétrica.

- ***Entropía Cruzada Binaria (Binary Cross Entropy):*** Cuantifica la diferencia entre dos distribuciones de probabilidad:

$$BCE = -\sum_{i}^{M} (y_i \log(Y_i) + (1 - y_i) \log(1 - Y_i)) \ ,$$

donde Y_i es el valor previsto, y_i es el valor obtenido y M es el número de clases totales.

- ***Entropía Cruzada (Cross Entropy):*** Cuantifica la diferencia entre dos distribuciones de probabilidad:

$$CE = -\sum_{i}^{M} y_i \log(Y_i) \ ,$$

donde Y_i es el valor previsto, y_i es el valor obtenido y M es el número de clases totales.

2.4 Evaluación de modelos

Para realizar un buen uso del conjunto de datos que disponemos para usar en nuestro modelo, este debe separarse en subconjuntos. Esta división es necesaria ya que, los datos que usemos para realizar el entrenamiento del modelo no deben ser los mismos que usemos para realizar sus test o su validación. Es por ello por lo que, este conjunto de datos se debe dividir en tres subconjuntos (Fig. 9):

- ***Conjunto de entrenamiento:*** Este conjunto debe ser el de mayor tamaño de los tres (50% - 70% del total de datos), ya que, a partir de estos datos se va a realizar el entrenamiento del modelo.

- ***Conjunto de test/prueba:*** Es una porción de mucho menor tamaño que el conjunto de entrenamiento (15% - 25% del total de datos), mediante el cual se evalúa el modelo. Normalmente a partir de este conjunto obtendremos la eficacia del modelo al enfrentarse a estos nuevos datos.

- ***Conjunto de validación:*** Esta porción (15% - 25% del total de datos) se usa para validar el modelo, es decir, para realizar una buena elección de los parámetros, como pueden ser el número de capas ocultas o el número de épocas, con el fin de obtener el mejor modelo posible.

Figura 9. Entrenamiento, Validación y Test

Una vez realizado el entrenamiento del modelo, nos disponemos a analizar cómo es de bueno su rendimiento en cuanto a la detección de vehículos. Para ello, debemos tener un conjunto de imágenes que no han sido empleadas para el entrenamiento del modelo, al cual llamaremos conjunto de validación.

En nuestro caso, el modelo empleado para realizar la detección dispone de una forma automática de crear un conjunto de datos de validación. Cuando nos disponemos a realizar el entrenamiento, el modelo nos solicita un conjunto de entrenamiento de imágenes a partir del cual aprenderá a detectar vehículos.

Opcionalmente nos solicita que también empleemos un conjunto de validación con el cual pueda en un futuro comprobar el rendimiento que ofrece después de realizar el entrenamiento. En caso de que se lo facilitemos, el modelo realizara la validación sobre este conjunto. En el caso de que no le aportemos ningún dataset de validación, este lo creará usando el 20% de los datos de entrenamiento.
De esta forma, nos aseguramos de que el modelo disponga de un conjunto de datos de validación con el fin de que pueda aportarnos una evaluación del entrenamiento realizado.

3. DETECCIÓN DE OBJETOS CON YOLO

3.1 Detección de objetos y YOLO

Muchos de los proyectos de Machine Learning que llevan a cabo las empresas en la actualidad incluyen de alguna manera la detección de objetos en imágenes. Esta detección no solo conlleva a identificar los tipos de objetos que se encuentran en la imagen, sino que también debemos localizarlos dentro de la imagen. De esta manera, podemos decir que la detección de objetos se basa en clasificación y localización.

Si investigamos un poco sobre técnicas para detección de objetos en imágenes, llegamos a la conclusión de que nos tendremos que centrar en hacer uso de Deep Learning. En este proyecto nos hemos centrado de manera exclusiva en hacer uso del algoritmo de detección de objetos YOLO (You Only Look Once) [7].

YOLO hace uso de Deep Learning y de redes neuronales convolucionales (CNN) para la detección de objetos. Los pasos que sigue YOLO (Fig. 10) a la hora de realizar la detección de objetos en imágenes son los siguientes:

1. Divide la imagen en una cuadricula de *NxN* celdas.

2. En cada una de estas celdas, el modelo ejecuta la localización de objetos que hemos descrito en el apartado 3.3 prediciendo S posibles *Bounding boxes* y calcula un nivel de certidumbre o probabilidad de cada una de ellas.

3. Una vez obtenido el nivel de certidumbre de cada una de las *Bounding boxes*, el algoritmo escoge aquellas que dan un mejor rendimiento, de esta forma los objetos quedarán bien localizados y clasificados en la imagen.

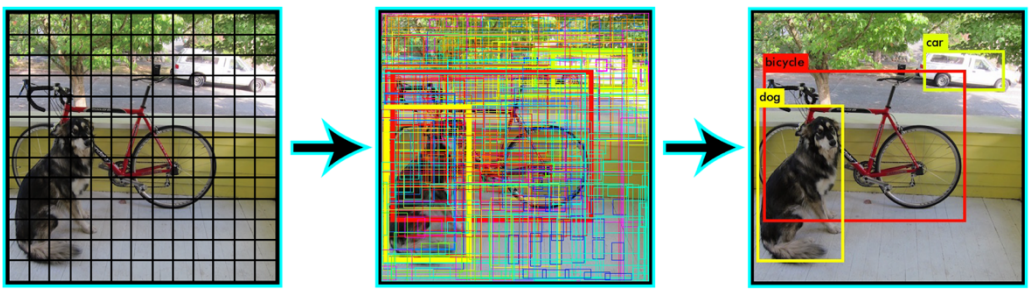


Figura 10. Pasos de YOLO [22].

Una de las principales ventajas del uso de YOLO en detección de objetos en imágenes es que es capaz de funcionar correctamente recorriendo la imagen una única vez, como indica su nombre. Aunque pierde un poco de exactitud, esta velocidad le permite ser uno de los algoritmos de detección de objetos en imagen más rápidos, así como ser capaz de realizar la detección de objetos en vídeos a tiempo real (hasta 30 FPS).

3.2 Descripción de la arquitectura de YOLO

La arquitectura con la que cuenta YOLO es una red neuronal convolucional. Las primeras capas de la red extraen características de la imagen, mientras que, las capas totalmente conectadas predicen las coordenadas de salida y coordenadas. La arquitectura de la red está inspirada en el modelo para la clasificación de imágenes GoogleNet [23].

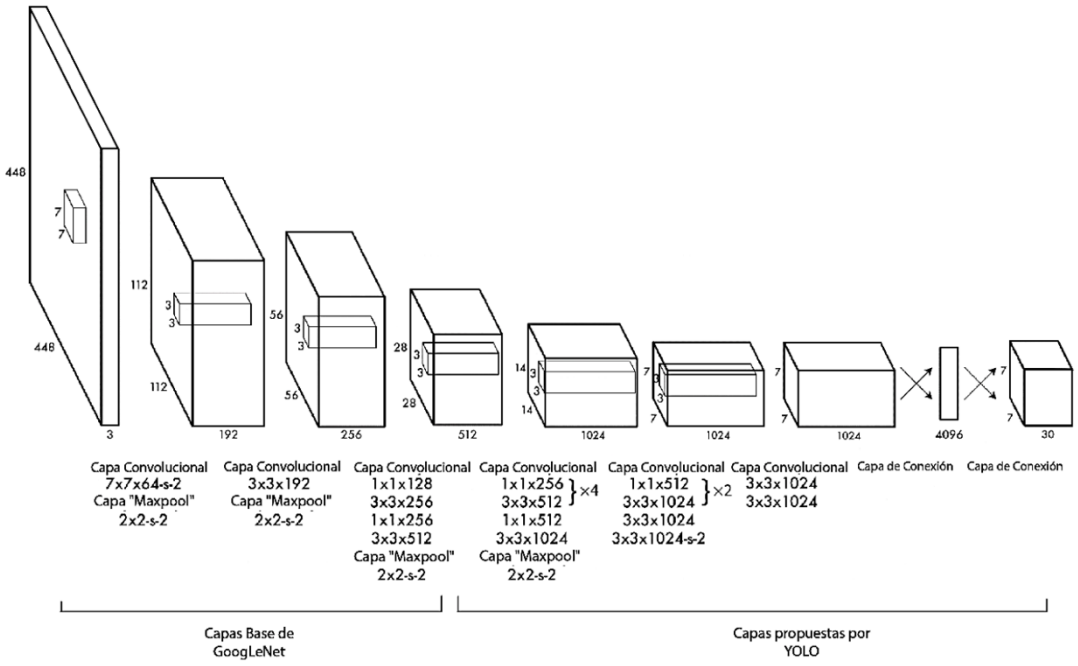


Figura 11. Arquitectura de la red neuronal YOLO.

Como podemos ver en la arquitectura (Fig. 11), la red está formada por 24 capas convolucionales seguidas por 2 capas completamente conectadas. En lugar de los módulos propuestos por GoogleNet, YOLO utiliza capas de reducción de 1x1 seguidas de capas convolucionales de 3x3.

La principal característica de YOLOv3 es que realiza detecciones en tres escalas diferentes de imágenes. Esta detección en las capas de reducción se realiza aplicando *kernels* de 1 x 1 en imágenes de tres tamaños diferentes en distintas partes de la arquitectura.

El tamaño del *kernel* es 1 x 1 x (B x (5 + C)), donde B se refiere al numero máximo de *Bounding Boxes* que se pueden predecir en una imagen y el "5" es para las cuatro coordenadas de las *Bounding Boxes* y la precisión de la detección. Por último, la C se refiere al número de clases de salida.

3.3 Localización de objetos mediante Bounding Box

La localización de objetos consiste en la capacidad de un modelo para indicar en qué posición dentro de una imagen se encuentra el objeto, colocando a su alrededor un rectángulo delimitador o Bounding Box.

Dada una imagen como entrada, nuestro modelo construirá como resultado un vector de las características de la misma con la siguiente forma:

$$y = \begin{bmatrix} p \\ b_x \\ b_y \\ b_w \\ b_h \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

En el vector anterior podemos ver varias características de la imagen procesada por el modelo. Empezaremos explicando el valor de p, el cual es un valor numérico entre 0 y 1 que nos indica el nivel de confianza del modelo sobre si hay un objeto en la imagen o no lo hay. En el caso de que este valor sea 1, nos indicará que el modelo está completamente seguro de que la imagen contiene el objeto, mientras que, si es un 0, el modelo estará completamente seguro de que la imagen no contiene el objeto.

Los siguientes cuatro parámetros que vemos en el vector hacen referencia a la posición que ocupa la *Bounding Box* dentro de la imagen. El par (b_x, b_y) nos indicará las coordenadas del punto central de la Bounding Box en la imagen, mientras que los valores de b_w y b_h representarán el ancho y el alto de la misma respectivamente.

Por último, los valores $(c_1, \dots, c_n)$ nos indican las posibles *n* categorías del objeto a localizar y clasificar en la imagen. Para ello se usará *One-hot Encoding* [24], es decir, si el objeto pertenece a la categoría *i*, el valor de c_i será 1, mientras que el valor en el resto de categorías será 0.
Para la mejor comprensión de lo explicado anteriormente, vamos a poner un ejemplo del vector obtenido para la detección de vehículos en una imagen.

$$y = \begin{bmatrix} 1 \\ 0.5 \\ 0.5 \\ 0.6 \\ 0.3 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

En este caso, en la imagen si que se habría localizado el objeto, ya que el valor de *p* es 1. Los valores de (b_x, b_y) son 0.5, lo que significa que el objeto aparece en el centro de la imagen. El valor de b_w es 0.6, lo que nos indica que el objeto ocupa el 60% de la anchura de la imagen, mientras que el valor de b_h es 0.3, indicándonos que el objeto ocupa el 30% de la altura. De esta forma, la Bounding Box quedaría bien situada dentro de la imagen. Por último, obtendríamos el valor para las distintas categorías, las cuales podrían ser c_1 = coche, c_2 = camión y c_3 = moto. En esta caso, vemos como el valor 1 en la primera categoría, nos indicaría que el objeto detectado está clasificado como coche.

En el caso de que el modelo no encuentre el objeto en la imagen, obtendríamos el siguiente vector de salida:

$$y = \begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$$

En este caso, el único valor que interesa es el de *p*, mientras que el resto de valores dejan de tener relevancia.

3.4 Métrica de evaluación Intersection over Union (IoU)

Intersection over Union [25] es una métrica de evaluación utilizada para medir la precisión de un detector de objetos en un conjunto de datos en particular. Cualquier tipo de algoritmo de detección de objetos que realice su detección mediante Bounding Boxes puede hacer uso de esta métrica.

Para poder aplicar esta métrica de evaluación necesitamos dos elementos (Fig. 12):

- El conjunto de las Bounding Boxes reales de donde se encuentran los objetos, es decir, las Bounding Boxes precisas etiquetadas a mano de los objetos a detectar.
- El conjunto de las Bounding Boxes obtenidas mediante la predicción con nuestro modelo.

Mediante estos dos conjunto de Bounding Boxes, podremos aplicar la métrica *Intersection over Union*.

