UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Anonimización en imágenes mediante la aplicación de Redes Convolucionales.

Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

AUTOR/A: Rodríguez Sánchez, Álvaro

Tutor/a: Paredes Palacios, Roberto

CURSO ACADÉMICO: 2021/2022

Resum

L'objectiu del present treball és l'anonimització de dades en imatges mitjançant l'ús de *Machine Learning*. Avui dia les dades són un atribut intrínsec a les persones, gran part d'aquestes són de caràcter confidencial, per això actualment el concepte d'anonimització de dades juga un paper molt important. Gràcies a això, és possible la manipulació i el processament de les dades sense que la confidencialitat de les persones se'n vegi afectada.

Per assolir l'objectiu proposat, es farà prèviament un estudi sobre diferents tècniques en la detecció d'objectes dins d'imatges, per justificar finalment l'ús d'una estructura *YOLOv5*. Posteriorment es crearà un sistema que generarà dades sintètiques i etiquetades, i s'utilitzaran diferents tècniques de maneig d'imatge per abastar més rang d'il·luminacions i enfocament. Després d'això, es va assegurar una eina que facilita la creació de diferents topologies i la variació de paràmetres d'una *YOLOv5*; aquestes seran entrenades a partir de la generació de *datasets* sintètics per posteriorment detectar cares i matrícules en imatges reals i procedir a anonimitzar-les aplicant un difuminat sobre la zona de detecció. Finalment, s'estudiarà i analitzarà la confiança i el comportament de les diferents topologies a l'hora d'anonimitzar.

Paraules clau: Anonimització en imatges, Machine Learning, Visió per Computador, YOLO, detecció, cares, matricules.

Resumen

El objetivo del presente trabajo es la anonimización de datos en imágenes mediante el uso de *Machine Learning*. Hoy en día los datos son un atributo intrínseco a las personas, gran parte de éstos son de carácter confidencial, es por ello por lo que actualmente el concepto de anonimización de datos juega un papel muy importante. Gracias a ello, es posible la manipulación y el procesado de los datos sin que la confidencialidad de las personas se vea afectada.

Para lograr el objetivo propuesto, se hará previamente un estudio sobre distintas técnicas en la detección de objetos dentro de imágenes, para finalmente justificar el uso de una estructura *YOLOv5*. Posteriormente se creará un sistema que genere datos sintéticos y etiquetados, y se aplicará distintas técnicas de manipulación de imagen para abarcar un mayor rango de iluminaciones y enfoque. Tras ello, se utilizará una herramienta que facilita la creación de distintas topologías y variación de parámetros de una *YOLOv5*; éstas serán entrenadas a partir de la generación de *datasets* sintéticos para posteriormente detectarse caras y matrículas en imágenes reales y proceder a anonimizarlas aplicando un difuminado sobre la zona de detección. Finalmente se estudiará y analizará la confianza y el comportamiento de las distintas topologías a la hora de anonimizar.

Palabras clave: Anonimización en imágenes, Machine Learning, Visión por Computador, YOLO, detección, caras, matrículas.

Abstract

The objective of this project is the anonymization of data in images through the use of *Machine Learning*. Nowadays, data is an intrinsic attribute to people, a large part of these are confidential, that is why currently the concept of data anonymization plays a very important role. Thanks to this, it is possible to manipulate and process data without affecting the confidentiality of individuals.

To achieve the proposed objective, a study will be previously done on different techniques in objects detection within images, to finally justify the use of the *YOLOv5* structure. Subsequently, a system will be created that generates synthetic and labeled data, and different image manipulation techniques will be applied to cover a greater range of illumination and focus. After that, a tool will be used for the creation of different topologies and variation of parameters of a *YOLOv5*; these will be trained from the generation of synthetic *datasets* to later detect faces and license plates in real images and proceed to anonymize them by applying a blur on the detection area. Finally, the confidence and behavior of the different topologies when anonymizing will be studied and analyzed.

Key words: Anonymization in images, Machine Learning, Computer Vision, YOLO, detection, faces, license plates.

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Me gustaría transmitir mi más sincero agradecimiento a todos aquellas personas que me han ayudado tanto directa como indirectamente a lo largo de la realización del actual trabajo. En primer lugar, a mis tutores Roberto Paredes Palacios y Mercedes García Martínez, tanto por apoyar mi propuesta de trabajo, como por consolidarla ayudándome en su planificación, organización y dudas surgidas a lo largo de su desarrollo. En segundo lugar, a mi familia, tanto mi padre José Manuel, como su mujer Ornela, como mis hermanos José Manuel, Aleksandar y Lucía, que siempre han estado dándome apoyo anímico a pesar de que algunos de ellos han estado residiendo en el extranjero durante mi periodo universitario. A los nuevos compañeros que he conocido en el máster, con los que he superado las distintas asignaturas del mismo, apoyándonos y ayudándonos mutuamente tanto en el ámbito de estudio como en el ámbito personal. También, expresar mi más sentido agradecimiento al departamento de DSIC, y por ende a la UPV, por ofrecerme la oportunidad de realizar el MIARFID y acogerme dentro de sus aulas, haciéndome sentir como en casa.

A todos ellos, mil gracias.

CAPÍTULO 1
Introducción

1.1 ¿Qué son los datos personales?

Hoy en día los datos personales son un concepto inherente a las personas. Los datos personales son aquellos mediante los cuales se identifican de forma directa o indirecta a una persona. Algunos ejemplos de datos personales son el nombre/apellidos, la foto, el número de teléfono, la dirección, así como la dirección IP o el nombre de usuario (*nick*).

La recopilación, procesamiento y almacenamiento de datos personales está estrictamente regulado por el Reglamento General de Protección de Datos (RGPD) (adaptado al marco legal español con el nombre de *LOPDGDD*) [3]. Éste consiste en un marco legal de la Unión Europea, vigente desde el 25 de mayo de 2018, que rige la recopilación y el tratamiento de los datos personales de los usuarios. Por ende, es de extrema importancia conocer la metodología para recopilar, gestionar y almacenar dichos datos. Es aquí donde entra en juego el concepto de anonimización, puesto que gracias a él se cumplirían las medidas que garantizan la privacidad de los datos y su posterior explotación.

1.2 Detección de objetos

La detección de objetos es un área de gran importancia dentro de la visión por computador, ésta se encarga de detectar instancias de objetos visuales de una determinada clase (como personas, animales o coches) en imágenes digitales [27]. El objetivo de la detección de objetos es desarrollar modelos y técnicas computacionales que proporcionen una de las informaciones más básicas que necesitan las aplicaciones de visión por computador: ¿Qué objetos están y dónde? Como uno de los problemas fundamentales dentro de la visión por computador, la detección de objetos constituye la base de muchas otras tareas de visión por computador, como puede ser la segmentación de instancias [9] [2], el subtitulado de imágenes [13] o el seguimiento de objetos [12].

Desde el punto de vista de la aplicación, la detección de objetos puede agruparse en dos temas de investigación principales:

- **Detección general de objetos**, el cual pretende explorar los métodos de detección de diferentes tipos de objetos bajo un marco unificado para simular la visión y la cognición humanas (Figura 1.1).
- **Aplicaciones de detección**, hace referencia a la detección dentro de un escenario de aplicación concreto, como la detección de peatones, la detección de caras o la detección de textos (Figura 1.2).

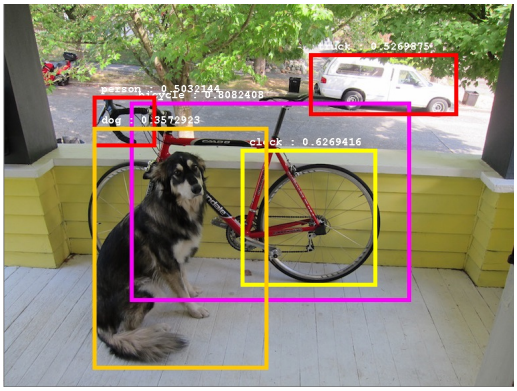


Figura 1.1: Detección de varios objetos distintos.
Source imagen

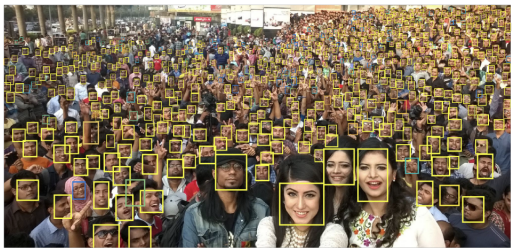


Figura 1.2: Detección de caras.
Source imagen

En los últimos años, el rápido desarrollo de las técnicas de aprendizaje profundo [26] ha aportado nuevos enfoques a la detección de objetos, lo que ha dado lugar a notables avances y la ha convertido en un gran área de investigación con una importancia sin precedentes. La detección de objetos se utiliza ampliamente en muchas aplicaciones del mundo real, como la conducción autónoma, la visión robótica o la videovigilancia, es por ello por lo que es de sumo interés su uso, desarrollo y estudio.

1.3 Motivación

El actual trabajo ha sido motivado por diversos aspectos. El primero de ellos es el entusiasmo y satisfacción personal de crear un sistema mediante el cual se detecte matrículas y caras dentro de imágenes para posteriormente anonimizarlos, y poder de esta forma explotar dichas imágenes en futuros procesos. Hoy en día se genera una gran cantidad de datos para los cuales se requiere alto tiempo de procesamiento.

El segundo motivo va relacionado con el ámbito profesional; la actual empresa para la que me encuentro trabajando, Pangeanic, está comercializando un producto basado en la anonimización de datos dentro de textos mediante el uso de *NLP* [1]. Es de especial interés que sobre los documentos de los que se extrae el texto a anonimizar, se detecte previamente ciertas imágenes para al igual que el texto, se puedan anonimizar aplicando un difuminado o eliminación sobre la zona en la que se encuentre. Es por ello que, este trabajo consiste en un paso y experimentación previa a dicho proceso; en él nos centraremos concretamente, como se ha mencionado al principio de esta sección, en matrículas y caras. Éstas son datos clave que identifican a un persona.

Finalmente, y no por ello menos importante, el último aspecto por el que me motiva la realización del actual proyecto es el de poner a prueba los conocimientos que se han adquirido en las distintas asignaturas cursadas a lo largo del Máster *IARFID*.

1.4 Objetivos

Con el presente trabajo se pretende investigar sobre distintas técnicas basadas en *Deep Learning* para la detección de objetos en imágenes. Seguidamente, aplicar la técnica que

[1] *Natural Language Processing*, campo de las ciencias de la computación, de la inteligencia artificial y de la lingüística que estudia las interacciones entre las computadoras y el lenguaje humano.

más se ajuste al caso de estudio, creando y entrenando para ello un modelo. Finalmente se aplicará una difuminación sobre las zonas detectadas para anonimizarlas.

Por tanto, los objetivos a abordar en el trabajo son:

1. Estudiar el estado del arte en la detección de objetos para justificar qué técnica llevar a cabo para la detección de caras y matrículas.
2. Desarrollar un sistema que genere un *dataset* sintético con matrículas y caras, junto con sus respectivas localizaciones de los *bounding boxes*.
3. Entrenar y evaluar distintas topologías de la técnica escogida, y analizar los resultados, las ventajas y desventajas de cada una de ellas.
4. Aplicar una técnica de difuminado sobre las zonas detectadas que asegure la anonimización.
5. Probar un modelo final y sólido en entornos realistas, como pueden ser documentos reales e incluso en adquisición de imagen en tiempo real.

1.5 Estructura de la memoria

La memoria se divide en distintos capítulos; en cada uno de ellos se hace énfasis en un determinado ámbito, estos capítulos son:

- **Primer capítulo** (actual), en él se expone un breve introducción a los datos personales y detección de objetos, además se habla sobre cuáles son los factores que motivan a la realización del presente trabajo y los objetivos a abordar.
- **Segundo capítulo**, se estudia el estado del arte de la detección de objetos en imágenes, explicando brevemente cada una de las estructuras existentes.
- **Tercer capítulo**, se explica las métricas fundamentales dentro de la detección de objetos, así como los *datasets* más relevantes; además, se justificará el uso de la estructura *YOLOv5*.
- **Cuarto capítulo**, se procede a explicar más profundamente la estructura *YOLOv5* y mostrar sus distintos tipos junto con sus respectivas precisiones, tiempo de detección y número de parámetros sobre uno de los conjuntos de datos presentados en el anterior capítulo.
- **Quinto capítulo**, se explica dos metodologías mediante las que se permite obtener un conjunto de datos etiquetado. A continuación, se expondrá la herramienta *YOLOv5* de Ultralytics la cual facilita el uso, entrenamiento y validación de diferentes topologías. Por último se muestra el desarrollo llevado a cabo para automatizar el entrenamiento, el reentrenamiento, la validación dentro de un mismo *script* único, y la creación de otro para proceder a la anonimización de las regiones que son reconocidas en etapa de predicción.
- **Sexto capítulo**, se muestran los diferentes experimentos que se realizan y los resultados obtenidos aplicando dos métodos diferentes para obtención de datos etiquetados. Esto ayuda a entender las ventajas y desventajas de cada una de las topologías.
- **Séptimo capítulo**, en él se cierra el trabajo desarrollado mediante una serie de conclusiones y de futuros trabajos.

CAPÍTULO 2
Estado del arte

En la presente sección se va a explicar y exponer el estado del arte en el ámbito de la detección de objetos dentro de imágenes. De esta forma, posteriormente en Capítulo 3 se justifica el uso de una de las topologías dentro del estado del arte. Tal y como se muestra en la Figura 2.1 existen multitud de estructuras y métodos para la detección de objetos. Puede apreciarse como los métodos previos a 2012 no hacian uso de *Deep Learning*, estos estaban basados en métodos más convencionales. A partir de 2012, tras la creación de la estructura *AlexNet*, los detectores comenzaron a basarse en técnicas de *Deep Learning*; el camino de los detectores se dividieron entonces en:

- *Two-stage detector* o detector de dos pasos, los cuales como su nombre indica, están formados por dos pasos dependiente el segundo del primero para extraer la información necesaria y detectar objetos en imágenes. Dentro de este grupo se encuentran estructuras como *RCNN*, *Fast RCNN* y *Fatser RCNN*.

- *One-stage detector* o detector de un pasos, salieron a la luz unos años más tarde que los primeros *One-stage detector*. Éstos unificar los 2 pasos necesarios de los *One-stage detector* para realizarlo todo en un único paso. Algunos de los más relevantes son *YOLO*, *SSD* y *RetinaNet*.

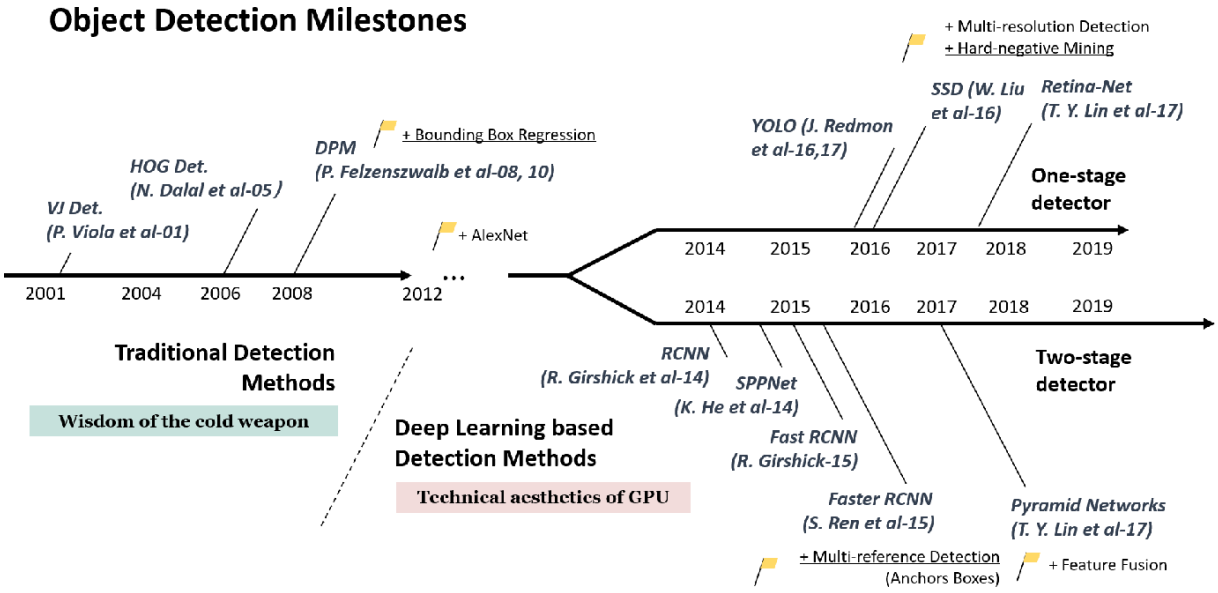


Figura 2.1: Evolución de los detectores de objetos.
Source imagen

Dependiendo de la precisión en la detección, el tiempo en detectar y carga computacional, existen varias estructuras, las cuales unas son más certeras que otras en alguno de los dichos puntos mencionados. El conjunto de técnicas que más se utilizan a día de hoy, y que resultan más punteras son:

- Region based CNN.
- You Only Look Once (YOLO).
- Single Shot MultiBox Detector (SSD).
- Retina-Net.

2.1 Region based CNN

En esta sección se expone el evolución desde la primera topología para la detección de objetos basada en *Deep Learning* (R-CNN) hasta su última mejora (Faster R-CNN).

Faster R-CNN consiste en una red convolucional profunda utilizada para la detección de objetos, ésta se muestra al usuario como una red unificada end-to-end. La red puede predecir con precisión y cierta rapidez la ubicación de diferentes objetos. Esta versión está basada en la red Fast R-CNN, la cual se basa a su vez en la topología R-CNN. A diferencia de estas dos últimas, la versión de Faster R-CNN, emplea menos tiempo a la hora de la detección de objetos en una imagen y consigue una mayor precisión [7] [6] [21], en la Tabla 2.1 se puede ver reflejado.

	Tiempo predicción (segundos)	**Carga computacional**	**mAP test Pascal VOC 2007(%)**	**mAP test Pascal VOC 2012 (%)**
R-CNN	40-50	Muy alto	58.5	53.3
Fast R-CNN	2	Alto	66.9	68.4
Faster R-CNN	0.2	Bajo	69.9	75.9

Tabla 2.1: R-CNN vs Fast R-CNN vs Faster R-CNN.

Es por ello por lo que la Faster R-CNN es un versión mejorada de la Fast R-CNN y a su vez de la R-CNN. El objetivo de las tres redes es la de detectar objetos en una imagen definiendo unos cuadros delimitadores (*bounding boxes* en inglés) alrededor de los objetos que detecte y para los que han sido entrenados.

2.1.1. R-CNN

En la Figura 2.2 se puede apreciar el procedimiento que sigue la R-CNN, a ésta le entra una imagen (227×227 píxeles) y se procesa por un mecanismo llamado «búsqueda selectiva» para extraer información sobre la región de interés. La región de interés puede representarse mediante los límites del rectángulo. Dependiendo del escenario, puede haber más de 2000 regiones de interés. Esta región de interés pasa por la CNN (formada por 5 capas convolucionales) para producir un vector de características de 4096 dimensiones. Estas características de salida pasan por tantos clasificadores SVM [1] como número de clases más 1 ($C + 1$ SVMs) para clasificar los objetos presentados bajo una región de interés

[1] Support Vector Machine

(ROI)[2] [7] y la localización de los *bounding boxes* asociados a cada uno de éstos. A la hora de establecer los ejemplos positivos y los ejemplos negativos, existen dos situaciones. Previamente se define el umbral de solapamiento IoU [3] como 0.5 en el proceso de ajuste, y entonces:

- Aquellas regiones propuestas por debajo del umbral, se consideran como negativas.
- Aquellas regiones propuestas por encima del umbral, se consideran como positivas.

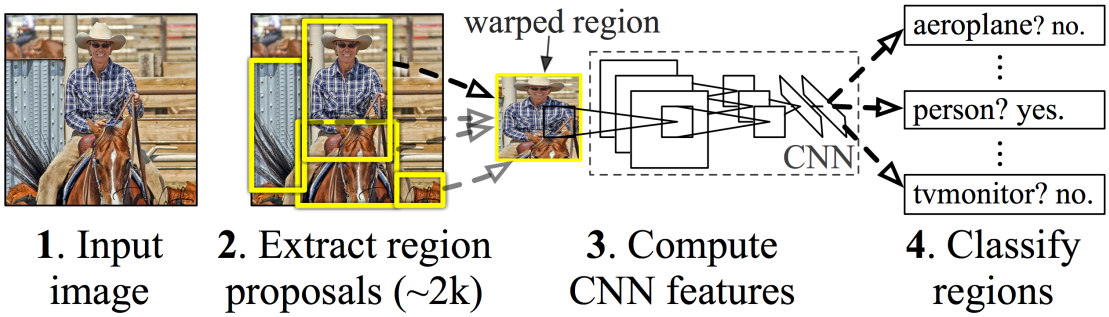


Figura 2.2: Procedimientos de la R-CNN.
Source imagen

2.1.2. Fast R-CNN

En el caso de la Fast R-CNN se extraen las características la totalidad de la imagen de entrada, y luego pasa a la «capa de agrupación de regiones de interés» (*RoI pooling layer* en inglés) para obtener un conjunto de características de tamaño fijo, siendo el tamaño de éste de 7x7x512 [6]. Posteriormente, cada uno de estos conjuntos de características se procesan por las capas de clasificación y regresión de los *bounding boxes*; dichos procesos pueden verse reflejados en la imagen 2.3.

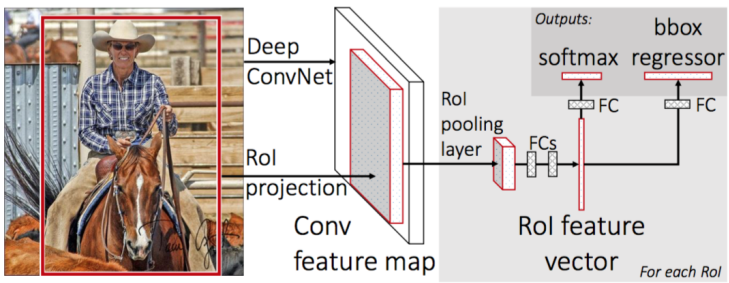


Figura 2.3: Estructura de la Fast R-CNN.
Source imagen

La *Fast R-CNN* es por lo tanto un proceso de entrenamiento de extremo a extremo (*end-to-end* en inglés) de una sola etapa que utiliza una pérdida multitarea en cada RoI etiquetado para entrenar conjuntamente la red. Otra mejora presenta *Fast R-CNN* respecto a *R-CNN* es que utiliza una capa *RoI pooling* para extraer un mapa de características de tamaño fijo a partir de propuestas de regiones de diferente tamaño; esta operación no necesita deformar las regiones y reserva la información espacial de las características de las propuestas

[2] Region of Interest
[3] Intertsection over Union

de regiones. Además, para realizar una detección rápida, se utiliza el *SVD* truncado, que acelera el paso hacia delante del cálculo de las capas totalmente conectadas. Gracias a estas características, *Fast R-CNN* resulta más rápida y precisa que la estructura *R-CNN*, tanto en el entrenamiento como en la detección.[6]

2.1.3. Faster R-CNN

Tal y como se presenta anteriormente en este apartado, la *Fast R-CNN* utiliza la búsqueda selectiva para proponer ROIs, esta búsqueda es lenta y necesita el mismo tiempo de ejecución que la red de detección. La *Fastest R-CNN* la sustituye por una RPN (red de propuesta de regiones, *Region Proposal Network* en inglés) que consiste en una red totalmente convolucional para predecir eficazmente las propuestas de región con una amplia gama de escalas y relaciones de aspecto. Éste nuevo módulo le dice a la red las regiones que resultan más relevantes y por ende, las regiones en las que tiene que centrarse y mirar.

