



Universidad Europea

**UNIVERSIDAD EUROPEA DE
MADRID**

MÁSTER EN BUSINESS ANALYTICS

TRABAJO FIN DE MÁSTER

**DETECCIÓN DE OBJETOS EN IMÁGENES
BASADO EN MODELOS DE DEEP LEARNING:
DETECCIÓN DE VEHÍCULOS APLICADO A LA
CONDUCCIÓN AUTONOMA MEDIANTE
IMÁGENES EN VIVO**

Rubén García Olivas

Dirigido por

Ingeniero Carlos Rodríguez Abellán

CURSO 2021-2022

TÍTULO: DETECCIÓN DE OBJETOS EN IMÁGENES BASADO EN MODELOS DE DEEP LEARNING:
DETECCIÓN DE VEHÍULOS APLICADO A LA CONDUCCIÓN AUTÓNOMA MEDIANTE IMÁGENES EN
VIVO

AUTOR: Rubén García Olivas

TITULACIÓN: MÁSTER EN BUSINESS ANALYTICS

DIRECTOR/ES DEL PROYECTO: Ingeniero Carlos Rodríguez Abellán

FECHA: 30 de abril de 2022

Índice General

Capítulo1	5
Introducción	5
1.1 Identificación del Proyecto	5
1.2 Intervinientes	6
1.3 Proyecto	7
1.3.1 Detección de objetos para la conducción autónoma.	7
1.3.2 Detección de cánceres de mama en imágenes.	7
1.3.3 Detección de movimiento de jugadores en entornos deportivos.	7
1.4 Antecedentes y Estudio del estado del arte	8
1.4.1 Abstract	8
1.4.2 Entorno	8
Capítulo 2	10
Fundamentos y Tecnologías a usar	10
2.1 Introducción a las estructuras de Deep Learning	10
2.2 Redes Neuronales	10
2.3 Estructura de una red convolucional y elementos que la componen	11
2.4 Algoritmos de clasificación y detección de objetos	12
2.4.1 Faster Region-based Convolutional Neural Networks (Faster R-CNN)	12
2.4.2 You Only Look Once (YOLOv5)	14
2.4.3 Detectron2	16
2.5 Estadísticos de Evaluación	17
Capítulo 3	19
Desarrollo del proyecto	19
3.1 Tratamiento de datos	19
3.2 Origen de los datos	21
3.2.1 Compilación y tratamiento del conjunto de datos para YOLOv5	22
3.2.2 Compilación y tratamiento del conjunto de datos para Detectron2	22
3.3 Sistemas de detección y clasificación	22
3.3.1 Sistemas de detección y clasificación para YOLOv5	22
3.3.2 Sistema de detección y clasificación para Detectron2	23
3.4 Entrenamiento y validación	24
3.5 Visualización de resultados	28
Capítulo 4	32
Conclusiones	32
4.1 Conclusiones	32
Capítulo 5 Trabajos citados	33

Ilustración 1 : Estructura neurona	10
Ilustración 2 : Estructura Red Neuronal	11
Ilustración 3: Idea general R-CNN	13
Ilustración 4 : Arquitectura modelo YOLOv5.....	15
Ilustración 5 : Bounding Boxes explained	15
Ilustración 6: Intersection over Union	17
Ilustración 7: Imagen original del conjunto de datos	20
Ilustración 8: Muestra de imagen con boxplot proyecto STREETS	21
Ilustración 9: Benchmarking configuraciones YOLOv5	23
Ilustración 10: Comparativa Detectron2	23
Ilustración 11: Comparativa configuraciones Faster R-CNN.....	24
Ilustración 12: Detección mediante YOLOv5	29
Ilustración 13: Detección mediante detectron2	29
Ilustración 14: Detección en escena nocturna YOLO (Izquierda) vs Detectron2(Derecha).....	30
Ilustración 15: Detección Detectron2 en situación sin tráfico (Izquierda) vs con tráfico (Derecha)	30
Ilustración 16: Detección YOLOv5 en situación sin tráfico (Izquierda) vs con tráfico (Derecha)	31

Capítulo 1

Introducción

1.1 Identificación del Proyecto

A continuación, detallamos título, alumnos y tutor del proyecto:

Titulo general: “Detección de objetos en imágenes basado en modelos de Deep Learning”

Acrónimo: “DOI”

Código UNESCO: 1203.04 Inteligencia Artificial

Alumnos participantes: Rubén García Olivas

Tutor: Carlos Rodríguez Abellán

Aunque de modo general el proyecto se desarrolla en el campo del uso de redes neuronales para la detección de objetos en imágenes se desplegará bajo 3 casos de uso diferentes:

- Detección de cánceres en imágenes.
- Detección de objetos para la conducción autónoma mediante imágenes en video.
- Detección de movimiento de deportistas en imágenes en video.

1.2 Intervinientes

Los intervinientes en el proyecto son los siguientes:

Alumnos: **Rubén García Olivas**

Director/Tutor: **Carlos Rodríguez Abellán**

1.3 Proyecto

El proyecto que se va a llevar a cabo se encuadra en el mundo de las técnicas de Deep Learning, en este caso en el campo del tratamiento de imágenes. Lo que se pretende es que, a partir de una imagen, seamos capaces de interpretar e identificar los objetos que hay en ella para tomar decisiones en función de ello. Para ello se utilizarán modelos de redes neuronales (dentro de ellas las redes convolucionales) que permitan identificar y clasificar las características de una imagen para posteriormente poder descomponerla en los diversos elementos que la forman. En este sentido se trabajará en tres grandes casos de uso:

1.3.1 Detección de objetos para la conducción autónoma.

Este es un campo de vanguardia actualmente ya que la identificación de elementos en la carretera, particularmente señales de tráfico, es imprescindible para el desarrollo de los coches de conducción autónoma. Lo que se pretende es identificar tanto señales de tráfico como cualquier objeto relacionado con el tránsito de vehículos y su significado para que en el caso de que estas técnicas fuesen usadas por un coche autónomo este pueda reducir o aumentar su velocidad, parar, desviarse o en general comportarse conforme a lo que la señal identificada le indique.

1.3.2 Detección de cánceres de mama en imágenes.

En este caso lo que se plantea es la ayuda a los radiólogos y demás personal médico en la identificación de posibles tumores en imágenes de tal modo que la posibilidad de fallo en estos diagnósticos se reduzca al no depender tanto del ojo humano. Al mismo tiempo mediante estas técnicas podrá realizarse de un modo más rápido y también en mucha mayor cantidad al tener ordenadores dedicados a esto. Esto permitirá al personal médico dedicarse a otras funciones.

1.3.3 Detección de movimiento de jugadores en entornos deportivos.

En este caso lo que se plantea es utilizar técnicas de detección de objetos para detectar y seguir en video los movimientos de deportistas, más concretamente, futbolistas. Esto permite que las cámaras de televisión se muevan autónomamente para dirigir la imagen hacia donde está realmente la acción en una determinada competición.

1.4 Antecedentes y Estudio del estado del arte

1.4.1 Abstract

Fue en 1997 cuando los investigadores de la NASA Michael Cox y David Ellsworth utilizaron por primera vez el termino de “*Big Data*”, refiriéndose a este como un problema para los sistemas informáticos actuales debido al gran aumento de los datos. Desde entonces hemos visto como el termino se ha convertido en una ciencia en sí, derivando en multitud de ramas y avances que han ido a un ritmo superior que el propio progreso tecnológico.

Es bien sabido que la conducción autónoma mediante IA es una de las tecnologías que mayor repercusión ha tenido los últimos años debido a los grandes avances que se están realizando en el campo, haciendo que muchas de las grandes marcas de automoción se hayan volcado en la implementación de estos modelos de inteligencia artificial en sus productos.

