Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

ÁREA DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES

RECONOCIMIENTO DE ACTIVIDAD AGRÍCOLA MEDIANTE DETECCIÓN DE OBJETOS EN IMÁGENES AÉREAS

D. González Lema, Darío
TUTOR: D. García Martínez, Daniel Fernando
COTUTOR: D. Usamentiaga Fernández, Rubén

FECHA: enero 2021

Índice general

Índice de figuras

Índice de tablas

1. Introducción

Tradicionalmente, el sector agrario ha ido rezagado con respecto al sector industrial en lo que se refiere a aplicación de técnicas de inteligencia artificial para ayudar a resolver problemas comunes. Por ejemplo, en plantas industriales se están utilizando distintas técnicas para poder identificar defectos en la producción.

Debido a que en los últimos años se han realizado grandes avances en este campo, desarrollándose, por ejemplo, varios clasificadores de imágenes o detectores de objetos, surge la posibilidad de poder aplicar estas técnicas al sector agrario.

Una de las tareas más necesarias en la actualidad es el **reconocimiento de actividad agrícola**. Es muy común que, por ejemplo, los animales estén pastando en extensiones muy grandes de terrero y que no se conozca con certeza dónde se encuentran. Tradicionalmente ha sido necesario acudir in situ al lugar donde se supone que se encuentran los animales para proceder al conteo. En los últimos años, se han incorporado los drones a las tareas de localización y conteo de animales en grandes extensiones de terreno, pero el uso de drones sin más, no es eficiente, debido a que es necesario que un operador revise cientos de imágenes.

En este proyecto se pretende detectar de forma automática objetos de interés, como pueden ser animales, con el objetivo de poder detectar actividad agrícola mediante el uso de imágenes aéreas.

Es necesario destacar que estas técnicas no solo se pueden aplicar a la detección de animales, sino a cualquier signo de actividad agrícola, como pudiera ser la presencia de comederos o bebederos, maquinaria agrícola, personas, etc.

La aplicación de esta tecnología, no se limita a la simple localización y/o conteo de animales o elementos que identifiquen la presencia de actividad agraria, sino que se puede aplicar a muchos otros ámbitos. A la hora de solicitar determinadas ayudas, como por ejemplo la PAC, es necesario justificar la presencia de actividad agrícola en un determinado terreno, abarcando en este caso toda la Unión Europea.

Debido a todo esta problemática surge este proyecto de investigación realizado por la Universidad de Oviedo, marco sobre el cual se ha desarrollado este Trabajo de Fin de Máster (TFM): **Reconocimiento de Actividad Agrícola mediante Detección de Objetos en Imágenes Aéreas**.

2. Objetivos y Alcance

2.1.- Objetivos

El principal objetivo de este trabajo es poder **detectar actividad agraria** utilizando técnicas de visión por computador, pudiendo así procesar una gran cantidad de datos (imágenes) de forma automática en un tiempo relativamente pequeño.

Para el cumplimiento de este objetivo se han establecido una serie de hitos que se describen a continuación:

- **Búsqueda y análisis de conjuntos de datos.**
  - Búsqueda de conjuntos de datos que se puedan utilizar con el objetivo de poder detectar actividad agraria.
  - Análisis de los conjuntos de datos encontrados para comprobar si se corresponden con las necesidades de este trabajo.
- **Análisis del estado del arte de las diferentes CNNs.**
  - Búsqueda de las diferentes familias de CNNs disponibles, y análisis de sus correspondientes versiones.
  - Análisis de los diferentes resultados obtenidos con las CNNs analizadas.
- **Estudio de las diferentes alternativas para el despliegue de un servicio de reconocimiento de actividad agrícola.**

2.2.- Alcance

Este TFM se enmarca dentro de un proyecto de colaboración entre la Universidad y Seresco [1]. Gracias a esta colaboración, se pretende cumplir con los objetivos establecidos, y poder así realizar un detector de objetos especializado en el reconocimiento de actividad agrícola.

Una vez finalizado el proyecto se pretende haber alcanzado las siguientes metas:

- Encontrar o crear al menos un conjunto de datos que se adapte a las necesidades del proyecto.
- Analizar el estado del arte de las herramientas disponibles actualmente en la tarea del reconocimiento de objetos, así como el estudio del funcionamiento de dichos detectores.
- Estudio de las diferentes métricas disponibles para comparar los resultados de los distintos detectores.
- Comparar los diferentes detectores disponibles.
- Seleccionar el detector que mejor se adapte al conjunto de datos utilizado.
- Configurar el detector de la forma que mejor se adapte al conjunto de datos seleccionado.
- Realizar una guía detallada de la instalación del detector seleccionado.