Dicha RPN, tal y como se aprecia en la Figura 2.4, se caracteriza por:

1. Se desliza una pequeña red sobre la salida del mapa de características convolucional. En cada ubicación de la ventana deslizante, se predicen k propuestas de regiones, denominadas *anchors*.
2. La ventana deslizante es de tamaño $n \times n$ (*3 x 3* en el caso de la Figura 2.4).
3. Cada ventana deslizante se mapea a un vector de características con menor dimensión, usualmente 256 o 512.
4. Dicho vector de características se le pasa a dos capas *Fully Connected* (totalmente conectadas), a la capa de regresión de *bounding boxes* y a la capa de clasificación.

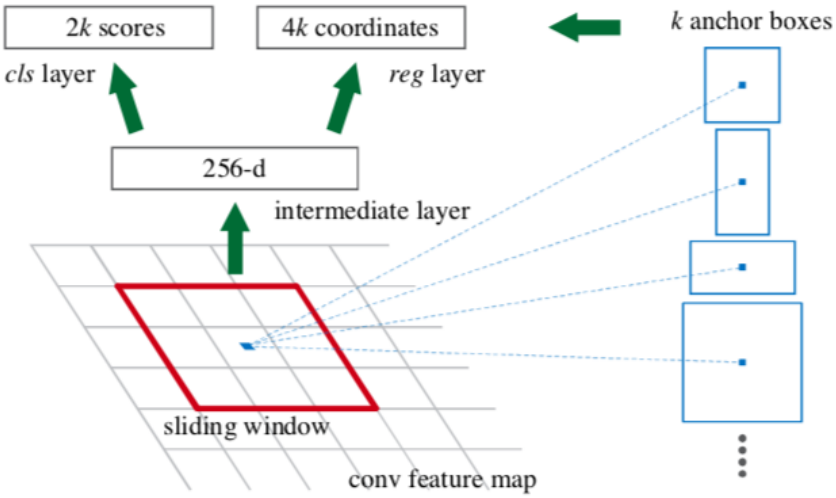


Figura 2.4: Estructura de la *Faster R-CNN*.
Source imagen

La *RPN* acelera la velocidad de generación de las propuestas de región porque comparte características convolucionales de imagen completa y un conjunto común de capas convolucionales con la red de detección; rompe de esta forma con el cuello de botella que presentaba la *Fast R-CNN*; pese a ello, sigue habiendo redundancia de cálculo en la siguiente fase de detección.

Tal y como se explica en esta sección, estas tres redes se basan en capas convolucionales para la extracción de características de una imagen, posteriormente hacen uso en redes *Fully Connected* para procesar dichas características y conseguir cuadrar los *bounding box* dentro de los objetos a reconocer. Pese a que distan unas de las otras en las metodologías que emplean para reconocer un objeto, hay una estructura común en ellas, y es que las tres son *Two-stage Detectors*, puesto que para hacer la detección de objetos tienen una primera fase en la que se extraer las regiones de interés convirtiendo éstas en un vector de características, y posteriormente otra fase en la que se trata dicho vector de características para ajustar un *bounding box* alrededor del objeto a reconocer.

2.2 You Only Look Once (YOLO).

En la actual sección se exponen los conceptos básicos y una idea general de una estructura YOLO, y de las diferentes modificaciones y mejoras que han ido teniendo a lo largo de sus sucesivas revisiones.

2.2.1. YOLOv1, estructura básica

A diferencia de lo que ocurre en los *Two-stage Detectors*, la estructura YOLO consiste en una *Single-stage Detector*; ésta trata la detección de objetos como un único problema de regresión, que va desde los píxeles de la imagen hasta las coordenadas de los *bounding boxes* y las probabilidades de clase. El sistema «sólo mira una vez» (You Only Look Once, en inglés (YOLO)) la imagen de entrada para predecir qué objetos están presentes y la región en la que están; todo ello en una única fase.

Como puede apreciarse en la Figura 2.5, una sola red convolucional será la que predice simultáneamente múltiples *bounding boxes* y las probabilidades de la clase asociada a cada uno de ellos. *YOLO* se entrena en imágenes completas y optimiza directamente el rendimiento de la detección.

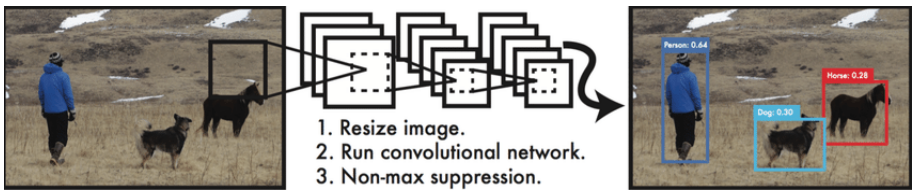


Figura 2.5: Estructura de la *Sistema detección YOLO.*.
Source imagen

Hay tres motivos principales por los que esta estructura resulta más interesante que las expuestas en la sección anterior (Two-stage Detectors) [18]:

- Al tratarse de un problema de regresión, la computación será más rápida y menos compleja, obteniendo resultados hasta el doble de mejores.
- Razona globalmente sobre la imagen cuando hace las predicciones. A diferencia de las técnicas basadas en ventanas deslizantes y en propuestas de regiones, la estructura YOLO ve la totalidad de la imagen durante el tiempo de entrenamiento y de test, por lo que codifica implícitamente la información contextual sobre las clases, así como su apariencia.
- Aprende representaciones generalizables de los objetos. Debido a esto, es menos probable que falle cuando se aplican nuevos dominios o a entradas inesperadas.

2.2.2. YOLOv2/9000 «Better, Faster, Stronger»

La segunda versión de YOLO, también conocida como YOLO9000 por tener la capacidad de diferenciar entre 9000 clases, mejora con respecto a la primera versión. Cabe destacar que es introducido el concepto de *anchor boxes* [19], áreas predefinidas para una imagen que ilustra la posición idealizada de los objetos que se van a detectar.

Se calcula la relación de solapamiento entre el *anchor box* previsto y el predefinido, para ello se utiliza la *IoU* [4] que hace función de umbral (*threshold* en inglés) y que se calcula tal y como se muestra en la Ecuación 2.1.

$$IoU = \frac{Area\_Interseccion}{Area\_Union} \tag{2.1}$$

Dependiendo si del valor del *IoU* entre los *anchor boxes* supera el umbral preestablecido, entonces serán tomados en consideración o no. Las *anchor boxes* no se calculan aleatoriamente; en su lugar, el algoritmo YOLO examina los datos de entrenamiento y realiza un *clustering* sobre ellos y de esta forma asegurar que los *anchor boxes* que se utilizan representan los datos sobre los que vamos a entrenar nuestro modelo, esto ayuda a mejorar mucho la precisión.

2.2.3. YOLOv3 «An Incremental Improvement»

La versión de *YOLOV3* consta de 75 capas convolucionales sin utilizar capas *fully connected* y *poolings*, concretamente hace uso de una *DarkNet53* [20]. De de esta forma reduce en gran medida el tamaño y el peso del modelo. Proporciona lo mejor de ambos mundos, es decir, el uso de modelos residuales (del modelo *ResNet*) para el aprendizaje de múltiples características con la red de pirámide de características (FPN), manteniendo tiempos reducidos en inferencia.

La *Feature Pyramid Network* (FPN) es un extractor de características que extrae diferentes tipos, formas y/o tamaños de características para una sola imagen. Concatena todas las características para que el modelo pueda aprender características locales y generales. Esta versión de *YOLO* ofrece, en inferencia, una precisión mayor que la que ofrecen las estructuras *RetinaNet* (concretamente *RetinaNet-50* y *RetinaNet-101*) invirtiendo menos tiempo, ésto se muestra en la Figura 2.6, donde la *YOLOv3* consigue una precisión de entre 51.5 % y 57.9 % (mAP50) empleando entre 22 y 51 milisegundos en inferencia sobre el conjunto de datos *COCO*.

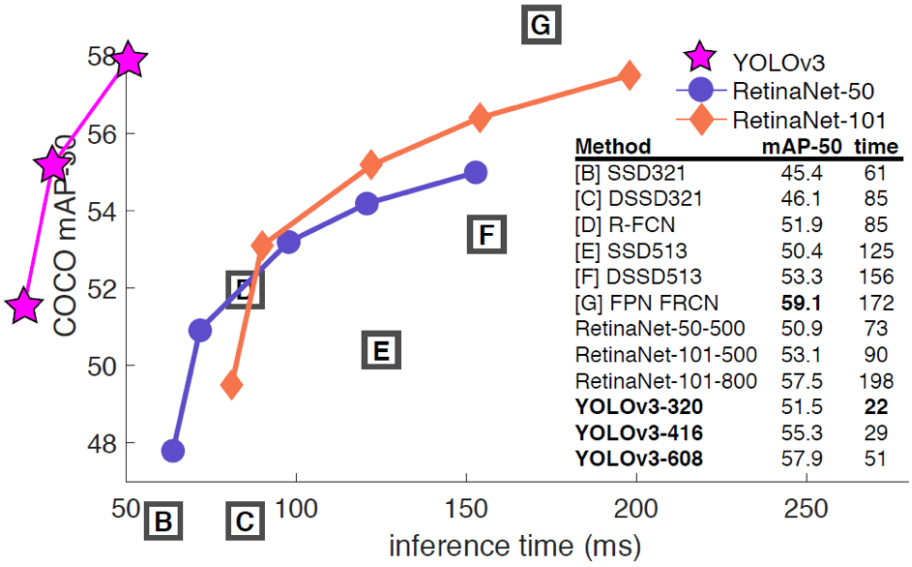
2.2.4. YOLOv4 «Optimal Speed and Accuracy of Object Detection»

La versión de *YOLOv4* salió a la luz en el año 2020, su modelo utiliza una *DarkNet53* al igual que su predecesor. Por otra parte, se introducen los conceptos de «*bag of freebies*» y «*bag of specials*» [1], éstos consisten en técnicas que aportan una mejora en el rendimiento del modelo sin aumentar el coste de inferencia y aumentan la precisión a la vez que el coste de cálculo respectivamente.

Bag of Freebies (BOF) se caracteriza por lo siguiente:

- Técnicas de aumento de datos: *Cutmix* (Cortar y mezclar múltiples imágenes que contienen objetos que queremos detectar), *Mixup* (Mezcla aleatoria de imágenes), *Cutout*, Aumento de datos en mosaico.

[4] Intersection over Union



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figura 2.6: Comparativa precisión y tiempo de detección *YOLOv3*.
Source imagen

- Pérdida por regresión de la *bounding box*: Experimentación de diferentes tipos de regresión de *bounding box*. Ejemplo: *MSE*, *IoU*, *CIoU*, *DIoU*.

- Regularización: Diferentes tipos de técnicas de regularización como *Dropout*, *DropPath*, *Spatial dropout*, *DropBlock*.

- Normalización: Se ha introducido la normalización cruzada de *mini-batches*, que ha demostrado aumentar la precisión. Junto con técnicas como la normalización por *batches* de iteración y la normalización GPU.

Por otro lado, Bag of Specials (BOS):

- Módulos de atención espacial (Spatial attention modules, SAM): Genera mapas de características utilizando la relación de características interespaciales. Ayudan a aumentar la precisión pero aumentan los tiempos de entrenamiento.

- Supresión no máxima (Non-max suppression, NMS): En el caso de los objetos agrupados, se obtienen múltiples *bounding boxes* como predicciones. La supresión no máxima reduce las cajas falsas o excesivas.

- Funciones de activación no lineales: Se han probado diferentes tipos de funciones de activación como por ejemplo: *ReLU*, *SELU*, *Leaky*, *Swish*, *Mish*.

- Conexiones de salto como las conexiones residuales ponderadas (Weighted Residual Connections, WRC) o las conexiones parciales entre etapas (Cross-Stage Partial connections, CSP).

Tal y como se aprecia en la Figura 2.7, esta nueva versión de *YOLO* consigue superar a su predecesor (*YOLOv3*) en términos de precisión, mientras que mantiene la velocidad de detección, consiguiendo una precisión de 43.5 % (AP) y 65.5 % (AP50), a una velocidad de 62 FPS sobre el *COCO dataset*.

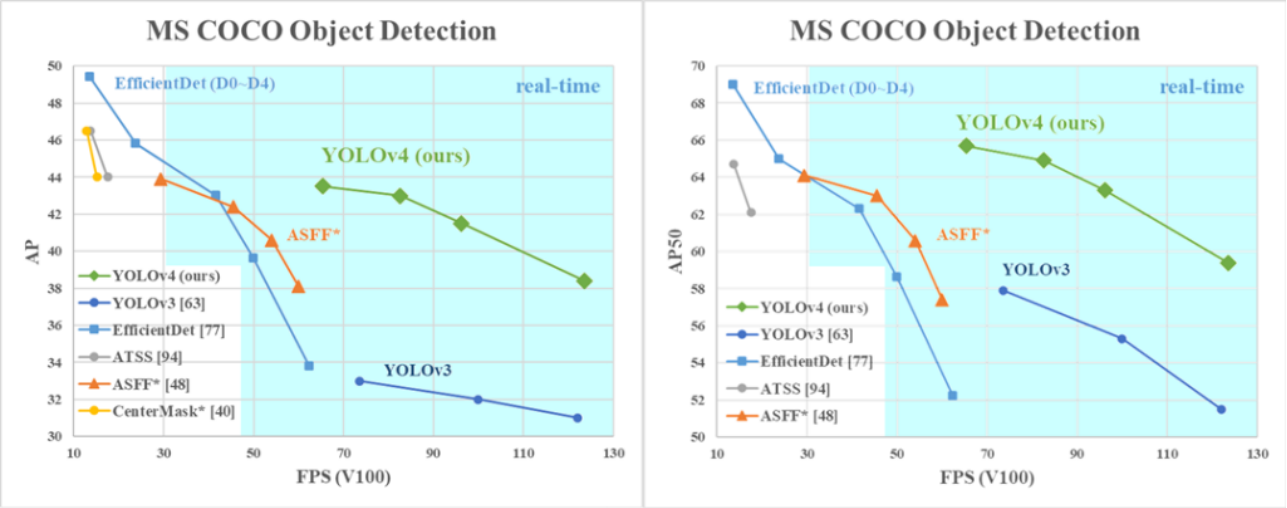


Figura 2.7: Comparativa precisión y tiempo de detección *YOLOv4*.
Source imagen

2.2.5. YOLOv5

YOLOv5 es el siguiente miembro de la familia *YOLO* lanzado en 2020 por la empresa *Ultranytics*, un tiempo más tarde de que saliese a la luz la *YOLOv4*. A día de hoy aún no hay un paper en el que se exponga de forma oficial los resultados de esta red; pero *Ultranytics* asegura que se obtiene una precisión de hasta 55 % (AP) en un tiempo de 20 milisegundos en inferencia sobre el conjunto de datos *COCO* [24]; lo cual supera a cualquier otra versión de *YOLO* en ambos campos.

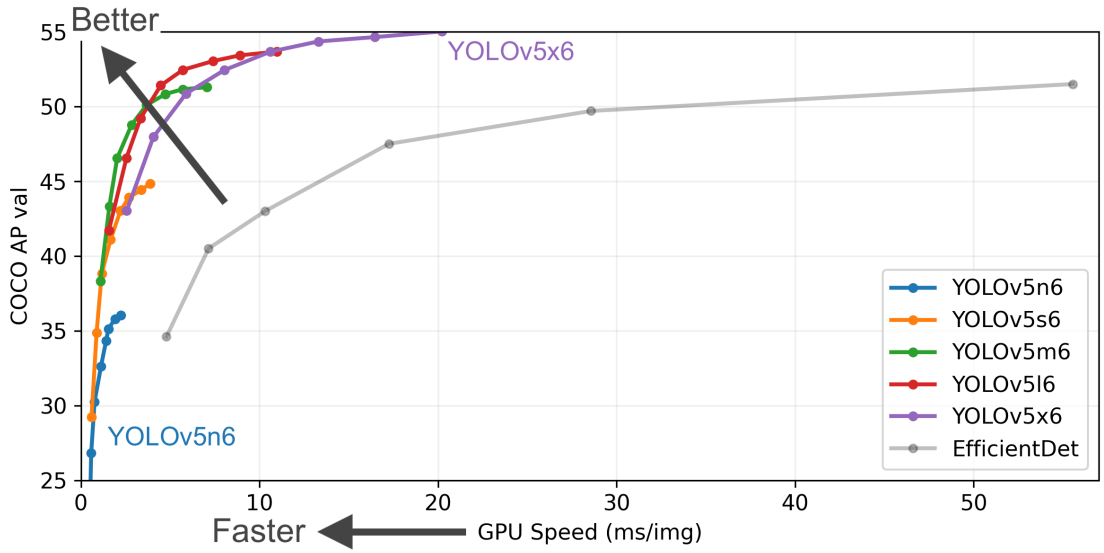


Figura 2.8: Comparativa precisión y tiempo de detección de las subversiones de la *YOLOv5*.
Source imagen

En la Figura 2.8 se puede observar la precisión (AP) y el tiempo de detección que obtiene cada una de las subversiones de la *YOLOv5*; estas subversiones distan una de otras en la cantidad de capas convolucionales que la forman. Cuando el modelo es más grande, se obtendrá mejor precisión a costa de tomar más tiempo en la detección.

En cuanto a las mejoras técnicas respecta, en comparación con la *YOLOv4*, la *YOLOv5* contiene las siguientes:

- Mejora en el *data augmentation* y cálculos de pérdidas.
- Uso de conexiones *Cross-Stage Partial* (CSP) en la red troncal.
- Uso de *Path Aggregation Network* (PAN) en el cuello del modelo.

2.3 Single Shot MultiBox Detector (SSD)

La estructura *SSD* consiste, al igual que *YOLO*, en una *Single-stage Detector*; propuesto por W. Liu et al. en 2015 [16]. Es el segundo detector de una etapa en la era del aprendizaje profundo.

La principal contribución del SSD es la introducción de las técnicas de detección multirreferencia y multirresolución, que mejora significativamente la precisión de detección de un *Single-stage Detector*, especialmente para algunos objetos pequeños. La SSD presenta ventajas tanto en términos de velocidad de detección como de precisión.

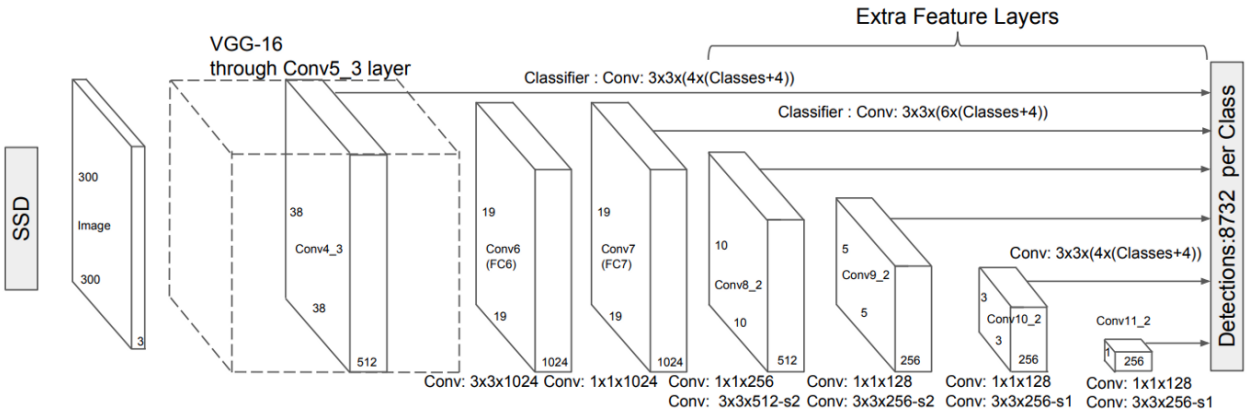


Figura 2.9: Estructura de la SSD.
Source imagen

El enfoque SSD se basa en una red convolucional *feed-forward* que produce una colección de tamaño fijo de *bounding boxes* y puntuaciones para la presencia de instancias de la clase de objeto en estas *bounding boxes*, seguido de un paso de *non-maximum supression* para producir las detecciones finales. Las primeras capas de la red se basan en una arquitectura estándar utilizada para la clasificación de imágenes de alta calidad (truncada antes de cualquier capa de clasificación), a la cual llamaremos la «red base». A continuación, añadimos una estructura auxiliar a la red para producir detecciones con las siguientes características clave:

- Mapas de características multiescala para la detección. Añadimos capas de características convolucionales al final de la «red base» truncada. Estas capas disminuyen de tamaño progresivamente y permiten predicciones de detecciones a múltiples escalas. El modelo convolucional para predecir las detecciones es diferente para cada capa de características.
- Predicciones convolucionales para la detección. Cada capa de características añadida puede producir un conjunto fijo de predicciones de detección utilizando un conjunto de filtros convolucionales. Éstos se indican en la parte superior de la arquitectura de la red *SSD*, tal y como se puede apreciar en la Figura 2.9. Para una capa de características de tamaño $m * n$ con p canales, el elemento básico para predecir los parámetros de una posible detección es un pequeño kernel de $3 * 3 * p$ que

produce una puntuación para una categoría o un desplazamiento de la forma en relación con las coordenadas de la caja por defecto. En cada una de las $m * n$ ubicaciones en las que se aplica el kernel, éste produce un valor de salida. Los valores de salida del desplazamiento de la *bounding box* se miden en relación con una posición de la caja por defecto en relación con cada ubicación del mapa de características.

- Cajas por defecto y relaciones de aspecto. Se asocia un conjunto de *bounding boxes* por defecto con cada celda del mapa de características, para múltiples mapas de características en la parte superior de la red. Los recuadros por defecto forman un mosaico del mapa de características de manera convolucional, de modo que la posición de cada recuadro en relación con su celda correspondiente es fija.

  En cada celda del mapa de características, se predicen los desplazamientos relativos a las formas de las cajas por defecto en la celda, así como las puntuaciones por clase que indican la presencia de una instancia de clase en cada una de esas *boxes*. En concreto, para cada casilla de las k que hay en un lugar determinado, se calculan c puntuaciones de clase y los 4 desplazamientos con respecto a la forma original de la casilla por defecto. Esto da lugar a un total de $(c+4)k$ filtros que se aplican alrededor de cada ubicación en el mapa de características, lo que produce $(c+4)kmn$ salidas para un mapa de características $m * n$.

  En la Figura 2.10 se ilustran los recuadros por defecto. Éstos por defecto son similares a los *anchor boxes* utilizados en *Faster R-CNN*, pero se aplican a varios mapas de características de diferentes resoluciones. Permitir diferentes formas de caja por defecto en varios mapas de permite discretizar eficientemente el espacio de posibles formas de caja de salida.

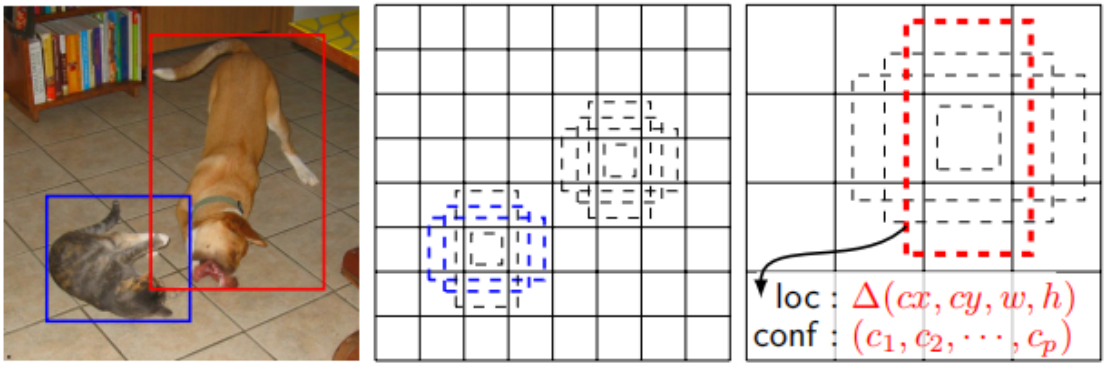


Figura 2.10: *Anchor boxes* predeterminados en SSD.
Source imagen

Finalmente, tras utilizar la SSD sobre el conjunto de datos *COCO* se consiguió una precisión de mAP@.5=46.5 % y mAP@[.5,.95]=26.8 %, a una velocidad de 59fps.