En los últimos años, los métodos de detección de imágenes han evolucionado considerablemente desde técnicas básicas de procesamiento, pasando por complejos algoritmos de Machine Learning, hasta llegar a la tecnología que parece que empieza a imponerse hoy por hoy en el mercado, la utilización de redes neuronales profundas (Deep Learning).

Para el presente proyecto detallaremos los fundamentos teóricos en los cuales se sustenta la detección y tracking de objetos orientado a la conducción autónoma, para más tarde realizar la implementación de varios sistemas basados en Deep Learning para la detección tridimensional de elementos del entorno de un vehículo autónomo utilizando informacion de cámara y poder compararlos con el fin de poder tomar una conclusión sobre qué modelo es mejor con esta aplicación.

1.4.2 Entorno

Hasta hace poco la detección de objetos mediante visión artificial solamente se limitaba a planos en 2D, pero esto no era suficiente para poder comprender el entorno y es ahí cuando los modelos de procesamiento de imágenes en 3D entran en escena y dan un salto cualitativo al campo de la detección de objetos. Esto ha sido posible principalmente mediante 2 tipos de tecnologías: La tecnología radar, que mediante ondas de radio que rebotan contra receptores fijos o móviles pueden medir distancias entre objetos, y la novedosa tecnología LiDAR, que aunque funciona de una forma parecida a la tecnología de radar, esta se basa en un láser

infrarrojo que mide el tiempo que tarda en llegar a un objeto, siendo lo asombroso de esto que cuando desarrollas un sensor de 360 grados, puedes obtener una nube de puntos que nos muestre un mapa bastante preciso de nuestro entorno.

Muchos prototipos de coches autónomos confían en el LiDAR como sensor principal para “ver” todo tipo de obstáculos, por ejemplo, Google, Ford, Toyota, Volkswagen o Uber lo emplean. Por otro lado, fabricantes como Tesla creen que esta herramienta es prescindible debido al alto coste y la complejidad para poder introducirla en sus diseños y recurren a alternativas tan consolidadas como cámaras de video de alta definición. Sin ir más lejos, Tesla con su Autopilot implementa 8 cámaras de video alrededor de todo el vehículo con el fin de poder cubrir el mayor campo visual y que teóricamente, se pueda reconocer todo.

Si profundizamos dentro del campo de la conducción autónoma, podemos encontrarnos con que esta está segmentada en 6 niveles distintos, enumerados desde el nivel 0 al nivel 5. Cada nivel que vamos avanzando en estos niveles, el grado de complejidad y dependencia de nuestra IA será mayor, requiriendo el nivel 0 una máxima implicación del ocupante del vehículo en todas las tareas de atención y de reacción, mientras que el máximo nivel de automatización se dará en el nivel 5 donde no hará falta que el conductor esté presente.

Actualmente la gran mayoría de vehículos se encuentran dentro del nivel 2, siendo este el techo actual ya que el paso hacia nivel 3 requiere de un salto cualitativo grande porque el vehículo en este caso tendría un sistema de conducción automatizado que podría realizar la totalidad de las tareas que engloba la conducción.

Los niveles 4 y 5 por el momento son terreno inexplorado, diferenciándose del nivel 3 en cuanto a la cantidad de ocasiones en donde es necesaria la intervención del humano, considerándose el nivel 5 un punto en el cual el vehículo no necesita de ayuda humana en ninguna situación.

Capítulo 2

Fundamentos y Tecnologías a usar

2.1 Introducción a las estructuras de Deep Learning

Las redes neuronales se enmarcan en el campo de la inteligencia artificial. Las neuronas artificiales se modelan de tal forma que imitan el comportamiento de una neurona cerebral. Tendrán unas ramificaciones y un núcleo o nodo.

Habrá ramificaciones de entrada al nodo, que serán las entradas de la neurona, procedentes de otras neuronas. Esta información se procesará en un nodo, y se generará una información de salida que se transmitirá por las ramificaciones de salida a otras neuronas. Podemos pensar en las conexiones entre neuronas artificiales como en las sinapsis de las neuronas del cerebro.

Para entender el funcionamiento y la implicación que tienen las redes neuronales en problemas complejos de detección de objetos mediante modelos de Deep Learning, es esencial que entendamos las dos estructuras principales de red neuronal: La red multicapa y la red convolucional.

2.2 Redes Neuronales

Como ya hemos mencionado, una red neuronal es un modelo simplificado que emula el modo en el que el cerebro humano procesa la información, esto quiere decir que la estructura de esta red está formada por nodos de canalización de información, denominados perceptrones, que emulan de alguna forma el comportamiento de las neuronas de un ser humano.

Estos nodos los cuales llamaremos neuronas estarán integrados dentro de capas, pudiendo distinguir dentro de nuestra red neuronal 3 tipos de capas:

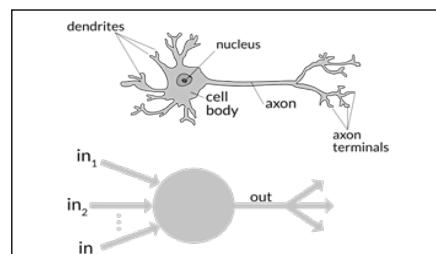


Ilustración 1 : Estructura neurona

- Capa de entrada: Son las unidades principales del modelo, y están formadas por neuronas que representan los campos de entrada de información.
- Capas ocultas: También denominadas capas de procesamiento son variables que no tienen una conexión directa con el entorno y normalmente realizan alguna función sobre sus predictores, esto quiere decir que realizan alguna función predefinida sobre sus variables de entrada, devolviendo un valor distinto.
- Capa de salida: Son aquellas cuya salida da una respuesta a la red neuronal

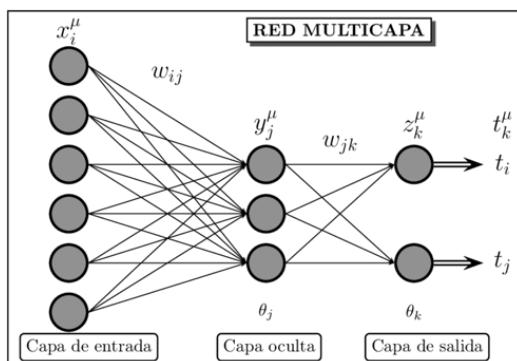


Ilustración 2 : Estructura Red Neuronal

Cuando la información ya se ha procesado dentro de cada neurona, la transmisión de la información entre capas se realiza mediante lo que se conoce como funciones de activación, el hecho de transformar la información que genera cada nodo se debe a la voluntad de dar una “no linealidad” al modelo, adaptando cada conjunto de datos al problema en concreto y haciendo al modelo capaz de resolver problemas más complejos. Las funciones de activación más comunes son: Función escalón, Sigmoid, ReLU, Tangente hiperbólica y SoftMax.

2.3 Estructura de una red convolucional y elementos que la componen

Las redes neuronales convolucionales son muy similares a las redes neuronales multicapas que acabamos de ver, en el sentido de que estas también se componen de neuronas, agrupadas en capas, las cuales tienen unos pesos y unas funciones de activación.