- Disponer de un detector de objetos capaz de detectar diferentes evidencias de la presencia de actividad agraria, en un determinado terreno.
- Evaluar redes detectoras en un entorno cloud: tanto redes propietarias como el propio despliegue de un posible servicio de reconocimiento de actividad agrícola.
- Transmitir el conocimiento generado a la empresa receptora.
- Realización del Trabajo de Fin de Máster (TFM) en el ámbito de este proyecto.

Este trabajo se estructura en los siguientes capítulos:

- **Introducción**: en este capítulo se comentan los motivos por los que surge este proyecto, la problemática actual y enfoque establecido para resolver el problema planteado.
- **Objetivos y Alcance**: en este capítulo se analizan los objetivos del proyecto así como su alcance.
- **Revisión del estado del arte**: antes de comenzar con el diseño experimental es necesario conocer las técnicas disponibles para poder resolver el problema planteado. Estas tareas de análisis se llevan a cabo en este capítulo.
- **Conjuntos de datos utilizados**: para poder llevar a cabo los experimentos es necesario haber analizado los datos previamente, ya que el resultado del proyecto depende en gran parte de estos datos.
- **Evaluación de redes para la detección de objetos**: una vez estudiado el estado del arte y presentados los conjuntos de datos a utilizar, se debe de realizar un diseño experimental. Finalizados todos los experimentos, disponibles en los **Anexos A, B y C**, se seleccionan los mejores para cada una de las redes utilizadas y se comparan, con el fin de seleccionar el mejor modelo posible.
- **Implementación y despliegue de un servicio de detección de objetos**: una vez desarrollados los mejores modelos posibles es necesario crear un prototipo para ilustrar el posible funcionamiento del servicio. Además se estudian diferentes escenarios de despliegue.
- **Hardware utilizado**: para que los resultados obtenidos durante este proyecto sean replicables, es necesario dejar documentado el hardware utilizado. Además se indican los costes de dicho hardware.
- **Conclusiones**: para finalizar el trabajo se incluye el capítulo de conclusiones. En este capítulo se comentan los resultados obtenidos, el cumplimiento o no de los objetivos e hitos marcados, además, de realizar una valoración personal de todo el proyecto.

3. Revisión del estado del arte

Para cumplir con el objetivo del proyecto es necesario utilizar imágenes aéreas, por lo que surgen tres posibilidades para llevar a cabo la detección de actividad agrícola: clasificación de imágenes, segmentación semántica y detección de objetos.

La clasificación de imágenes consiste en decidir si una determinada clase está presente o no en una determinada imagen. Por ejemplo, si se utilizase la Figura 3.1 (imagen de la izquierda) en una tarea de reconocimiento de imágenes, el objetivo sería predecir si en esa imagen hay un gato o no.

La segmentación semántica es similar a la clasificación de imágenes, pero en vez de clasificar a nivel de imagen, se clasifica a nivel de píxel. En la Figura 3.1 (imagen central) se puede observar una clasificación de cada uno de los píxeles de la imagen, así como las diferentes clases con sus correspondientes códigos de colores.

En este proyecto, se opta por utilizar la detección de objetos. La detección de objetos consiste en localizar los objetos de interés en una imagen, y clasificarlos (ver Figura 3.1, imagen de la derecha).

Figura 3.1.- Diferencia entre la clasificación y la detección de objetos

3.1.- Conceptos Previos

En esta sección se definen una serie de términos que se van a utilizar en el resto del presente documento.

- **GroundTruth:** para poder entrenar un modelo es necesario disponer de unos datos. En el caso de la detección de objetos, estos datos son las propias imágenes y las anotaciones. Las anotaciones son, simplemente, las coordenadas (en píxeles) en las que se encuentran los objetos, **realizadas de forma manual** por un humano. Estas anotaciones al ser realizadas manualmente se **suponen** que son 100 % correctas. Gracias al GroundTruth durante la fase de entrenamiento, se le puede indicar al modelo, dónde están los objetos, y así éste puede aprender a reconocer dichos objetos. Además, gracias al GroundTruth se puede evaluar el modelo una vez finalizado el entrenamiento, ya que se pueden comparar las predicciones realizadas (BoundingBoxes) con las anotaciones que se **suponen** correctas (GroundTruth). En la Figura 3.2 se muestra un ejemplo de GroudTruth.