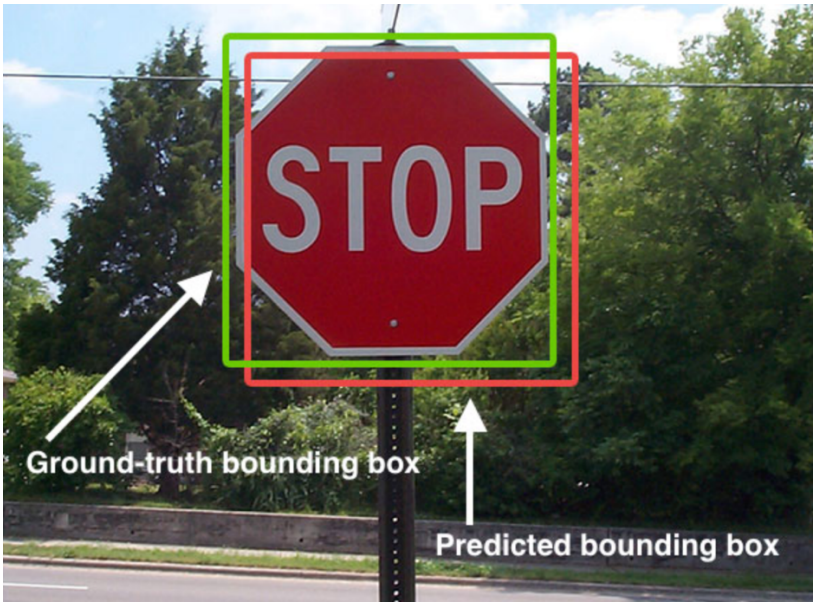


Figura 12. Bounding Boxes en Intersection over Union

Una vez tenemos las Bounding Boxes de un objeto, realizar el cálculo de *Intersection over Union* (Fig. 13) es tan sencillo como aplicar lo que su propio nombre indica, es decir, dividir el área de superposición entre las dos Bounding Boxes por el área de unión de ambas.

$$IoU = \frac{Intersección\ de\ BBox}{Unión\ de\ BBox}$$

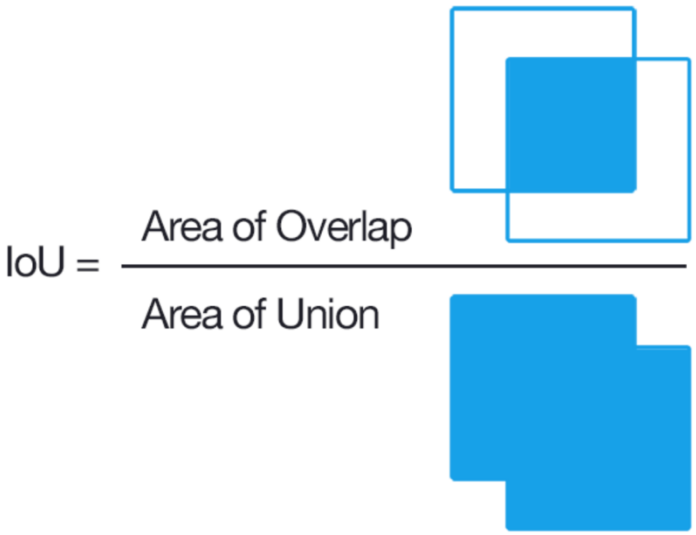


Figura 13. Formula visual Intersection over union

Una vez realizada la operación, si el valor obtenido de *Intersection over Union* es mayor de 0.5, podremos decir que la predicción es buena.

3.5 Métrica mAP (mean Average Precision)

Para la evaluación del modelo implementado, vamos a hacer uso de una métrica llamada mAP (mean Average Precision) [26], mediante la cual, podremos obtener el valor de precisión promedio de nuestro modelo al procesar nuevas imágenes. Es decir que, esta métrica nos devolverá un valor entre 0 y 1, siendo el 0 el peor de los casos, así como el 1 el mejor resultado posible.

Para entender el uso de esta métrica, debemos hacer referencia previamente a varios términos:

- **True Positive (TP):** Es el caso en el que algo es positivo y es predicho como tal. En nuestro caso, se refiere a cuando etiquetamos un vehículo como tal y verdaderamente lo es.
- **True Negative (TN):** Es el caso en el que algo es negativo y es predicho como tal. En nuestro caso, se refiere por ejemplo, a cuando algo que no es un vehículo no es detectado como vehículo.
- **False Positive (FP):** Es el caso en el que algo es negativo y es predicho como positivo. En nuestro caso, cuando nuestro detector cataloga algo como vehículo cuando no lo es.

- **False Negative (FN):** Es el caso en el que algo es positivo y es predicho como negativo. En nuestro caso, cuando el detector no detecta un coche en la imagen o vídeo.

Con estos términos, podemos hacer uso de las siguientes métricas, las cuales son necesarias para realizar el cálculo del valor mAP:

- Precisión:

  Cuando hablamos de precisión, nos referimos al desempeño del modelo a la hora de realizar sus predicciones, es decir, que porcentaje de estas son correctas.

$$Precision = \frac{TP}{TP + FP}.$$

- Recall:

  Cuando hablamos de recall, nos referimos al desempeño del modelo a la hora de encontrar todos los casos positivos. Es decir, que tanto porciento de estos los evalúa como tal.

$$Recall = \frac{TP}{TP + FN}.$$

Por último, también se hará uso de la métrica *Intersection over Union (IoU)*, explicada en apartados anteriores. Esta métrica será utilizada para determinar si una predicción la catalogamos como correcta o no. Si el valor de *IoU* es mayor que un valor prefijado, diremos que la predicción es correcta, en caso contrario diremos que es incorrecta.

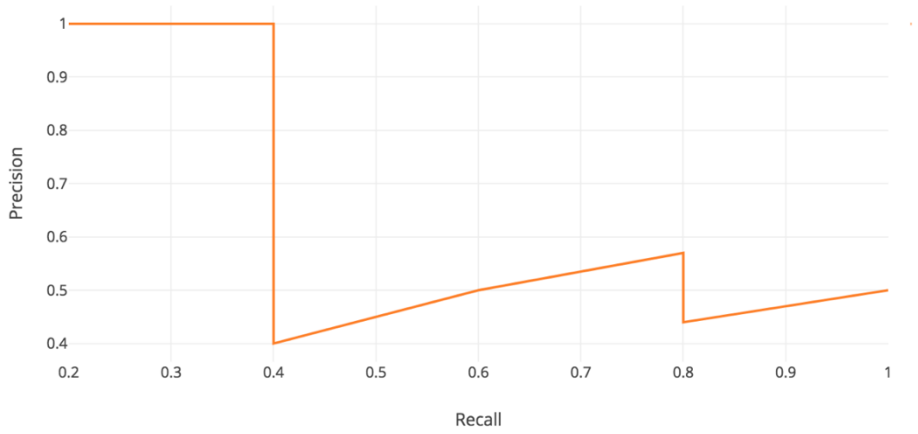
Para el entendimiento de esta métrica vamos a hacer uso de un ejemplo [26] :

Disponemos de un conjunto de datos el cual únicamente contiene 5 manzanas. Recopilamos todas las predicciones hechas para las manzanas en todas las imágenes, ordenándolas de mayor a menor valor de confianza, de forma que obtenemos un ranking como el que podemos ver en la Tabla 8. Para el segundo valor de la tabla consideraremos que la predicción es correcta si tiene un valor *IoU > 0.5*.

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

Tabla 2. Valores precision-recall

A continuación, representaremos los datos de precisión y recall enfrentados en una gráfica, de modo que obtendremos como resultado una curva (Gráfica 1).

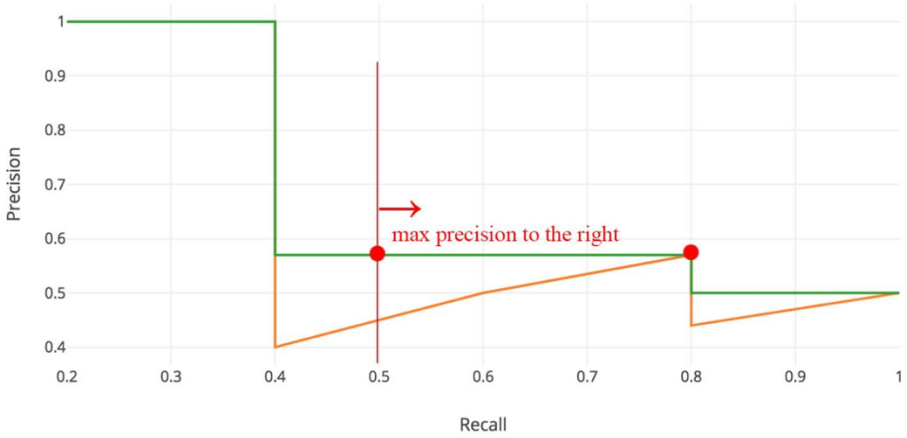


Gráfica 1. Curva precision-recall - Parte 1

La definición general de la métrica AP (Average Precision) es el valor del área bajo la curva obtenida en la gráfica anterior.

$$AP = \int_0^1 p(r)\, dr$$

Los valores de precisión y recall siempre se encuentran entre 0 y 1, por lo que el valor de AP también estará dentro de estos valores. Antes de calcular el valor de AP, se suaviza la curva, donde para cada cifra de recall se sustituye su valor de precisión por el máximo de sus valores a la derecha.



Gráfica 2. Curva precision-recall - Parte 2

De esta manera, nuestro valor de AP se calcularía con el área obtenida por debajo la curva verde, en vez de la curva naranja (Gráfica 2).

4. OBJECT TRACKING

El objetivo de este trabajo es realizar una estimación del flujo de vehículos que pasan por un determinado punto. Para ello, nos basamos en la detección de los vehículos en imágenes mediante YOLO, pero no con eso basta. Para extraer información acerca de la densidad del tráfico de un vídeo, debemos realizar de alguna forma, un seguimiento concreto de cada vehículo detectado con el fin de contar cada vehículo diferente una única vez. Es decir, necesitamos poder estimar en el tiempo la ubicación de cada uno de los vehículos mediante únicamente el uso del vídeo.

4.1 Algoritmo Simple Online and Realtime Tracking (SORT)

Para realizar el tracking de los vehículos en nuestro proyecto, vamos a utilizar el algoritmo *Simple Online and Realtime Tracking (SORT)* [27], el cual está dirigido principalmente al seguimiento en línea donde solo se presentan las detecciones del frame anterior y el frame actual.

Este algoritmo está basado en técnicas de detección de cambios, es decir, el principal objetivo es la detección de los píxeles del objeto y los píxeles de fondo. Asumen que el fondo es estacionario y que los cambios entre frames consecutivos son debidos al movimiento de los objetos.

Durante la primera fase de este algoritmo, el mismo realiza una detección del punto central del objeto detectado y así, extraer características del mismo, de modo que en su segunda etapa pueda clasificar el objeto detectado. Por defecto, SORT utiliza el algoritmo de detección *Faster Region CNN (FrRCNN)* [28], sin embargo, en nuestro proyecto este algoritmo de detección será reemplazado por YOLO.

Cada objeto detectado se representará mediante un vector el cual será usado para propagar la identidad del objeto en el siguiente fotograma. El estado de cada objeto se modela como:

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]^T,$$

donde *u* y *v* representan las coordenadas horizontal y vertical del píxel central del objeto, mientras que, *s* y *r* representan respectivamente la escala (área) y la relación de aspecto de la Bounding Box. Por último, con $\dot{u}, \dot{v}$ y $\dot{s}$, nos referimos a los componentes de la velocidad, calculados de manera óptima mediante un filtro Kalman [29].

El seguimiento de los objetos detectados se realiza por medio de la distancia obtenida mediante *Intersection over Union (IoU)* entre cada detección nueva y todas las Bounding Box detectadas en el frame anterior. De esta manera, todas aquellas nuevas detecciones que tengan un valor de IoU mayor que un valor prefijado de IoU_{min} , se consideraran válidas y se asignaran al Bounding Box del objeto del frame anterior con el cual su valor de IoU sea máximo.

A cada uno de los objetos se les asigna un ID, de manera que cuando un objeto entre o sale de la imagen, este ID deberá ser creado como nuevo o destruido. Para ello, se establece que cualquier nueva detección con un valor de IoU menor que IoU_{min} será considerado como nuevo objeto. De la misma forma, los ID que no sean detectados para un número determinado de fotogramas, se eliminarán automáticamente.

5. ESTIMACIÓN DE LA DENSIDAD DEL TRÁFICO

En esta sección nos centraremos en explicar cada uno de los pasos a realizar para el uso de las técnicas descritas en el problema de estimar la densidad del tráfico en un punto concreto.

Como primer paso veremos la elección del dataset que contendrá los datos con los que entrenaremos nuestro modelo. La elección de una buen dataset es fundamental a la hora de sacarle un rendimiento óptimo a nuestro modelo.

5.1 Estudio de diferentes Datasets

El dataset es una parte fundamental a la hora de obtener buenos resultados de detección en nuestro modelo, ya que, es el lugar de donde nuestro modelo aprende para posteriormente detectar de una manera correcta y eficaz. Es por ello que, para la elección del dataset vamos a realizar un análisis sobre algunos dataset de detección de vehículos para así poder sacar conclusiones sobre cual de ellos nos interesa más o encaja mejor en nuestro proyecto.