Lo que las hace especiales es que las capas de esta red neuronal están formadas por múltiples filtros convolucionales, la estructura de estas podría verse como:

- **Entrada:** Son los pixeles de un área local de la imagen que queremos introducir en el modelo, compuesto por 2 dimensiones (alto y ancho) o 3, en el caso de que la imagen tuviera color (profundidad).
- **Capa de convolución:** procesará la salida de neuronas que están conectadas en «regiones locales» de entrada (es decir pixeles cercanos), calculando el producto escalar entre sus pesos (valor de píxel) y una pequeña región a la que están conectados en el volumen de entrada.
- **Capa de reducción o de pooling:** Tiene como objetivo reducir la cantidad de parámetros, quedándose solamente con los que sean más comunes. Cabe destacar que esta reducción de parámetros solamente se producirá en las dimensiones alto y ancho, no en profundidad.
- **Capa “tradicional” o de clasificación:** Esta tiene una estructura similar a la de una red neuronal multicapa común, estará totalmente conectada y nos dará el resultado de nuestro modelo.

2.4 Algoritmos de clasificación y detección de objetos

Aunque el campo de la detección y clasificación de objetos mediante aprendizaje automático ha tenido un gran rendimiento y progreso, esta empieza a caer un poco ante nuevos retos o problemas dinámicos que requieren un ajuste de parámetros para un rendimiento razonable. La puesta en escena de las redes neuronales y más tarde las redes neuronales convolucionales ha hecho que se puedan superar estos problemas y en concreto permiten una mejora significativa en el rendimiento de los detectores de objetos.

Dentro de esta mejora sustancial que se ha comentado previamente, de entre todas las técnicas y modelos que se utilizan para la clasificación y localización de objetos mediante Deep Learning, los más populares a día de hoy podríamos decir que son (por orden alfabético): *Faster R-CNN*, *Histogram of Oriented Gradients (HOG)*, *Region-based Convolutional Neural Networks (R-CNN)*, *Region-based Fully Convolutional Network (R-FCN)*, *Single Shot Detector (SSD)*, *Spatial Pyramid Pooling (SPP-net)* y *YOLO (You Only Look Once)*.

2.4.1 Faster Region-based Convolutional Neural Networks (Faster R-CNN)

Para poder entender el funcionamiento de Faster R-CNN es preciso que conozcamos en primer lugar a su predecesor y modelo que sirve como arquitectura base, el *Region-based Convolutional*

Neural Network. La principal diferencia entre un clasificador de red convolucional de red neuronal y R-CNN es que el primero es principalmente utilizado para la clasificación de imágenes mientras que R(región)-CNN se utiliza principalmente para la detección de objetos por lo que añade una funcionalidad extra y por lo tanto es capaz de clasificar y localizar objetos mediante imágenes.

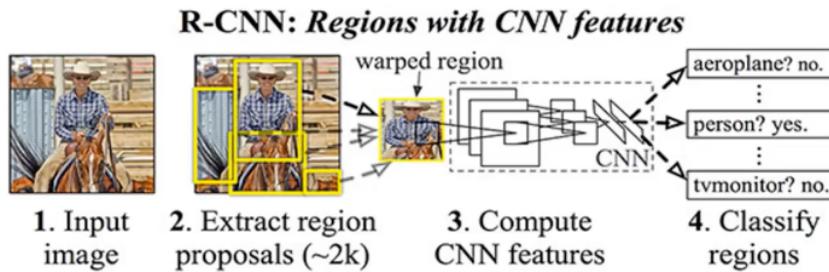


Ilustración 3: Idea general R-CNN

La forma en la que R-CNN trabaja consiste en introducir un input al modelo dividiendo esta en una cuadricula de entre 1.500 a 2.000 regiones, sirviendo cada una de ellas como input para un modelo de red neuronal convolucional. El proceso de selección de regiones está basado en la idea de segmentar la imagen de entrada en miles de posibles objetos, y mediante una combinación recursiva de las regiones propuestas en base a su color, textura, tamaño y relleno de sus regiones vecinas más cercanas, poco a poco vamos combinando regiones similares para hacer regiones más grandes (Shaoqing Ren, 2016).

La gran ventaja de este algoritmo de detección reside en sus buenos resultados originados por la CNN, que permite predecir resultados con una precisión muy alta. Por el contrario, el punto negativo de esta es que es más lenta respecto de otros modelos y dependiente de la aplicación en donde se vaya a hacer uso del modelo, puede ser el indicado o no.

En 2015 se publica el primer artículo sobre la nueva versión de la familia R-CNN. La gran ventaja y problema que venía a solucionar esta versión viene relacionada con la eficiencia computacional y los tiempos de *test-training*, esto ocurría porque como ya hemos comentado anteriormente, tanto en R-CNN como en versiones posteriores se utilizaban algoritmos de propuesta de región basados en la CPU, por ejemplo, el algoritmo de búsqueda selectiva que se usa en estos modelos tarda alrededor de 2 segundos por imagen y se ejecuta en la propia CPU.

Faster R-CNN soluciona esto utilizando otra red convolucional

(*Region Proposal Network*) para generar las propuestas de región y de este modo liberar a la arquitectura del algoritmo de la búsqueda selectiva, no solo permitiendo rebajar el tiempo de los 2s a 17ms por imagen, sino que también nos permite que se comparten capas con las siguientes etapas de detección, lo que provoca una mejora general en la representación de características.

2.4.2 You Only Look Once (YOLOv5)

YOLO es uno de los algoritmos de IA preferido por los ingenieros dedicados a la investigación en detección y clasificación de objetos, además suele ser la primera opción cuando de problemas de detección en tiempo real se trata. Si por algo se caracteriza YOLO y el motivo por el cual es de los algoritmos más usados para usos como los que se abordan en este trabajo es por su velocidad de detección y clasificación de objetos (En YOLOv5 entre los 2ms/fps a 10ms/fps).

La versión que vamos a tratar en el siguiente trabajo es en concreto YOLOv5, lanzado originalmente en junio de 2020 no deja de ser la actualización de la primera versión original de YOLO, lanzado en su origen 4 años atrás ha basado sus mejoras principalmente a la optimización de las métricas de evaluación (ocuparemos este tema más tarde) cuando tratamos inputs con altos FPS.

El proceso de aplicación del algoritmo se divide en 3 fases, en la primera de estas se recibe el input del modelo que de forma generalizada será una imagen o el frame de un video, donde dividimos el mismo en una cuadricula de dimensión $S \times S$ y en cada cuadricula se lleva a cabo un proceso de detección independiente y paralelo. Este proceso de detección tiene sus propias particularidades que pasaremos a detallar a continuación y donde reside el verdadero secreto del algoritmo.

Como YOLOv5 es un modelo de detección de una sola etapa, consta de 3 bloques principales dentro de su arquitectura: Columna vertebral del modelo (*Model backbone*), cuello del modelo (*Model neck*) y cabeza del modelo (*Model head*).