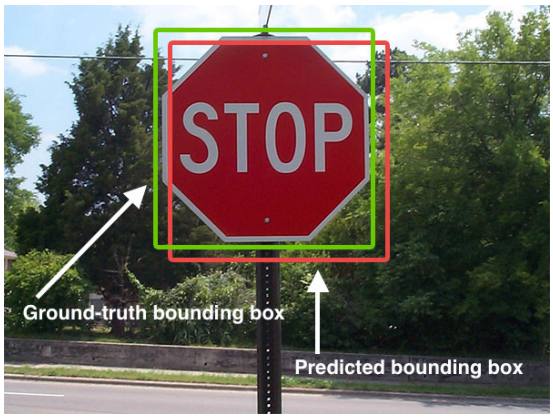


Figura 3.2.- Ejemplo de GroundTruth

- **BoundingBox:** una BoundingBox es una caja imaginaria que rodea a los objetos detectados [2]. Visualmente, simplemente es una caja alrededor de un objeto. Analíticamente, son una serie de coordenadas. Uno de los posibles formatos de estas coordenadas es el siguiente: x1, y1 (esquina superior izquierda), ancho y alto medido en píxeles. En la Figura 3.3 se muestra un ejemplo de BoundingBox.

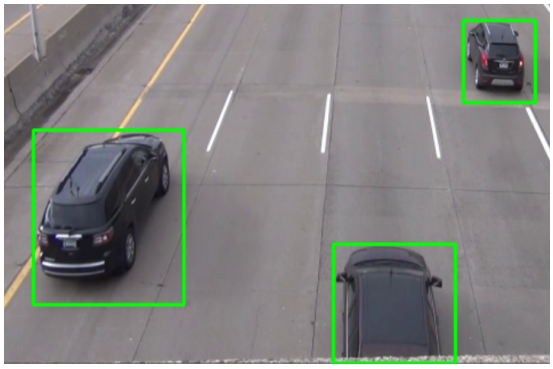


Figura 3.3.- Ejemplo de BoundingBox

- **Confianza:** también conocido como confidence score o simplemente score, es un indicador producido por el modelo que indica cómo de seguro está este sobre una predicción. Este valor oscila entre 0 y 1.

- **Conjunto de datos:** en el caso de la detección de objetos, el dataset o conjunto de datos es el conjunto de imágenes y anotaciones (GroundTruth) utilizadas.

- **Conjunto de entrenamiento, validación y test:** para desarrollar un detector de objetos es necesario dividir el conjunto de datos (o dataset) en tres subconjuntos independientes entre sí [3].

  La forma de trabajar es la siguiente: se entrena el algoritmo con el conjunto de entrenamiento, y se aplica el modelo obtenido al conjunto de test, de manera que para medir el rendimiento se comparan las BoundingBoxes predichas (sobre el conjunto de test) con el GroundTruth. Adicionalmente, se utiliza el conjunto de validación. Este conjunto sirve para medir el rendimiento del modelo durante la fase del entrenamiento. Mientras que se realiza el entrenamiento se realiza un test con el conjunto de validación, de esta manera se puede hacer una estimación del rendimiento del modelo durante el entrenamiento.

- **CrossTesting:** esta técnica consiste en dividir el conjunto, en este caso en cuatro partes iguales. Una vez hecho esto, una de esas partes se utiliza como test (en este proyecto el conjunto de test y validación será el mismo) y la combinación de las otras tres como entrenamiento. De esta forma, se pueden realizar diferentes combinaciones, como se explica a continuación.

  En la Figura 3.4 se puede observar cómo se divide el conjunto en cuatro partes iguales, cada una con el 25 % de las imágenes. Para que la división no sea arbitraria, primero **se desordena el conjunto de imágenes de forma aleatoria**.

Figura 3.4.- División del conjunto para el CrossTesting

Una vez realizada esta división, se generan cuatro conjuntos de entrenamiento y test, combinando las diferentes divisiones realizadas previamente. Los subconjuntos formados son los siguientes.

Test 1	1
Train 1	2, 3 y 4
Test 2	2
Train 2	1,3 y 4
Test 3	3
Train 3	1, 2 y 4
Test 4	4
Train 4	1, 2 y 3

De esta manera se entrena con el conjunto de entrenamiento X y se realiza el test con el conjunto de test X. Nótese que al entrenar por ejemplo con el conjunto de entrenamiento uno y evaluar los resultados con el conjunto de test uno, durante el entrenamiento no se habrán utilizado las imágenes destinadas a la evaluación, por lo que se podrá medir correctamente la eficacia del detector.

Se espera que los resultados obtenidos en cada uno de los cuatro conjuntos de test realizados sean similares, habiendo entrenado, como es evidente, con su correspondiente conjunto de entrenamiento. En caso de que los resultados no sean similares, se habrá detectado algún fallo que habría que corregir.