- **Dataset 1 (CrowdAI** [30]**):**

  El primer dataset consiste en una colección de imágenes tomadas en EE.UU. durante el día (Fig. 14). Contiene la cantidad de 9.423 frames tomadas a una resolución de 1920x1200 px. El dataset fue creado por CrowdAI mediante el uso de Machine Learning y humanos.

  El tipo de etiquetado utilizado es del tipo BBox, es decir se emplean Bounding Box que engloban las etiquetas de salida. Algunas de las características del dataset pueden verse en la Tabla 3.

LABELS	CSV FORMAT	TAMAÑO
• Coches • Camiones • Peatones	• Xmin • Ymin • Xmax • Ymax • Frame • Label • Preview URL for frame	• 1.5 GB

Tabla 3. Especificaciones del Dataset CrowdAI

Figura 14. Imagen de dataset CrowdAI.

- **Dataset 2 (Autti** [31]**):**

  El segundo dataset es similar a nuestro primer dataset, pero con la particularidad de que capta la oclusión y aumenta su número de etiquetas, ya que, es capaz de detectar semáforos en la calzada. Está compuesto por un total de 15.000 frames y fue etiquetado completamente por humanos.

  El tipo de etiquetado utilizado es del tipo BBox, es decir, se emplean bounding box que engloban las etiquetas de salida (Fig. 15). Algunas de las características del dataset pueden verse en la Tabla 4.

LABELS	CSV FORMAT	TAMAÑO
• Coches • Camiones • Peatones • Semáforos	• Frame • Xmin • Ymin • Xmax • Ymax • Oclusión • Label • Atributos (Solo en semáforos)	• 3.3 GB

Tabla 4. Especificaciones del Dataset Autti

Figura 15. Imagen de dataset Autti.

- **Dataset 3 (Cityscapes** [32]**):**

  Consiste en un dataset muy completo el cual está formado por hasta 30 etiquetas diferentes. Contiene imágenes de unas 50 ciudades, tanto en primavera, verano u otoño. Todas las imágenes, han sido tomadas en unas condiciones climáticas buenas y en diferentes horas del día. Consta de unas 5.000 imágenes.

  El tipo de etiquetado utilizado es del tipo píxel a píxel (Semantic Segmentation), es decir, se asocia a cada píxel de la imagen con una etiqueta de clase de salida (Fig. 16). Los tipos de etiquetas que podemos encontrar en este dataset pueden verse en la Tabla 5.

GRUPO	CLASES
Suelo	Carretera, Acera, Parking, Railes
Humanos	Personas, Ciclista
Vehículos	Coche, Camión, Autobús, Tren, Moto, Bicicleta, Caravana, Tráiler
Construcciones	Edificio, Muro, Alambrada, Guarda rail, Puente, Túnel
Objetos	Señales, Semáforos, Postes
Naturaleza	Vegetación, Terreno
Cielo	Cielo
Vacío	Asfalto, Situación dinámica, Situación estática

Tabla 5. Especificaciones del Dataset CityScapes

Figura 16. Imagen del dataset Cityscapes.

- **Dataset 4 (ApolloScape** [33]**):**

  Consiste en un dataset formado por más de 5.000 frames de alta calidad, en los cuales tenemos etiquetas de coches, peatones, ciclistas, etc.
  El dataset contiene diferentes conjuntos de vídeos grabados en distintas ciudades.
  La curiosidad de este dataset es que, se basa en un sistema 3D y que contiene modelos de vehículos guardados para ser capaz de detectar estos tipos de vehículos en concreto.

  El tipo de etiquetado utilizado es del tipo píxel a píxel (Semantic Segmentation), es decir, se asocia a cada píxel de la imagen con una etiqueta de clase de salida (Fig. 17).

Figura 17. Imagen del dataset Apolloscape.

- **Dataset 5 (BBD100K** [34]**):**

  Consiste en un dataset de la Universidad de Berkeley, el cual, contiene 100.000 secuencias de vídeo, cada una de aproximadamente 40 segundos y con una calidad de imagen de 720p y 30 frames por segundo. Los vídeos han sido realizados en distintas localizaciones de EE.UU.

  Uno de los puntos positivos es que está realizado en distintas condiciones climatológicas, ya que, hay vídeos con tiempo soleado, nublado, lloviendo o incluso con niebla.

  Es un dataset que también se puede utilizar para detectar peatones en la calzada, ya que contiene más de 85.000 instancias de peatones, lo que lo hace ideal para este tipo de situaciones.

  El tipo de etiquetado utilizado es del tipo píxel a píxel (Semantic Segmentation), es decir, se asocia a cada píxel de la imagen con una etiqueta de clase de salida (Fig. 18).

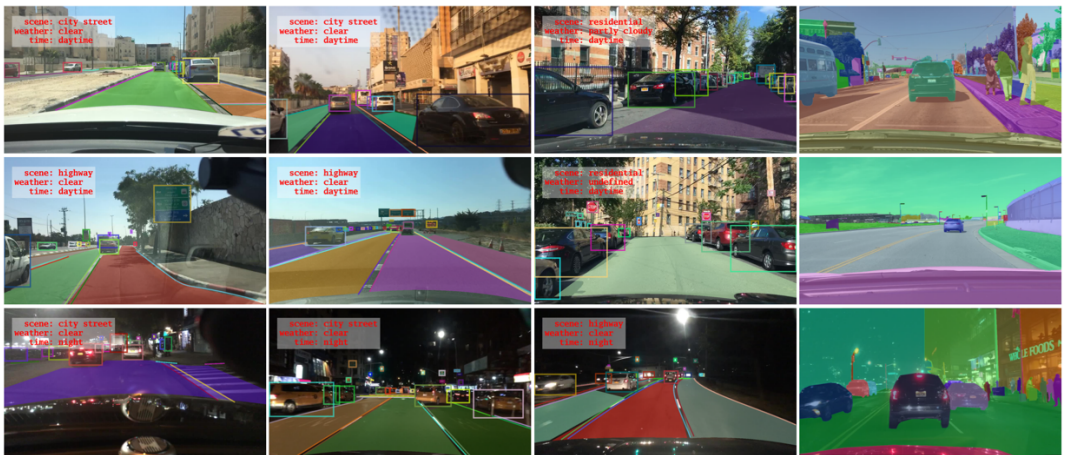


Figura 18. Imagen del dataset BBD100K.

- **Dataset 6 (Detrac** [35]**):**

  Este dataset está compuesto por un conjunto de entrenamiento, así como de un conjunto de test (Tabla 6). Todas las imágenes incluidas en ambas partes del dataset son imágenes en color de 24 bits y con formato .jpeg, cuya resolución es de 960x540 px.
  Está compuesto por distintas etiquetas de clasificación, entre las que podemos encontrar coches, autobuses, furgonetas…

  El tipo de etiquetado utilizado es del tipo BBox, es decir, se emplean bounding box que engloban las etiquetas de salida (Fig. 19).

CONJUNTO DE DATOS	FRAMES	BOUNDING BOXES	TAMAÑO
Entrenamiento	83791	577899	5.22 GB
Test	56340	632270	3.94 GB

Tabla 6. Conjunto de datos del Dataset Detrac

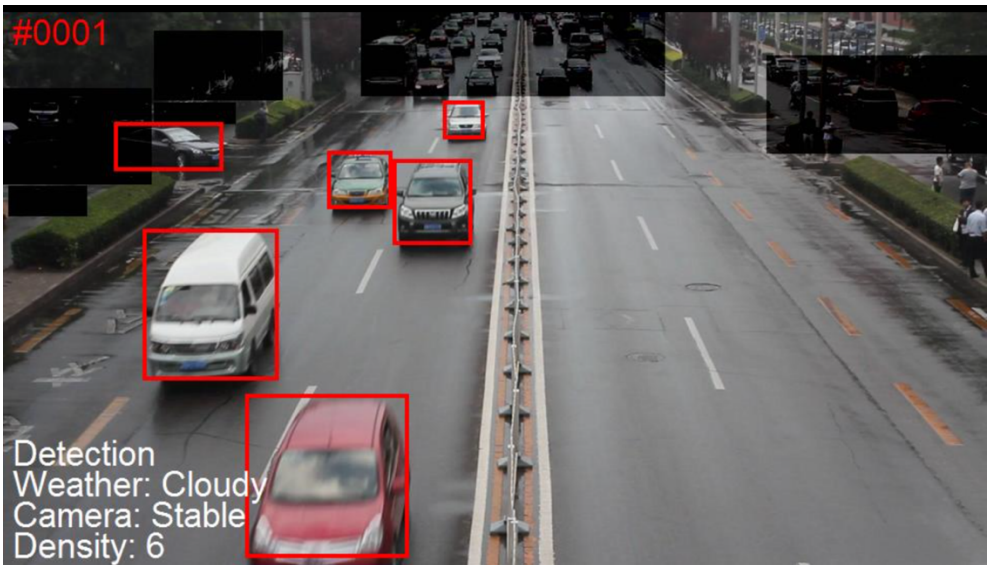


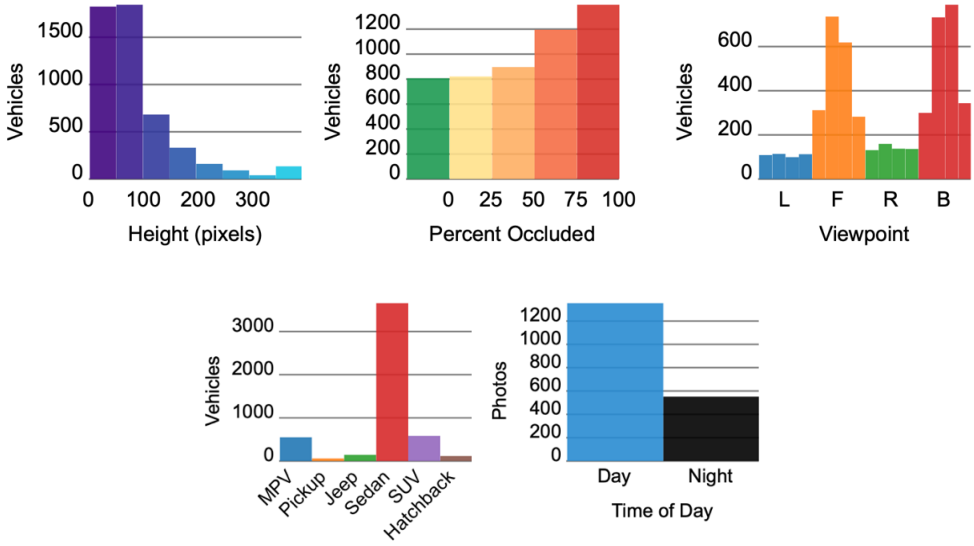
Figura 19. Imagen del dataset Detrac.

- **Dataset 7 (Nyc3Dcars** [36]**):**

  Este dataset está compuesto por imágenes de internet del tráfico de la ciudad de Nueva York. Todas las imágenes contenidas han sido etiquetadas por humanos, los cuales añaden a cada una de ellas la información de la posición del vehículo desde 6 puntos distintos de vista, el tipo de vehículo, una *Bounding box* sobre el vehículo, así como la hora del día en que se ha realizado la foto. Como información adicional, el dataset tiene un tamaño de 3.68 GB. Para una mejor comprensión, éste nos aporta algunas gráficas (Gráfica 3).

  El tipo de etiquetado utilizado es del tipo píxel a píxel (Semantic Segmentation), es decir, se asocia a cada píxel de la imagen con una etiqueta de clase de salida (Fig. 20).

Figura 20. Imagen del dataset Nyc3Dcars.



Gráfica 3. Dataset Nyc3DCars

- **Dataset 8 (Karlsruhe)** [37] **:**

  Consiste en un dataset compuesto por unas 1.000 imágenes en blanco y negro, en las cuales se etiquetan tanto a peatones como a vehículos.

  Además, contiene la información sobre la orientación de cada uno de los vehículos o de los peatones. En el caso de los vehículos, lo tenemos discretizado hasta en 8 clases, mientras que para los peatones tenemos 4 clases.

  Como información adicional, el dataset tiene un tamaño de 342 MB.

  El tipo de etiquetado utilizado es del tipo BBox, es decir, se emplean bounding box que engloban las etiquetas de salida (Fig. 21).

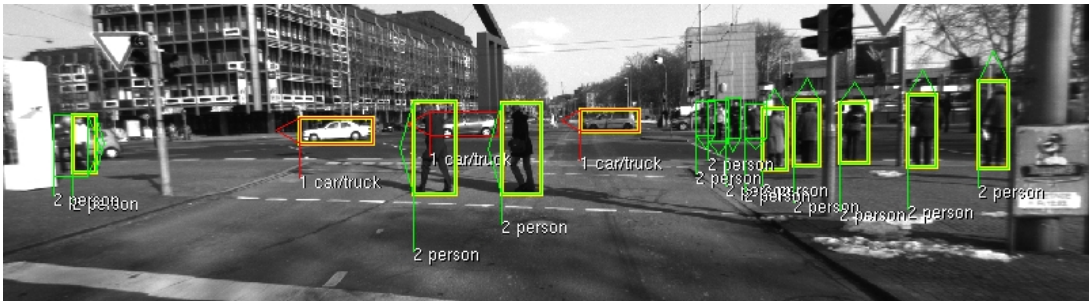


Figura 21. Imagen del dataset Karlsruhe.