2.4 Retina-Net

A pesar de la alta velocidad y simplicidad, los *Single-stage Detector* (*SSD*, *YOLOv1* y *YOLO9000/v2*) estuvieron a la par de los *Two-stage Detectors* en lo que a la precisión respecta durante años. T.-Y. Lin et al. descubrieron los motivos y propusieron la estructura *RetinaNet* en 2017 [14].

Afirman que el gran desequilibrio existente entre las clases de primer plano y de fondo durante el entrenamiento de los detectores densos es la causa central. Debido a esto, se introduce en la *RetinaNet* una nueva función de pérdida denominada «*focal loss*» (pérdida focal en castellano) mediante la remodelación de la pérdida de la «*standard cross-entropy*» (entropía cruzada estándar) para que el detector se centre más en los ejemplos difíciles y mal clasificados durante el entrenamiento. La pérdida focal permite que los *Single-stage Detector* alcancen una precisión comparable a la de los de los detectores de dos etapas, manteniendo al mismo tiempo una velocidad de velocidad de detección.

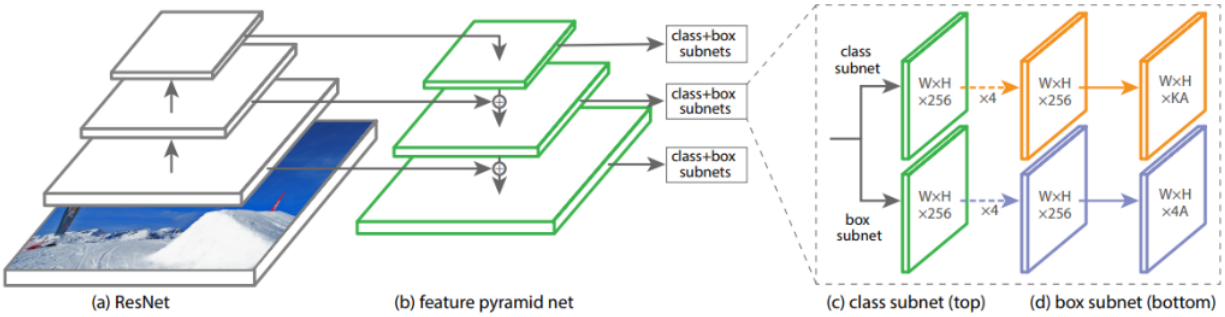


Figura 2.11: Topología de la RetinaNet
Source imagen

RetinaNet es una red única y unificada compuesta por una red troncal y dos subredes específicas para cada tarea. La red troncal se encarga de calcular un mapa de características convolucional sobre toda la imagen de entrada y es una red convolucional *off-the-self*. La primera subred realiza la clasificación convolucional de objetos en la salida de la red troncal; la segunda subred realiza la regresión convolucional de las *bounding boxes*. Las dos subredes presentan un diseño sencillo que proponemos específicamente para la detección densa en una etapa, tal y como se muestra en la Figura 2.11. Aunque hay muchas opciones posibles para los detalles de estos componentes, la mayoría de los parámetros de diseño no son especialmente sensibles a los valores exactos.

Finalmente, aplicando la estructura RetinaNet-101, se consigue una precisión en *COCO dataset* de mAP@.5=59.1 % y mAP@[.5, .95]=39,1 %.

CAPÍTULO 3
Introducción a la detección de objetos

A lo largo de este capítulo se explican las métricas usadas en la detección de objetos, la existencia de los principales *datasets* dentro de la detección de objetos, y finalmente el motivo por el cual se ha decidido escoger la estructura *YOLO* para la detección de imágenes (y posterior anonimización).

3.1 Métricas principales en la detección de objetos

Entre los diferentes conjuntos de datos anotados utilizados para la detección de objetos de objetos y por la comunidad científica, la métrica más métrica utilizada para medir la precisión de las detecciones es el AP (*Average Precision*) [17]. Antes de examinar las variaciones del AP, es de especial interés revisar algunos conceptos que se comparten entre ellos. Los más básicos son los que se definen a continuación:

- Verdadero positivo o ***True Positive*** (*TP*): detección correcta de un cuadro delimitador de verdad.
- Falso positivo o ***False Positive*** (*FP*): detección incorrecta de un objeto inexistente o una detección errónea de un objeto existente.
- Falso negativo o ***False Negative*** (*FN*): Un cuadro delimitador no detectado sobre el *ground-truth*.

Para definir estos términos, se necesita otra métrica de ayuda llamada *Intersection over Union* (IoU).

3.1.1. Intersection over Union (IoU)

Como ya se introdujo en la Sección 2.2.2 (Capítulo 2), esta métrica evalúa el grado de solapamiento de las *bounding boxes* entre el *ground-truth*(gt) y la predicción (pd); ambos pueden tener cualquier forma (caja rectangular, círculo o incluso una forma irregular). Se calcula como sigue:

$$IoU = \frac{Area\_Interseccion}{Area\_Union} = \frac{Area(gt \cap pd)}{Area(gt \cup pd)} \tag{3.1}$$

En la Figura 3.1 se puede apreciar de una manera más gráfica el cálculo de la *IoU*. El valor que se obtiene mediante dicha expresión (3.1) oscila entre 0 y 1, donde 0 indica que no hay

solapamiento y 1 significa un solapamiento perfecto entre *gt* y *pd*. La *IoU* es útil a través del *tresholding*, es decir, se necesita un umbral (α, digamos) y utilizando este umbral se puede decidir si una detección es correcta o no.

$$\text{IOU} = \frac{\text{area of overlap}}{\text{area of union}} =$$

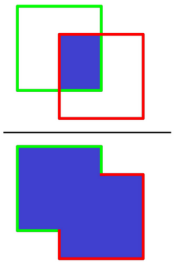


Figura 3.1: Intersection over Union.
Source imagen

Para la *IoU* teniendo en cuenta el umbral α, el *True Positive* es una detección para la cual $IoU(gt, pd) \geq \alpha$ y *False Positive* es una detección para la cual $IoU(gt, pd) < \alpha$. El *False Negative* es una una detección que se encuentra en el *ground-truth* pero cuya detección falla, por lo que se considera como negativa.

3.1.2. Precisión y Recall

La precisión es el grado de exactitud del modelo a la hora de identificar sólo los objetos relevantes (expresión 3.2). Es la proporción de TPs (*True Positives*) sobre todas las detecciones realizadas por el modelo.

$$P = \frac{TP}{TP + FP} = \frac{TP}{All\_detections} \tag{3.2}$$

Por otro lado, el *Recall* mide la capacidad del modelo para detectar todas los valores del *ground-truth*; consiste en la proposición de TPs (*True Positives*) entre todas los valores del *ground-truth* (expresión 3.5).

$$R = \frac{TP}{TP + FN} = \frac{TP}{All\_ground\_truths} \tag{3.3}$$

Se dice que un modelo es bueno si tiene una alta precisión y un alto *Recall*. Un modelo perfecto tiene cero FNs y cero FPs (es decir, precisión=1 y recall=1); normalmente no es posible conseguir un modelo perfecto.

3.1.3. Curva Precisión-Recall

Al igual que la IoU descrita en esta sección, la puntuación de confianza también depende del umbral. Aumentar el umbral de confianza significa que el modelo no detectará más objetos (más FNs y, por lo tanto, bajo *recall* y alta precisión), mientras que una puntuación de confianza baja denota que el modelo obtiene más FPs (por lo tanto, baja precisión y alta recuperación). Esto significa que es necesario desarrollar algún tipo de compensación para la precisión y el *recall*.

La curva de precisión-recall (PR) es un gráfico de la precisión y la recuperación con distintos valores de confianza (Figura 3.2); ésta puede ser de utilidad para decidir qué umbral es el óptimo dependiendo del objetivo a tratar.

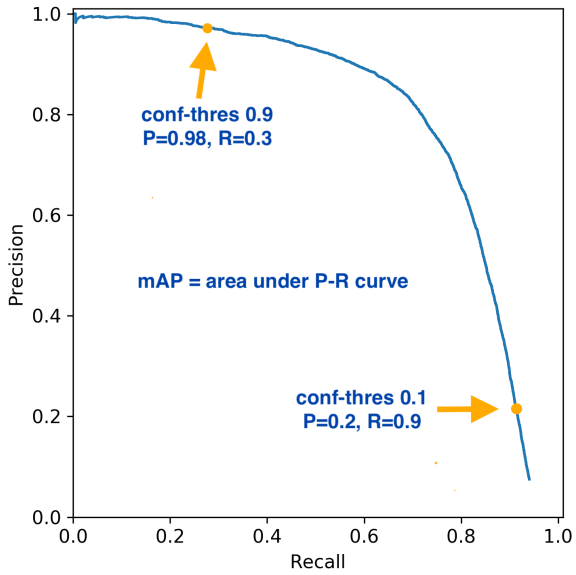


Figura 3.2: Ejemplo de curva PR.
Source imagen

3.1.4. Average Precision (AP)

AP@α consiste en el Área Bajo la Curva de Precisión-Recall (AUC-PR) evaluada en el umbral α IoU. Formalmente, se define como:

$$AP@\alpha = \int_0^1 p(r)dr \tag{3.4}$$

Una valor alto en *AUC-PR* significa un alto *recall* y una alta precisión. Normalmente, la curva PR es un gráfico en forma de zig-zag. Esto significa que no es monótonamente decreciente; esta propiedad se elimina utilizando métodos de interpolación como son:

- Método de interpolación de 11 puntos o *11-point interpolation method*.
- Enfoque de interpolación de todos los puntos o *All-point interpolation approach*.

3.1.5. Mean Average Precision (mAP)

El *Mean Average Precision*[1] se calcula individualmente el *Average Precision* (AP) para cada clase. Esto significa que hay tantos valores de PA como el número de clases. Estos valores de PA se promedian para obtener la métrica: Precisión media (mAP). Precisamente, la Precisión Media (mAP) es la media de los valores AP sobre todas las clases, y se define como:

$$mAP@\alpha = \frac{1}{n}\sum_{i=1}^{n} AP_i \tag{3.5}$$

3.1.6. F1 score

El objetivo de la *F1 score* es combinar las métricas de precisión y *recall* en una sola métrica. Al mismo tiempo ésta está diseñada para funcionar bien con datos desequilibrados, ésta

[1]Microsoft COCO, calculó el PA de una categoría/clase determinada en 10 IoU diferentes que van del 50 % al 95 % con un tamaño de paso del 5 %, normalmente denotado como AP@[.50:.5:.95]; esta es la notación que se muestra en el Capítulo 2 del presente trabajo.

se calcula como muestra en la ecuación 3.6.

$$f1\_score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.6}$$

Dado que el *F1 score* es una media de Precisión y *Recall*, significa que ésta da el mismo peso a Precisión que al *Recall*:

- Un modelo obtendrá un *F1 score* alto si tanto la Precisión como la *Recall* son altas.
- Un modelo obtendrá un *F1 score* bajo si tanto la Precisión como la *Recall* son bajas.
- Un modelo obtendrá un *F1 score* medio si uno de los valores de Precisión y *Recall* es bajo y el otro es alto.

3.2 Datasets principales en la detección de objetos

La creación de conjuntos de datos amplios con el mínimo sesgo posible es fundamental para desarrollar algoritmos avanzados de visión por ordenador. En el campo de la detección de objetos, en los últimos 10 años se han publicado una serie de conjuntos de datos y puntos de referencia muy conocidos, entre los que se incluyen:

- PASCAL VOC (por ejemplo, VOC2007, VOC2012).
- Reconocimiento visual a gran escala de ImageNet, como ILSVRC2014.
- MS-COCO.

3.2.1. Pascal VOC

El concurso *PASCAL Visual Object Classes* (VOC) (de 2005 a 2012) [5] es una de las competiciones más importantes en la comunidad de visión por ordenador. Hay múltiples tareas en PASCAL VOC, incluyendo la clasificación de imágenes, la detección de objetos y la segmentación semántica.

Figura 3.3: Algunas imágenes del conjunto de datos *PASCAL VOC*.
Source imagen

Dentro de la detección de objetos se utilizan principalmente dos versiones de Pascal-VOC: VOC07 y VOC12, donde la primera consta de 5000 imágenes (para training) + 12000

objetos anotados, y la segunda consta de 11000 imágenes (para training) + 27000 objetos anotados. El *dataset* esta formado por objetos de la vida cotidiana de 20 clases distintas (Persona: persona; Animal: pájaro, gato, vaca perro, caballo, oveja; Vehículo: avión, bicicleta, barco, autobús, coche moto, tren; Interior: botella, silla, mesa de comedor, maceta planta, sofá, televisor/monitor), tal y como puede apreciarse en la Figura 3.3.

En los últimos años, el VOC ha ido pasado gradualmente de moda y se ha convertido en un banco de pruebas para la mayoría de los nuevos detectores.

3.2.2. ILSVRC

El *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) [22] ha impulsado el estado del arte en la detección de objetos genéricos. El ILSVRC se ha organizado cada año desde 2010 hasta 2017. Contiene un desafío de detección utilizando imágenes de *ImageNet* [4]. El conjunto de datos de detección del ILSVRC contiene 200 clases de objetos visuales. En la Figura 3.4 se muestra algunos tipos de imágenes que contiene *Imagenet*

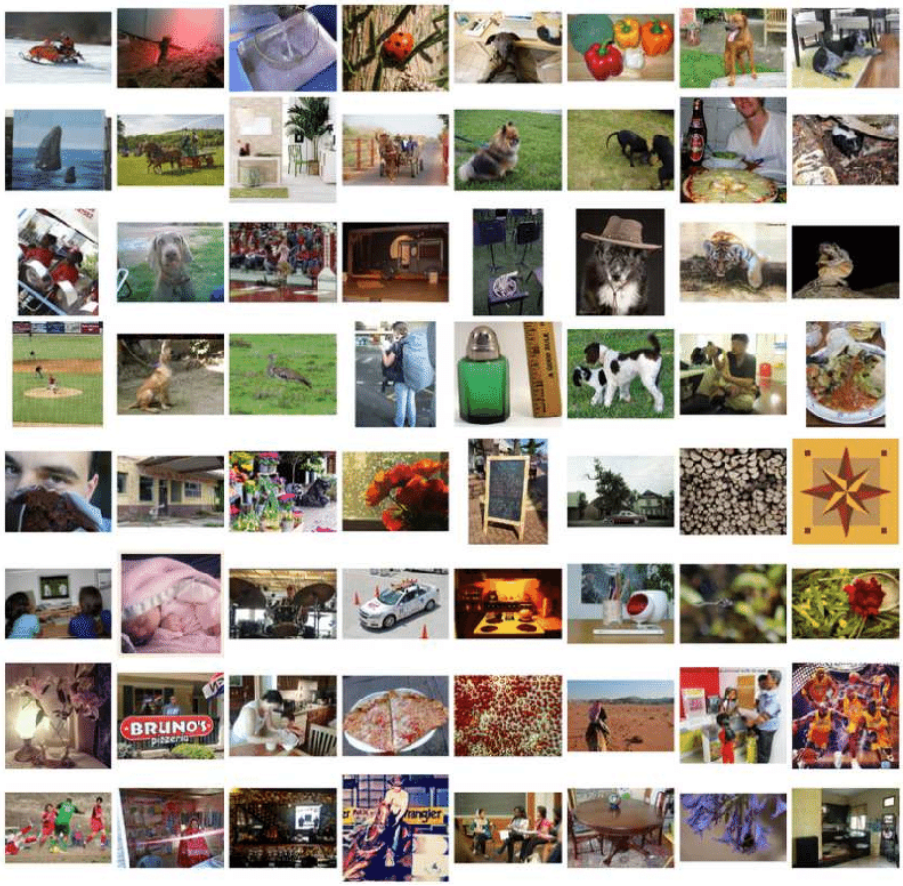


Figura 3.4: Algunas imágenes del conjunto de datos *Imagenet*.
Source imagen

El número de sus imágenes/objetos es dos órdenes de magnitud mayor que VOC. Por ejemplo, ILSVRC-14 contiene 517000 imágenes y 534000 objetos anotados.

3.2.3. MS-COCO

Microsoft-Common Objects in Context (MS-COCO) es el conjunto de datos más utilizado en la actualidad, convirtiéndose en el estándar de facto para la comunidad de detección

de objetos [15]. La competición anual basada en este *dataset* se celebra desde 2015; está compuesto por menos número de categorías de objetos que *ILSVRC*, pero más instancias de de objetos. Por ejemplo, *MS-COCO-17* contiene 164000 imágenes y 897000 objetos anotados de 80 categorías. En la Figura 3.5 se puede visualizar algunos de sus imágenes.

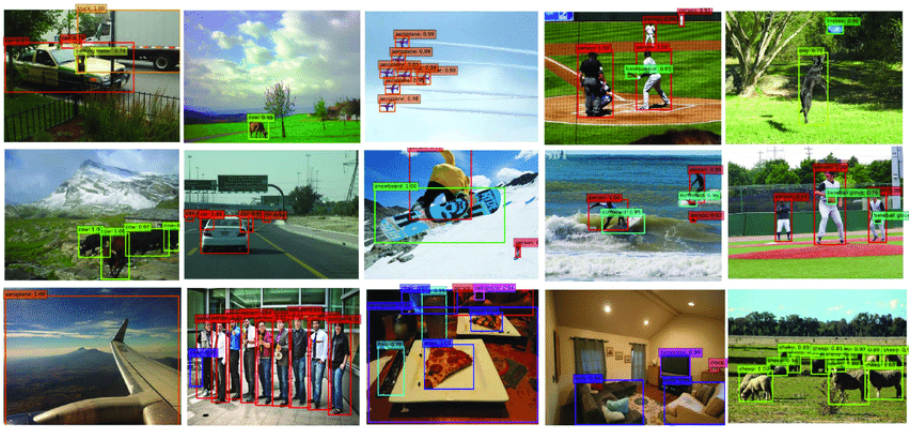


Figura 3.5: Algunas imágenes del conjunto de datos *MS-COCO*.
Source imagen

En comparación con *PASCAL VOC* e *ILSVRC*, el mayor avance de *MS-COCO* es que, además de las anotaciones de los *bounding boxes*, cada objeto se etiqueta adicionalmente mediante la segmentación por instancia para ayudar a una localización precisa. Además, *MS-COCO* contiene más objetos pequeños (cuya área es inferior al 1 % de la imagen) y más objetos densamente localizados que *VOC* e *ILSVRC*.

3.2.4. Open Images

En el año 2018 se presenta el desafío Open Images Detection (OID), siguiendo a MS-COCO pero a una escala sin precedentes. Hay dos tareas en Open Images:

1. La detección de objetos estándar.
2. La detección de relaciones visuales que detecta objetos emparejados en relaciones particulares.

Para la tarea de detección de objetos, el conjunto de datos consta de 1910000 imágenes con 15440000 cuadros delimitadores anotados en 600 categorías de objetos.

Dataset	train		validation		trainval		test	
	images	objects	images	objects	images	objects	images	objects
VOC-2007	2,501	6,301	2,510	6,307	5,011	12,608	4,952	14,976
VOC-2012	5,717	13,609	5,823	13,841	11,540	27,450	10,991	-
ILSVRC-2014	456,567	478,807	20,121	55,502	476,688	534,309	40,152	-
ILSVRC-2017	456,567	478,807	20,121	55,502	476,688	534,309	65,500	-
MS-COCO-2015	82,783	604,907	40,504	291,875	123,287	896,782	81,434	-
MS-COCO-2018	118,287	860,001	5,000	36,781	123,287	896,782	40,670	-
OID-2018	1,743,042	14,610,229	41,620	204,621	1,784,662	14,814,850	125,436	625,282

Figura 3.6: Características de cada *dataset*.
Source imagen

Para finalizar esta sección, a modo de resumen, en la Figura 3.6 se recoge las principales características de cada uno de los *dataset* mostrados.

3.3 ¿Por qué YOLOv5?

Como se observa a lo largo del Capítulo 2, tanto los métodos basados en *Two-stage detector* como los *One-stage detector* obtienen distintos resultados sobre el conjunto de datos *COCO* y además lo hacen con diferente velocidad. En la Tabla 3.1 se puede observar la velocidad de procesamiento en tiempo de detección y la precisión obtenida sobre el conjunto de datos *COCO* Y *VOC07*.

	MS-COCO		*PASCAL VOC07*	
	mAP@.5	**mAP@[.5,.95]**	**mAP@[.5,.95]**	**Velocidad de detección (FPS)**
R-CNN	-	-	33.7 % - 58.5 %	0.5
Fast R-CNN	-	-	70.0 %	5 - 7
Faster R-CNN	42.7 %	21.9 %	73.2 %	17
YOLO	65.5 % - -	43.5 % - -	- 52.7 % 63.4 %	62 155 45
SSD	6.5 %,	26.8 %	76.8 %	22 - 59
RetinaNet	59.1 %	39.1 %	81.2 %	-

Tabla 3.1: Comparación de las distintas estructuras en detección de objetos.

Los resultados que se muestran en dicha tabla (Tabla 3.1) sobre la precisión (mAP) y los *fps* pueden variar en función de la topología red que se utilice de fondo (como por ejemplo *Inception*, *Darknet*, *AlexNet*, etc.). Además, el tiempo de detección depende de si se hace sobre *CPU* o sobre *GPU* (y también del modelo de *GPU*).

Teniendo en cuenta los resultados de la Tabla 3.1 y las diferentes estructuras que se han expuesto en el capítulo referente al estado del arte, se ha decidido que la estructura que se va a utilizar para la detección de caras y matrículas (y su posterior anonimización) es la estructura *YOLO*, concretamente la *YOLOv5*.

Los motivos por los que se justifica su uso son los siguientes:

1. **Buena precisión**. Pese a que en la tabla sea la *RetinaNet* la que obtiene mayor precisión sobre el conjunto de datos *PASCAL VOC07*, *YOLO* la supera en los resultados que se obtienen en el *COCO dataset*. Dicho *dataset*, tal y como se ha explicado en el sección anterior, hay objetos cuya área son inferior al 1 % del tamaño de la imagen, lo cual resulta de mayor interés en el presente caso de estudio.

2. **Rapidez**. En cuanto a velocidad de detección, la estructura *YOLO* es la que resulta victoriosa. Esto resulta de especial interés puesto que, cuando el detector que se va a entrenar (se explica en el siguiente capítulo) sea acoplado a uno de los servicios de anonimización de *Pangeanic*, es necesario que su funcionamiento sea óptimo, minimizando de esta forma el tiempo de detección y posterior anonimización.

Una vez ya decidido que se va a hacer uso de la estructura *YOLOv5*, en el próximo capítulo se muestra la parte teórica de ésta, para posteriormente en el Capítulo 5 se pueda aplicar y entender los conceptos sin ningún tipo de problemas.

CAPÍTULO 4
You Only Look Once, YOLOv5

A lo largo de este capítulo se explica la topología de la *YOLOv5*, el procedimiento que se lleva a cabo en la etapa de *training*, y finalmente se exponen los distintos tipos existentes de YOLOv5 y sus resultados asociados al entrenamiento sobre el conjunto de datos *COCO*.

4.1 Topología

Tal y como se explica en la Sección 2.2, *YOLO* es un algoritmo de detección de objetos basado en regresión lineal; es decir, se introduce una imagen dentro de la red profunda, y *YOLO* devuelve la predicción de la clasificación y la información de localización de los objetos (*bounding boxes*) de acuerdo con el cálculo de la función de pérdida, por lo que hace que el problema de detección de objetivos se transforme en una solución de problemas de regresión [20].

YOLOv5 [24] se basa en la arquitectura de detección *YOLO* y utiliza una excelente estrategia de optimización del algoritmos en el campo de las *CNNs* como:

- Auto-aprendizaje de los *bounding box anchors*.
- *Data Augmentation* basado en mosaico; éste consiste en una extensión del algoritmo basado en *CutMix*, mediante el que se crea una imagen formada por imágenes del *dataset* [8].
- Uso de *Cross Stage Partial Network* (CSPNet) diseñada para atribuir el problema a la información duplicada del gradiente dentro de la optimización de la red. Gracias a ella la complejidad se puede reducir en gran medida, manteniendo la precisión.