- **Aumento de datos:** también conocido como Data Augmentation en inglés, es un conjunto de técnicas comúnmente utilizadas en el ámbito del Machine Learning con el fin de mejorar el proceso de entrenamiento. Estas técnicas se utilizan con el fin de generar un modelo más robusto. Cuando se disponen de pocas imágenes para el entrenamiento se puede aplicar estas técnicas con el fin de generar nuevas imágenes, similares a las ya disponibles, pero con ligeras modificaciones. **Estas técnicas no se deben de utilizar sobre el conjunto de test**, debido a que este debe reflejar la realidad, sin alteraciones. Además, de incrementar la cantidad de datos (imágenes) disponibles, estas técnicas sirven para evitar que el modelo se sobreajuste a los datos originales, permitiendo al modelo generalizar, por lo que podrá realizar predicciones correctamente. Por este motivo, se recomienda utilizar estas técnicas aunque se dispongan de suficientes datos, ya que el modelo podrá **generalizar** mejor. Las técnicas de aumento de datos más populares son las siguientes [4]:

  - **Giro o Flip:** esta técnica consiste en girar las imágenes horizontalmente y verticalmente. En la Figura 3.5 se muestra un ejemplo.

Figura 3.5.- Aumento mediante giros. De izquierda a derecha: imagen original, imagen girada horizontalmente e imagen girada verticalmente

- **Rotación:** esta técnica consiste en rotar una imagen una cantidad determinada de grados. En la Figura 3.6 se muestra un ejemplo.

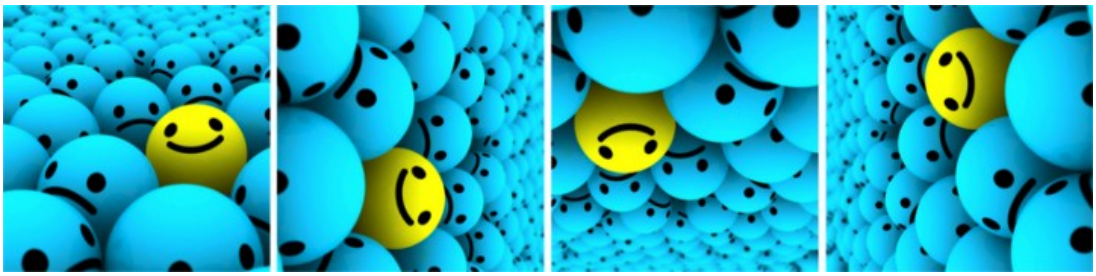


Figura 3.6.- Aumento mediante rotaciones. De izquierda a derecha: imagen original e imagen rotada cada 90º

- **Escalado:** esta técnica consiste en incrementar o disminuir el zoom sobre una imagen. Cuando se incremente el zoom, la imagen tendrá un tamaño más grande que la imagen original. Lo más normal es recortar una parte la imagen aumentada. De esta forma el tamaño de la nueva imagen coincidirá con el de la original. En la Figura 3.7 se muestra un ejemplo.

Figura 3.7.- Aumento mediante variación de escala. De izquierda a derecha: imagen original, imagen con el zoom incrementado en un 10 % y en un 20 %

- **Recorte:** esta técnica consiste en recortar una sección de la imagen original. Posteriormente, se redimensiona cada uno de los recortes generados al tamaño original de la imagen. En la Figura 3.8 se muestra un ejemplo.

Figura 3.8.- Aumento mediante recortes. De izquierda a derecha: imagen original, recorte sobre la esquina superior izquierda y recorte sobre la esquina inferior derecha

- **Translación:** esta técnica consiste en mover la imagen sobre el eje X o Y (o ambos). En la Figura 3.9 se muestra un ejemplo.

Figura 3.9.- Aumento mediante translaciones. De izquierda a derecha: imagen original, imagen translada a la derecha e imagen translada hacia arriba

- **Ruido gaussiano:** esta técnica consiste en introducir ruido en la imagen de manera de que el modelo no se sobreajuste con imágenes de alta calidad. Esto se debe a que una red neuronal intenta aprender patrones que se repiten. Una versión alternativa al ruido gausiano es el ruido "salt and pepper" o "sal y pimienta" que consiste en introducir píxeles en blanco y negro a lo largo de la imagen. Es similar al ruido gausiano, pero reduce menos la información de la imagen original. En la Figura 3.10 se muestra un ejemplo.

Figura 3.10.- Aumento mediante variación del ruido gaussiano. De izquierda a derecha: imagen original, imagen con ruido gaussiano e imagen con ruido salt and pepper

- **Alteración del color:** aparte de las técnicas mencionadas anteriormente se pueden aplicar otras técnicas: modificación del brillo, la saturación, el tono o el contraste. En la Figura 3.11 se muestra un ejemplo.