- **Dataset 9 (Boxy** [38]**):**

  Consiste en un dataset desarrollado por Bosch, el cual contiene alrededor de 2 millones de vehículos etiquetados. Dicho etiquetado se realiza mediante bounding box tanto en 2D como en 3D, lo que nos permite una detección más precisa de los vehículos.

  El tipo de etiquetado utilizado es del tipo BBox, es decir, se emplean bounding box que engloban las etiquetas de salida (Fig. 22).

  Datos generales del dataset:

  - 200.000 imágenes.
  - 1.900.000 vehículos detectados.
  - Imágenes en resolución de 5 Megapíxeles.
  - Todo tipo de condiciones climáticas: Nublado, lloviendo, noche, día, soleado…
  - Todo tipo de situaciones: Poco trafico, Centros urbanos, atascos…

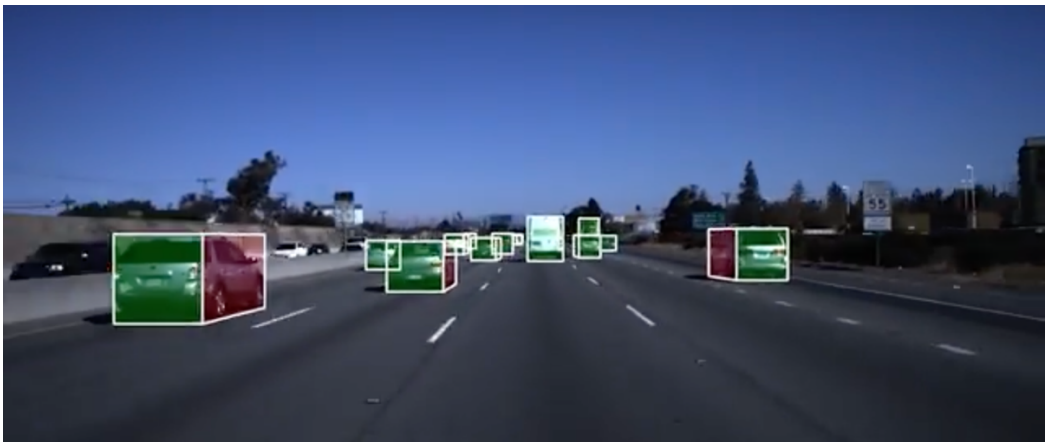


Figura 22. Imagen del dataset Boxy.

Comparación de todos los dataset analizados:

En las siguientes tablas (Tabla 7 y Tabla 8), mostraremos una comparativa de todos los datasets analizados para su uso con nuestro modelo de detección de vehículos. En ella se compararán las características principales que sean más relevantes a la hora de decantarnos por uno u otro.

	CrowdAI	Autti	Cityscape	Apolloscape	BDD100K
# Imágenes	9.423	15.000	5.000	150.000	100.000
Resolución Imágenes	1920x1200	1920x1200	1024x2048	3384x2710	1280x720
Múltiples Ciudades	SI	NO	SI	SI	SI
Múltiples Meteorologías	NO	NO	SI	NO	SI
Múltiples horas del día	SI	NO	NO	NO	SI
Múltiples tipos de escenas	SI	SI	NO	NO	SI
Tipo de etiquetado	BBox	BBox	Semantic Segmentation	Semantic Segmentation	Semantic Segmentation

Tabla 7. Comparación Dataset - Parte 1

	Detrac	Nyc3Dcars	Karlsruhe	Boxy
# Imágenes	140.131	-	1.020	200.000
Resolución Imágenes	960x540	Varias	1344x391	2464x2056
Múltiples Ciudades	-	NO	-	SI
Múltiples Meteorologías	SI	SI	NO	SI
Múltiples horas del día	SI	SI	SI	SI
Múltiples tipos de escenas	NO	SI	SI	SI
Tipo de etiquetado	BBox	Semantic Segmentation	BBox	BBox

Tabla 8. Comparación Dataset - Parte 2

5.1.1 Dataset escogido

Hemos decidido usar el dataset 9, *Boxy*, ya que, las imágenes que nos proporciona están realizadas en autopistas y este tipo de vía es el que nos interesa para hacer la estimación del tráfico una vez implementado nuestro modelo. Otro de los motivos por el cual decidimos usar, es la variedad tanto de climatologías, como de franjas horarias en las que las imágenes han sido realizadas.

Por último, es un dataset cuyas imágenes tienen el tipo de etiquetado mediante *Bounding Box*, característica que para nosotros es fundamental, ya que, nuestro modelo realizará la detección de vehículos basándose en este tipo de etiquetado.

5.2 Preprocesamiento de datos

Una vez elegido el dataset con el que entrenaremos nuestro modelo, debemos adaptar su conjunto de datos para que sea adecuado al formato de entrada de los datos en nuestro modelo.

En primer lugar, observamos que los datos recibidos durante la descarga del dataset *Boxy*, son un conjuntos de imágenes en formato .PNG y un único fichero .JSON en el que encontramos las anotaciones de cada una de las imágenes.

Como punto de partida, tendremos en cuenta que nuestro modelo YOLO requiere de dos tipos de ficheros para realizar su entrenamiento. En primer lugar, una carpeta llamada *images* la cual contendrá las imágenes y otra carpeta llamada *annots* la cual contendrá un fichero .XML por cada imagen con el mismo nombre que su imagen correspondiente.

- **Ficheros .PNG (Imágenes)**

Las imágenes en formato .PNG utilizadas en el entrenamiento de nuestro modelo pueden ser de cualquier resolución, pero hay que tener en cuenta que cuanto mayor sea la resolución, mayor tiempo de computo llevará a nuestro modelo ser entrenado.

Con la descarga de las imágenes del dataset *Boxy,* obtenemos un conjunto de imágenes con una resolución excesivamente alta (2464 x 2056). Por ello, creamos un script en Python llamado *allresize.py* mediante el cual reducimos la resolución de las mismas una cuarta parte la original (616 x 514).

Es un script muy sencillo (Fig. 23), en el cual, nos ayudamos de la librería PIL para realizar la conversión de todas la imágenes contenidas en la carpeta que nosotros le indiquemos a la resolución fija (616 x 514), guardando cada imagen nueva en el directorio que nosotros mismos le señalemos como salida.

```
from PIL import Image
import os, sys

path = "/Users/Alejandro/Desktop/Imagenes/"
dirs = os.listdir(path)
outdir = "/Users/Alejandro/Desktop/ImResize/"

for item in dirs:
    if os.path.isfile(path+item):
        im = Image.open(path+item)
        f, e = os.path.splitext(path+item)
        imResize = im.resize((616,514), Image.ANTIALIAS)
        imOut = item
        imResize.save(outdir + item)

print("PROCESO FINALIZADO")
```

Figura 23. Script allresize.py

Este proceso ayudará notablemente al modelo a la hora de realizar el entrenamiento con las imágenes que le aportemos, ya que, reducir su resolución disminuirá notablemente su tiempo de entrenamiento.

- **Ficheros .XML (Anotaciones)**

Al realizar la descarga del fichero .JSON del dataset *Boxy,* nos encontramos con un fichero en el cual están todas las anotaciones de todas las imágenes que contiene el dataset. Esto implica que es un fichero muy grande y, además, ninguno de los elementos contenidos en él presenta un formato legible a primera vista (Fig. 24).

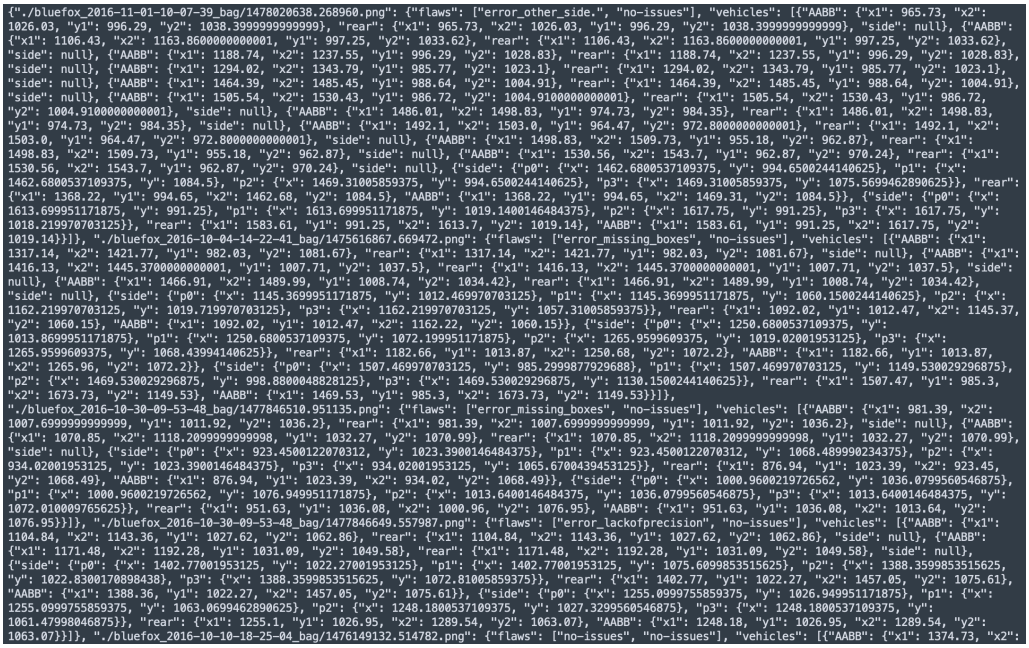
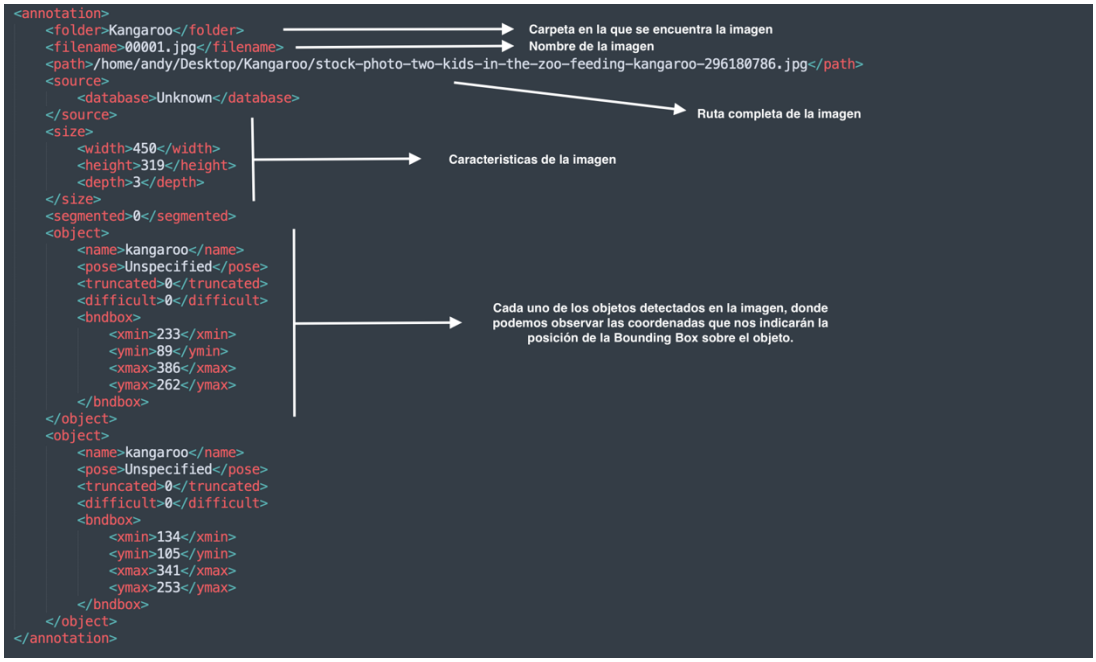


Figura 24. Archivo .JSON - Boxy dataset

Viendo la complejidad de analizar el archivo .JSON para extraer las anotaciones de las imágenes y adaptar cada una de estas al formato .XML de YOLO, se decide por automatizar este proceso creando un script en Python llamado Split.py, mediante el cual, se obtendrá la anotación de la imagen deseada transformándola a .XML y cumpliendo su estructura.