La arquitectura YOLOv5 consta de cuatro partes principales:

- **Entrada** o ***input***.
- ***Backbone network***.
- ***Neck Network***.
- **Salida o *Output***.

4.1.1. Entrada o *input*

El terminal de entrada contiene principalmente el pre-procesamiento de los datos, incluyendo el *data augmentation* basado en mosaico y el relleno adaptativo de la imagen

(conocido como *image filling*) tal y como puede visualizarse en la Figura 4.1. Para adaptarse a diferentes conjuntos de datos, integra el cálculo adaptativo del *anchor frame* en la entrada, de modo que puede establecer automáticamente el tamaño inicial del *anchor frame* cuando el conjunto de datos cambie.

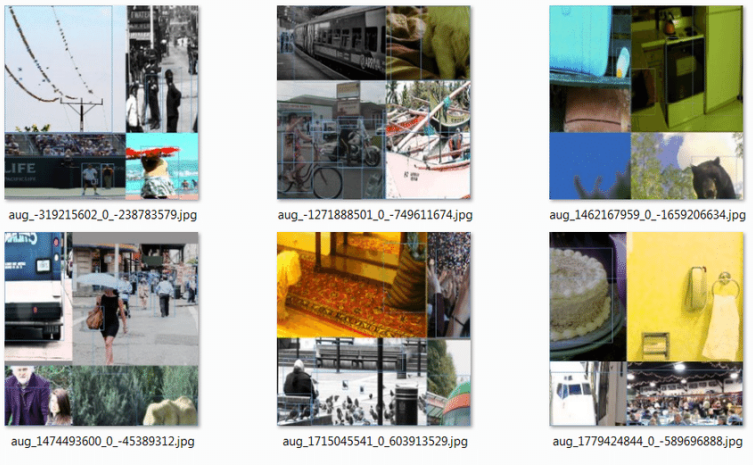


Figura 4.1: *Data Augmentation* de mosaico
Source imagen

4.1.2. *Backbone network*

Está formada por una *CSPNet* (mejora que se ha comentado anteriormente) y un *Spatial Pyramid Pooling* (SPP) para extraer mapas de características de diferentes tamaños de la imagen de entrada mediante convolución múltimple y *pooling*.

Cross Stage Partial Network (CSPNet)

Como se ha explicado a lo largo del trabajo, la *YOLO* está formada por redes neuronales profundas. En el estado del arte de las distintas versiones de *YOLO* dentro de las redes neuronales profundas, se utiliza la arquitectura *DenseNet*.

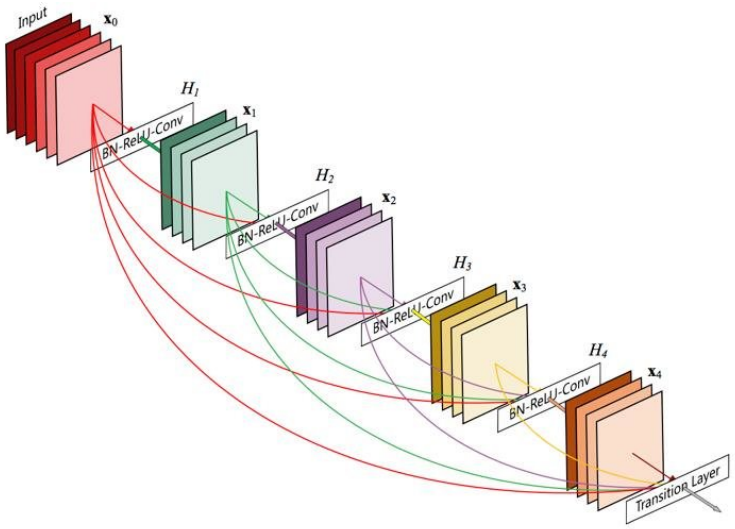
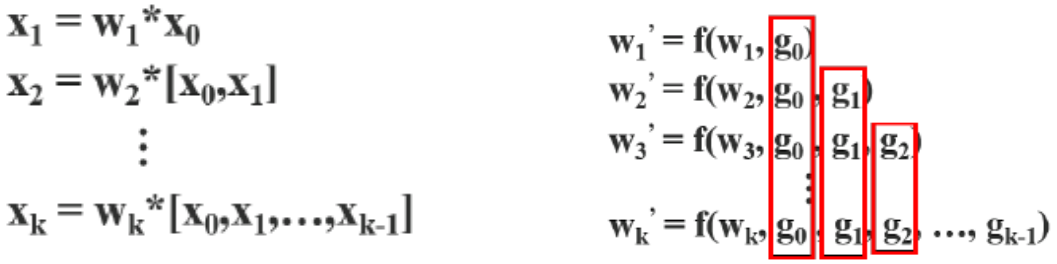


Figura 4.2: Arquitectura de un bloque denso.
Source imagen

En *DenseNet*, dentro de cada bloque denso, la salida de la i-ésima capa densa se concatena con la entrada de la i-ésima capa. Este resultado concatenado se convierte en la entrada de la capa densa (i+1); tanto en la Figura 4.2 como en la Figura 4.3 puede verse reflejado esto. De esta forma, uniendo secuencialmente varios bloques densos se construye una *Densenet* [11].



(a) Concatenaciones. **(b)** *Backpropagation* del gradiente.

Figura 4.3: Operaciones internas en un bloque denso.

Como puede observarse en la Figura 4.3b existe gran cantidad de información del gradiente que se reutiliza para actualizar los pesos de las diferentes capas densas. Esto hace que las diferentes capas densas aprendan repetidamente información copiada del gradiente; es aquí donde entra en juego la *CSPNet*.

El principal cometido de la *CSPNet* es la de evitar esa información duplicada/copiada; ésta separa el mapa de características de la capa base en dos partes, una de las cuales pasa por un bloque denso y una capa de transición; la otra parte se combina con el mapa de características transmitido a la siguiente etapa. De esta forma los gradientes procedentes de las capas densas se integran por separado. En la Figura 4.4 puede apreciarse el estado anterior de un bloque denso (4.4a), y la evolución con esta nueva característica (4.4b).

Finalmente se obtiene la *CSPDenseNet*, la cual conserva las ventajas de las características de reutilización de *DenseNet*, pero al mismo tiempo evita una cantidad excesiva de información de gradiente duplicada al truncar el flujo del gradiente.

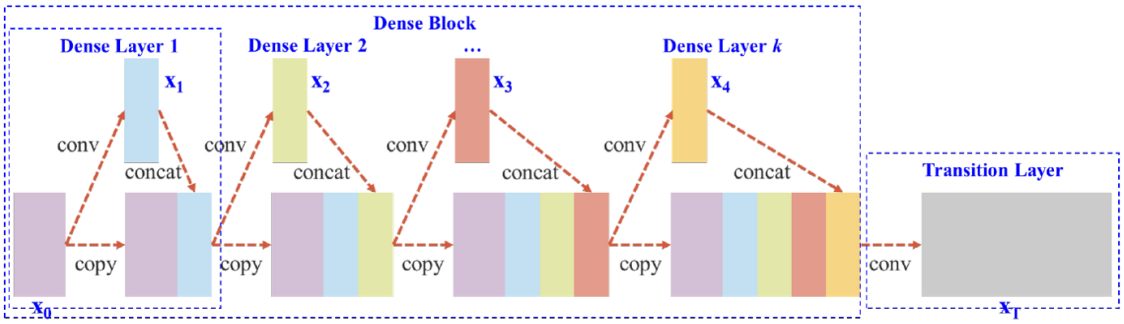
Spatial Pyramid Pooling (SPP)

Spatial Pyramid Pooling (SPP) consiste una capa de agrupación o *pooling* que elimina la restricción de tamaño fijo de la red, es decir, una *CNN* no requiere una imagen de entrada de tamaño fijo. En concreto, se añade una capa *SPP* sobre la última capa convolucional. Dicha capa extrae tres mapas de características a diferentes escalas y posteriormente genera una salida de longitud fija con dichos mapas (Figura 4.5), que luego se introducen en capas totalmente conectadas (u otros clasificadores) [10].

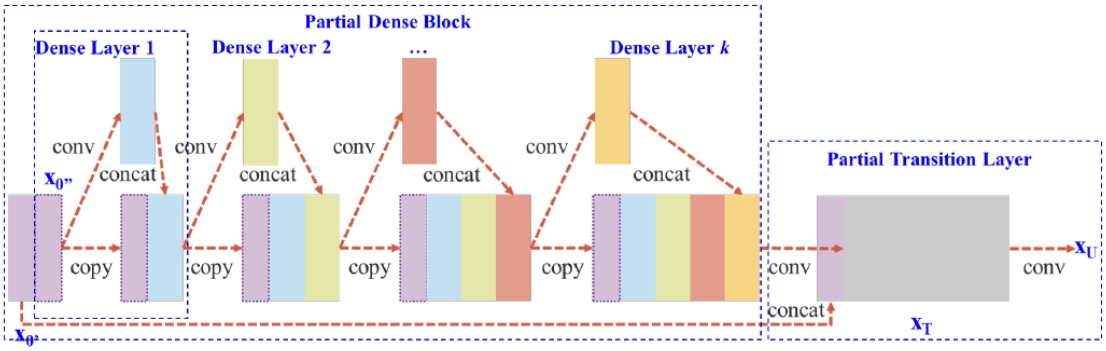
En definitiva, lo que se realiza es una agregación de información en la etapa más profunda de la jerarquía de la red (entre las capas convolucionales y las capas totalmente conectadas) para evitar la necesidad de recortar o deformar al principio la imagen de entrada.

4.1.3. *Neck Network*

En esta parte de la *YOLOv5*, se hace uso de las estructuras de características FPN y PAN, las cuales se describen a continuación.



(a) Bloque denso en *Densenet*.



(b) Bloque denso en *Densenet* con *CSP*.

Figura 4.4: *CSP* aplicada a *Densenet*.

Feature Pyramid Network (FPN)

Tal y como se expone en la Sección 2.2.3, el propósito de la *Feature Pyramid Network* (FPN) es combinar el mapa de características de alto nivel con el mapa de características de bajo nivel de una manera determinada para obtener el mapa de características con el equilibrio de la resolución y la información semántica y mejorar de esta forma en la detección [20].

La FPN se divide principalmente en dos procesos:

1. ***Bottom-up Pathway***: como se muestra en la parte izquierda de la Figura 4.6, se obtienen cierto número diferente de mapas de características reduciendo cada vez su tamaño.

2. ***Top Down Pathway* y *Lateral Connection***: representado en la derecha y en el centro de la Figura 4.6 respectivamente. Se obtiene un nuevo mapa de características el cual se realiza con un *upsampling* del mapa de características de la etapa actual a la vez que se le suma el mapa de características de la etapa anterior del proceso *Bottom-up*.

Path Aggregation Network (PAN)

PAN toma un camino adicional del *Bottom-up Pathway* al *Top Down Pathway* que realiza la *FPN* tal y como se muestra en la Figura 4.7 representado por las flechas discontinuas de color rojo y verde. Esto ayuda a acortar ese camino utilizando conexiones laterales directas desde las capas inferiores a las superiores. Esto se denomina conexión «*shortcut*», que sólo tiene unas 10 capas.

Teniendo en cuenta estas dos estructuras (*FPN* y *PAN*), se consigue en la parte *Neck Network* de la *YOLOv5*, se extraigan las características de diferentes tipos, formas y tamaños

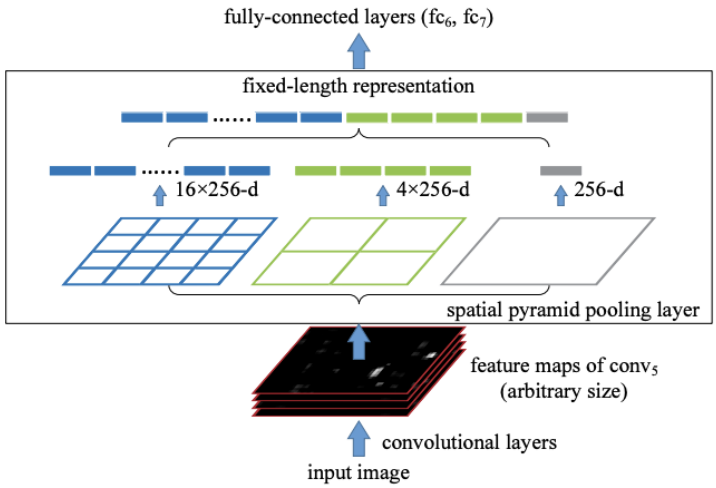


Figura 4.5: Representación gráfica del *Spatial Pyramid Pooling*.
Source imagen

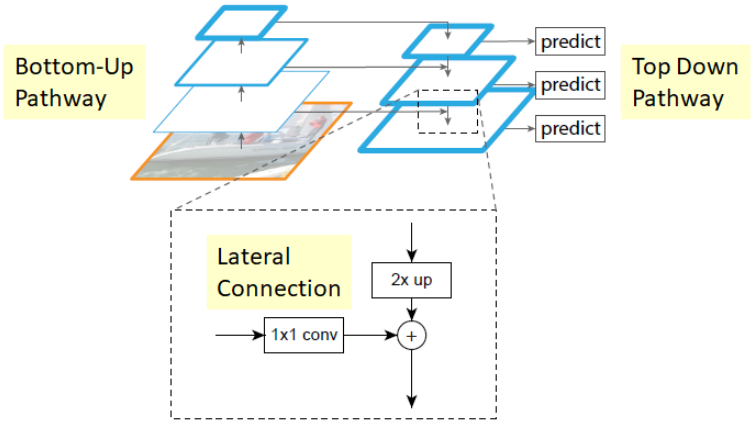


Figura 4.6: Estructura de la *Feature Pyramid Network*.
Source imagen

para una imagen dada, se concatenen dichas características para que el modelo pueda aprender las características locales y generales, y finalmente se transmita las características de localización fuertes de los mapas de características inferiores a los mapas de características superiores; ofreciendo una mejora en la capacidad de detección a *YOLO*.

4.1.4. *Output*

Como paso final en la detección, el resultado del modelo se utiliza principalmente para predecir objetivos de diferentes tamaños en los mapas de características. Previamente, en la parte de la entrada de la *YOLOv5*, la imagen se divide en un *grid* de tamaño *SxS* donde cada celda resultante puede predecir hasta *B bounding boxes* y asignar una *confidence* (confianza, en castellano) cada una de ellas, esto se puede ver reflejado en la Figura 4.8.

Cada una de las posibles *bounding boxes* consisten en un conjunto de 5 valores: x, y, w, h y *conficence*, donde:

- **(*x*,*y*)** representan las coordenadas del centro del *bounding box* en relación con los límites de la celda donde se encuentre.

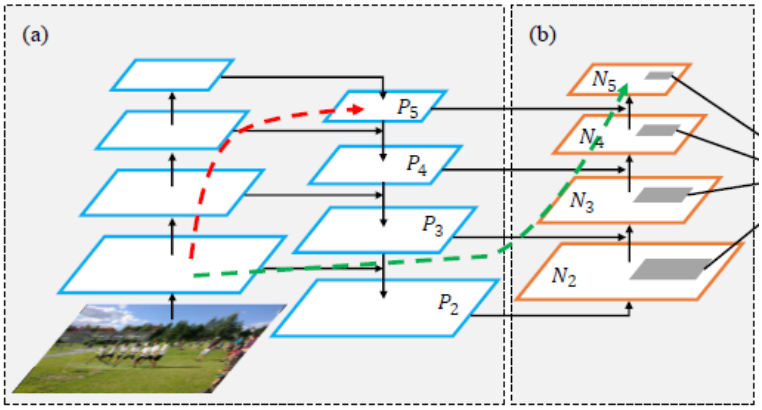


Figura 4.7: Estructura de la *Path Aggregation Network*.
Source imagen

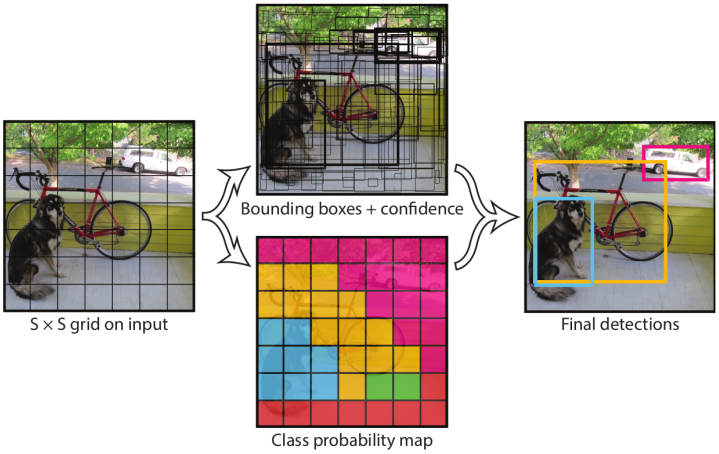


Figura 4.8: Proceso detección de objetos *YOLOv5*.
Source imagen

- ***w*** es la anchura del *bounding box* en relación con la imagen completa.
- ***h*** es la altura del *bounding box* en relación con la imagen completa.
- ***confidence*** representa el *IOU* entre la *bounding box* predicha y cualquier *bounding box* del *ground-truth*: $Pr(Objeto) * IOU_{pred}^{trhth}$

Finalmente, tras aplicar un *Non-maximal suppression*, lo que devuelve la salida de la *YOLOv5* es un tensor con toda la información necesaria para dibujar los *bounding boxes* y su clase asociada; dicho tensor es de tamaño:

$$SxSx(B * 5 + C) \tag{4.1}$$

Donde:

- **SxS** es el tamaño del *grid*.
- **B** en número de *bounding boxes* posibles por celda del *grid*.
- «**5**» hace referencia al número de elementos del conjunto de los atributos de cada *bounding box* (*x*,*y*,*w*,*h*,*confidence*).

- **C** el número de clases a detectar.

Una vez explicadas todas las partes de la *YOLOv5*, en la Figura 4.9 se puede apreciar la topología final separada por dichas partes.

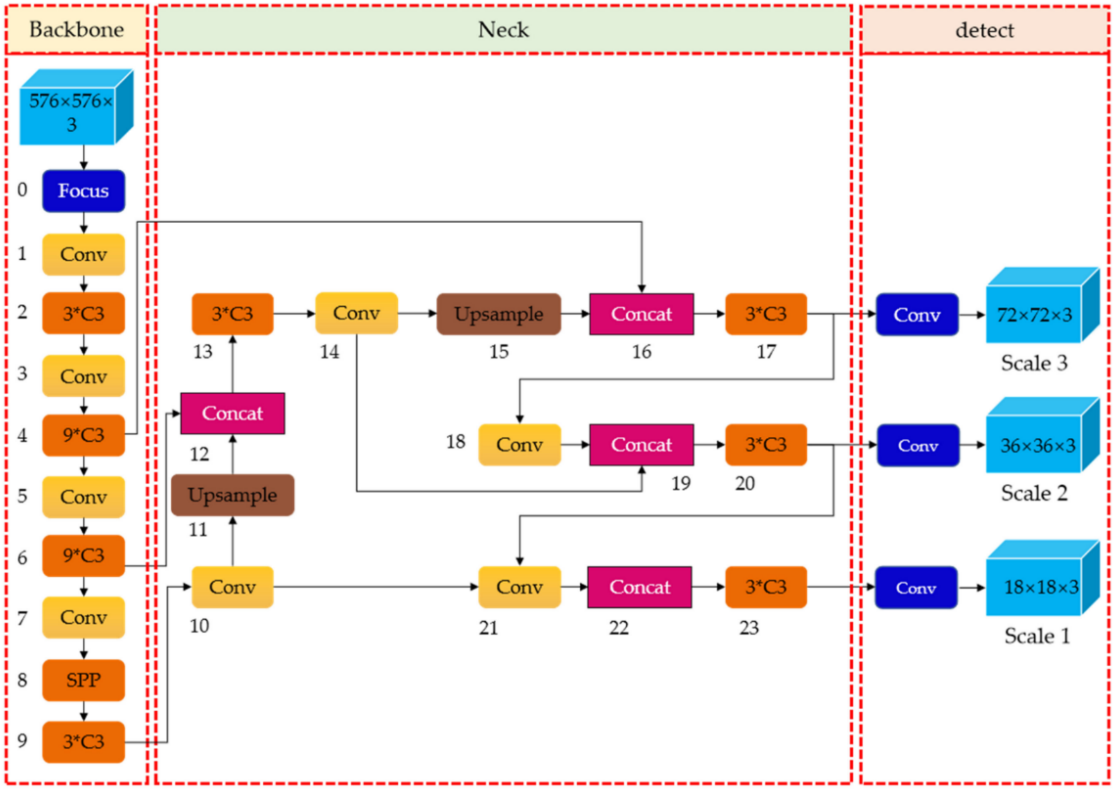


Figura 4.9: Topología final *YOLOv5*
Source imagen

4.2 Tipos de YOLOv5 y resultados

Como se menciona en el Capítulo del estado del arte del actual trabajo, dentro de la estructura *YOLOv5*,*Ultralytics* decidió crear varias topologías distintas en las que, tal y como se comenta en el siguiente capítulo, modifican el valor de unas variables, concretamente *depth_multiple* y *width_multiple* [25], que se encargan de escalar la profundidad y la anchura de la *backbone*; de esta forma se consigue aumentar o disminuir el número de parámetros para una misma red [23]. Dicha técnica se basa en la utilizada por la *EfficientNet*.

En total existen 5 tipos distintos de YOLOv5, éstas son:

- *YOLOv5n*, versión *nano*.
- *YOLOv5s*, versión *small*.
- *YOLOv5m*, versión *medium*.
- *YOLOv5l*, versión *large*.
- *YOLOv5x*, versión *xlarge*.

En la Tabla 4.1 se pueden ver reflejado los valores de ciertas características asociadas a cada uno de los tipos de *YOLOv5*, dichas características son: el tamaño de entrada de imagen, la precisión, la velocidad de detección, el número de parámetros, la profundidad y la anchura. Es importante recalcar que dichas características han sido obtenidas mediante el entrenamiento, evaluación y testeo sobre el conjunto de datos *COCO*.

	size (pixels)	mAP 0.5:0.95	mAP 0.5	Speed CPU (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)	Depth multiple	Width multiple
YOLOv5n (nano)	640	28.0	45.7	45	6.3	0.6	1.9	4.5	0.33	0.25
	1280	36.0	54.4	153	8.1	2.1	3.2	4.6		
YOLOv5s (small)	640	37.4	56.8	98	6.4	0.9	7.2	16.5	0.33	0.50
	1280	44.8	62.7	385	8.2	3.6	12.6	16.8		
YOLOv5m (medium)	640	45.4	64.1	224	8.2	1.7	21.2	49.0	0.67	0.75
	1280	51.3	69.3	887	11.1	6.8	35.7	50.0		
YOLOv5l (large)	640	49.0	67.3	430	10.1	2.7	46.5	109.1	1.0	1.0
	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4		
YOLOv5x (xlarge)	640	50.7	68.9	766	12.1	4.8	86.7	205.7	1.33	1.25
	1280	55.8	72.7	3136	26.2	19.4	140.7	209.8		

Tabla 4.1: Características de cada tipo de *YOLOv5*.

Lo primero que puede observarse es que se realiza un par de pruebas por cada tipo de *YOLOv 5*, en cada una de estas se varía el tamaño de entrada de la imagen (640x640 y 1280x1280). En todos los casos, cuando se escoge el tamaño de imagen de 1280x1280 píxeles para realizar el entrenamiento se obtiene una mayor precisión (tanto mAP[0.5:0.95] como mAP[0.5]) en etapa de evaluación.

Por otro lado, respecto la velocidad de detección como cabe esperar, en GPU es mucho mayor que en CPU. Además según se vaya ampliando el modelo, cada vez el tiempo de ejecución es mayor, siendo el mínimo en la *YOLOv5n* y el máximo en la *YOLOv5x*. Esta característica también se ve reflejada en la Figura 2.8 del Capítulo 2.

Finalmente, resulta de gran interés como a medida que aumenta el valor de las variables *Depth multiple* y *Width multiple*, también aumenta el número de parámetros. Tal y como se expone al principio de esta sección, dichas variables son las encargadas de lo profundo y ancha que es la topología de cada uno de los tipos de *YOLOv5*.