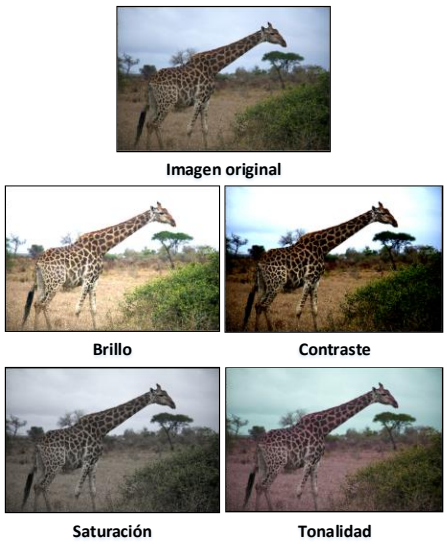


Figura 3.11.- Aumento mediante alteración del color de las imágenes

- **IoU:** Intersección sobre la Unión o índice de Jaccard mide el grado de similitud entre dos BoundingBoxes. Matemáticamente se representa como:

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

En el campo de la detección de objetos se utiliza el IoU para comprobar cuál es el grado de solapamiento entre la BoundingBox predicha y el GroundTruth [5], tal y como se puede apreciar en la Figura 3.12. Normalmente, si el $IoU \geq 0.5$ se considera que una predicción es correcta, pudiendo oscilar su valor en el intervalo [0, 1].

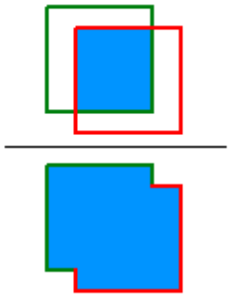


Figura 3.12.- Cálculo de IoU

- **Sobreajuste:** también conocido como **overfitting** en inglés, es uno de los principales factores por los que se obtienen unos malos resultados en el campo del Machine Learning [6]. El sobreajuste u overfitting hace referencia al problema de la **generalización de datos**. En otras palabras, el modelo aprende muy bien los datos de utilizados en el entrenamiento, pero no se adapta a los datos de test. En la Figura 3.13 se muestra un ejemplo de un modelo sobreajustado a los datos de entrenamiento (a la izquierda), y que por lo tanto no será capaz de realizar buenas predicciones, y un modelo no sobreajustado (a la derecha), que será capaz de realizar buenas predicciones.

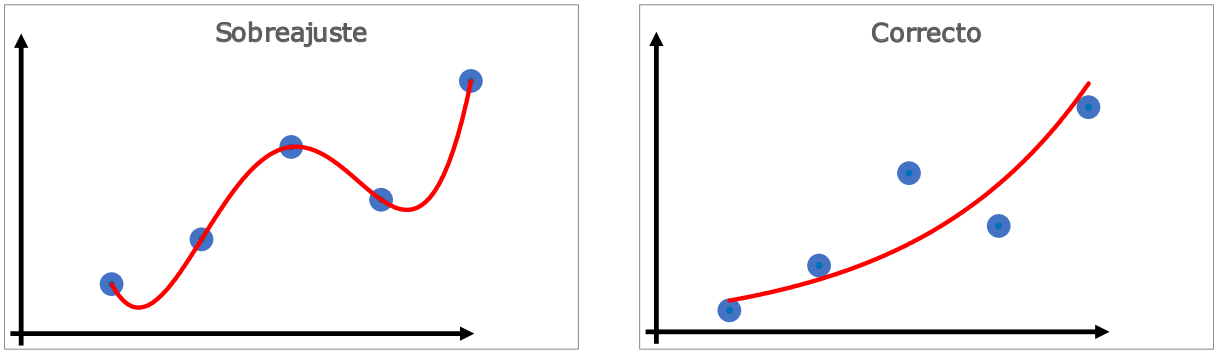


Figura 3.13.- Ejemplo de modelo sobreajustado y correcto

Para prevenir el sobreajuste se pueden llevar a cabo una serie de medidas que se citan a continuación:

- Utilizar un número elevado de muestras.

- Conjunto balanceado. En el caso de utilizar varias clases de objetos, es conveniente disponer de una gran cantidad de muestras de cada una de las clases.
- Conjunto de test independiente. Para comprobar la eficacia del modelo, es necesario utilizar un conjunto que no se haya utilizado durante el entrenamiento. Esto permite obtener una valoración real de cómo de preciso en el modelo, además, de poder detectar fácilmente el overfitting.

3.2.- Redes Neuronales Convolucionales (CNNs)

Las redes neuronales convolucionales o CNNs por sus siglas en inglés son un tipo de redes neuronales artificiales cuyo procesamiento se inspira en el córtex visual del ojo humano, con el objetivo de poder así, identificar características, y por lo tanto, identificar objetos [7].