Overview of YOLOv5

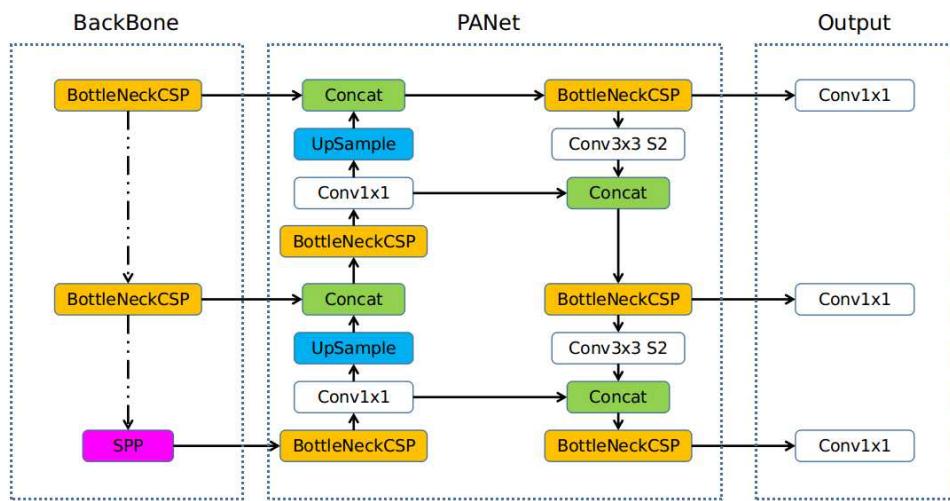


Ilustración 4 : Arquitectura modelo YOLOv5

La columna vertebral de un modelo (*Model backbone*) YOLOv5 se denomina a la fase en la cual se extraen todas las características importantes del input/ímagenes, para este fin se utiliza el *CSP-Cross Stage Partial Network* que trata de mitigar el problema de duplicidad de información del gradiente dentro de la optimización de la red, mediante la integración de mapas de características desde el principio al final de la etapa (Wu, 2019). Entendemos como cuello del modelo (*Model Neck*) como la implementación de una pirámide de características que ayuden a generalizar objetos mediante sus características a diferentes escalas y podamos clasificar el mismo objeto cuando se encuentre en distintos tamaños o escalas, siendo el método más utilizado el *PANet (Path Aggregation Network)*. Por último, la cabeza del modelo (*Model head*) es la parte final del modelo de detección, esta consiste en la ilustración de cajas delimitadoras de los objetos y vectores que son capaces de determinar la probabilidad de aparición de estos.

La segunda fase de aplicación enlaza con lo comentado previamente y es el proceso en el cual se delimitan todas las cajas allí donde se ha detectado el objeto en cuestión. La siguiente imagen (*Ilustración 5*) ofrece un ejemplo de cómo se muestran

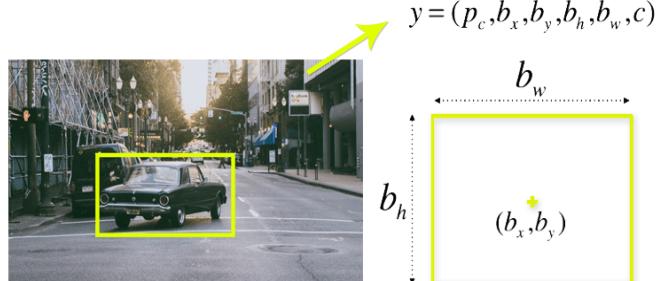


Ilustración 5 : Bounding Boxes explained

estas cajas. El proceso termina con lo que llamamos *Intersection Over Union (IOU)*, que consiste en solapar todas las cajas delimitadoras que han salido como resultado de la detección de cada cuadricula y formar las cajas definitivas allí donde la densidad de cajas es mayor, descartando las áreas donde la densidad es menor. Esto nos da como resultado cajas mucho más precisas y fieles a la realidad.

2.4.3 Detectron2

Detectron2 es la evolución natural de su primera versión, Detectron, diseñada por *Facebook AI Research (FAIR)*. Esta es una reescritura completa de su versión original y tiene como punto de referencia maskrcnn-benchmark, su implementación se encuentra sobre el framework Pytorch y la utilización de este algoritmo requiere del uso de CUDA, que es una plataforma informática paralela patentada por NVIDIA, que permite utilizar el procesamiento basado en GPU en lugar de CPU debido a los pesados cálculos involucrados (Girshick, 2019).

Mask R-CNN es una extensión de Faster R-CNN (Kaiming He, s.f.) y funciona añadiendo una rama con el objetivo de predecir una máscara en paralelo a la rama existente para el reconocimiento de cajas delimitadoras. Encontramos dos etapas dentro de Mask R-CNN:

- Primera etapa: El sistema genera propuestas sobre las regiones donde puede haber un objeto en base a la imagen de entrada.
- Segunda etapa: En esta segunda fase, el algoritmo predice las clases de los objetos, crea las cajas delimitadoras y genera la máscara a nivel de píxel del objeto en base al resultado de la primera etapa.

Una de las mayores ventajas que se asumen cuando se utiliza detectron2 como algoritmo de detección y clasificación es que ofrece el soporte para la inferencia, entrenamiento, uso de modelos preentrenados e incluso la modificación de la configuración de los modelos.

Para este trabajo, la opción escogida para entrenar el modelo ha sido la de “*Faster-RCNN R 50 FPN 3x*” mediante Detectron2.

2.5 Estadísticos de Evaluación

Durante el entrenamiento y evaluación de los modelos que más tarde veremos, utilizaremos una serie de métricas con el fin de poder comparar y sacar conclusiones sobre los modelos que son más convenientes para nuestra aplicación, pero es necesario previamente explicar que métricas utilizaremos para comparar los distintos resultados:

- Verdadero Positivo o *True Positive (TP)*: Acierto de detectar la señal que existe en la imagen acertando tanto la localización como la clasificación de este.
- Falso positivo o *False Positive (FP)*: Error que aparece cuando se detecta dentro de la imagen una señal que no existe, o bien cuando se detecta, pero esta no se ha clasificado correctamente.
- *Intersection over union (IoU)*: Como ya introdujimos conceptualmente más atrás, el IoU se encarga de medir que superficie del cuadro predicho se encuentra por encima del cuadro real

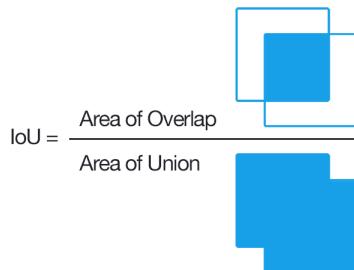


Ilustración 6: *Intersection over Union*

- Precisión o *Accuracy*: Mide como de precisa es la predicción mediante un rango de 0 a 1. Consideraremos valida o buena la predicción si supera cierto umbral que delimitaremos en $\text{IoU} \geq 0,5$.

$$\text{Accuracy} = \frac{TP}{TP + FP}$$

- Recall: Es la relación que cuantifica como de bien se han encontrado todos los objetos existentes en la imagen.
- Función de error o *Loss Function*: Seguramente sea uno de los estadísticos más importantes ya que nos orientaran en que tal medida se está entrenando nuestro modelo.
- Precisión media o *Average Precision (AP)*: Área de la curva precisión-recall

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

- Promedio de precisión media o *Mean Average Precision* (mAP): Mide la precisión media para cada clase y realiza un promedio. Cabe destacar que el mAP es usado como una de las métricas estándar cuando se trata de problemas de detección de objetos y para el caso que trataremos, el cual es un “*Custom Dataset*” y no uno predefinido en formato COCO, utilizaremos esta métrica de comparación principalmente en lugar del AP anterior.

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i$$

Capítulo 3

Desarrollo del proyecto

A continuación, pasaremos a detallar paso a paso como hemos ido desarrollando el modelo desde la primera fase en la que definimos la idea y obtenemos el conjunto de datos que entrenaremos hasta la parte final donde visualizamos los resultados obtenidos y evaluamos la calidad de los mismos. El objetivo de este capítulo es como hemos mencionado, acompañar durante todo el proceso de entrenamiento-evaluación y justificar por qué tomamos las decisiones que tomamos en cuanto a configuraciones y demás opciones.