CAPÍTULO 5
Desarrollo de la anonimización en imágenes

A lo largo de este capítulo se muestra el proceso que se ha llevado a cabo para que dada una imagen, se detecte caras y/o matrículas dentro de ésta y las anonimice dando como resultado una nueva imagen con un efecto de difuminación dentro de las zonas de detección.

Primero de todo se expone el *framework* de *YOLOv5* que facilita el entrenamiento y validación de los modelos. Posteriormente se exponen los *datasets* que se han escogido y las distintas formas en las que se tratan:

1. Crear *synthetic data* (datos sintéticos en castellano) mediante el uso de tres *datasets* distintos, a partir de ahora esto consistirá en el *Synthetic Method*.

2. Crear un *dataset* etiquetado de forma manual a partir de dos no etiquetados, de aquí en adelante *Real Method*.

Para ambas formas de tratar los datos se ha desarrollado su propia metodología; en el primer caso hace uso de un *script* para coger datos aleatorios de los *datasets* y crear de forma automática un *dataset* etiquetado para estructuras *YOLO*; mientras que en el segundo caso se hace uso la plataforma de visión por computador Roboflow.

Tras ello, se explica la estructura del *script* general que se encarga automatiza la creación del *data synthetic* (en el caso del *Synthetic Method*), el entrenamiento, el *fine-tuning* y la validación sobre ambos métodos. Finalmente, se expone la realización del *script* para anonimizar, que se encargará de cargar el modelo entrenado y anonimizar imágenes, vídeos e incluso video en *streaming* de cámara.

5.1 API YOLOv5 de Ultralytics

La compañía *Ultralytics* es la encargada del desarrollo de YOLOv5; en junio de 2020 lanzó dicha estructura de forma pública en su repositorio de GitHub, su implementación se realiza sobre el *framework* PyTorch.

El motivo por el que se ha decidido utilizar esta *API* es debido a:

- Facilidad de instalación y despliegue.
- Variantes de *YOLOv5* predefinidas (las mostradas en la Sección 4.2) y facilidad de definir nuevas.

- Comandos totalmente parametrizables para el entrenamiento, validación y test
- Obtención automática de gráficas de precisión, *recall*, curva *precision-recall* entre otras.
- Fácil de incorporar en la librería de *OpenCV* y a dispositivos móviles.

Comandos principales

Para la realización de los *scripts* de los métodos explicados al comienzo del actual capítulo, se hace uso de comandos principales que ofrece la *API* de *YOLOv5*, éstos son:

- ***train.py***, para el entrenamiento del modelo. Como se aprecia en el Listing 5.1, se lanza el *script* de entrenamiento con ciertos parámetros que ofrece, entre los cuales se encuentran los que más se utilizan en el actual trabajo.

```
python ./train.py --batch -1 --epochs 100 --data './anonymization.yaml'
    --project 'runs_yolov5n/freeze/' --name 'training' --cache --freeze 12
     --imgsz 1024 --device 0 --weights 'yolov5_type.pt'
```

Listing 5.1: Ejemplo de ejecución del entrenamiento.

  Donde caben destacar:

  - ***–batch***, se indica el tamaño del batch, en este caso el valor -1 hace referencia a la elección automática del tamaño.
  - ***–epochs***, se indica el número de epochs a realizar.
  - ***–data***, se le pasa un fichero *.yaml* el cual se le indica los *paths* de donde se escogen las particiones de *train*, *val* y *test*, tanto las imágenes como los archivos *.txt* con la información de cada uno de los *bounding boxes* para cada imagen.
  - ***–freeze*** para indicar el número de capas que se desea congelar.
  - ***–imgsz*** indica la resolución de las imágenes de entrada.
  - ***–weights*** se les pasa los pesos que se quieran utilizar, en función de la forma en la que esté estructurada los pesos se utilizará una versión de *YOLOv5* u otra.

- ***val.py***, para el entrenamiento la validación del modelo. Tal y como se muestra en el Listing 5.3, se lanza el *script* de evaluación con ciertos parámetros, muchos de éstos comparten semejanza con los de entrenamiento.

```
python ./val.py --weights 'runs_yolov5_type/freeze/training/weights/best.
    pt' --batch 32 --data './anonymization.yaml' --task test --project '
    runs_{yolov5_type}/freeze/' --name 'validation' --augment --imgsz 1024
     --device {GPU}")
```

Listing 5.2: Ejemplo de ejecución del entrenamiento.

- ***detect.py***, gracias a dicho *script* a partir de los pesos que se han obtenido en la fase de entrenamiento, se puede detectar las caras y matrículas en una imagen, en un vídeo o a través de la imagen que envía una cámara web.

```
python3 ./anonymize.py --weights 'runs_yolov5n/training/weights/best.pt'
    --conf 0.2 --source 0
```

Listing 5.3: Ejemplo de ejecución del entrenamiento.

Donde caben destacar:

- ***–source***, se indica el medio o *path* por el cual va a introducirse las imágenes o video:
  - *–source n*, donde n es el número de cámara a utilizar.
  - *–source 'directorio_imágenes'*, para detectar las caras y matrículas dentro de un conjunto de imágenes.
  - *–source 'directorio_vídeos'*, para detectar las caras y matrículas dentro de un vídeo.
- ***–conf***, indica la precisión a partir de la cual se considera una cara y una matrícula un *True Positive*, es decir, el umbral del sistema se establece a el valor que se le pase junto con este parámetro.

5.2 Metodología para el *Synthetic Method*

Tal y como se muestra al comienzo del actual capítulo, una de las maneras de obtener un *dataset* etiquetado de forma automática es mediante la creación de un conjunto de datos sintéticos. En el *Synthetic Method* se combina tres *datasets* diferentes para crear uno nuevo etiquetado con la información de cada uno de los *bounding boxes* para cada imagen. A lo largo de esta sección se explica el proceso que se ha llevado a cabo para realizar esto.

5.2.1. *Datasets*

La idea principal es que se combinen tres *datasets*, uno que haga función de fondo y otros dos que pongan caras y matrículas sobre dicho fondo. Dichos *datasets* son:

- ***BG-20k***, consiste en un conjunto de 20.000 imágenes de fondo de alta resolución que excluyen los objetos destacados, y que pueden utilizarse para ayudar a generar datos sintéticos de alta calidad. En la Figura 5.2 puede verse algún ejemplo de dicho *dataset*.

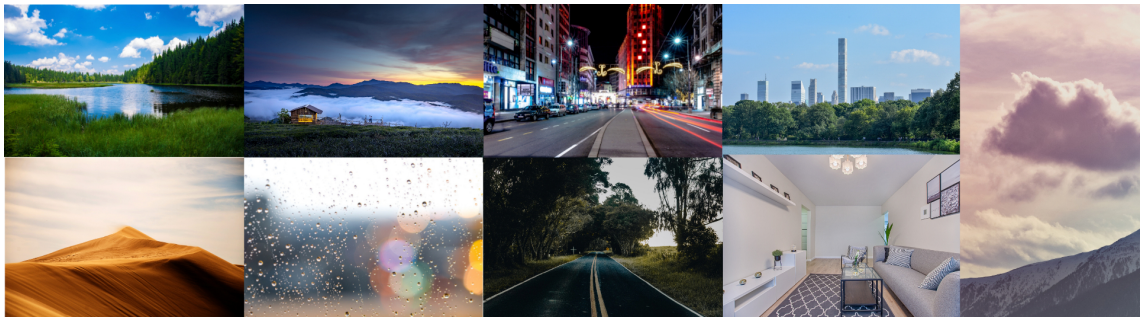
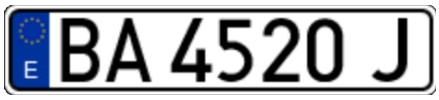


Figura 5.1: Ejemplos del *dataset BG-20K*.
Source imagen

- ***UTKFace***, es un conjunto de datos de rostros a gran escala con un amplio rango de edad (entre 0 y 116 años), éste consta de más de 20.000 imágenes faciales con diferentes rangos de edad, sexo y origen étnico. Las imágenes cubren una gran variación en cuanto a pose, expresión facial, iluminación, oclusión, resolución, etc. Concretamente se ha utilizado la versión alineada y recortada de la cara, para ajustar de esta forma más al reconocimientos de ésta. En la Figura **??** se muestra algunos ejemplos del *dataset*.

Figura 5.2: Ejemplos del *dataset UTKFace*.
Source imagen

- ***Plates Mania***, consiste en una web en la que se pueden encontrar matrículas de un gran número de países del mundo. Ya que se quiere abarcar un gran rango de matrículas, a través de el uso de *web scraping* con la librería *selenium* para Python, se recoge una gran variedad de matrículas de Europa, Estados Unidos, Japón y Reino Unido. Tal y como se muestra en la Figura 5.3 se muestra algunas de éstas.



(a) Matrícula de Europa.

(b) Matrícula de Reino Unido.

(c) Matrícula de Japón.

(d) Matrícula de Estados Unidos.

Figura 5.3: Matrículas del *dataset* creado.

5.2.2. Estrategia del *script* para la creación del *dataset*

Para el *Synthetic Method* se han creado dos *scripts* de *Python*. El primero se encarga de generar los datos necesarios, de aquí en adelante *DataGenerator.py*; mientras que el segundo se encarga de hacer una llamada al primero para que le genere un numero determinado de datos para *training*, validación y *test* para posteriormente organizarlo en ciertas carpetas, de aquí en adelante se referirá a el como *CreateDataset.py*.

DataGenerator.py

Su objetivo principal es el de superponer un número determinado de caras y matrículas sobre un fondo, obteniendo de esta forma automáticamente la imagen y los atributos

asociados necesarios (posición, anchura, altura y clase) para almacenarlos dentro de un archivo *.txt* que es el formato que utiliza *YOLOv5* para leer los *bounding boxes*, tal y como se explica en la Secciones 5.1 y 4.1.4.

Los pasos principales que realiza el *DatatGenerator.py*, tal y como pueden verse reflejados en la Figura 5.4, son los siguientes:

1. Cargar los datos de los tres *datasets* y barajarlos.
2. Escoger una imagen del dataset cargado y barajado de *backgrounds*.
3. Para cada un conjunto de *x* pares de caras y matrículas:
   a) Se comprueba que la imagen de la cara o de la matrícula a colocar sobre el fondo no solape a otra ya existente. Para ello se hace uso de la función *make_fit* que encarga de buscarle espacio disponible a la imagen sobre el fondo sin que solape otra cara o matrícula; internamente se encarga de mover su posición y de hacer un reescalado de la imagen.
   b) En función del valor de la variable *PROB_CHANGE_BRIGHTNESS* se le aplicará un aumento o disminución del brillo.
   c) Se le aplica un difuminado en función del valor de la variable *PROB_CHANGE_BLUR*.
4. Enviar la imagen y el *.txt* con los atributos necesarios a *CreateDataset.py* y comprobar si hay que generar más pares de imágenes matrícula-cara.

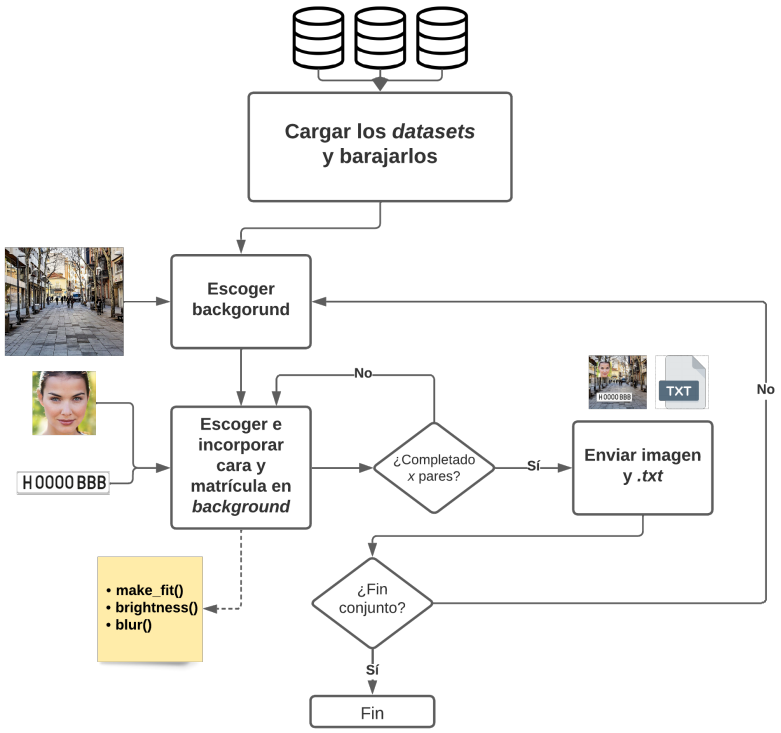


Figura 5.4: Diagrama de ejecución de *DataGenerator.py*.

CreateDataset.py

Este *script* se encarga esencialmente en crear la jerarquía de carpetas correspondiente y hacer llamadas a *DataGenerator* para ir guardando el par imagen-txt en el directorio correspondiente.

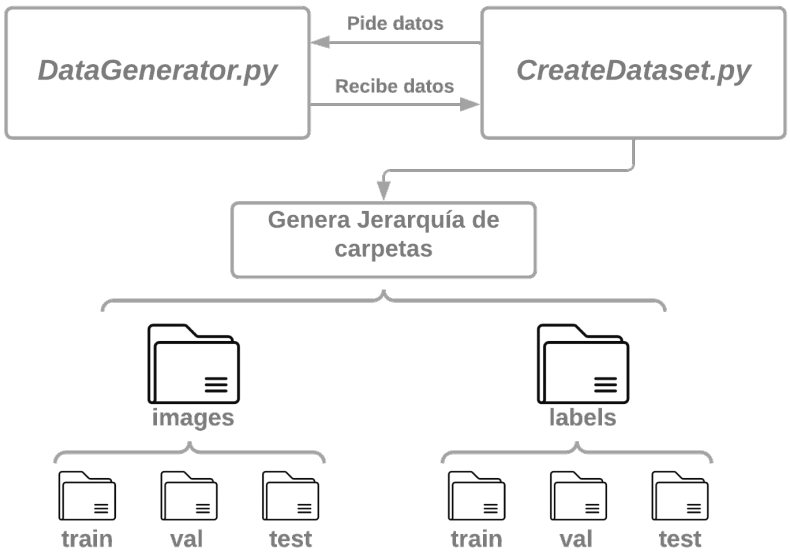


Figura 5.5: Esquema conexión entre *DataGenerator.py* y *CreateDataset.py*.

Tal y como se muestra en la Figura 5.5 la jerarquía de carpetas que se genera se divide en dos principales, y éstas se dividen en tres cada una:

- ***images***, contiene únicamente las imágenes que le ha devuelto el *DataGenerator.py*; dichas imágenes son contenidas en los subdirectorios *train*, *val* y *test* en función de si éstas son creadas para un conjunto u otro.
- ***labels***, alberga los *txts* con la información necesaria de cada imagen generada. Al igual que antes, éstos están organizados de la misma manera que las imágenes, distribuyéndose en los subdirectorios *train*, *val* y *test*.

Un aspecto importante a destacar es que el nombre de una imagen y su txt correspondiente es el mismo (a excepción de la extensión del archivo), y es *YOLOv5* el que se encarga automáticamente de su gestión y enlazado de información.

El actual *script* está pensado para que se lance desde consola de comandos, de esta forma es más sencillo indicar las opciones que deseadas, en el Listing 5.4 se ver reflejado un ejemplo.

```
python CreateDataset.py --clean_directory True --n_train 5000 --n_val 500 --
    n_test 500
```

Listing 5.4: Ejemplo de ejecución de *CreateDataset.py*.

Donde:

- ***–clean_directory*** se encarga de vaciar los directorios que genera *CreateDataset.py* antes de generar un nuevo *dataset* en caso de que se le pase como parámetro *True*, de lo contrario simplemente se añadirá el nuevo *dataset* al ya existente.
- ***–n_train***, indica el número de imágenes (*background* con imágenes y caras) para el conjunto de datos que se utiliza en *training*.
- ***–n_val***, indica el número de imágenes para el conjunto de datos que se utiliza en *validation*.

- ***–n_test***, indica el número de imágenes para el conjunto de datos que se utiliza en *test*.

Tras el desarrollo de estos dos *scripts* se consigue la posibilidad de obtener una cantidad de datos prácticamente ilimitada, esto resulta de gran interés puesto que a más cantidad de datos, mejor puede ser la precisión del sistema. En la Figura 5.6 se puede apreciar una muestra generada por el *DataGenerator.py*.

Figura 5.6: Muestra obtenida mediante el *Synthetic Method*.

5.3 Metodología para el *Real Method*

Al principio del actual capítulo se menciona la alternativa a los datos sintéticos; esta consisten en, a partir de ciertos *datasets*, etiquetarlos de forma manual. Entra en juego el *Real Method*, por lo que en esta sección se describe la metodología y los pasos empleados para su desarrollo.

5.3.1. *Datasets*

En este caso, se tienen 2 *datasets* uno referente únicamente a vehículos con su respectiva matrícula y otro con caras. Dichos *datasets* se han adaptado ligeramente al caso de uso del presente trabajo, estos *datasets* son:

- License-plates_us_es, consiste en un conjunto de datos que contiene 350 imágenes de vehículos con sus respectivas matrículas, cada imagen pueden contener una o más matrículas. Este dataset está ya etiquetado previamente en el formato *.txt* que admite la *YOLOv5*. Las clases que ofrece estos datos etiquetados son las de matrícula y la de coche (Figura 5.7).

  Debido a esto último, se modificaron las estructuras de los *.txt* para que únicamente se tuvieran en cuenta la información relacionada con la clase «*matrícula*», se llevó a cabo gracias a un *script* de *Python* desarrollado para tal caso.

- WIDER FACE, es un conjunto de datos de referencia para la detección de rostros. En total contiene 32.203 imágenes con 393.703 caras con un alto grado de variabilidad

Figura 5.7: *Dataset License-plates_us_es*.

en la escala, la pose y la oclusión, tal y como se muestra en la Figura 5.9. Dicho conjunto de datos está organizado en base a 61 clases de eventos. Para cada clase de evento, se selecciona aleatoriamente un 40 %/10 %/50 % de datos como conjuntos de entrenamiento, validación y *test* respectivamente.

En este caso, la forma en la que se encuentra etiquetadas las imágenes era totalmente distinta a lo esperado, el etiquetado se encontraba unificado en un formato *.mat* que a su vez hace referencia a un fichero *.txt* el cual hace referencia a las distintas clases de eventos que ofrece el *dataset*. Teniendo en cuenta esto, se decidió en una primera instancia recolectar 200 imágenes con distintas características (escala, iluminación, pose, etc.) del dataset y etiquetarlo mediante el uso de la herramienta *Roboflow*, la cual se explica en la Sección 5.3.2.

Figura 5.8: *Dataset WIDER FACE*.

5.3.2. Herramienta *Roboflow*

La herramienta *Roboflow* consiste en una herramienta *online* para la organización y el anotado de *datasets*; de esta forma, ofrece gran facilidad en el ámbito de la visión por computador. En el caso del presente trabajo, se van a añadir ambos datasets descritos en la Sección 5.3.1 y únicamente se va a etiquetar el *dataset* reducido de 200 imágenes de *WIDER FACE*.

Etiquetar un dataset

Lo primero que se realiza es subir ambos *datasets* a un nuevo proyecto en la plataforma; tras ello se accede y etiquetan las imágenes que tienen relación con las caras. Para cada una de las imágenes, se procede marcando la zona/s en la que aparece la cara/s, tras ello se selecciona la clase a la que pertenece cada uno de las regiones marcadas (en este caso la clase es «*Face*»), de esta forma se irá creando un *bounding box* para cada cara existente. Todo lo descrito puede verse reflejado en la Figura 5.9

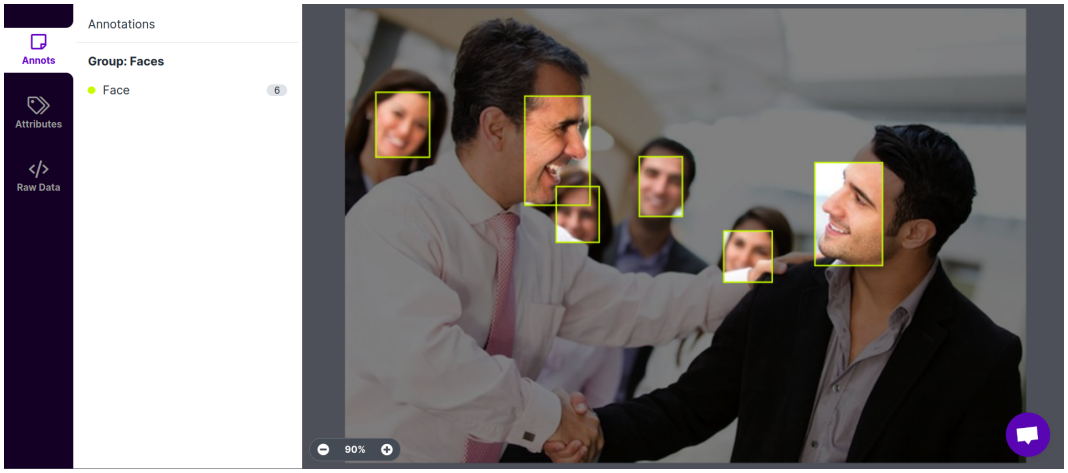


Figura 5.9: Proceso de etiquetar una imagen en *Roboflow*.

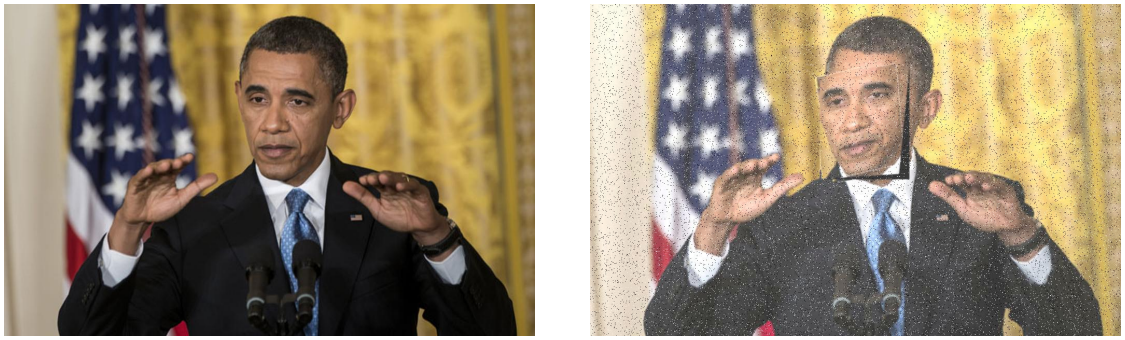
Aplicar *data augmentation* sobre el dataset

Tras finalmente tener el conjunto de matrículas (etiquetado antes de subirlo a la plataforma) y de caras etiquetados, se procede a crear un *dataset* definitivo al cual se le va a aplicar *data augmentation*; éste consiste en aplicar ciertas técnicas de post-procesado de imagen para tener para un misma imagen, diferentes versiones de ésta.

La herramienta *Roboflow* permite añadirle al *dataset* final *data augmentation* tanto a nivel de *bounding box* como a nivel de toda la imagen, en el caso del *dataset* creado se ha utilizado la siguiente configuración:

- A nivel de *bounding box*:
  - Rotación entre -15º y +15º.
  - Recortado horizontal y vertical entre el -15 % y el +15 %.
- A nivel de toda la imagen:
  - Escala de grises al 15 % de las imágenes.
  - Saturación entre el -25 % y el +25 %.
  - Brillo entre el -25 % y el +25 %.
  - Difuminado de hasta 10px.
  - Ruido hasta el 8 % de los píxeles.

En la Figura 5.10 se puede apreciar un ejemplo del *data augmentation* que se lleva a cabo.



(a) Original. **(b)** Con data *augmentation*

Figura 5.10: Ejemplo del *data augmentation* aplicado.