La estructura de cada uno de los ficheros .XML empleados para el entrenamiento será el siguiente (Fig. 25):




```
<annotation>
    <folder>Kangaroo</folder>
    <filename>00001.jpg</filename>
    <path>/home/andy/Desktop/Kangaroo/stock-photo-two-kids-in-the-zoo-feeding-kangaroo-296180786.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>450</width>
        <height>319</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>kangaroo</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>233</xmin>
            <ymin>89</ymin>
            <xmax>386</xmax>
            <ymax>262</ymax>
        </bndbox>
    </object>
    <object>
        <name>kangaroo</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>134</xmin>
            <ymin>105</ymin>
            <xmax>341</xmax>
            <ymax>253</ymax>
        </bndbox>
    </object>
</annotation>
```

Figura 25. Estructura .xml YOLO

El script *Split.py* comienza con la creación de una lista de los nombres de aquellas imágenes de las cuales queremos obtener sus anotaciones para el posterior entrenamiento (Fig. 26).

```
import json
import os, sys
import xml.etree.cElementTree as ET

print(" Split a JSON file ")

archivo = str(raw_input("Escribe el nombre de el fichero a abrir: "))

print("Procesando archivo " +archivo+ " .....")

# Indicamos la ruta donde se encuentran las imagenes del dataset
path = "/Users/Alejandro/Desktop/ImResize/"
imagenes = os.listdir(path)
```

Figura 26. Script Split.py parte 1

De esta manera, el siguiente paso para el script es leer el fichero .JSON comprobando si el nombre de la anotación leída se corresponde con alguno de los nombres de las imágenes (Fig. 27).

```
with open(archivo) as file:

    data = json.load(file)

    for img in data:

        # Las variable cont y num las usamos unicamente para acceder a aquellas imagenes que
        # tienen BBox en 3D.
        cont = 0
        num = 0

        # Obtenemos el nombre de la imagen que estamos tratando, para luego anadirlo al .xml
        nombre = str(img)
        nombreFile = nombre[34:51]
        nomFinal = nombreFile+".png"

        # Si el .json a analizar es de una imagen que esta en nuestro dataset, lo analizamos y
        # creamos su .xml, en caso contrario no lo analizamos ni creamos su .xml.
        if nomFinal in imagenes:


            # Creacion del XML

            annotation = ET.Element("annotation")
            folder = ET.SubElement(annotation, "folder")
            folder.text = "Vehiculos"
            filename = ET.SubElement(annotation, "filename")
            filename.text = nomFinal
            source = ET.SubElement(annotation, "source")
            database = ET.SubElement(source, "database")
            database.text = "Unknown"
            size = ET.SubElement(annotation, "size")
            width = ET.SubElement(size, "width")
            width.text = "616"
            #width.text = "2464"
            height = ET.SubElement(size, "height")
            height.text = "514"
            #height.text = "2056"
            depth = ET.SubElement(size, "depth")
            depth.text = "3"

            # Iteramos por los datos de cada coche en cada imagen.
            for coche in data[img]["vehicles"]:
```

Figura 27. Script Split.py parte 2

En caso de que corresponda con el nombre de alguna de las imágenes, se tratará esa anotación convirtiéndola al formato .XML con la estructura necesaria. Por otro lado, si la anotación leída no corresponde con ninguna de la imágenes, no la trataremos y pasaremos a leer la siguiente anotación.

Para la creación del nuevo .XML, lo que haremos será iterar por cada uno de los objetos de la imagen (Fig. 28), obteniendo las coordenadas de sus Bounding Box y adaptándolas a la estructura de .XML que requiere nuestro modelo para poder entrenar. Hay partes de la anotación en .JSON que no se han tenido en cuenta a la hora de realizar la transformación, ya que, son irrelevantes y no aportan ninguna información útil para realizar el entrenamiento del modelo.

```
# Iteramos por los datos de cada coche en cada imagen.
for coche in data[img]["vehicles"]:

    # Creamos dos vectores, uno para todos los puntos x del coche en la imagen,
    # y otro para todos los puntos y del coche en la imagen.
    vectorX = []
    vectorY = []

    # En el caso de que el coche tenga BBox en 3D, anadimos los puntos de las dos
    # ventanas que generan el 3D a los vectores.
    if (coche["side"] != None):

        num = cont

        for p in data[img]["vehicles"][num]["side"]:
            vectorX.append(data[img]["vehicles"][num]["side"][p]["x"])
            vectorY.append(data[img]["vehicles"][num]["side"][p]["y"])

        vectorX.append(coche["AABB"]["x1"])
        vectorX.append(coche["AABB"]["x2"])
        vectorY.append(coche["AABB"]["y1"])
        vectorY.append(coche["AABB"]["y2"])

        x1 = (min(vectorX)/4)
        y1 = (min(vectorY)/4)
        x2 = (max(vectorX)/4)
        y2 = (max(vectorY)/4)

    # Caso en el que el coche tiene una BBox en 2D.
    else:

        x1 = ((coche["AABB"]["x1"])/4)
        y1 = ((coche["AABB"]["y1"])/4)
        x2 = ((coche["AABB"]["x2"])/4)
        y2 = ((coche["AABB"]["y2"])/4)

    # Seguimos con la creacion del .XML
    object = ET.SubElement(annotation, "object")
```

Figura 28. Script Split.py parte 3

Para la creación del .XML, se ha utilizado la librería de Python ET, la cual nos ha permitido ir creando cada uno de los elementos del .XML con el nombre y jerarquía que nosotros hemos deseado (Fig. 29). Esta librería funciona empleando una estructura en forma de árbol, lo que nos permite crear de manera intuitiva y sencilla el .XML.

```
    name = ET.SubElement(object, "name")
    name.text = "vehiculo"

    pose = ET.SubElement(object, "pose")
    pose.text = "Unspecified"

    truncated = ET.SubElement(object, "truncated")
    truncated.text = "0"

    difficult = ET.SubElement(object, "difficult")
    difficult.text = "0"

    bndbox = ET.SubElement(object, "bndbox")

    xmin = ET.SubElement(bndbox, "xmin")
    xmin.text = str(x1)

    ymin = ET.SubElement(bndbox, "ymin")
    ymin.text = str(y1)

    xmax = ET.SubElement(bndbox, "xmax")
    xmax.text = str(x2)

    ymax = ET.SubElement(bndbox, "ymax")
    ymax.text = str(y2)

    # El contador se una para llevar en cuenta el numero de coche de la imagen,
    # para que en caso de que sea con una BBox en 3D, podamos acceder a sus datos.
    cont = cont + 1

# Finalizamos la creacion del .XML
arbol = ET.ElementTree(annotation)
nombreXml = nombreFile+".xml"
ruta = "/Users/Alejandro/Desktop/Archivos_XML/"+nombreXml
arbol.write(ruta)
```

Figura 29. Script Split.py parte 4

Una vez realizado todo el proceso de transformación mediante el script Split.py, obtendremos la salida de las anotaciones de cada una de las imágenes deseadas con el formato .XML (Fig. 30).

```
<annotation>
    <folder>Vehiculos</folder>
    <filename>1475273024.455387.png</filename>
    <size>
        <width>616</width>
        <height>514</height>
        <depth>3</depth>
    </size>
    <object>
        <bndbox>
            <xmin>304.9075</xmin>
            <ymin>250.105</ymin>
            <xmax>307.98</xmax>
            <ymax>255.345</ymax>
        </bndbox>
    </object>
    <object>
        <bndbox>
            <xmin>302.6625</xmin>
            <ymin>250.2625</ymin>
            <xmax>306.01</xmax>
            <ymax>256.4475</ymax>
        </bndbox>
    </object>
</annotation>
```

Figura 30. Salida .XML - Split.py

5.3 Entrenamiento

El entrenamiento se ha realizado en un entorno virtual *Google Colab*, ya que, es la mejor forma de obtener una mayor capacidad de computo respecto a nuestro ordenador personal. Este entorno nos permite hacer uso de GPUs para el entrenamiento de nuestro modelo, lo que agiliza notablemente el proceso, ahorrándonos gran cantidad de tiempo.

Para comenzar nuestro entrenamiento, lo primero que debemos hacer es importar nuestro proyecto a *Google Colab*. Esto se puede realizar conectando este entorno a nuestro propio *Google Drive* o incluso podemos clonar repositorios para hacer uso de estos.

Para realizar el entrenamiento de nuestro modelo YOLO, vamos a usar los datos del dataset *Boxy*. Estos datos, como hemos visto en el apartado anterior, han sido previamente preprocesados con el fin de adaptarlos al tipo de ficheros y estructuras que requiere YOLO para poder realizar un correcto entrenamiento.

Este conjunto de datos lo tenemos que dividir en varias partes:

- Una carpeta que contenga las imágenes que van a ser usadas en el entrenamiento del modelo. En nuestro caso, el numero de imágenes usadas para realizar el entrenamiento es de 100.
- Una carpeta que contenga las anotaciones en formato VOC (.XML) de las imágenes de entrenamiento, las cuales, tendrán el mismo nombre que las imágenes a las cuales pertenecen.
- Una carpeta que contenga las imágenes que van a ser usadas en la validación del modelo. En nuestro caso, usaremos 10.
- Una carpeta que contenga las anotaciones en formato VOC (.XML) de las imágenes de validación, las cuales, tendrán el mismo nombre que las imágenes a las cuales pertenecen.

De esta forma, tenemos organizados nuestros datos para realizar el entrenamiento del modelo YOLO. También, cabe destacar que, en el caso de que no se creen carpetas con las imágenes y anotaciones para la validación, nuestro modelo las generará dividiendo el total de datos para el entrenamiento en dos partes.

Un 80% del conjunto de los datos de entrenamiento seguirán siendo de entrenamiento, mientras que, el 20% restante pasará a ser parte del conjunto de datos de validación. De esta forma, nos aseguramos siempre de que nuestro modelo dispone de datos de entrenamiento, así como de validación.

Tanto las carpetas de entrenamiento, como de validación, deben ser indicadas en nuestro archivo de configuración (Fig. 31), en el cual también podremos elegir si queremos realizar un entrenamiento haciendo uso de GPU o de otros parámetros como podrían ser el tamaño de batch, la tasa de aprendizaje, el número de épocas…

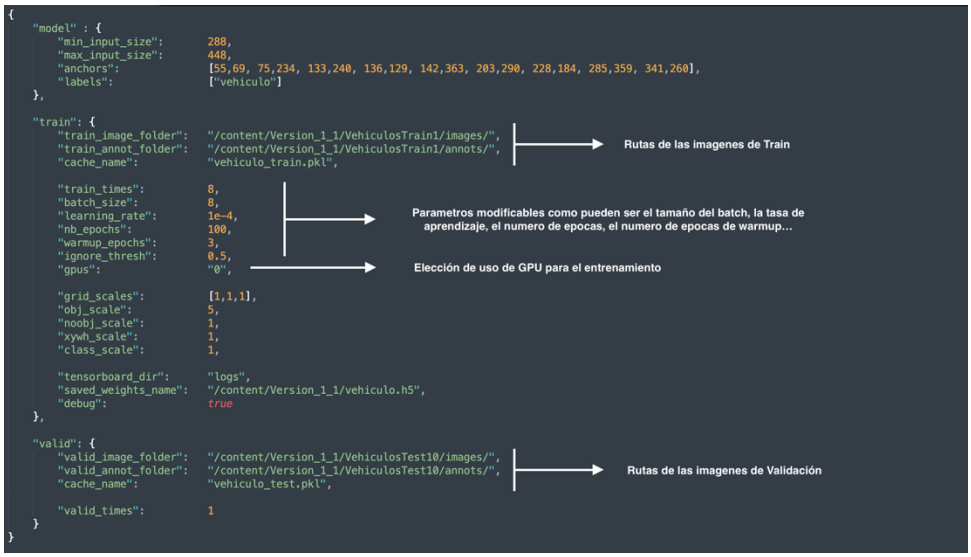


Figura 31. Archivo de configuración YOLO

El entrenamiento consta de un número máximo de épocas prefijado en nuestro archivo de configuración, en el cual, también podremos hacer uso del parámetro *warmup_epochs*. Al realizar un entrenamiento, si el error obtenido durante una época no disminuye durante el número fijado en este parámetro, la ejecución termina automáticamente dándolo por finalizado.

Un parámetro importante al hacer uso de nuestro modelo YOLO, es el parámetro *gpus*, en el cual tenemos la opción de fijar a “1” o a “0”. Cuando se fija a “1” significará que el entrenamiento del modelo no hará uso de GPUs, mientras que si se fija a “0”, si que hará uso de las GPUs para llevar a cabo el entrenamiento. Es altamente recomendable hacer uso de GPU en el entrenamiento del modelo, ya que, reduciremos en gran medida el tiempo necesario de computo para el entreno.

Hay que tener en cuenta también que para realizar el entrenamiento es necesario un archivo .h5 con los pesos iniciales del modelo. De esta manera, el modelo dispondrá de unos pesos iniciales los cuales irán cambiando conforme se realice el entrenamiento.

Una vez configurado nuestro modelo para el entrenamiento, comenzaremos a entrenarlo. La ejecución del mismo, nos irá mostrando el resultado en cada una de las épocas realizadas (Fig. 32). Nos indicará el tiempo que ha empleado en cada época, el loss global, así como el loss en tres capas distintas de YOLO.

```
Epoch 00017: loss improved from 10.38438 to 9.89909, saving model to /content/Version_1_1/vehiculo.h5
Epoch 18/103
 - 34s - loss: 9.8056 - yolo_layer_1_loss: 0.1885 - yolo_layer_2_loss: 0.4965 - yolo_layer_3_loss: 9.1206

Epoch 00018: loss improved from 9.89909 to 9.80556, saving model to /content/Version_1_1/vehiculo.h5
Epoch 19/103
 - 34s - loss: 9.5512 - yolo_layer_1_loss: 0.0028 - yolo_layer_2_loss: 0.4764 - yolo_layer_3_loss: 9.0720

Epoch 00019: loss improved from 9.80556 to 9.55115, saving model to /content/Version_1_1/vehiculo.h5
Epoch 20/103
 - 37s - loss: 9.5903 - yolo_layer_1_loss: 0.0034 - yolo_layer_2_loss: 0.5506 - yolo_layer_3_loss: 9.0363
```

Figura 32. Entrenamiento de YOLO – épocas

Al llegar al número máximo de épocas o cuando se cumple el parámetro *warmup_epochs*, el entrenamiento finaliza. De esta manera, obtendremos los pesos del modelo entrenado un archivo .h5 indicado en el parámetro de configuración *saved_weight_name*.