3.1 Tratamiento de datos

Para este trabajo, como hemos mencionado, entrenaremos el mismo conjunto de datos mediante diferentes algoritmos de clasificación, en concreto utilizaremos YOLOv5 y Faster R-CNN mediante detectron2. La motivación por la cual hemos escogido ambos modelos es principalmente porque a día de hoy son los dos algoritmos más utilizados en el campo de la detección de objetos aplicado a la conducción autónoma, aunque con una clara diferenciación de ámbito de funcionamiento y es que YOLO tiene como una de sus características principales la gran velocidad de detección y clasificación aunque para ello sacrifique algo de precisión (Principalmente debido a que es un detector de una sola etapa), hecho que lo hace propicio para su implementación en situaciones de detección en tiempo real o donde se busca una clara predominancia de la velocidad en contraposición a la precisión. Por otro lado, detectron2 alimentado por *Faster R-CNN* es uno de los detectores más equilibrados que hay desarrollados en la actualidad, teniendo uno de los mejores niveles de precisión (Debido principalmente a que es un detector de dos etapas) sin ser tampoco un detector “lento”, es utilizado principalmente en aplicaciones de cámaras de tráfico donde la velocidad no es un factor determinante y sí que vamos a necesitar una mayor precisión.

Esto supone el primer reto ya que hemos decidido con el fin de que la conclusión sea lo más acertada posible no utilizar datos preentrenados ni conjuntos ya creados de la base de datos COCO. Por ello hemos utilizado un conjunto de datos que venía diferenciado en 3 subcategorías de imágenes: “*train*”, “*test*” y “*validation*”.



Ilustración 7: Imagen original del conjunto de datos

Dentro de cada subconjunto a su vez hemos trabajado con archivos en formato *.jpg* y su homónimo en formato *.txt* donde hemos podido en ocasiones encontrar y en otras generar las etiquetas que delimitaban los objetos de nuestras imágenes para poder entrenar el modelo. Las características de nuestro “*custom dataset*” son las siguientes:

<i>Característica</i>	<i>Valor</i>
Formato de imágenes	JPG; TXT
Formato de anotación	Filename; x_left;x_right;y_bottom;y_top; class_id
Tamaño de cuadriculas	416x416
Clases	1
Imágenes de entrenamiento	5248
Imágenes de validación	582
Imágenes de test	291
Total imágenes	6121

Tabla 1: Características conjunto de datos

Cabe añadir a lo comentado, que el conjunto de datos fue concebido en su origen para ser trabajado sobre YOLO, al querer implementar el mismo conjunto de datos y ser entrenado bajo el algoritmo de detectron2, se tuvo que transformar los *.txt* que contenían la información relativa a cada imagen en unos *.json* de estructura estándar a fin de que la implementación más tarde fuera lo más limpia y sencilla posible.

También a método de nota aclaratoria, se tomó la decisión de que solo hubiera una clase de imagen porque el trabajo y el conjunto de datos está obtenido mediante cámaras de vigilancia estáticas en la carretera, lo que en contexto quiere decir esto, es que los únicos objetos dinámicos que a priori nos interesan detectar son los automóviles.

3.2 Origen de los datos

Las imágenes que utilizamos en el presente trabajo proceden de un estudio basado en los datos utilizados por el proyecto STREETS (Snyder, 2019). Este conjunto de datos es una colección de capturas de pantalla aleatorias de cámaras de tráfico en Chicago. En los datos, todos los vehículos están etiquetados en una sola categoría, como se ha comentado previamente, llamada “car”. Las etiquetas consisten en cajas delimitadoras (*bounding boxes*) en formato YOLOv5 PTorch.

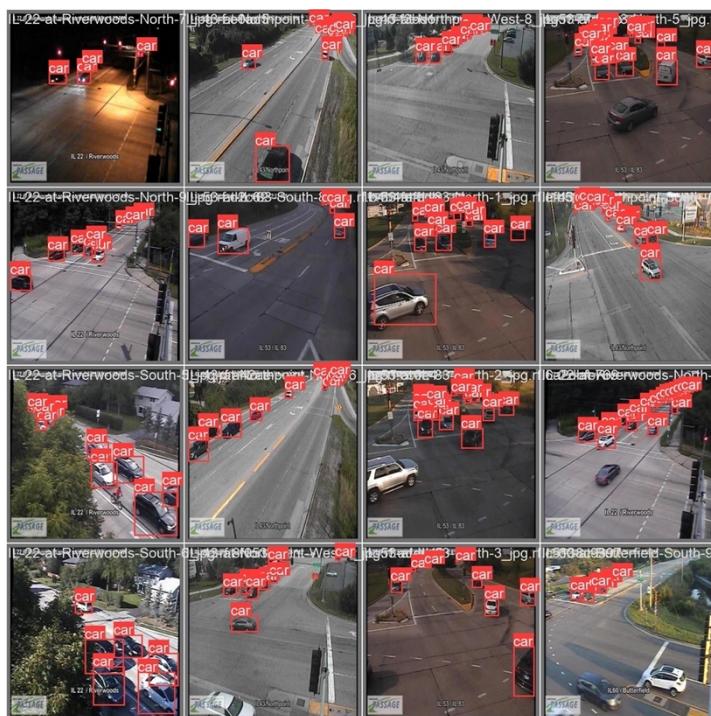


Ilustración 8: Muestra de imagen con boxplot proyecto STREETS

Los datos de origen están a disposición del público y fueron obtenidos entre 2018 y 2021, teniendo en el momento actual de extracción de estos, licencia CC0.

3.2.1 Compilación y tratamiento del conjunto de datos para YOLOv5

Con el objetivo de poder simplificar al máximo la carga de código y aprovechando una de las ventajas de YOLO, creamos un archivo *.yaml* que contenga todas las rutas y características del modelo que queremos entrenar. YAML es un lenguaje de declaración de datos que facilita la legibilidad y la capacidad de escritura del usuario, permitiendo al mismo evitar la especificación de todos los parámetros en la línea de comandos (YAML Language Development Team, 2021), en concreto, nuestro *.yaml* contiene la siguiente información:

```
1  train: ../traffic/train/images
2  val: ../traffic/valid/images
3
4  nc: 1
5  names: ['car']
```

3.2.2 Compilación y tratamiento del conjunto de datos para Detectron2

Como comentamos previamente, necesitamos por simplicidad y fluidez convertir los archivos de etiquetas en JSON, que simplemente es otro formato de texto y para realizar dicha conversión utilizamos la librería *asposecells*.

Una vez tenemos nuestro conjunto de datos en el formato de origen que queremos, es hora de transformarlo en el formato necesario para crear un conjunto de datos COCO, que finalmente será el conjunto que entrenemos. Para ello simplemente utilizamos las etiquetas para crear una tabla y posteriormente mediante una función previamente definida transformamos la información de la tabla a formato COCO.

3.3 Sistemas de detección y clasificación

3.3.1 Sistemas de detección y clasificación para YOLOv5

Debido a que para nuestro ejemplo en concreto solo nos centraremos en la clasificación de un objeto por los motivos que ya mencionamos previamente, tanto la configuración para la detección como para la clasificación será la misma.

Siguiendo el hilo de lo que hemos comentado previamente, de cara a la simplicidad del modelo en sí, decidimos implementar toda la configuración y rutas necesarias dentro de unos documentos *.yaml* por lo que especificamos toda la configuración dentro del mismo, aunque por regla general es cierto que estos archivos ya vienen preconfigurados y en nuestro caso hemos podido obtenerlos directamente de la web del desarrollador (Jocher, 2020).

Para el posterior entrenamiento decidimos utilizar en concreto una configuración de YOLO llamada “*YOLOv5m*”, ya que creemos que para la aplicación que queremos utilizar es la ideal debido a que nos ofrece la ratio más baja de ms/fps con unas métricas de evaluación más que aceptables.