La red toma como entrada los píxeles de una imagen. Si por ejemplo la imagen es de 28 x 28, la red tendría 784 neuronas de entrada y esto suponiendo que se estuviese utilizando una imagen en escala de grises. En el caso de utilizar una imagen en color (RGB) la red tendría 28 x 28 x 3 = 2 352 neuronas en la primera capa.

Una convolución consiste en seleccionar grupos de píxeles próximos y realizar una serie de operaciones matemáticas (producto escalar) con una pequeña matriz denominada kernel. El kernel recorre todas las neuronas de entrada y genera una nueva matriz de salida, que será la nueva capa resultante.

En realidad no se utiliza un kernel, sino que se utilizan varios. A este conjunto de kernels se les denomina filtros. Para seguir con el ejemplo anterior, si se utilizan 32 filtros, se obtendrían 32 matrices de salida. El conjunto de estas matrices se conoce como mapa de características (feature map). Cada mapa de características sería de $28 \times 28 \times 1$, y como se utilizan 32 filtros la red estaría utilizando 25 088 neuronas.

A medida que se van realizando más convoluciones se van extrayendo más características. Las primeras convoluciones permiten extraer las características más elementales, como curvas o siluetas, mientras que las últimas convoluciones permiten extraer las características más relevantes.

Antes de continuar realizando convoluciones, es necesario reducir el número de neuronas utilizadas, ya que 25 088 neuronas en la primera convolución son demasiadas (y esto teniendo en cuenta que se trata de una imagen de 28×28). Para reducir el número de neuronas se pueden utilizar varias técnicas de downsampling pero la más utilizada es Max-Pooling.

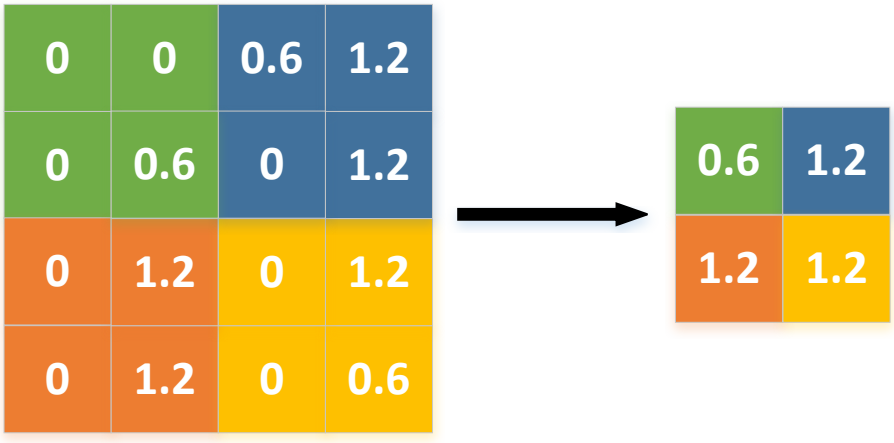


Figura 3.14.- Ejemplo de Max-Pooling

Para explicar en qué consiste el Max-Pooling se utiliza la Figura 3.14. En este ejemplo para realizar el Max-Pooling se utiliza un matriz de 2×2. Esto quiere decir que para cada mapa de características obtenidos (32 mapas de 28×28) en lugar de seleccionar cada uno de sus píxeles, se selecciona el más alto (por lo tanto más representativo) de los cuatro píxeles más próximos. Gracias a este proceso, se pasa de 32 mapas de características de 28×28 a 32 mapas de 14×14, reduciendo así el número de neuronas de 25 088 a 6 272. Estas nuevas neuronas siguen almacenando la información más relevante.

Para poder identificar distintos objetos es necesario realizar una gran cantidad de convoluciones a partir de la salida de la anterior capa de convolución. Evidentemente, esto solo es una introducción debido a que en este proyecto el foco se pone sobre las redes específicas para la detección de objetos.

En el ámbito de la **detección de objetos** mediante Redes Neuronales Convolucionales o CNNs, existen tres familias principales: RCNN, SSD y YOLO. Existen más redes, algunas anteriores a las mencionadas, aunque la RCNN se considera la primera debido a que sus predecesoras no consiguen unos resultados satisfactorios. En este proyecto se podrá el foco sobre la familia YOLO, debido a que son las mejores relación prestaciones/tiempo de entrenamiento. También se analiza la red SSD debido a que se evaluarán sus resultados.

En la Figura 3.15 se puede ver cómo han ido evolucionado en el tiempo las diferentes redes, habiendo sido creadas YOLOv4 y YOLOv5 en 2020.