Como podemos observar la perdida (loss) durante el entrenamiento se ve reducida rápidamente a medida que avanzamos en las épocas (Gráfica 2). A medida que estas pasan, la pérdida se va estabilizando, hasta que llega un momento en el cual no consigue mejorar durante tres épocas (es el valor que tenemos en el parámetro *warmup_epoch*) y, por lo tanto, finaliza el entrenamiento.

Gráfica 4. Loss entramiento YOLO

Una vez efectuado el entrenamiento de nuestro modelo, podremos realizar predicciones pasándole tanto imágenes como vídeos. Debemos aclarar que, el modelo ha sido entrenado para la detección únicamente de vehículos, sin distinguir tipos de los mismos.

La configuración de los parámetros, en el archivo *config.json*, usados para realizar el entrenamiento se pueden ver en la Tabla 9.

PARÁMETRO	VALOR	DESCRIPCIÓN
Train_times	8	Número de ciclos en el conjunto de entrenamiento.
Batch_size	8	Número de imágenes a leer en cada lote.
Learning_rate	1e-4	Tasa de aprendizaje de nuestra red neuronal.
Nb_epoch	100	Número de épocas a realizar en el entrenamiento.
Warmup_epochs	3	Número de épocas en las que se permite que el valor de *loss* no mejore.
Ignore_thresh	0.5	Valor de IoU (Intersection over Union) mínimo.
Gpus	0	Uso de GPU para el entrenamiento del modelo. Si su valor es “0” haremos uso de GPU. Si su valor es “1” no haremos uso de GPU.

Tabla 9. Parámetros de entrenamiento de YOLO

5.4 Detección de vehículos

Con nuestro modelo ya entrenado, este ya es capaz de realizar la detección de vehículos situando Bounding Boxes sobre cada uno de los vehículos detectados. La detección se puede realizar tanto para imágenes, como para vídeos.

Al realizar la detección, los ficheros aportados como entrada (imágenes/vídeo) no serán modificados en ningún momento, sino que se creará un nuevo fichero de salida (imagen/vídeo) en el que obtendremos los resultados.

Para realizar la detección, únicamente es necesario indicar la ruta del fichero de entrada, así como la ruta y el nombre del fichero de salida, el cual se creará automáticamente.

A continuación, se muestran varios resultados (Fig. 33 y Fig. 34) de realizar la detección de vehículos.

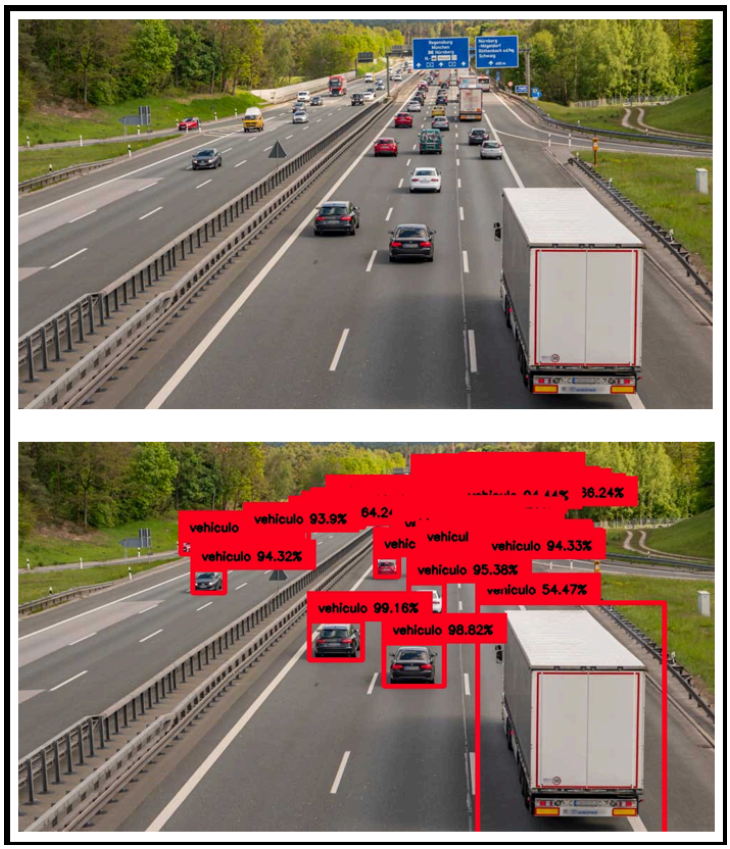


Figura 33. Detección de vehículos - Ejemplo 1

Figura 34. Detección de vehículos - Ejemplo 2

5.5 Conteo de vehículos

El conteo de vehículos es una parte fundamental en nuestro trabajo, ya que, es la parte que nos permitirá estimar el flujo de tráfico para una zona en un determinado tiempo. Para ello, necesitamos de dos partes ya implementadas en el proyecto:

- En primer lugar, debemos disponer de un modelo de detección de vehículos ya entrenado, capaz de mostrarnos las Bounding Box sobre los vehículos detectados en la imagen/vídeo que le aportemos. En este caso, usaremos nuestro modelo de detección de vehículos ya entrenado YOLO.
- En segundo lugar, debemos disponer de un algoritmo de tracking de vehículos con el fin de poder estimar en el tiempo la ubicación de cada uno de ellos. En este caso, haremos uso de nuestro algoritmo de tracking de vehículos SORT.

Para el desarrollo de esta parte, nos hemos encontrado con la dificultad de que la mayoría de los algoritmos de tracking implementados en Python, están desarrollados para ser usados mediante modelos de detección implementados en *Tensorflow* o en *OpenCV*. Nuestro principal problema es que nuestro modelo de detección YOLO, está

implementado en el lenguaje *Keras*, por lo que, era imposible su utilización con los algoritmos de tracking vistos.

Es por ello que, hemos tenido que adaptar el algoritmo de tracking para que pueda ser plenamente funcional con nuestro modelo Keras-YOLO, de forma que hemos realizado los siguiente pasos con el fin de conseguir su funcionamiento:

i. Leemos el vídeo de entrada para el cual queremos realizar el conteo de vehículos. Para ello, hacemos uso de la librería *openCV*, mediante la cual leeremos el vídeo frame a frame para poder ir tratándolos uno a uno.
ii. Para cada frame, nuestro modelo Keras-YOLO nos devuelve las Bounding Box de cada uno de los vehículos detectados en el frame, así como la precisión de la detección (valor entre 0 y 1).
iii. Para cada Bounding Box perteneciente a un vehículo, obtenemos las coordenadas que nos permiten dibujar la Bounding Box sobre el vehículo detectado.
iv. Estas coordenadas son pasadas al algoritmo de tracking SORT junto con la precisión de la detección.
v. En el algoritmo SORT trataremos los vehículos detectados con una precisión mayor que un valor preestablecido, obteniendo la asignación de un ID para cada uno de estos vehículos, así como su seguimiento y posición en cada uno de los frames siguientes.
vi. Por último, situaremos una línea virtual en el vídeo, la cual usaremos con el fin de realizar el conteo de los vehículos trackeados. Es decir, todo vehículo trackeado que cruce esa línea será sumado a un contador, el cual llevará la cuenta de los vehículos (Fig. 35).

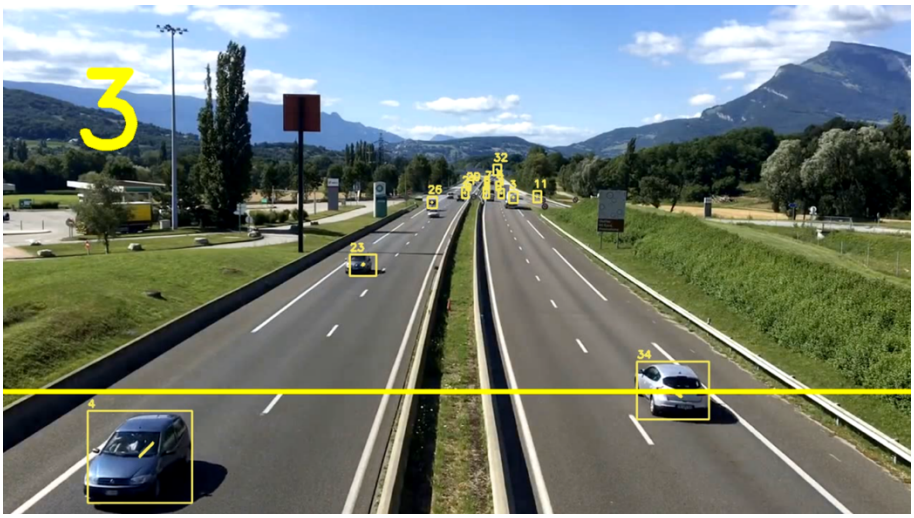


Figura 35. Conteo de vehículos

5.6 Resultados experimentales

5.6.1 Evaluación YOLO mediante Loss

Comenzaremos la evaluación de nuestro modelo YOLO calculando el valor de *loss* tanto para el conjunto de *train,* como para el conjunto de validación haciendo uso de cuatro configuraciones diferentes. Nuestro conjunto de *train,* está formado por 100 imágenes junto con sus anotaciones, mientras que, nuestro conjunto de validación está formado por un conjunto de 10 imágenes junto a sus anotaciones.

Para comparar el desempeño del modelo en ambos conjuntos, realizaremos una gráfica en la que se mostrará el valor de *loss* del modelo en ambos conjuntos para cada una de las etapas que han sido empleadas para el entrenamiento.

- **Configuración 1**

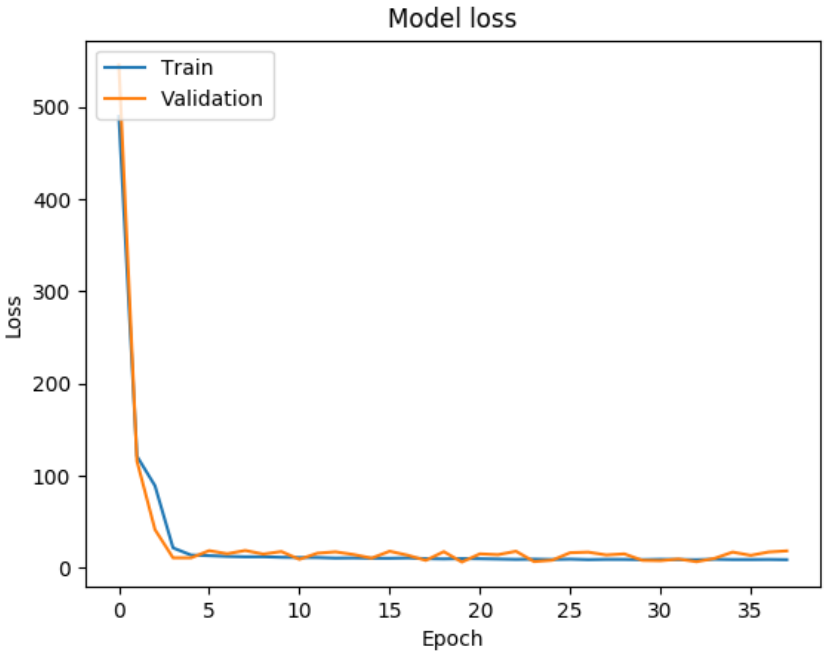
En esta configuración 1, los parámetros utilizados para la realización de las pruebas son los que podemos ver en la tabla 10.

PARÁMETRO	VALOR
Train_times	8
Batch_size	8
Learning_rate	1e-4
Nb_epoch	100
Warmup_epochs	3
Ignore_thresh	0.5
Gpus	0

Tabla 10. Parámetros de YOLO - Configuración 1

Después de las pruebas realizadas, obtenemos un parámetro de *loss* de 9.80 para el conjunto de *train* y de 16.41 para el conjunto de *validación*.

Como podemos ver (Gráfica 5), los valores de *loss* comienzan siendo bastante altos, pero con el paso de las épocas se van reduciendo notablemente hasta un punto en el cual se estabilizan. Podemos observar que tanto en *train*, como en *validación* obtenemos unos valores muy parecidos, por lo que, podemos decir que disponemos de un modelo capaz de generalizar de una manera bastante notable.