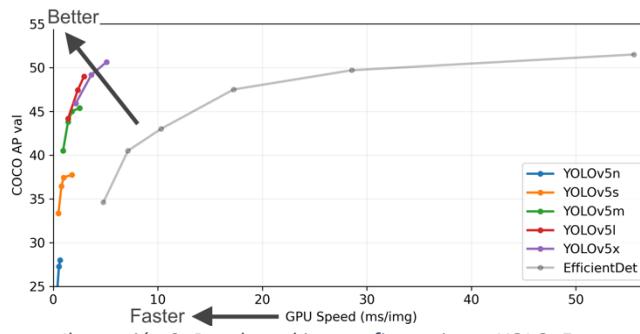


Ilustración 9: Benchmarking configuraciones YOLOv5

La función de perdida que hemos escogido para este modelo ha sido una función “*CrossEntropy*”, utilizando un entrenamiento en 12 épocas y un “*batch size*” de 3.

3.3.2 Sistema de detección y clasificación para Detectron2

Aprovechando que Detectron2 puede implementar tanto Mask-RCNN como otras versiones de la familia, para el caso de uso decidimos utilizar Faster R-CNN principalmente porque es el que más velocidad de entrenamiento nos ofrece a la vez de un buen AP. Descartamos en su origen “Keypoint R-CNN” ya que con cierta configuración suele dar bastantes problemas y corregirlos puede ser contraproducente.

Name	lr sched	train time (s/iter)	inference time (s/img)	train mem (GB)	box AP	mask AP	kp. AP	model id
Faster R-CNN	1x	0.219	0.038	3.1	36.9			137781054
Keypoint R-CNN	1x	0.313	0.071	5.0	53.1		64.2	137781195
Mask R-CNN	1x	0.273	0.043	3.4	37.8	34.9		137781281

Ilustración 10: Comparativa Detectron2

Dentro de Faster-RCNN, la configuración que hemos elegido es “*faster_rcnn_R_50_FPN_3x*” debido a que es el más eficiente en cuanto a carga de hardware y mejor velocidad de entrenamiento nos ofrece.

Faster R-CNN:								
Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download	
R50-C4	1x	0.551	0.102	4.8	35.7	137257644	model metrics	
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model metrics	
R50-FPN	1x	0.210	0.038	3.0	37.9	137257794	model metrics	
R50-C4	3x	0.543	0.104	4.8	38.4	137849393	model metrics	
R50-DC5	3x	0.378	0.070	5.0	39.0	137849425	model metrics	
R50-FPN	3x	0.209	0.038	3.0	40.2	137849458	model metrics	
R101-C4	3x	0.619	0.139	5.9	41.1	138204752	model metrics	
R101-DC5	3x	0.452	0.086	6.1	40.6	138204841	model metrics	
R101-FPN	3x	0.286	0.051	4.1	42.0	137851257	model metrics	
X101-FPN	3x	0.638	0.098	6.7	43.0	139173657	model metrics	

Ilustración 11: Comparativa configuraciones Faster R-CNN

Otros valores que marcamos dentro de la configuración del algoritmo son los siguientes:

- cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE: Decidimos fijar un valor de 128 ya que después de realizar pruebas llegamos a la conclusión de que con un valor de 64 los tiempos eran mucho más lentos, aunque la precisión aumentara, para nuestro caso marcar un valor de 128 nos ofrece más velocidad aunque menor precisión.
- cfg.MODEL.WEIGHTS: Utilizamos los pesos que vienen por defecto dentro de “*faster_rcnn_R_50_FPN_3x*”
- Loss_type: *CrossEntropy*
- cfg.SOLVER.BASE_LR: El valor escogido como ratio de aprendizaje ha sido por defecto de 0.0005.

3.4 Entrenamiento y validación

Utilizando la configuración previamente descrita en los apartados 3.3.1 y 3.3.2 procedemos a entrenar nuestro modelo en el entorno base de *Google Colab* y con una configuración adicional de procesamiento mediante GPU en lugar de CPU con el fin de poder reducir los tiempos de entrenamiento.

Una vez hemos entrenado los modelos, podemos comparar ambos donde sacamos las siguientes conclusiones:

Por una parte, los resultados del entrenamiento de YOLO se presentan en el Gráfico 1, donde con la configuración elegida vemos que tanto la precisión (línea azul) como el *Mean Average Precision* (línea naranja) alcanzan cierto techo sobre la época 12 donde obtenemos valores de 0,82038 y 0,76036 respectivamente (Gráfico 2). El tiempo de detección con este entrenamiento es de 12ms/img mientras que el tiempo de entrenamiento de todo el conjunto de imágenes ha sido de 44 minutos con el acelerador de hardware mediante GPU de *Google Colab*.

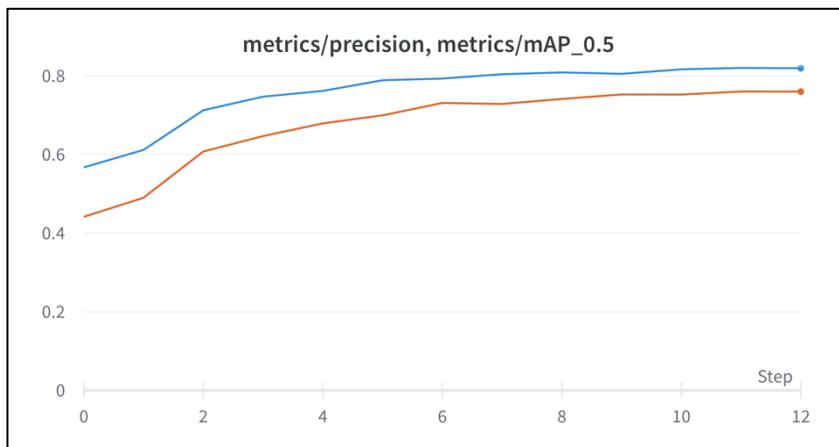


Gráfico 1: Accuracy, mAP_0.5

Por otro lado, los resultados en cuanto a caída de la función de perdida también podemos considerar que es buena, ya que como podemos observar llegados a cierta etapa dejamos de ver una caída significativa y este es uno de los motivos por el cual decidimos no aumentar las épocas de entrenamiento, ya que muy probablemente llegados a este punto hubiéramos observado señales de *overfitting*. El resultado de las principales métricas las podemos encontrar en el Grafico 3 donde veremos los indicadores previamente mencionados tanto para la etapa de entrenamiento como de evaluación.

Gráfico 2: Run Summary

```

Run summary:
  best/epoch 11
  best/mAP_0.5 0.76036
  best/mAP_0.5:0.95 0.3759
  best/precision 0.82038
  best/recall 0.69268
  metrics/mAP_0.5 0.76011
  metrics/mAP_0.5:0.95 0.37574
  metrics/precision 0.81956
  metrics/recall 0.69269
  train/box_loss 0.04211
  train/cls_loss 0.0
  train/obj_loss 0.09035
  val/box_loss 0.04364
  val/cls_loss 0.0
  val/obj_loss 0.04203
  x/lr0 0.00175
  x/lr1 0.00175
  x/lr2 0.00175