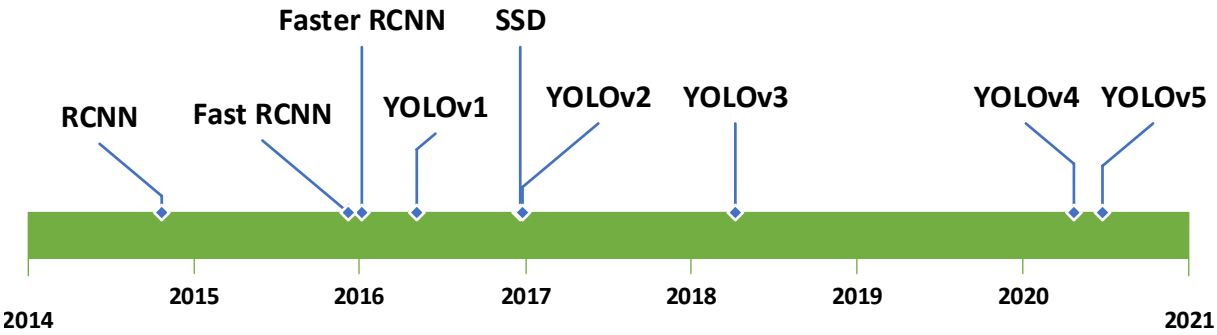


Figura 3.15.- Evolución de las CNNs en el ámbito de la detección de objetos

3.2.1.- YOLOv1

YOLOv1 [8] es una red especializada en la detección de objetos. Una de sus características más destacada es que trabaja con la imagen completa, de ahí su nombre You Only Look Once.

Esta red trabaja de una forma relativamente sencilla. Primero hay que redimensionar la imagen a 448 × 448, se utiliza una CNN sobre la imagen y se obtienen los resultados de la detección basándose en un modelo de confianza (ver Figura 3.16).

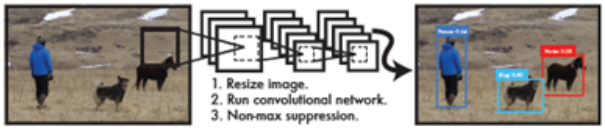


Figura 3.16.- Funcionamiento de YOLOv1

La red predice simultáneamente las BoundingBoxes y la clase. YOLOv1 entrena con imágenes completas y optimiza el rendimiento de la detección. Por este motivo, YOLOv1 es muy rápida.

Otro beneficio de entrenar con imágenes completas es conocer el contexto global de toda la imagen. Es decir, al entrenar con imágenes completas, se posee información sobre todas las clases. Como consecuencia, YOLOv1 comete pocos fallos en lo referente al fondo de las imágenes.

Por último, el tercer beneficio que tiene entrenar con imágenes completas es la mejora producida al realizar generalizaciones de nuevos dominios. La Figura 3.17 es un ejemplo de esta generalización a nuevos dominios ya que el modelo se ha entrenado con imágenes de personas reales, pero la red es capaz de detectar la figura de una persona en un cuadro.

Figura 3.17.- Ejemplo de generalización de nuevos dominios

Hay que tener en cuenta que a pesar de todo esto, YOLOv1 tiene una precisión inferior a otros detectores, especialmente en lo que se refiere a objetos pequeños, aunque su principal ventaja es que es más rápido.

3.2.1.1.- Principios en los que se basa la red

Como se ha explicado en el apartado anterior, YOLOv1 utiliza características de toda la imagen, por lo que tiene un contexto global de toda la imagen, y de todos sus objetos [8] [9].

La detección de objetos que realiza la red se basa en considerar que la imagen de entrada se puede dividir en un grid de $S \times S$ (típicamente S = 7). Cada elemento del grid se considera una celda de la imagen, y para cada celda de la imagen la red calcula un conjunto de características que utiliza para la detección (ver Figura 3.18).

Si el centro de un objeto recae en una celda del grid, esa celda es la responsable de detectar ese objeto (celda detectora). En la Figura 3.19, la celda roja es la encargada de detectar el coche, debido a que el centro del coche (punto verde, obtenido del GroundTruth) ha caído en esa celda.