Distribución del *dataset* final

Una vez ya se ha aplicado el *data augmentation*, se procede a realizar la división de las imágenes en *train*, *validation* y *test*; simplemente hay que indicárselo a *Roboflow* antes de exportar el *dataset* y éste lo hace automáticamente.

Tras aplicar el *data augmentation*, se obtiene un *dataset* final de 1389 imágenes (419 imágenes de caras y 970 imágenes de matrículas) . En el caso del presente trabajo se ha decidido maximizar al máximo la cantidad de imágenes al conjunto *train* para mejorar todo lo posible los modelos, por lo que se ha optado por la siguiente distribución de los datos:

- Imágenes de caras: 90 % para *train* y 10 % para *validation*.
- Imágenes de matrículas: 95 % para *train* y 5 % para validation.

Es importante destacar que se ha decidido no destinar ningún porcentaje para el conjunto *test* puesto que, en función del modelo que obtenga los mejores valores en las métricas en la fase de *validation*, se le podrá se podrá pasar fácilmente cualquier imagen o vídeo para verificar el reconocimiento (y anonimización) del sistema.

Finalmente, se ha decidido separar previamente el dataset en función de la clase para evitar el des-balanceo de clases en el conjunto de *train* o en el de *validation*, posteriormente se juntaron ambos para obtener el *dataset* final a utilizar.

5.4 Automatizar entrenamiento y validación

Una vez que ya se conocen las funciones que ofrece la API de *ultralitycs* y con los datasets preparados (tanto el del *Real Method* como el de *Synthetic Method*), se procede a la creación de un *script* que automatice el entrenamiento y la validación de diferentes modelos. En la Figura 5.11 se muestra un esquema de como se forma y funciona el *script* a rasgos generales (*script* base), puesto que tal y como se explica más adelante en esta misma sección, se ha decidido crear cuatro *subscripts* distintos; todos ellos programados en *Python*.

Como puede apreciarse en la Figura 5.11, para cada uno de los distintos modelos que se entrenen, se le cargarán sus correspondientes pesos pre-entrenados (con el *COCO dataset* visto en la Sección 3.2.3) y se entrenará durante un número determinado de *epochs*. En dicho entrenamiento se congelan un número determinado de capas y se entrena durante un número concreto de *epochs* con el conjunto de *train*. Tras ello se obtienen los pesos *best.pt* los cuales pasaremos como parámetro a la validación; en este proceso se obtendrán los resultados de las estadísticas explicadas en la Sección 3.1 sobre el conjunto de

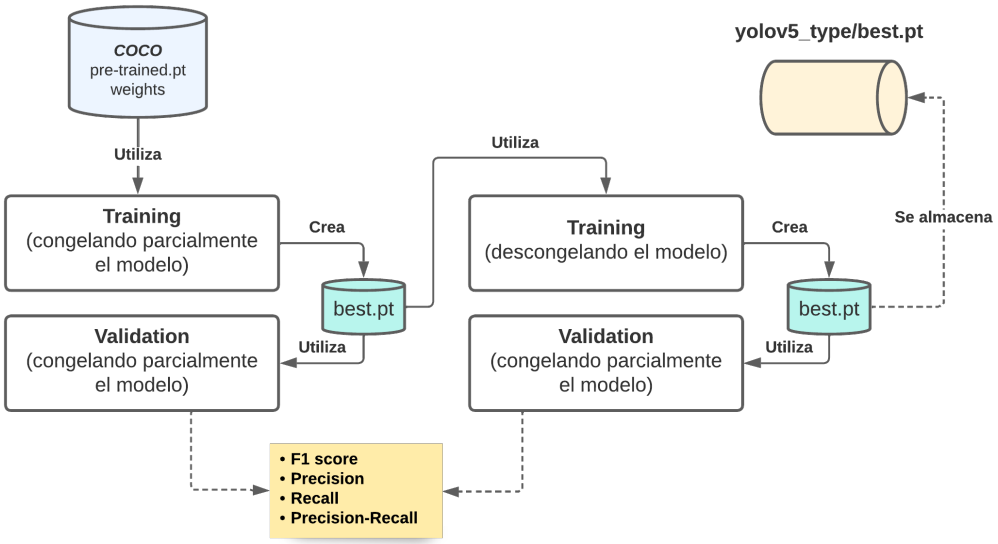


Figura 5.11: Esquema general del *script* base.

validation. Posteriormente se descongelan las capas anteriormente congeladas del modelo y se procede a cargar los pesos *best.pt* anteriores y entrenar el modelo de nuevo durante un número determinado de *epochs*. Finalmente, se volverá a evaluar sobre el conjunto de *validation* tomando los nuevos pesos *best.pt* que devuelve el entrenamiento anterior (dichos pesos también se guardan en disco para poder cargarlos en cualquier momento) y guardando las estadísticas obtenidas para su posterior estudio e interpretación.

Tras saber el funcionamiento del *script* base, se procede a crear cuatro *scripts* diferentes para poder hacer un estudio sobre qué implica el uso de un tipo de *dataset* u otro en las distintas etapas.

La nomenclatura para los nombres de los *scripts* es la siguiente:

- Las primeras dos letras indican qué método se ha utilizado para la creación del *dataset* dentro de la fase de *training*:
  - ***RE*** para el *Real Method* (datos etiquetados de forma manual).
  - ***SY*** para el *Synthetic Method* (datos sintéticos).
- Las segundas dos letras indican qué método se ha utilizado para la creación del *dataset* dentro de la fase de *validation*:
  - ***RE*** para el *Real Method*.
  - ***SY*** para el *Synthetic Method*.

Por lo tanto, los cuatro *scripts* son:

- ***RERE script***, donde tanto en *training* como en *evaluation* se utilizarán los conjuntos de *train* y *evaluation* respectivamente que hayan sido creados con el *Real Method*.
- ***RESY script***, donde en *training* se utilizará el conjunto de *train* creado con el *Real Method* y en la etapa *evaluation* se utiliza el conjunto de datos *evaluation* creado por el *Synthetic Method*.

- ***SYSY script***, donde tanto en *training* como en *evaluation* se utilizarán los conjuntos de *train* y *evaluation* respectivamente que hayan sido creados con el *Synthetic Method*.

- ***SYRE script***, donde en *training* se utilizará el conjunto de *train* creado con el *Synthetic Method* y en la etapa *evaluation* se utiliza el conjunto de datos *evaluation* creado por el *Real Method*.

5.4.1. RERE script

Tal y como se explica al comienzo de la actual sección, en *RERE* se va a utilizar los conjuntos de datos de *training* y *validation* obtenidos mediante el *Real Method* para el entrenamiento y la evaluación del modelo. Su diferencia respecto el diagrama del *script* base es mínima, pues como se muestra en la Figura 5.12, lo único distinto es que el *dataset* es aquel que se crea mediante el *Real Method*, por lo que internamente el *script* será prácticamente igual que el base.

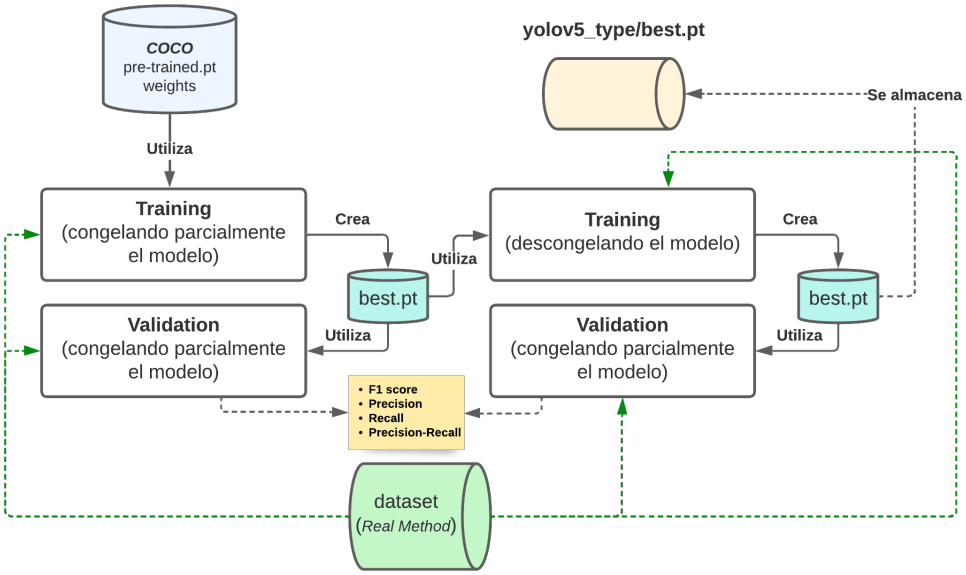


Figura 5.12: Esquema del *RERE script*.

Con este *script* se obtendrá un único archivo con los pesos (*best.pt*) y los resultados de las métricas a evaluar. Esto se hace para cada uno de los distintos modelos a probar.

5.4.2. RESY script

En este caso, tal y como se expone en actual sección cuando se mencionan los distintos *scripts*, en *RESY* se va a utilizar los conjuntos de datos de *training* y *validation* obtenidos mediante *Real Method* y *Synthetic Method* para el entrenamiento y la evaluación del modelo, respectivamente. La diferencia respecto el diagrama del *script* base es que, tal y como muestra en la Figura 5.13, lo que varía en este caso es que se va a tener los dos tipos de datasets *dataset*. El dataset creado con el *Synthetic Method* se utiliza para la validación tanto del modelo con las capas congeladas como en el modelo con las capas descongeladas. Por otro lado el dataset creado mediante *Real Method* se utiliza para ambos entrenamientos, en el modelo con los pesos congelados y en el modelo con los pesos descongelados.

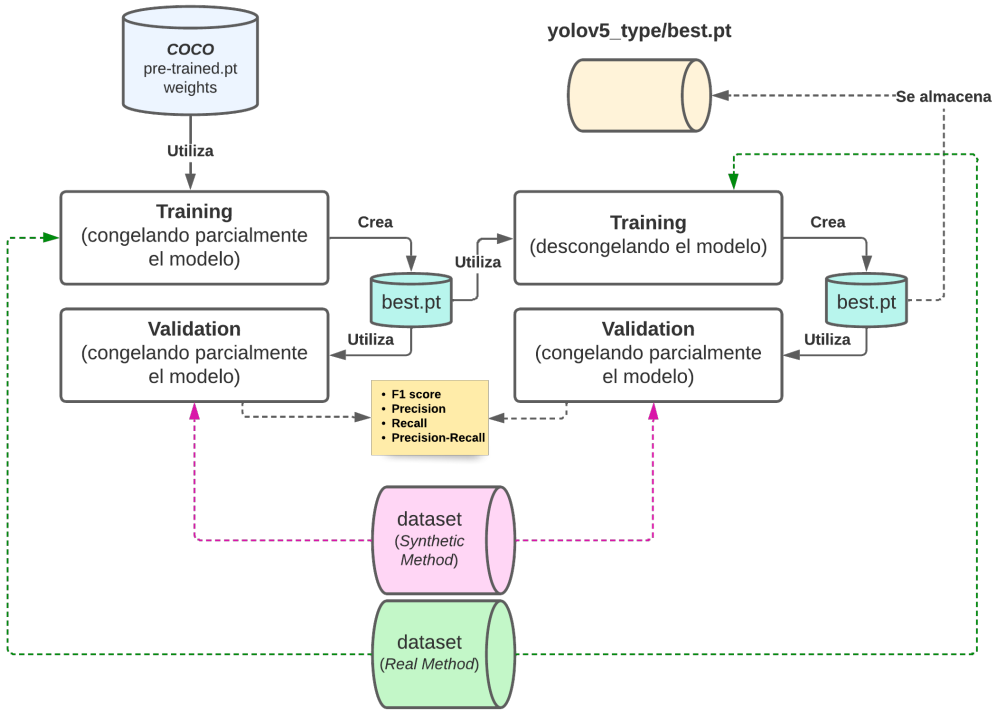


Figura 5.13: Esquema del *RESY script*.

Con este *script* se obtendrá un único archivo con los pesos (*best.pt*) referente al reentrenamiento realizado, y los resultados de las métricas a evaluar en la evaluación del modelo con los pesos descongelados, todo ello para cada uno de los distintos modelos a probar.

5.4.3. SYSY script

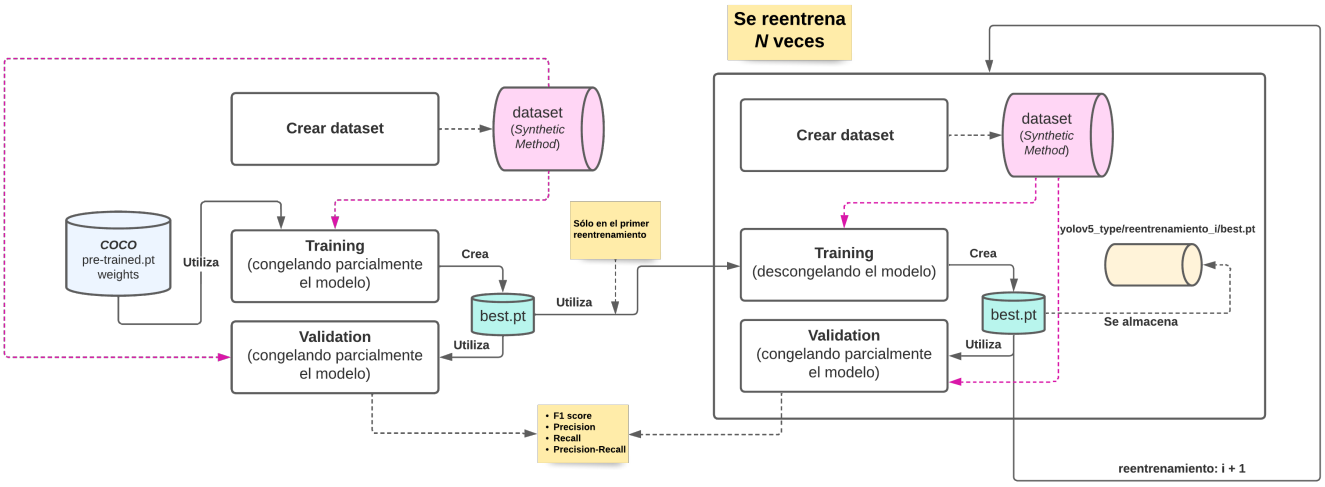


Figura 5.14: Esquema del *SYSY script*.

En este caso, el esquema del *script* base cambia considerablemente, tal y como se aprecia en la Figura 5.14. Lo primero que se realiza es una llamada al *CreateDataset.py* para crear un *dataset* sintético (*Synthetic Method*); posteriormente se cargan los pesos preentrenados en el modelo correspondiente y se congelan ciertas capas del mismo, tras ello y se entrena

el modelo durante cierto número de *epochs* y se guardan los pesos resultantes (*best.pt*). Al igual que en los otros *scripts*, se evalúa utilizando estos pesos y utilizando el conjunto de *evaluation* del *dataset* sintético.

Posteriormente, se van a realizar *N* reentrenamientos en los cuales se descongelan los pesos del modelo. En cada uno de estos reentrenamientos se realizan los siguientes pasos:

1. Se crea un nuevo dataset sintético (*Synthetic Method*).
2. Se cargan los pesos del reentrenamiento anterior. En caso de ser el primer reentrenamiento, se cargan los pesos obtenidos en el primer entrenamiento. Se entrena el modelo durante un numero concreto de *epochs*.
3. Se guardan los pesos del modelo en *best.pt*.
4. Se evalúa sobre el conjunto de *evaluation* del dataset sintético creado y se obtienen las estadísticas resultantes.

De esta forma se puede obtener un *dataset* prácticamente ilimitado y aspirar a un modelo más sólido. Con este *script* se obtiene un archivo con los pesos (*best.pt*) y los resultados de las métricas en la evaluación del modelo para cada uno de los reentrenamientos realizados, y a su vez para cada uno de los distintos modelos a probar.

5.4.4. SYRE script

En este caso, tal y como se puede observar en la Figura 5.15, en el esquema del *script* es prácticamente igual que el esquema del *SYSY script*; lo único que cambia es que ahora en lugar de evaluar sobre un dataset sintético que va modificándose en cada reentramiento, se tiene el conjunto de *evaluation* del *Real Method*; en cuanto a los demás pasos es exactamente igual que *SYRE script*.

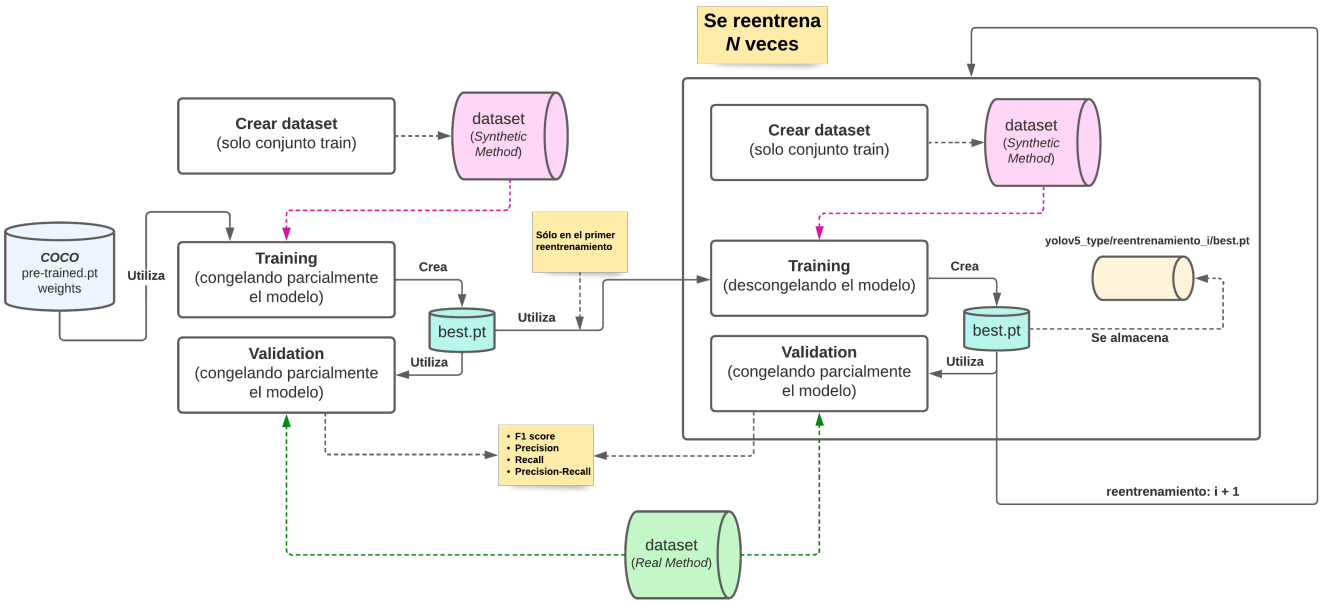


Figura 5.15: Esquema del *SYRE script*.

Se obtiene por lo tanto un archivo con los pesos (*best.pt*) y los resultados de las métricas en la evaluación del modelo para cada uno de los reentrenamientos realizados, y a su vez para cada uno de los distintos modelos a probar. Y de paso, se podrá comprobar si el

entrenar con datos sintéticos sirve para la correcta detección (y posterior anonimización) en un entorno realista.

5.4.5. Comandos para lanzar los scripts automatizados

Los scripts realizados se lanzan con un comando en el cual se indican distintos valores para distintos parámetros, estos son:

- ***–n_retrain***, indica el número de reentrenamientos a realizar para cada uno de los modelos. Esta opción solo está disponible para los *scripts SYSY* y *SYRE*, pues son los únicos que contienen reentrenamiento.
- ***–n_train***, indica el número de muestras a crear para el conjunto de *train* cada vez que se haga una llamada al *script CreateDataset.py*; por lo que esta opción no está disponible en los scripts *RERE* y *RESY*, ya que en ninguno de los dos crea un dataset para el *training*.
- ***–n_val***, hace referencia al número de muestras a crear para el conjunto de *validation* cada vez que se haga una llamada al *script CreateDataset.py*; por lo que esta opción no está disponible en los scripts *RERE* y *SYRE*, ya que en ninguno de los dos crea un dataset para la *evaluation*.
- ***–epochs***, indica el número de *epochs* que se van a realizar tanto en fase de entrenamiento como de reentrenamiento.
- ***–gpu***, indica el número de *GPU* en la que se va a lanzar el entrenamiento y validación de los distintos modelos.
- ***–batch***, indica el tamaño del *batch* a tomar para el entrenamiento y la validación; normalmente se le asigna el valor -1 para que se escoja automáticamente el tamaño del *batch*.

Por lo tanto si se lanzase el *script SYSY* el comando quedaría tal y como se muestra en el Listing 5.5.

```
python script_sysy.py --n_retrain 10 --n_train 1000 --n_val 100 --n_test 100 \\
--epochs 100 --gpu 0 --batch -1
```

Listing 5.5: Ejemplo lanzamiento del *script SYSY*.

5.5 Anonimizador

Una vez ya se tiene uno o varios modelos entrenados, se procede a utilizarlos para detectar y anonimizar las caras y las matrículas dentro de las imágenes.

Para ello, se modifica el script *detect.py* (de aquí en adelante *anonymize.py*) que ofrece la API de *YOLOv5* de *ultralitycs* para que no sólo detecte las caras y las matrículas, sino que además las anonimice. Tal y como se explica en la Sección 5.1, el *script detect.py* se encarga de cargar el modelo entrenado (junto con sus pesos) y de dada una imagen (no importa su procedencia, ya sea simplemente una imagen o de un vídeo) la introduce dentro del modelo entrenado, y devuelve los *bounding boxes* en las zonas donde se encuentren matrículas y/o caras. Posteriormente en dichas regiones en las que se encuentran las respectivas caras y/o matrículas, se aplica un efecto de difuminado mediante la función *blur* que ofrece la librería *OpenCV*, la cantidad de difuminación que se aplica va en relación

con la resolución de la imagen, para de esta forma asegurarnos que independientemente de ésta, asegurar que la anonimización se realiza correctamente.

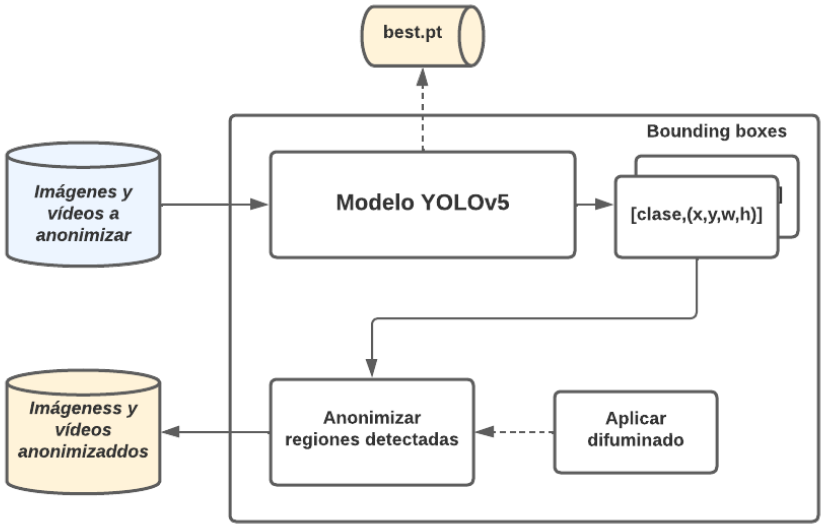
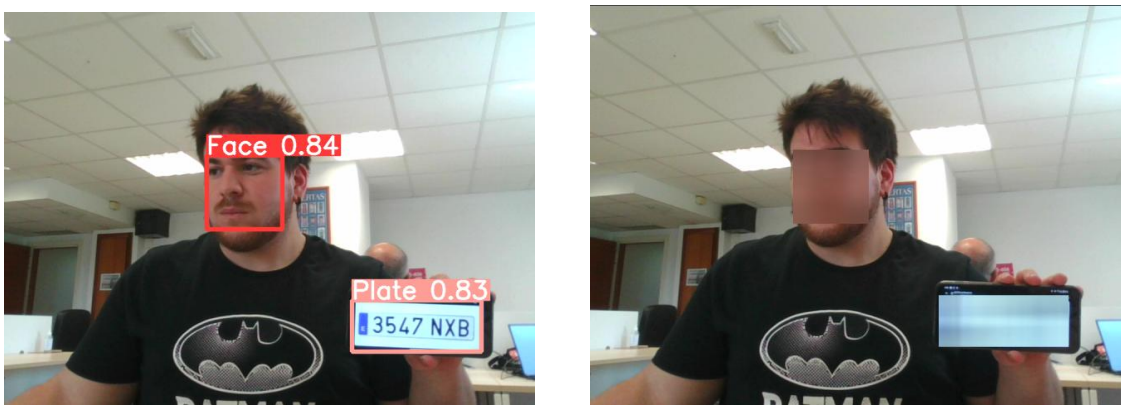


Figura 5.16: Esquema de *anonimizer.py*.