```

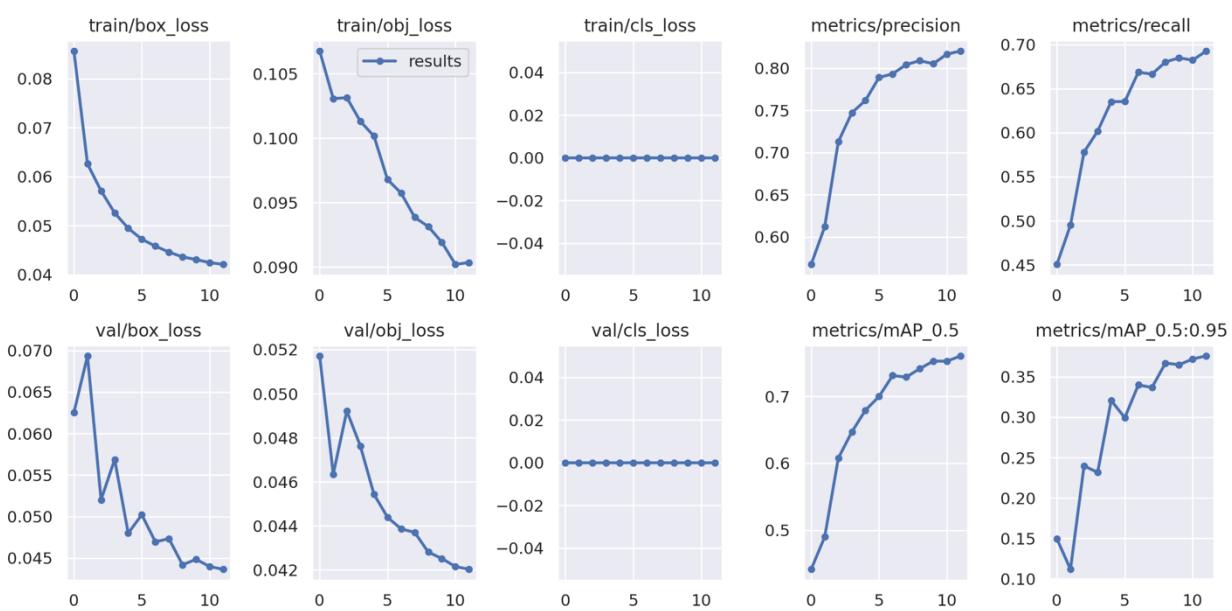


Gráfico 3: Métricas train y validation

Por último, y tras observar la F1-Curve vemos que el valor óptimo del parámetro “confidence” para este caso se sitúa en 0,378. Este parámetro en concreto es especialmente útil para maximizar la precisión y el recall de la detección, el mismo puede encontrarse en un intervalo de 0 a 1 y no necesariamente un valor más alto está directamente relacionado a un mejor resultado final.

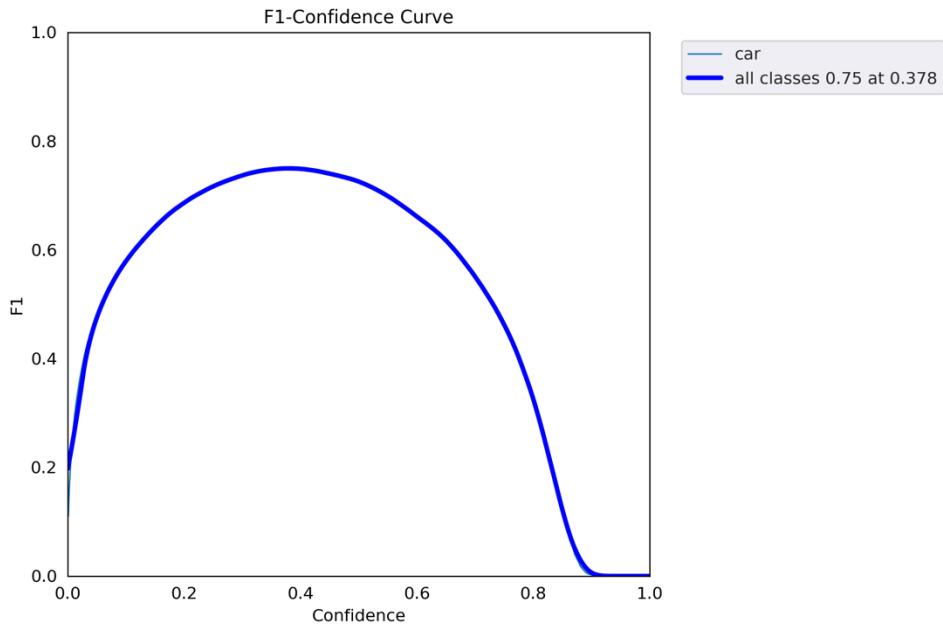


Gráfico 4: F1-Confidence Curve

Por otro lado, de cara a evaluar los resultados de detectron2 hemos observado que tras entrenar el modelo obtenemos unos resultados que sobre el papel podríamos esperar. Siguiendo el mismo orden que utilizamos con YOLO, vemos que los resultados de precisión son mejores para este modelo, en cambio el mejor valor del *mAP* no mejora respecto al algoritmo anterior (Grafico 5). Esto es algo que podríamos esperar principalmente debido a que YOLO al igual que otros algoritmos como SDD son algoritmos de una sola etapa (Wei Liu, 2016), ideados en un origen para primar la velocidad sobre la precisión.

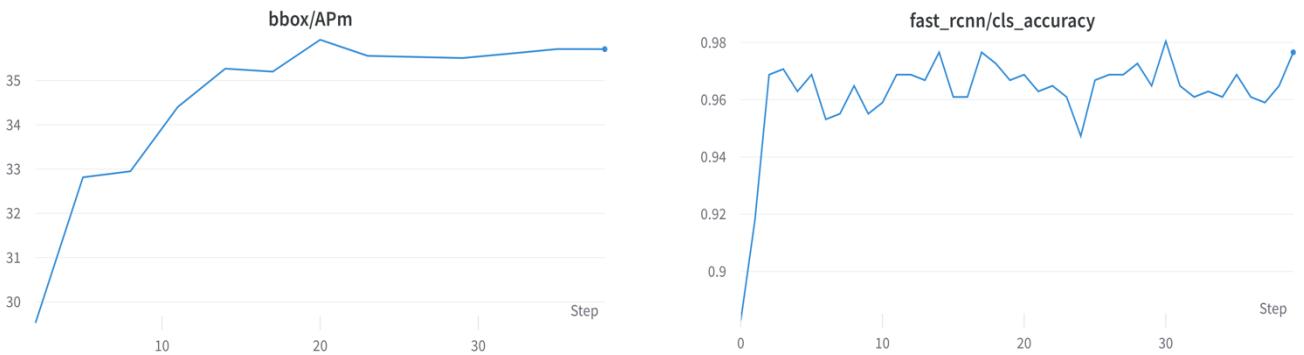


Gráfico 5: Accuracy, mAP Faster-RCNN

Así pues, a nivel general los resultados de este modelo serían los siguientes en cuanto a las métricas principales (Grafico 6). Por último, los tiempos medios de detección por imagen han estado entorno a los 15-22ms/img mientras que el tiempo total del entrenamiento ha sido superior a los 45 minutos.

AP	AP50	AP75	APs	APm	APl
34.748	52.219	41.311	26.210	35.816	37.392

Gráfico 6: Summary detectron2

3.5 Visualización de resultados

Por la parte de la visualización de datos, hemos elegido imágenes de muestra en diferentes situaciones detectadas por ambos modelos, a fin de que las conclusiones sobre las distintas técnicas sean más esclarecedoras.