Gráfica 5. Loss YOLOv3 - Configuración 1

- **Configuración 2**

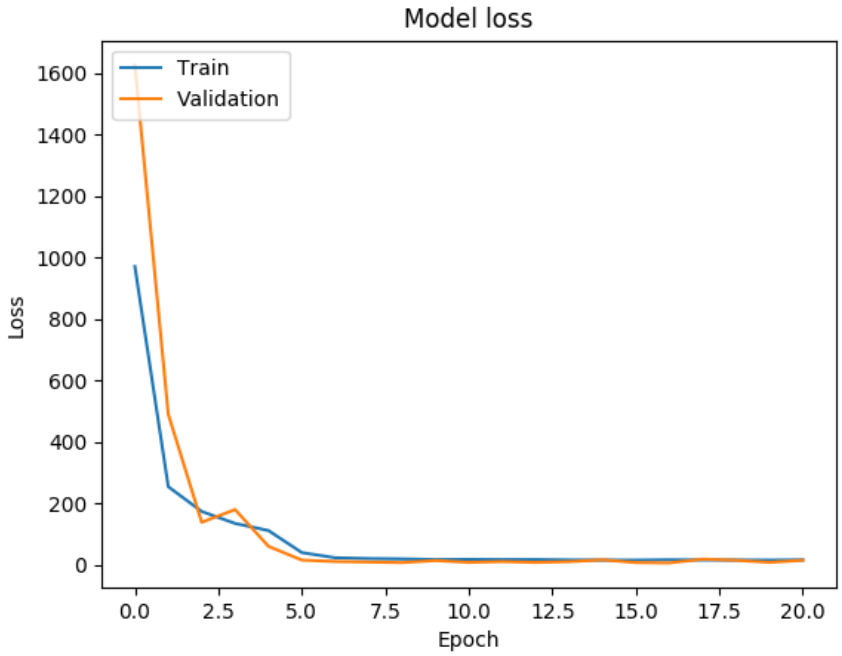
En esta configuración 2, los parámetros utilizados para la realización de las pruebas son los que podemos ver en la tabla 11. En este caso, hemos aumentado el tamaño de *batch*, así como el *Learning_rate* y el parámetro *warmup_epochs* respecto a la primera configuración.

PARÁMETRO	VALOR
Train_times	8
Batch_size	16
Learning_rate	1e-2
Nb_epoch	100
Warmup_epochs	5
Ignore_thresh	0.5
Gpus	0

Tabla 11. Parámetros de YOLO - Configuración 2

Después de las pruebas realizadas, obtenemos un parámetro de *loss* de 15.00 para el conjunto de *train* y de 22.23 para el conjunto de *validación*.

En este caso (Gráfica 6), vemos que el uso de una mayor tasa de aprendizaje nos permite avanzar más rápidamente en la reducción del *loss* en las primeras épocas, sin embargo, a medida que avanzamos en ellas podemos observar como el valor de *loss* se empieza a estancar. De esta forma, al llegar al final podemos observar como el menor valor de *loss* alcanzado es mayor (tanto en *train* como en *validación*) que los obtenidos para la configuración 1, por lo que se descarta el uso de esta configuración.



Gráfica 6. Loss YOLOv3 - Configuración 2

- **Configuración 3**

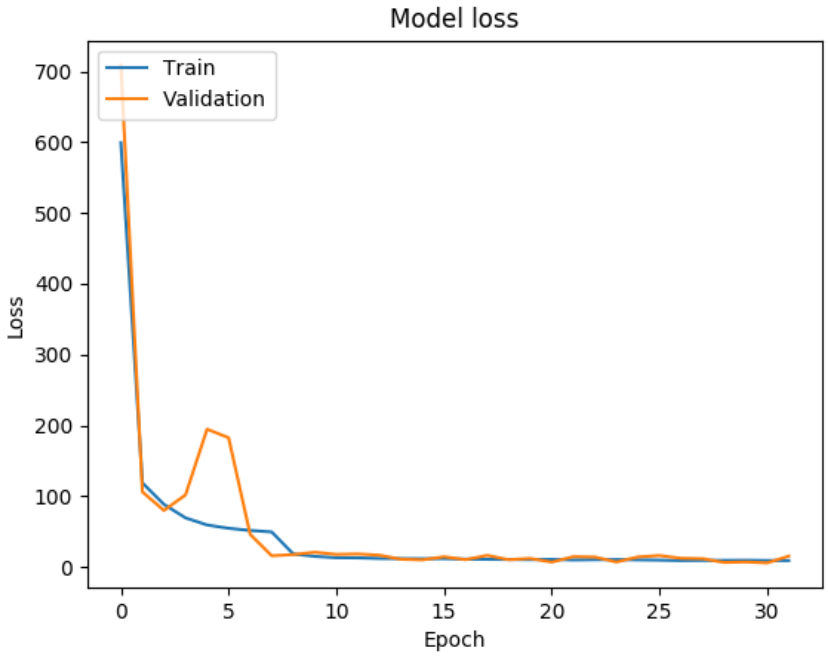
En esta configuración 3, los parámetros utilizados para la realización de las pruebas son los que podemos ver en la tabla 12. En este caso, usaremos un tamaño de *batch* y de *Learning_rate* igual a los empleados en la configuración 1, mientras que el parámetro *warmup_epochs* lo aumentaremos.

PARÁMETRO	VALOR
Train_times	8
Batch_size	8
Learning_rate	1e-4
Nb_epoch	100
Warmup_epochs	8
Ignore_thresh	0.5
Gpus	0

Tabla 12. Parámetros de YOLO - Configuración 3

Después de las pruebas realizadas, obtenemos un parámetro de *loss* de 8.99 para el conjunto de *train* y de 15.27 para el conjunto de *validación*.

En este caso, los resultados son muy parecidos a los que hemos obtenido en la configuración 1, aunque, como podemos ver en la gráfica 7 los valores son más inestables durante la ejecución.



Gráfica 7. Loss YOLOv3 – Configuración 3

- **Configuración 4**

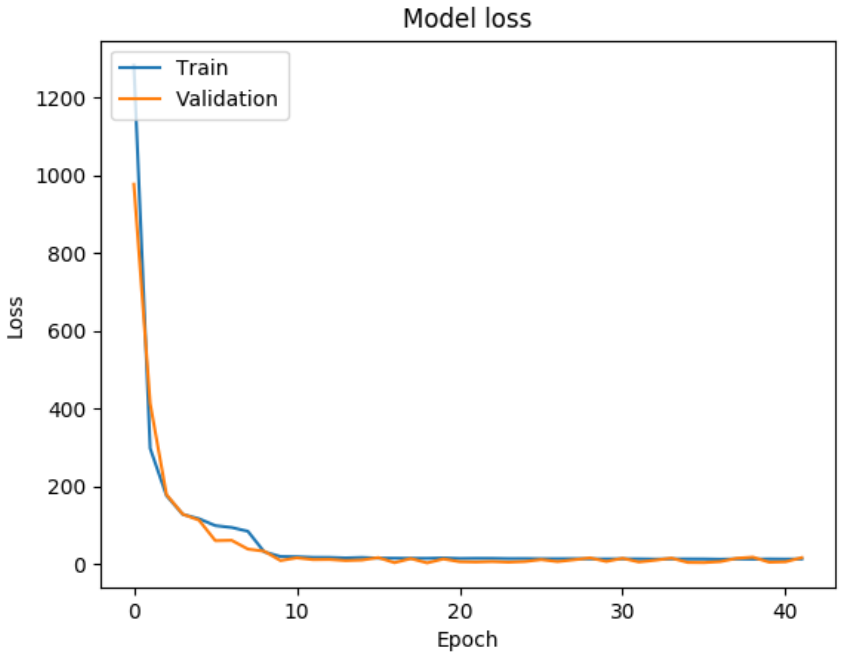
En esta configuración 4, los parámetros utilizados para la realización de las pruebas son los que podemos ver en la tabla 13. En este caso los parámetros a usar son los mismos que en la configuración 3, excepto que usaremos un tamaño de *batch* de 16.

PARÁMETRO	VALOR
Train_times	8
Batch_size	16
Learning_rate	1e-4
Nb_epoch	100
Warmup_epochs	8
Ignore_thresh	0.5
Gpus	0

Tabla 13. Parámetros de YOLO - Configuración 4

Después de las pruebas realizadas, obtenemos un parámetro de *loss* de 13.87 para el conjunto de *train* y de 17.42 para el conjunto de *validación*.

En este caso (Gráfica 8), el aumento del tamaño del *batch* hace que nuestro valor de *loss* comience en unas cifras mayores, sin embargo, rápidamente desciende obteniendo unos resultados bastante buenos.



Gráfica 8. Loss YOLOv3 – Configuración 4

- **Elección de configuración:**

Después de las pruebas realizadas con las distintas configuraciones, vemos que el parámetro más relevante es *Learning_rate*, donde un valor muy grande del mismo hace que el valor de *loss* descienda rápidamente durante las primeras épocas, pero luego se estanque de manera que no consigue avanzar. De manera contraria, un valor muy pequeño hace que el modelo sea incapaz de rebajar el valor de *loss*, impidiendo que avance desde las primera épocas.

El tamaño de batch, hace que los valores de *loss* en las primera épocas sean grandes, aunque a medida que avanzamos en las épocas este valor se reduce notablemente, por lo que, no afecta demasiado al rendimiento que llega a conseguir el modelo. Sin embargo, hemos visto que cuanto menor sea este parámetro unos mejores valores de *loss* llegamos a conseguir.

En definitiva, la configuración 3 es la que un menor valor de loss nos ha dado, por lo que, es la configuración elegida para continuar con la implementación del proyecto.

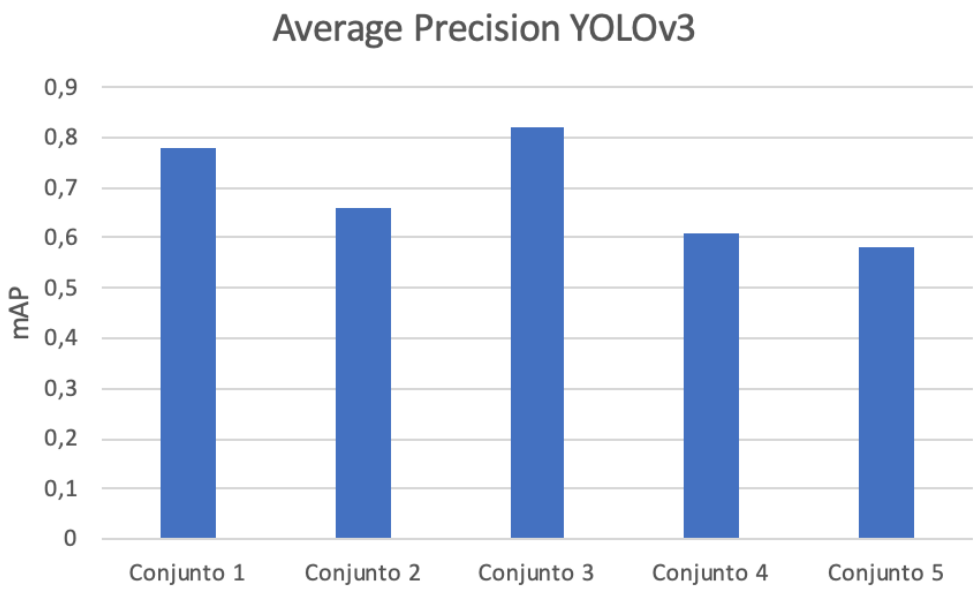
5.6.2 Evaluación YOLO mediante mAP

Tras la elección de la mejor configuración para nuestro modelo, también haremos uso de la métrica *mAP*, explicada en la sección 3.5, donde después de realizar el entrenamiento del modelo, se pasarán distintos conjuntos de imágenes en los cuales para cada uno de ellos obtendremos un valor *mAP* que nos dirá la precisión media de nuestro modelo.

Estos conjuntos de imágenes, se han obtenido del dataset Boxy, el cual dispone de varios paquetes de imágenes en distintas situaciones de tráfico y lugares.
Los resultado obtenido para las pruebas se pueden ver en la tabla 14.

Dataset	Valor mAP obtenido
Conjunto 1 (bluefox_2016-10-10-15-17-24_bag.zip)	0.78
Conjunto 2 (bluefox_2016-09-27-14-43-04_bag.zip)	0.66
Conjunto 3 (bluefox_2016-10-26-13-00-25_bag.zip)	0.82
Conjunto 4 (bluefox_2016-10-30-10-01-47_bag.zip)	0.61
Conjunto 5 (bluefox_2016-11-03-15-40-30_bag.zip)	0.58

Tabla 14. Valores mAP sobre distintos conjuntos de datos - YOLOv3



Gráfica 9. Mean Average Precision YOLO sobre distintos conjuntos de datos

Podemos observar (Gráfica 9) que, en todos ellos obtenemos unos valores similares de mAP, esto se debe a que, aunque los lugares donde han sido tomadas las imágenes son distintos en cada uno de los conjuntos, todas la imágenes son tomadas desde la cámara delantera de un vehículo, por lo que todas ellas comparten el mismo punto de vista.

Hay que añadir que la evaluación del parámetro mAP en otros dataset de vehículos es posible, sin embargo, es necesario que las anotaciones de la imágenes que componen a los dataset deben estar en formato VOC pascal, ya que este tipo de formato es el usado por nuestro modelo.