En la Figura 5.16 se puede observar de forma gráfica el flujo de la información que se acaba de describir anteriormente. Finalmente, cada uno de los archivos que se ha enviado a anonimizar, se quedan alojados dentro de un directorio concreto. En la Figura 5.17 se puede observar el proceso que lleva a cabo el *anonymizer.py*; a la izquierda se encuentra la imagen que le entra al modelo entrenado, y a la derecha la imagen resultado que se devuelve tras la detección y el anonimizado.



(a) Detección. **(b)** Anonimizado.

Figura 5.17: Ejemplo de anonimizado.

CAPÍTULO 6
Experimentos y resultados

A lo largo del este capítulo se van a mostrar los experimentos planteados, los resultados obtenidos para cada uno de ellos, y una comparación final entre ellos. Como se ha ido exponiendo a lo largo del trabajo (Secciones 2.2.5 y 4.2), *YOLOv5* de *Ultralytics* ofrece una serie de subversiones las cuales cada una de ellas es más profunda y por ende más pesada computacionalmente.

Por lo tanto, en el presente capítulo se estudia la diferencia entre todas estas subversiones aplicando los métodos vistos en las secciones 5.1 y 5.2 a cada una de ellas; se hace una comparación en función de las métricas vistas en la Sección 3.1 para finalmente en función de ellas justificar el uso de una u otra.

6.1 Explicación de los experimentos

Tal y como se comenta al comienzo del presente capítulo, los experimentos que se realiza el actual estudio vienen marcados por las diferentes subversiones de *YOLOv5* y por los dos métodos de creación/etiquetado de datos.

Además de lo anterior, es de especial importancia tener en cuenta lo que se comenta en [18]: «*La detección suele requerir información visual detallada (grano fino), por lo que aumentamos la resolución de entrada de la red.*»; en dicha referencia se habla sobre que los entrenamientos de *YOLO* logran mejorar los resultados cuando se aplican imágenes de mayor resolución. En una primera instancia se pensó en únicamente entrenar con imágenes cuya resolución es de 640x640 píxeles; pero finalmente se decidió también aplicar el entrenamiento con imágenes de 1024x1024 píxeles.

De este modo, y teniendo en cuenta los *scripts* para la automatización expuestos en la Sección 5.4, se realizan un total de ocho experimentos los cuales son:

- **RERE 640x640**, aplicar *Real Method* en entrenamiento y validación con imágenes de 640x640.
- **RERE 1024x1024**, aplicar *Real Method* en entrenamiento y validación con imágenes de 1024x1024.
- **RESY 640x640**, aplicar *Real Method* en entrenamiento y *Synthetic Method* en validación con imágenes de 640x640.
- **RESY 1024x1024**, aplicar *Real Method* en entrenamiento y *Synthetic Method* en validación con imágenes de 1024x1024.

- **SYSY 640x640**, aplicar en entrenamiento y validación *Synthetic Method* con imágenes de 640x640.
- **SYSY 1024x1024**, aplicar en entrenamiento y validación *Synthetic Method* con imágenes de 1024x1024.
- **SYRE 640x640**, aplicar en entrenamiento *Synthetic Method* y validación *Real Method* con imágenes de 640x640.
- **SYRE 1024x1024**, aplicar en entrenamiento *Synthetic Method* y validación *Real Method* con imágenes de 1024x1024.

En todos los entrenamientos se han utilizado la misma configuración de hiper-parámetros para facilitar y agilizar los entrenamientos, estos simplemente varían en función de si es entrenamiento o reentrenamiento, en la Tabla 6.1 se puede apreciar el valor de los parámetros más relevantes.

	Epochs	**Oprimizador**	**Learning Rate inicial**	**N. capas congeladas**
entrenamiento (pesos congelados)	150	SGD	0.01	12
reentrenamiento (pesos descongelados)	100	SGD	0.01	0

Tabla 6.1: Hiper-parámetros para el entrenamiento y reentrenamiento.

Como se observa en la Tabla 6.1, el único valor que varía es el número de *epochs*. El estudio presente se centra más en las subversiones de *YOLOv5* que en la variación de los hiper-parámetros, es por ello por lo que se propone como un *Futuro Trabajo* en el último capítulo del trabajo.

6.2 Resultados obtenidos

En esta sección se explican los diferentes experimentos que se realizan; además se exponen los resultados obtenidos para cada uno de ellos para posteriormente en el capítulo final barajar cual de ellos es mejor en cuanto a eficiencia y eficacia.

La estrategia que se sigue para la realización de diferentes experimentos, teniendo en cuenta los experimentos expuestos en la Sección 6.1 y para cada una de las subversiones de la YOLOv5 vistas en la Sección 4.2, se realiza lo siguiente:

- **Experimento *RERE***. Las fases de entrenamiento y reentrenamiento se realizan con el *dataset* etiquetado de forma manual (*Real Method*) durante 100 y 150 *epochs* respectivamente. Posteriormente se valida sobre el conjunto *validation* también proveniente del *dataset* obtenido mediante *Real Method*. Para ello se hace uso del *RERE script* tanto con imágenes de 640x640 como con imágenes de 1024x1024.
- **Experimento *RESY***. Al igual que en el caso del experimento *RERE*, las fases de entrenamiento y reentrenamiento se realizan con el *dataset* etiquetado de forma manual (*Real Method*) durante 100 y 150 *epochs* respectivamente, pero posteriormente se valida sobre el conjunto *validation* proveniente del *dataset* obtenido mediante *Synthetic Method*, es decir, del conjunto de datos sintéticos. Para ello se hace uso del *RESY script* tanto con imágenes de 640x640 como con imágenes de 1024x1024.

- **Experimento *SYRE***. En este caso, la fase de entrenamiento se realiza con el conjunto de *train* de un *dataset* obtenido mediante el *Synthetic Method* a lo largo de 100 *epochs*, posteriormente se realizan 10 reentrenamientos, de 150 *epochs* cada uno, y en los que en cada uno de ellos se genera un *dataset* completamente nuevo mediante el uso del *Synthetic Method*. Posteriormente se valida sobre el conjunto *validation* proveniente del *dataset* obtenido mediante *Real Method*. Para ello se hace uso del *SYRE script* tanto con imágenes de 640x640 como con imágenes de 1024x1024.

- **Experimento *SYSY***. Tal y como se realiza en *SYRE*, la fase de entrenamiento se realiza con el conjunto de *train* de un *dataset* obtenido mediante el *Synthetic Method* durante 100 *epochs*, para posteriormente realizar 10 reentrenamientos, donde en cada uno de ellos se reentrena durante 150 *epochs* y se genera un *dataset* completamente nuevo mediante el uso del *Synthetic Method*. Posteriormente se valida sobre el conjunto *validation* proveniente de un *dataset* obtenido mediante también con el *Synthetic Method*. Para ello se hace uso del *SYSY script* tanto con imágenes de 640x640 como con imágenes de 1024x1024.

En todos lo casos, en el primer entrenamiento se utilizan unos pesos preentrenados con el conjunto *MS-COCO* dataset, tal y como se comenta en la Sección 5.4, por lo que los entrenamientos y validaciones resultarán más fructíferos y rápidos en fase de *training*.

6.2.1. Resultados RERE y RESY

Como en ambos experimentos se utiliza el *Real Method* en la fase de entrenamiento y reentrenamiento, se han unificado los resultados de ambas fases de entrenamiento en la Tabla 6.2.

	Res. entrada	**mAP_0.5**	**mAP_0.5:0.95**	**Precisión**	**Recall**
YOLOv5n	640x640	0.71984	0.56652	0.87852	0.61886
	1024x1024	0.78665	0.4267	0.88548	0.6481
YOLOv5s	640x640	0.76193	0.60225	0.84702	0.66927
	1024x1024	0.82481	0.48036	0.81272	0.72007
YOLOv5m	640x640	0.7245	0.55422	0.75714	0.6231
	1024x1024	0.84632	0.50608	0.87194	0.50581
YOLOv5l	640x640	0.69637	0.54579	0.8685	0.56987
	1024x1024	0.81667	0.49624	0.80149	0.4053
YOLOv5x	640x640	0.6905	0.47883	0.78811	0.5879
	1024x1024	0.75335	0.68082	0.93449	0.3457

Tabla 6.2: Resultados de las métricas en entrenamiento de RERE y RESY.

En ella se puede apreciar el valor de las diferentes métricas para cada una de las distintas subversiones de *YOLOv5* después de ejecutarse la fase de entrenamiento y reentrenamiento. Como se puede observase, se obtiene mayor *mAP* en los casos en los que se usan la resolución de 1024x1024 pues tal y como se menciona en [18], cuando se entrena un modelo de detección, éste suele requerir información visual detallada por lo que al aumentamos la resolución de entrada de la red se consigue que esto se cumpla. El modelo que mejor estadísticas obtiene es el modelo *YOLOv5x* consiguiéndose un *mAP* de **0.86266**, un *mAP_0.5:0.95* de **0.68082**, una precisión de **0.93449**; por lo tanto este modelo es el que mejor anonimización ofrece de todos.

Una vez ya se sabe qué modelo es el que más cobertura puede ofrecer, se procede a estudiar como se comporta en la fase de validación en los experimentos *RERE* y *RESY*.

Validación en RERE

En este caso, la validación se realiza sobre el conjunto de *validation* del *dataset* creado mediante *Real Method*. En la Figura 6.1 se puede observar la evolución de las métricas *f1-score*, *precision* y *recall* en función del umbral (*confidence*) tras la fase de validación para el experimento *RERE*.

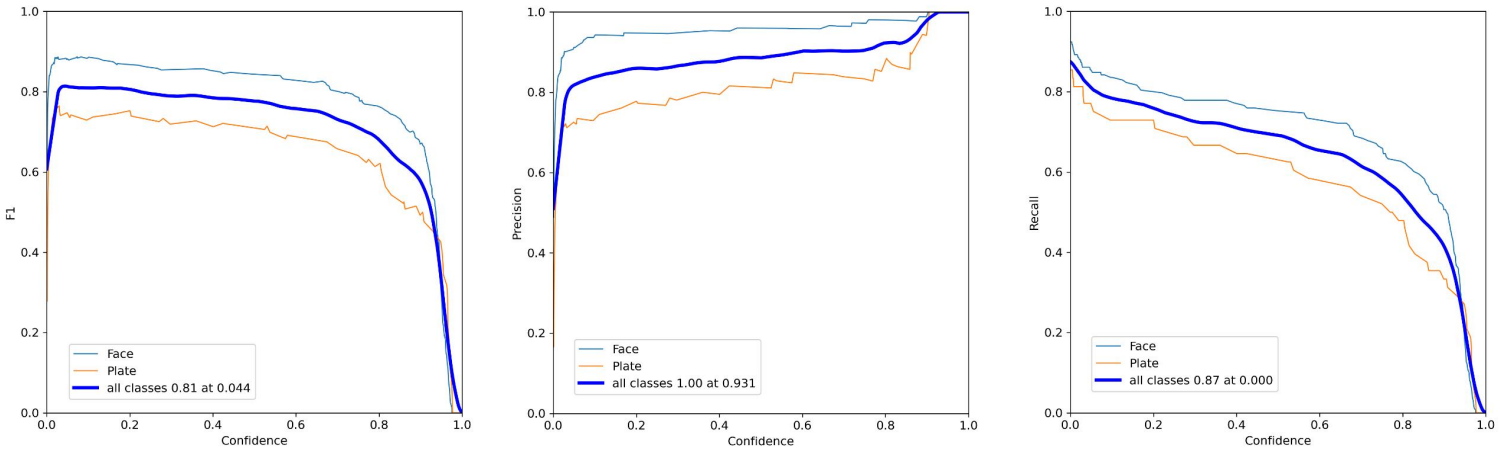


Figura 6.1: Resultados tras validación del experimento *RERE*.
F1-score (izquierda), *Precision* (medio), *Recall* (derecha).

Como se puede observar en dicha figura, dependiendo del umbral, se obtiene un valor mayor o menor de *f1-score*, precisión y *recall*, en función de la cantidad de *TP* a maximizar y *FP* a minimizar, se escogerá un cierto umbral (*confidence*) a la hora de la detección de caras y matrículas. En la Figura 6.2 se muestra la curva *Precision-Recall* mediante la cual también ayuda a decidir que umbral puede ser el más conveniente, tal y como se explica en la Sección 3.1.3.

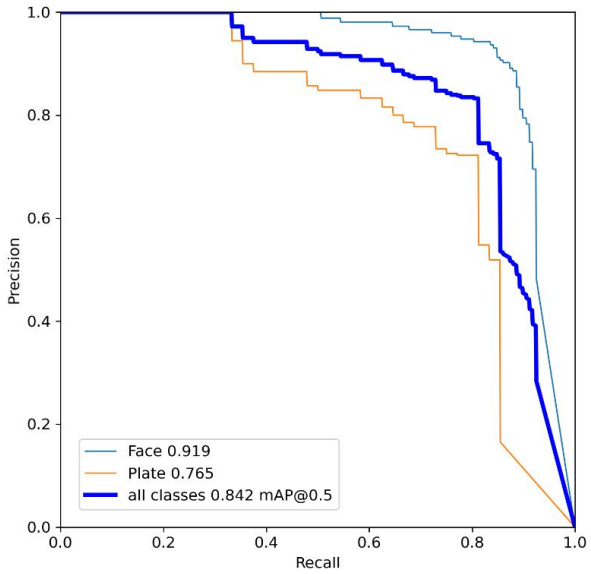


Figura 6.2: Curva *Precision-Recall* tras validación del experimento *RERE*.

Validación en RESY

En este caso, la validación se realiza sobre el conjunto de *validation* del *dataset* creado mediante *Synthetic Method*. Tal y como se explica en el caso del experimento *RERE*, en la

Figura 6.3 se puede observar la variación del valor asociado al *f1-score*, la precisión y el *recall* dependiendo del valor que tome el umbral.

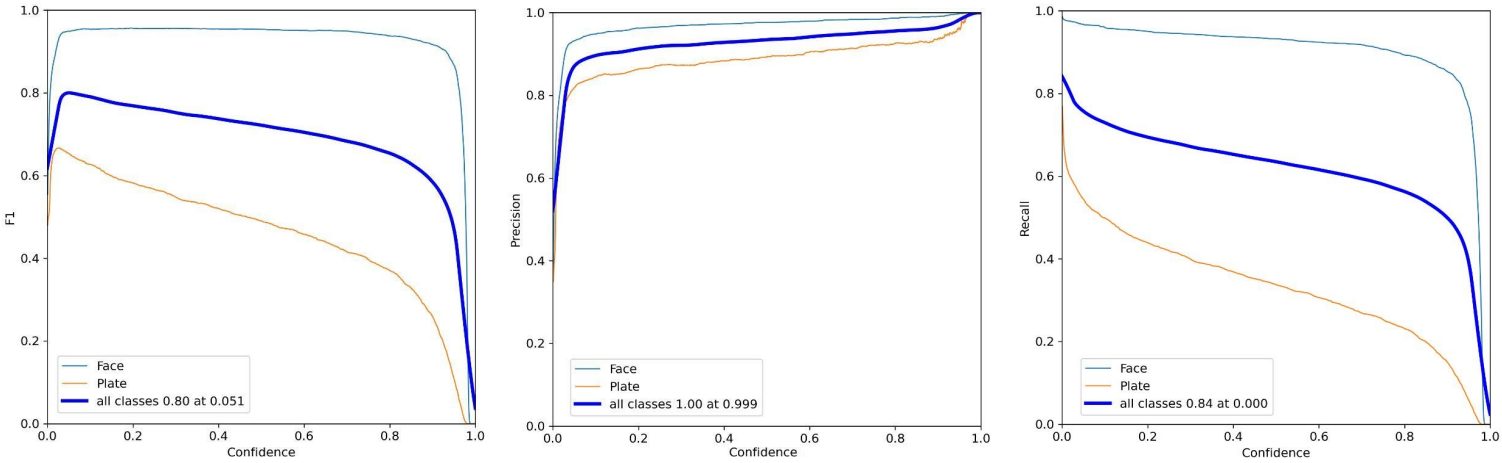


Figura 6.3: Resultados tras validación del experimento *RESY*.
F1-score (izquierda), *Precision* (medio), *Recall* (derecha).

Por otro lado, en la Figura 6.4 se muestra la curva *Precision-Recall* asociada al experimento *RESY*. Al igual que en el primer experimento, esta gráfica puede ser de ayuda para determinar el umbral óptimo a escoger para la detección.

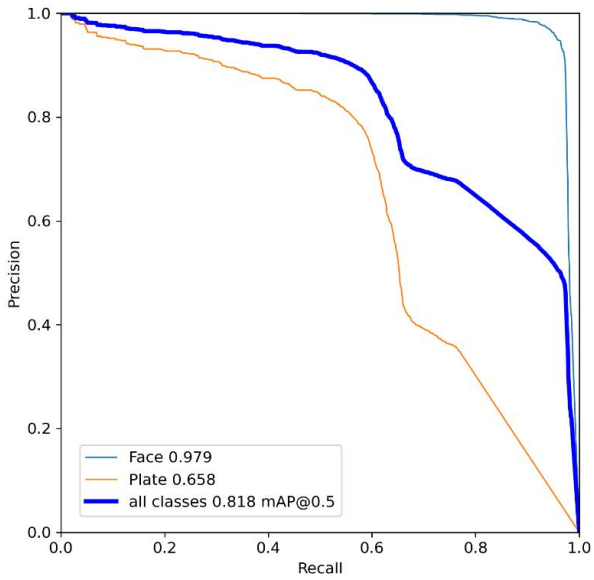


Figura 6.4: Curva *Precision-Recall* tras validación del experimento *RESY*.

Comparando ambos experimentos, se puede concluir que el experimento *RESY* obtiene mejores resultados tanto en *f1-score* como en precisión; esto se debe a que el conjunto de datos de *validation* es mucho más idílico en *RESY* que en *RERE* ya que el *dataset* creado mediante *Synthetic Method*, la mayoría de caras y matrículas están de frente, es decir, su posición respecto a los ejes de coordenadas y y z no varían, por lo que su posición será mucho mas neutral que en un entorno realista como es el que se utiliza en el dataset de *RERE*. Por otra parte, si se observa a nivel de clase (Figura 6.3), el *f1-score* y el *recall* dista mucho; es decir, el *f1-score* de la clase cara es cercano al 0.9 hasta el umbral 0.9 (lo cual es bastante positivo), pero el *f1-score* de las matrículas es al contrario, el mayor *f1-score* se da cuando el umbral toma el valor aproximado de 0.05, a partir de ahí comienza a descender a medida que se incrementa el umbral (lo cual no es positivo).

Por otra parte, en cuanto a las estadísticas que ofrece el experimento *RERE* (Figura 6.1) aunque sean ligeramente peores, el equilibrio que existe entre las clases cara y matrícula es mucho más estable que en el caso del experimento *RESY*. Por lo tanto, en un entorno realista, donde se busque maximizar la anonimización de ambas clases, el modelo *YOLOv5x* del experimento *RERE* resulta más interesante que el modelo YOLOv5x del experimento *RESY*.

Pese a todo esto, tal y como se explica en las conclusiones del Capítulo 7, en el entorno de producción también se busca la rapidez, por lo que dependiendo de ello se podría escoger otro de los modelos mostrados en la Tabla 6.2 para cumplir con la rapidez, pero a costa de decrementar ligeramente la precisión de anonimizado.

6.2.2. Resultados SYRE y SYSY

En este caso, no ocurre lo mismo que en los experimentos *RERE* y *RESY*, los experimentos *SYSY* y *SYRE* no comparten el comparten el mismo *dataset* durante la fase entrenamiento y reentrenamientos, ya que en cada reentrenamiento el *dataset* se genera de forma automática mediante el *Synthetic Method*. Por otra parte, a la hora de la evaluación, el experimento *SYRE* utiliza el conjunto de datos de evaluación creado con el *Real Method*, es decir con el *dataset* real etiquetado a mano. El experimento *SYSY* en su caso, utiliza como conjunto de datos de evaluación un *dataset* creado a partir del *Synthetic Method*.

Entrenamiento y evaluación SYRE

Tras la ejecución del *script SYRE*, los resultados que se obtienen tras 10 reentrenamientos no son nada satisfactorios. Tras todos los reentrenamientos, y para todos los modelos y para las distintas resoluciones de entrada, el entrenamiento no consigue no consigue incrementar el *mAP* y la precisión, sino que el valor de éstos basculan. En las Figura 6.5 y 6.6 se pueden observar los valores que toman las distintas métricas a lo largo del último reentrenamiento y la curva *Precision-Recall* respectivamente.

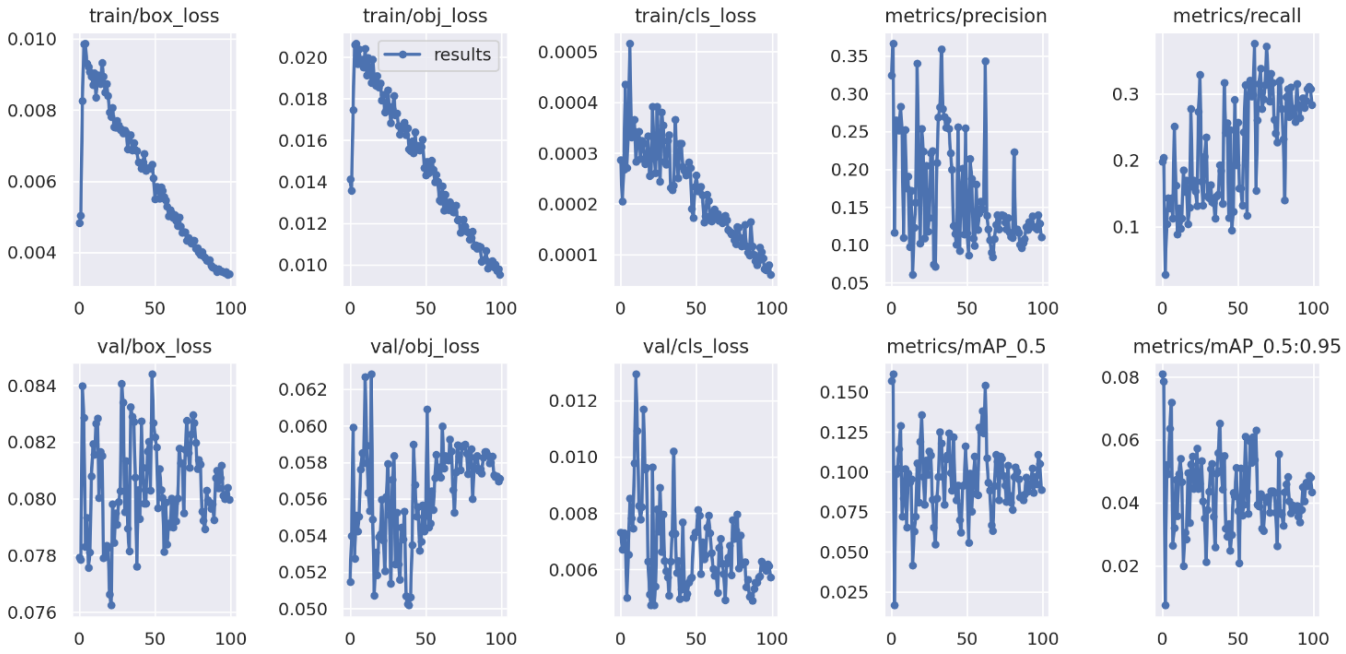


Figura 6.5: Resultados último reentrenamiento del experimento *SYRE* (YOLOv5x 1024x1024).