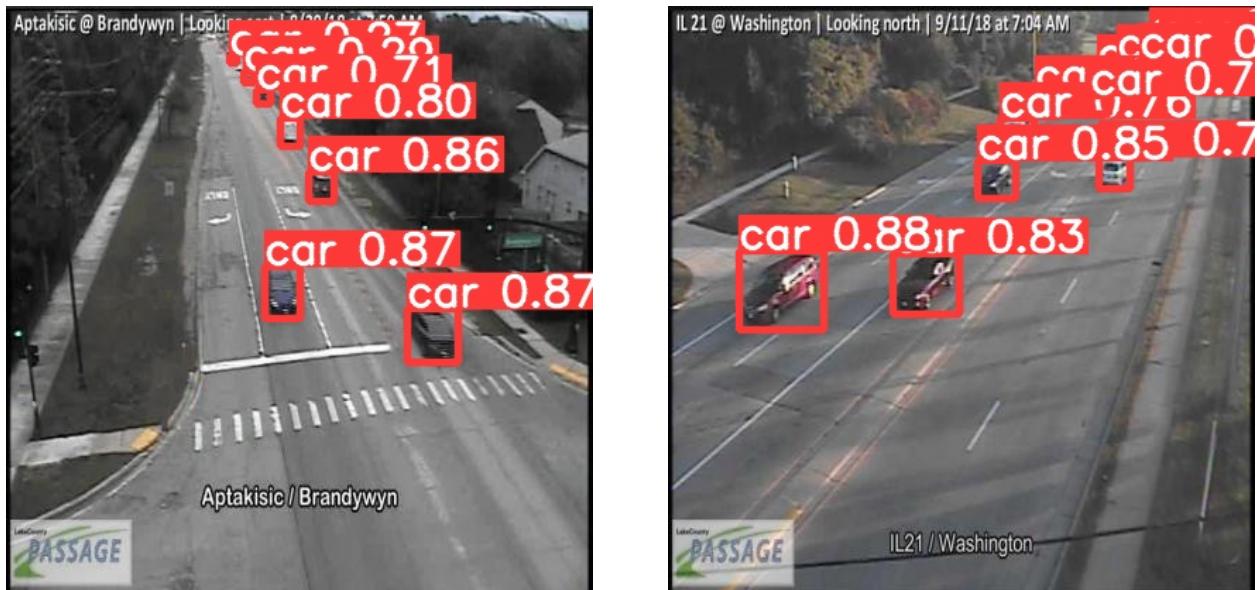


Ilustración 12: Detección mediante YOLOv5



Ilustración 13: Detección mediante detectron2



Ilustración 14: Detección en escena nocturna YOLO (Izquierda) vs Detectron2(Derecha)

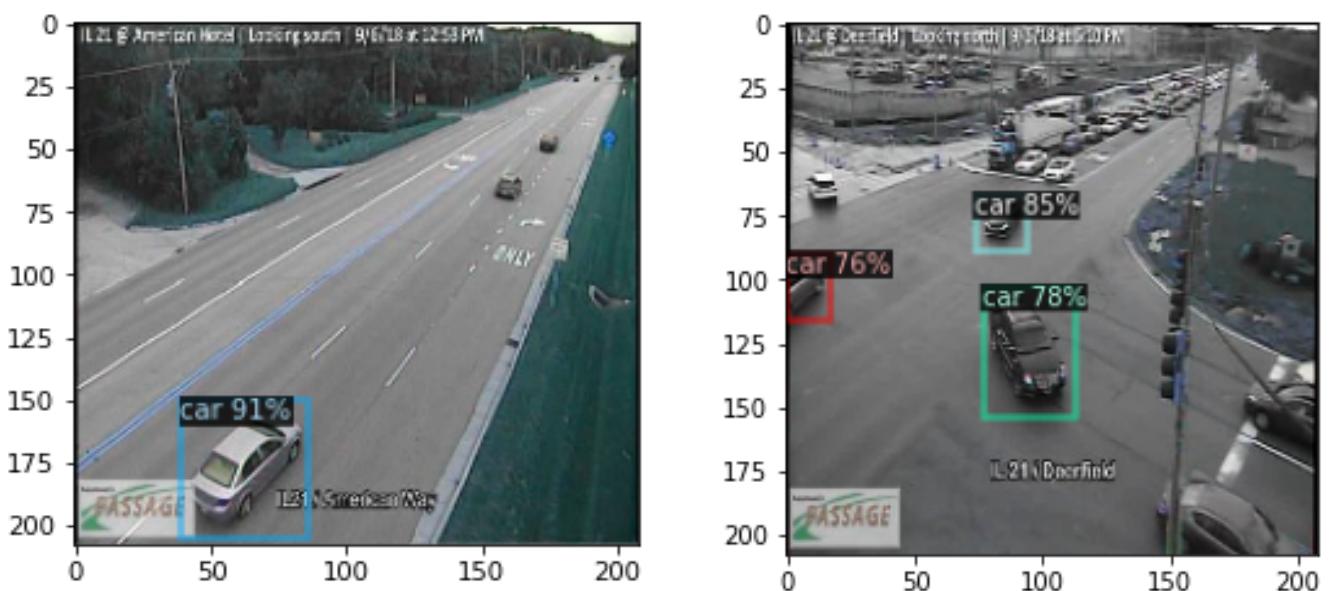


Ilustración 15: Detección Detectron2 en situación sin tráfico (Izquierda) vs con tráfico (Derecha)



Ilustración 16: Detección YOLOv5 en situación sin tráfico (Izquierda) vs con tráfico (Derecha)

Capítulo 4

Conclusiones

4.1 Conclusiones

La conducción autónoma probablemente se trate de uno de los campos de estudio más atractivos que existen en la actualidad. A pesar de que todavía sea un mercado poco desarrollado, hemos podido observar que los últimos avances en algoritmos de detección de objetos, principalmente alimentados por la puesta en escena de las redes neuronales hace que muy probablemente en menos tiempo del que creamos podremos disfrutar de vehículos sin volantes ni pedales.

Tal y como hemos podido observar, parece claro que el algoritmo de YOLO se sitúa a día de hoy un peldaño por delante que otros modelos ya que ha llegado a un punto de madurez donde sigue siendo el modelo con más rapidez a la hora de detectar y clasificar y ha conseguido mejorar mucho los niveles de precisión haciendo que incluso dependiendo de las circunstancias pueda ser más preciso que otros modelos que sobre el papel disfrutan de tener mejores *benchmark* en cuanto a precisión. Hoy en día sigue en desarrollo YOLOv6 y YOLOv7, centrándose en ser los reyes en detección con inputs de altos fps.

Dejamos a continuación enlace al repositorio de GitHub (<https://github.com/rubgarcia97/TFM>) donde se encuentra alojado un video que ha sido tratado con los pesos y la configuración resultante del modelo entrenado previamente.

Capítulo 5 Trabajos citados

- Girshick, Y. W.-Y. (2019). *Detectron2*. Obtenido de Github:
<https://github.com/facebookresearch/detectron2>
- Jocher, G. (2020). *GitHub*. Obtenido de github.com/ultralytics/yolov5
- Kaiming He, G. G. (s.f.). *Mask R-CNN: Facebook AI Research (FAIR)*. Obtenido de arXiv:1703.06870v.
- Shaoqing Ren, K. H. (2016). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. Obtenido de arXiv:1506.01497 .
- Snyder, C. a. (2019). *STREETS: A Novel Camera Network Dataset for Traffic Flow*. Chicago: H. Wallach and H. Larochelle and A. Beygelzimer.
- Wei Liu, D. A.-Y. (2016). *SSD: Single Shot MultiBox Detector*. Obtenido de arXiv:1512.0232.
- Wu, Y.-H. (2019, Noviembre 27). *CSPNet: A New Backbone that can Enhance Learning Capability of CNN*. Retrieved from arXiv from Cornell University:
<https://doi.org/10.48550/arXiv.1911.11929>
- YAML Language Development Team. (2021). *YAML Ain't Markup Language (YAML™)*.