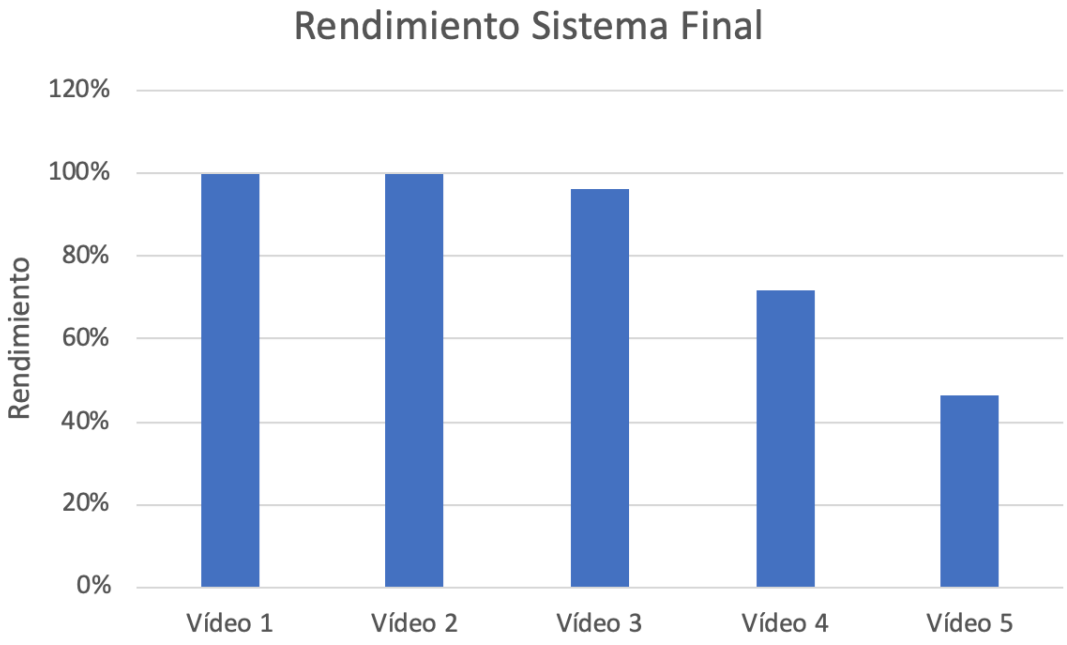
5.6.3 Evaluación sistema final

Por último, vamos a realizar la evaluación de nuestro sistema de conteo, es decir, vamos a comprobar que el sistema implementado es capaz de realizar un buen desempeño a la hora de realizar conteos de vehículos desde una posición fija.

Para ello vamos a hacer uso de 5 vídeos, en los cuales, vamos a contar de forma manual el número de vehículos que pasan por nuestra posición. Después, realizaremos el mismo conteo pero de forma automática, procesando el vídeo mediante nuestro sistema implementado.

Vídeo	Descripción	Conteo Manual	Conteo Automático	Rendimiento sistema
Vídeo 1	Vídeo propio, de día en la AP-15.	8	8	100%
Vídeo 2	Sacado de internet, de día y en ciudad.	4	4	100%
Vídeo 3	Sacado de internet, de día y en autopista.	26	25	96.15%
Vídeo 4	Sacado de internet, de día, en autopista y a cámara rápida.	53	38	71.69%
Vídeo 5	Sacado de internet, de noche y en autopista	13	6	46.15%

Tabla 15. Datos de conteo Sistema Final



Gráfica 10. Rendimiento sistema de conteo

A partir de los datos obtenidos (Tabla 15 y Gráfica 10), podemos observar que el rendimiento del clasificador varía según las situaciones en las que nos encontremos. Cuando evaluamos vídeos de día, el rendimiento suele ser bastante bueno, viendo que el sistema es capaz de detectar la mayoría de los vehículos de forma autónoma. Sin embargo, en situaciones con mala visibilidad como pueden ser lloviendo mucho o de noche a el sistema le cuesta mucho trabajo detectar los vehículos y, por lo tanto, contarlos.

Cuando las pruebas han sido realizadas con vídeos de noche, observamos que los coches que mejor detecta son aquellos que van inmediatamente delante de otro vehículo. Con ellos, podemos decir que gracias a la luz trasera extra que le aporta el vehículo hace que sea más fácil la detección del vehículo delantero para nuestro sistema.

Otro de los factores a tener en cuenta es la velocidad del vídeo. En vídeos a cámara rápida, es muy difícil que el sistema sea capaz de contar todos los vehículos, no porque no los detecte, sino porque a la hora de realizar el tracking vemos como a veces pierde el seguimiento de alguno de los vehículos.

En general, podemos decir que el rendimiento del sistema es bastante bueno para situaciones de día y con una velocidad normal. Esto es un punto positivo, ya que, pruebas como el vídeo a cámara rápida simplemente son retos a los que enfrentamos nuestro sistema para observar su rendimiento en situaciones inusuales.

Su mayor punto débil es el conteo en situaciones de tráfico nocturnas, por lo que, este sería un punto clave para mejorar en versiones futuras.

Por último, en las siguientes figuras (Fig. 36, Fig.37 y Fig. 38), correspondientes al primer vídeo, podemos observar una secuencia de cómo el detector ha realizado un conteo de vehículos en una autovía en buenas condiciones.

Figura 36. Secuencia de conteo de vehículos - Parte 1

Figura 37. Secuencia de conteo de vehículos - Parte 2

Figura 38. Secuencia de conteo de vehículos - Parte 3

CONCLUSIONES Y LÍNEAS FUTURAS

Viendo los resultados obtenidos, podemos llegar a la conclusión de que el rendimiento del sistema para la mayoría de las situaciones es bastante bueno, por lo que, actualmente podría ser un sistema funcional. Sin embargo, tanto la meteorología como las situaciones nocturnas influyen notablemente en el rendimiento del sistema.

El sistema desarrollado permite el conteo de vehículos en un determinado punto mediante la instalación de únicamente una cámara fija. Esto nos permite prescindir de otros sensores para realizar esta tarea, lo que implica un ahorro económico a la hora de implantar el sistema.

Tras ver los buenos resultados obtenidos a la hora de detectar vehículos en imágenes las cuales eran bastante diferentes a las usadas para el entrenamiento del modelo, podemos decir que la capacidad de generalización de nuestro sistema de detección de vehículos YOLO ha sido realmente buena. Aunque para determinadas situaciones que pueden ser complicadas, como imágenes nocturnas, el rendimiento baja notablemente.

Hemos de tener en cuenta que, uno de los mayores inconvenientes a la hora de realizar proyectos de detección de objetos mediante Deep Learning, es la necesidad de disponer de una GPU para poder realizar el entrenamiento de los modelos. Sin una GPU, se hace casi imposible realizar entrenamientos de los modelos, ya que, demora mucho tiempo en ello. Por este motivo, el uso de herramientas como Google Colab se convierte casi en obligatorio para afrontar un proyecto de estas características.

Este tipo de sistemas requieren de una gran cantidad de datos de entrenamiento, ya que normalmente una mayor cantidad de estos implicará la construcción de modelos fiables y con buenos resultados, sin embargo, la calidad de los datos también es un punto fundamental a la hora de obtener unos resultados más reales y representativos.

Por una parte, el desarrollo de este tipo de sistemas nos motiva en el sentido de que en implementaciones futuras se tendrá un conocimiento mayor sobre este tipo de sistemas, así como, una mayor capacidad de resolución de los problemas que puedan surgir.

Líneas futuras

El uso de un sistemas de detección de objetos como YOLO, nos ofrece la posibilidad de mejorar este proyecto de muchas formas:

- Ampliar la capacidad de nuestro modelo de detección, el cual puede ser capaz de detectar más de una clase de objetos. En nuestro caso, únicamente detecta vehículos, sin embargo, podría aprender a distinguir entre distintos tipos de vehículos (coches, motos, camiones, autobuses …).
- Mejorar la capacidad de tracking, el sistema podría ser capaz de detectar la velocidad de cada uno de los vehículos que son detectados, así como estimar si el flujo de tráfico es fluido o denso.
- Una de las líneas futuras que podría tener una buena aplicación, sería dotar al sistema de la capacidad de detectar si un vehículo está averiado en la calzada o se produce un accidente en la misma, esto se podría realizar ya que trackeamos cada vehículo que es detectado en nuestro vídeo.
- Realizar conteo de vehículos en tiempo real, mediante cámaras fijas situadas en autovías o autopistas.
- Mejora del rendimiento del sistema en situaciones nocturnas.

BIBLIOGRAFÍA

[1] "Inteligencia Artificial – Qué es y por qué es importante | SAS." [Online]. Available: https://www.sas.com/es_es/insights/analytics/what-is-artificial-intelligence.html.

[2] "Aprendizaje automático: Qué es y por qué es importante | SAS." [Online]. Available: https://www.sas.com/es_es/insights/analytics/machine-learning.html.

[3] "¿Qué es deep learning? | SAS." [Online]. Available: https://www.sas.com/es_es/insights/analytics/deep-learning.html.

[4] "Red neuronal artificial - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/Red_neuronal_artificial.

[5] "Los 10 usos más comunes en machine learning e inteligencia artificial." [Online]. Available: https://blogthinkbig.com/los-10-usos-mas-comunes-en-machine-learning-e-inteligencia-artificial.

[6] "Redes neuronales convolucionales - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016-Decem, pp. 779–788.

[8] "Qué es Machine Learning, cómo funciona y a qué se aplica | APD." [Online]. Available: https://www.apd.es/que-es-machine-learning/.

[9] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," 2016.

[10] "Historia del Deep Learning (1): tres etapas fundamentales - datahack, expertos en Big Data, el mejor máster." [Online]. Available: https://www.datahack.es/historia-deep-learning-etapas.

[11] "La IA ya supera a los humanos en los videojuegos." [Online]. Available: https://www.muyinteresante.es/tecnologia/articulo/actualidad-la-ia-ya-supera-a-los-humanos-en-los-videojuegos-611559292003.

[12] "Deep Learning: Tres cosas que es necesario saber - MATLAB & Simulink." [Online]. Available: https://es.mathworks.com/discovery/deep-learning.html#howitworks.

[13] "Un nuevo método basado en inteligencia artificial detecta el cáncer del futuro." [Online]. Available: https://www.nationalgeographic.es/ciencia/2019/05/un-nuevo-metodo-basado-en-inteligencia-artificial-detecta-el-cancer-del-futuro.

[14] "Corti: el asistente digital que detecta ataques al corazón." [Online]. Available: https://www.nobbot.com/personas/inteligencia-artificial-para-salvar-vidas-con-una-llamada/.

[15] "El lenguaje secreto de la Inteligencia Artificial." [Online]. Available: https://hipertextual.com/2016/11/lenguaje-inteligencia-artificial-google.

[16] D. Amodei *et al.*, "Deep speech 2: End-to-end speech recognition in English and Mandarin," *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 1, pp. 312–321, 2016.

[17] “Filtros de imágenes por convolución de matrices.” [Online]. Available: http://acodigo.blogspot.com/2017/05/filtros-de-imagenes-por-convolucion-de.html.
[18] “Max-pooling / Pooling - Computer Science Wiki.” [Online]. Available: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling.
[19] “Maxpooling vs minpooling vs average pooling - Madhushree Basavarajaiah - Medium.” [Online]. Available: https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9.
[20] “Rectificador (redes neuronales) - Wikipedia, la enciclopedia libre.” [Online]. Available: https://es.wikipedia.org/wiki/Rectificador_(redes_neuronales).
[21] “Función SoftMax - Wikipedia, la enciclopedia libre.” [Online]. Available: https://es.wikipedia.org/wiki/Función_SoftMax.
[22] “Detección de objetos con YOLO: implementaciones y como usarlas.” [Online]. Available: https://medium.com/@enriqueav/detección-de-objetos-con-yolo-implementaciones-y-como-usarlas-c73ca2489246.
[23] “Review: GoogLeNet (Inception v1)— Winner of ILSVRC 2014 (Image Classification).” [Online]. Available: https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7.
[24] “One-hot - Wikipedia.” [Online]. Available: https://en.wikipedia.org/wiki/One-hot.
[25] “Intersection over Union (IoU) for object detection - PyImageSearch.” [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/.
[26] “mAP (mean Average Precision) for Object Detection - Jonathan Hui - Medium.” [Online]. Available: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.
[27] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking,” *Proc. - Int. Conf. Image Process. ICIP*, vol. 2016-August, pp. 3464–3468, 2016.
[28] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017.
[29] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *J. Fluids Eng. Trans. ASME*, vol. 82, no. 1, pp. 35–45, 1960.
[30] “CrowdAI - The world at your fingertips.” [Online]. Available: https://crowdai.com/.
[31] “(No Title).” [Online]. Available: http://autti.co/.
[32] M. Cordts *et al.*, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-December, pp. 3213–3223, 2016.
[33] P. Wang, X. Huang, X. Cheng, D. Zhou, Q. Geng, and R. Yang, “The ApolloScape Open Dataset for Autonomous Driving and its Application,” *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–1, 2019.

[34] F. Yu *et al.*, “BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling,” pp. 1–16, 2018.
[35] S. Lyu *et al.*, “UA-DETRAC 2018: Report of AVSS2018 IWT4S Challenge on Advanced Traffic Monitoring,” *Proc. AVSS 2018 - 2018 15th IEEE Int. Conf. Adv. Video Signal-Based Surveill.*, 2019.
[36] K. Matzen and N. Snavely, “NYC3DCars: A dataset of 3D vehicles in geographic context,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 761–768, 2013.
[37] “Autonomous Vision Group | MPI for Intelligent Systems.” [Online]. Available: http://www.cvlibs.net/datasets/karlsruhe_sequences/.
[38] K. Behrendt and B. A. Driving, “Boxy Vehicle Detection in Large Images.”