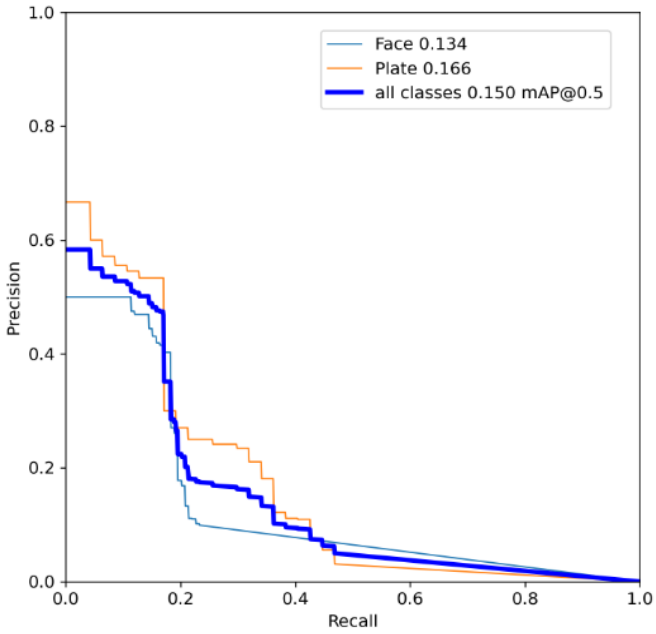


Figura 6.6: Curva *Precision-Recall* tras la fase de validación en el experimento *SYRE* (YOLOv5x 1024x1024).

En ambas figuras (Figuras 6.5 y 6.6) se puede apreciar que el modelo no consigue prosperar y por ende no aprende a detectar de forma correcta las caras y matrículas. Es posible que esto se deba a que al final de cada *epoch*, cuando tiene que validar el entrenamiento hasta el momento, se utiliza el conjunto de datos de evaluación; en este caso dicho conjunto ha sido creado mediante el *Real Method*, por lo que al entrenar con datos sintéticos y validar posteriormente con datos en un entorno real, hace que no se obtenga un modelo sólido que funcione correctamente en un entorno real, que es a lo que se aspira.

Entrenamiento y evaluación SYSY

Tras la ejecución del *script SYSY*, los resultados que se obtienen para todos los modelos y para las dos resoluciones posibles de entrada son realmente satisfactorios. Tal y como se muestra en la Figura 6.7, en el último reentrenamiento se consigue alcanzar valores ideales en las métricas de *mAP* y precisión, llegando a alcanzar ambos un valor comprendido entre 0.995 y 0.999; dichos valores mostrados son los que se alcanzan con el modelo *YOLOv5x* con imágenes de 1024x1024 píxeles de entrada.

En cuanto a la fase de validación, en la Figura 6.8 se puede observar que al igual que en fase de entrenamiento y reentrenamiento, se obtiene un valor muy optimista en las métricas llegando a ser estas casi perfectas. El motivo por el que ocurre esto puede deberse a que el modelo, de cierta forma consigue sobreentrenarse para este tipo de datos sintéticos (ya que se utiliza tanto en entrenamiento como en validación el *Synthetic Method*), de forma que pese a que las imágenes de las matrículas y caras sean distintas, éstas no son realmente diferentes entre sí. Esto se debe a que el *data augmentation* no es lo suficientemente eficaz.

Por ejemplo, las matrículas que se colocan sobre el fondo pese a que se le apliquen algunos efecto para variar la luminosidad y enfoque, la posición en las que están éstas siempre son de la misma forma, es decir, la posición es siempre horizontal y nunca se le realizan rotaciones en ninguno de los ejes. Esto último sí que ocurre dentro de un dataset realista, de ahí que el experimento *SYRE* sea tan poco fructífero.

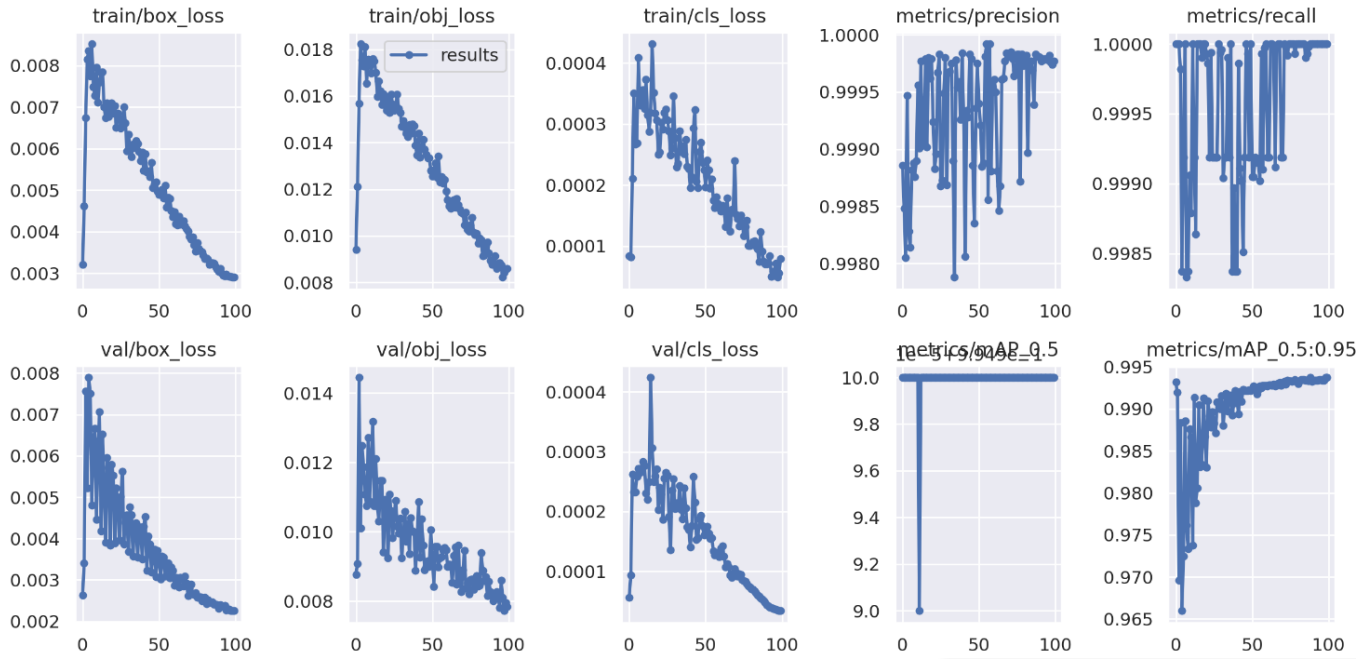


Figura 6.7: Resultados último reentrenamiento del experimento *SYSY* (YOLOv5x 1024x1024).

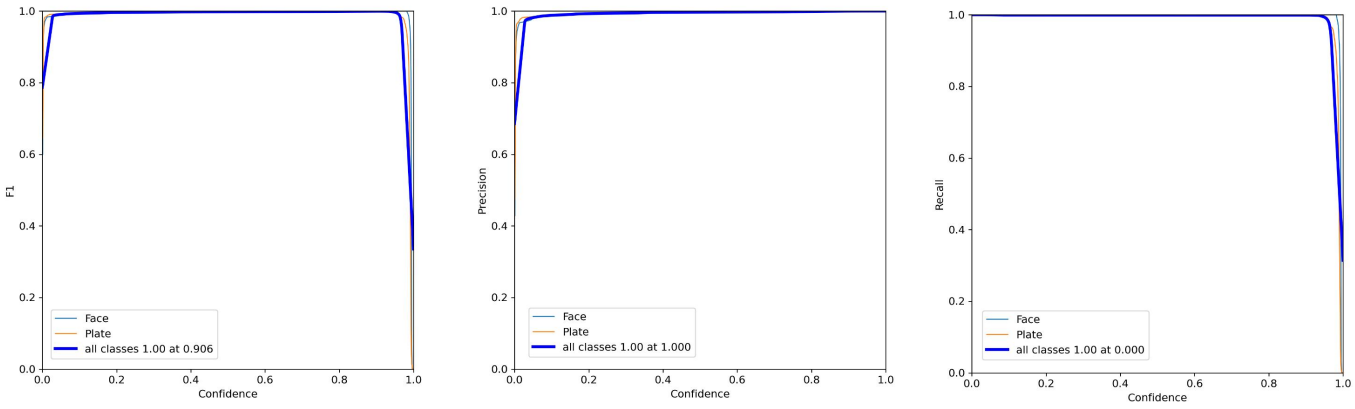


Figura 6.8: Resultados validación del experimento *SYSY* (YOLOv5x 1024x1024).
F1-score (izquierda), *Precision* (medio), *Recall* (derecha).

Por otro lado, en la Figura 6.9 se aprecia como en ningún momento, pese que que se vaya aumentando el umbral, la precisión sigue constante (muy próximo a 1.0) y el *recall* no se va viendo afectado. Este se debe a lo comentado anteriormente, se da por supuesto que el modelo está sobreentrenando.

6.3 Precisión *vs* *Recall* para la anonimización

Una vez presentados y evaluados los resultados de los distintos experimentos, es momento extrapolar e interpretar éstos al ámbito del actual trabajo, la anonimización. Dentro de la anonimización lo que se desea es maximizar el número de *TPs* (*True Positives*) acercándose lo máximo posible a 0 *FNs* (*False Negatives*).

Teniendo esto último en cuenta, se procede a escoger el modelo *yolov5x 1024x1024* del experimento *RERE*, que es con el que se obtienen mejores resultados de *mAP* y precisión

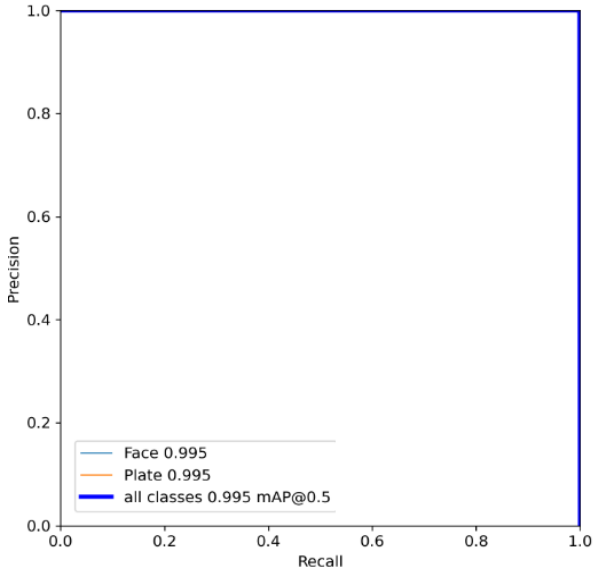
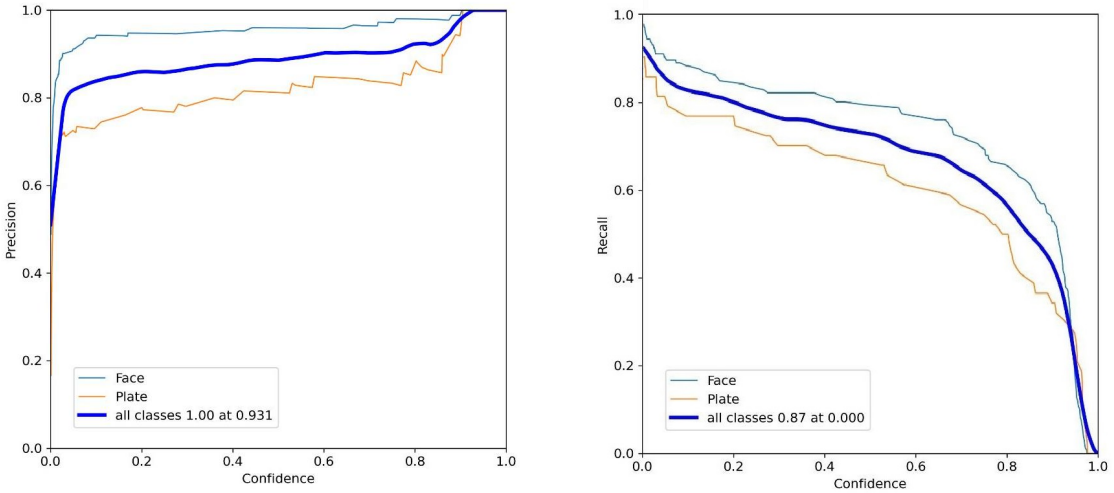


Figura 6.9: Curva *Precision-Recall* tras la fase de validación en el experimento *SYSY* (YOLOv5x 1024x1024).

sobre un conjunto de datos reales. Sobre dicho modelo, se procede a realizar el análisis sobre la variación en la precisión y el *recall* en función del umbral que se escoja.

Tras la fase de validación, se procede a ejecutar el *anonymization.py* sobre el conjunto de datos reales, para obtener la precisión y el *recall* para cada una de los posibles umbrales:

$$umbrales \in \{0{,}010{,}050{,}10{,}20{,}30{,}40{,}50{,}60{,}70{,}80{,}9\}$$



(a) Precisión **(b)** *Recall*.

Figura 6.10: Resultados de la precisión y *recall* en función del umbral

En la Figura 6.10 se puede apreciar la precisión y el *recall* para el modelo *yolov5x 1024x1024* en función del valor del umbral. Como se observa, a medida que incrementa el umbral, se decrementa la precisión e incrementando el *recall*; por lo que a incrementar el umbral se pierde en precisión pero se gana en sensibilidad (*recall*).

Esto último, es un punto a tener muy en cuenta, puesto que cuando aumenta la sensibilidad, el sistema es menos preciso pero detecta más cantidad de objetos aunque alguno de

ellos sean *FP* (*False Positive*). Lo que se desea en la anonimización es entonces, maximizar al máximo los *TPs* sin darle demasiada importancia a los *FPs*; esto es así puesto que es preferible anonimizar el máximo posible de objetos (en este caso matrículas y caras), aunque algunos de éstos no sean realmente los objetos a detectar/anonimizar, de esta forma se asegura una mayor cobertura en la anonimización.

En el caso del presente modelo, se puede apreciar que cuando se hace uso de un umbral pequeño, como es el caso de 0.05, se mantiene la precisión a un 0.8 (80 %) y el *recall* superior al 0.95 (95 %); lo cual hace que, pese a que se anonimicen regiones en las que no hay ninguna cara/matrícula, se asegure la anonimización de las caras y matrículas que sí que aparecen en la imagen. De esta forma, se obtendrán imágenes en las que se anonimizan las caras y matrículas que se encuentran en la imagen, y además de otras regiones en las que no hay dichos objetos. En el servicio *Google Street View*[1] ocurre lo mismo que en el modelo *yolov5x 1024x1024* entrenado para el actual trabajo.

Figura 6.11: Ejemplo de *Google Street View*.

A éste, le ocurre lo comentado en el anterior párrafo; para asegurar que se anonimizan todas las matrículas y caras posibles, hay regiones de la imagen que se anonimizan sin que ésta contenga ninguno de estos dos objetos. Tal y como se muestra en la Figura 6.11, se puede visualizar con *bounding boxes* de color verde, éstos consisten en los *TP*, pues anonimizan las matrículas de los coches. Por otro lado los *bounding boxes* de color rojo representan los *FP*, pues se anonimizan regiones de la imagen que no corresponden a caras o matrículas. Por lo tanto, se seguirá la estrategia seguida por *Google Street View* para la elección del modelo utilizado para anonimizar.

Como se ha podido observar a lo largo del capítulo, se muestran los resultados obtenidos para los distintos experimentos propuestos. Los experimentos con los que mejor resultados se obtiene son los *RERE*, *RESY* y *SYSY*, siendo el peor el experimento *SYRE*. Pese a que el experimento *SYSY* sea el que mejor resultado obtiene, si en lugar de validar con un conjunto de datos sintético, se evalúa con un conjunto de datos reales, se obtendrían los resultados del experimento *SYRE*, lo cual no es conveniente para un entorno realista. Entre los experimentos *RERE* y *RESY*, es más conveniente tener en cuenta los datos de

[1] Es una presentación de *Google Maps* y de *Google Earth* que proporciona panorámicas a nivel de la calle (360 grados de movimiento horizontal y 290 grados de movimiento vertical), permitiendo a los usuarios ver partes de las ciudades seleccionadas y sus áreas metropolitanas circundantes.

validación de *RERE* puesto que éste esta formado por un conjunto de datos reales, y tal y como se comenta en la Sección 6.3, esto es algo a tener muy en cuenta en un entorno realista.

Finalmente, en el siguiente capítulo se comenta qué modelo resulta más interesante en función de la precisión, el tiempo de respuesta y el *recall* (lo comentado en la actual sección) al anonimizar.

CAPÍTULO 7
Cierre de trabajo

Tal y como se ha mostrado a lo largo del actual trabajo, se ha conseguido el objetivo principal de anonimizar caras y matrículas. Ésto se ha conseguido mediante el uso de técnicas de *Deep Learning* aplicadas a *CNN*, concretamente el uso de la estructura *YOLOv5*. En esta capítulo final se recoge cuál de los experimentos ha resultado más interesante, y dentro de éste que modelos son los ideales en cuanto eficiencia y eficacia; finalmente se cierra el trabajo con una serie de proposiciones y ampliaciones que podrían realizarse para futuros trabajos.

7.1 Conclusiones

En el Capítulo 6 se muestra que el experimento que mejor resultados ofrece es el *RERE*; lo cual resulta lógico debido a que los entrenamientos con datos reales etiquetados manualmente (*Real Method*) ofrecen mejores resultados que los sintéticos (*Synthetic Method*). Esto se debe a que *YOLO* aprende teniendo en cuenta el contexto alrededor del objeto a detectar. En caso del *Real Method* esto es óptimo ya que existe contexto con sentido alrededor de los objetos a detectar, pero en caso del *Synthetic Method* no lo es, ya que al por ejemplo añadir la imagen de una matrícula sobre un fondo cualquiera, no permite que exista un contexto real, pues normalmente las matrículas se encuentran dentro en un coche, pese a ello. Los resultados de evaluar un modelo entrenado con *Real Method* sobre un conjunto de datos creado con *Synthetic Method* son también fructíferos, pero no tanto como la evaluación de un modelo entrenado con *Real Method* sobre un conjunto de datos creado con *Real Method*.

Por otro lado, dentro de los distintos modelos evaluados en los experimentos *RERE*, se puede apreciar que aquellos que han sido entrenados con imágenes de entrada de resolución 1024x1024 píxeles obtienen mejores resultados que los entrenados con imágenes de entrada 640x640 píxeles. Es de esperar que esto ocurriese puesto que la detección suele requerir información visual detallada para una mejor precisión [18]. Por otro lado, los valores del *recall* son altos tanto en el experimento de 640x640 como en el de 1024x1024, pero tal y como se comenta en la Sección 6.3, tener un alto *recall* no va a tenerse en cuenta, puesto que de esta forma se aumenta la sensibilidad del modelo, abarcando de esta forma el máximo número de objetos posibles y asegurando así una buena anonimización.

Una vez teniendo claro que el experimento *RERE 1024x1024* contiene los mejores modelos, la primera cuestión que se presenta es saber cual de esos modelos (*nano*, *small*, *medium*, *large*, *extra-large*) resulta el más interesante a la hora de anonimizar. La respuesta ante esta duda es que cualquiera de los modelos resultará más interesante que otro dependiendo del enfoque que se le de. En el caso que sea necesario un velocidad de detección muy alta, se debería escoger el modelo *nano* ya que consigue tasas muy altas de

frames procesados por segundo, eso sí, a costa de disminuir la precisión del anonimizado (no llegando a anonimizar todos los objetos que se requieran). Por otro lado, si lo que se busca es que la precisión de la anonimización sea muy alta, la mejor decisión es optar por el modelo *extra-large*, a costa de que el tiempo de anonimización se vea drásticamente afectado. Si lo que se busca es un equilibrio entre eficacia (anonimización) y eficiencia (tiempo), lo mejor será optar por los modelos *small*, *medium* e incluso en algunos casos el modelo *large*.

7.2 Trabajos futuros

Pese a que el presente trabajo propuesto ha resultado fructífero para los objetivos fijados, es de especial interés proponer algunas modificaciones y ampliaciones que bajo el punto de vista del autor, podría ayudar a mejorar la precisión del sistema o a ampliar las funcionalidades. Dichas propuestas son:

- Realizar un híbrido con el *Real Method* y el *Synthetic Method*, de forma que para algunas de las imágenes etiquetadas a mano, se sustituyese la región de detección por otro objetos de su misma clase, pero de los dataset de matrículas y caras que se utilizan en *Synthetic Method*, de esta forma se consigue una especie de *data augmentation* en el que se seguiría manteniendo el contexto de la imagen.
- Para la detección de caras, y con el objetivo de minimizar los falsos positivos, resulta de interés el uso de un detector de rasgos faciales para asegurar que la posible cara que a detectar, sea realmente una cara y no algo que se parezca a una cara.
- Dentro de la detección de matrículas, podría resultar de especial interés aplicar un sistema de *OCR* para realizar una pseudononimización[1] y posteriormente tener acceso a dicho número de matrícula ya sea para la recuperación de datos o para el estudio y explotación de los mismos.
- Dentro de la anoninimización de caras podría ser interesante aplicar pseudononimización mediante el uso de caras creadas artificialmente utilizando redes *DCGAN*, éstas sustituirían a las caras a anonimizar.
- Puesto que el actual trabajo se centra sobre todo en la experimentación sobre las distintas estructuras de la YOLOv5, es interesante hacer un estudio basado en la modificación de los hiperparámetros en el entrenamiento, para posteriormente contrastar los resultados que se obtienen con diferentes experimentos.

Finalmente, me gustaría concluir el presente trabajo expresando mi máxima satisfacción personal por el procedimiento llevado a cabo para conseguir los objetivos propuestos. Personalmente, quedo muy contento de haber podido poner a prueba los conocimientos adquiridos en el máster para crear parte de un producto que se va a utilizar en un futuro muy cercano en *Pangeanic*. El máster (Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital (MIARFID)) me ha aportado el conocimiento necesario como para poder abarcar todas las partes del trabajo con soltura; concretamente las asignaturas de Visión por Computador y Redes Neuronales Artificiales son las que han hecho esto posible. Espero que dicho trabajo ayude a mi actual empresa, *Pangeanic*, a aplicar la anonimización también dentro del mundo de la imagen, y no sólo en caras o matrículas, sino en cualquier cosa que se requiera.

[1] Procedimiento de gestión de datos donde se reemplazan campos de información personal dentro de un registro de datos por uno o más identificadores artificiales o pseudónimos.

Bibliografía

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[2] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades, 2015.

[3] Diario Oficial de la Unión Europea. Ley orgánica 3/2018, de 5 de diciembre, de protección de datos personales y garantía de los derechos digitales. https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf, 06 de diciembre de 2018. [Online; Consulta 18 mayo 2022].

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[5] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 2009.

[6] Ross Girshick. Fast r-cnn, 2015.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[8] Wang Hao and Song Zhili. Improved mosaic: Algorithms for more complex images. *Journal of Physics: Conference Series*, 1684(1):012094, nov 2020.

[9] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision (ECCV)*, 2014.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Computer Vision – ECCV 2014*, pages 346–361. Springer International Publishing, 2014.

[11] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[12] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, and Wanli Ouyang. T-CNN: Tubelets with convolutional neural networks for object detection from videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2896–2907, oct 2018.

[13] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions, 2015.

[14] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2017.

[15] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[17] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, 2020.

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[19] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[20] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[21] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2014.

[23] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *CoRR*, abs/1911.09070, 2019.

[24] Ultralytics. Yolov5, 2020.

[25] Tian-Hao Wu, Tong-Wen Wang, and Ya-Qi Liu. Real-time vehicle and distance detection based on improved yolo v5 network. In *2021 3rd World Symposium on Artificial Intelligence (WSAI)*, pages 24–28, 2021.

[26] Geoffrey Hinton Yann LeCun, Yoshuam Bengio. Deep learning, 2015.

[27] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey, 2019.