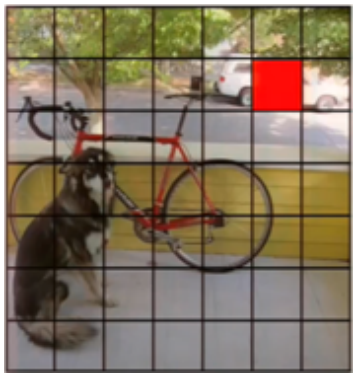


Figura 3.18.- División de la imagen en un grid

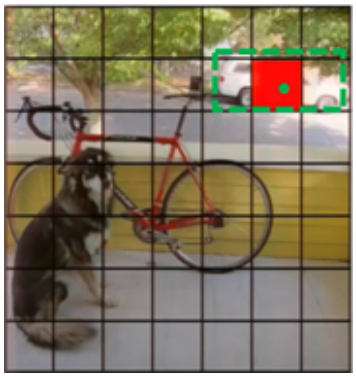


Figura 3.19.- Celda responsable de detectar un objeto

Cada celda predice B BoundingBoxes (típicamente B = 2) (ver Figuras 3.20 y 3.21) y proporciona una puntuación de la confianza (confidence score) para estas BoundingBoxes. Esta confianza no es más que un valor, pero conceptualmente refleja dos aspectos:

- La probabilidad de que la BoundingBox predicha contenga un objeto (de cualquier clase).
- Lo precisa que es esta BoundingBox. Es decir que se ajuste (IoU) lo más posible a la anotación del GroundTruth.

La confianza, conceptualmente se puede representar mediante la siguiente fórmula: $Pr(Object) \times IoU_{pred}^{truth}$. Esta fórmula, en otras palabras, se puede ver como la multiplicación de la probabilidad de que haya un objeto y del grado de solapamiento entre la BoundingBox predicha con la BoundingBox del GroundTruth (anotación realizada a mano).

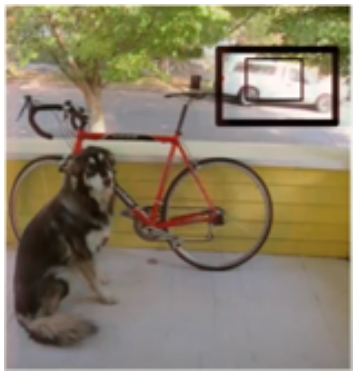


Figura 3.20.- Predicciones de la celda de la Figura 3.19

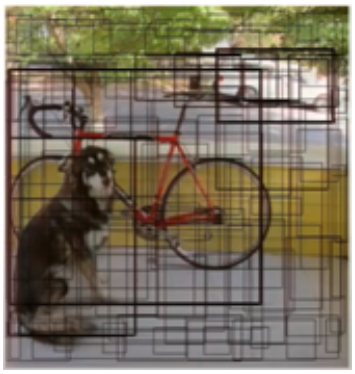


Figura 3.21.- Predicciones de todas las celdas

Cada BoundingBox (predicción) tiene 5 valores: x, y, w (ancho), h (alto) y confianza. Las coordenadas (x, y) representan el centro de la caja (ver Figura 3.19).

A parte de las BoundingBoxes, cada celda predice las C probabilidades de que el objeto (si lo hay) pertenezca a cada una de las C clases establecidas (ver Figura 3.22).

Una vez obtenidas las BoundingBoxes (Figura 3.21) y la probabilidad de las clases (Figura 3.22), se combinan estos resultados. Conceptualmente este resultado se puede apreciar en la Figura 3.23.

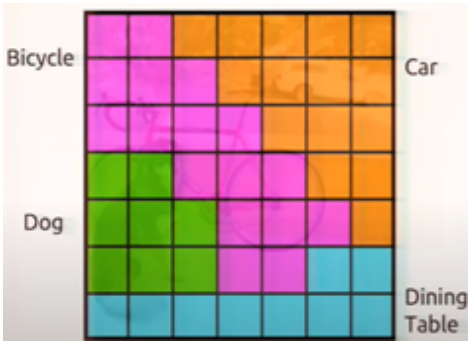


Figura 3.22.- Predicción de clases

Figura 3.23.- Combinación de bboxes y probabilidades de clase

Tal y como se puede apreciar en la Figura 3.23 se generan demasiadas predicciones. Para solucionar este problema se aplica el algoritmo Non Maximal Suppresion o NMS (ver sección 3.2.1.2). El resultado de aplicar NMS se puede apreciar en la Figura 3.24.

En la Figura 3.25 se muestra un resumen del funcionamiento de YOLOv1:

- División de la imagen en un grid.
- Propuesta de BoundingBoxes por las celdas detectoras.
- Propuesta de las clases de cada celda.
- Resultado final tras aplicar NMS.

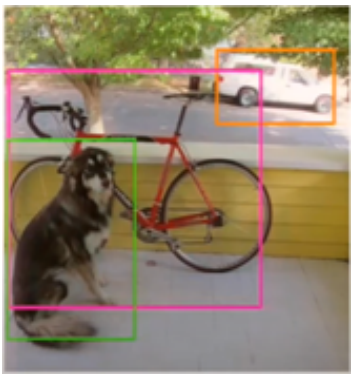


Figura 3.24.- Resultado después de aplicar NMS

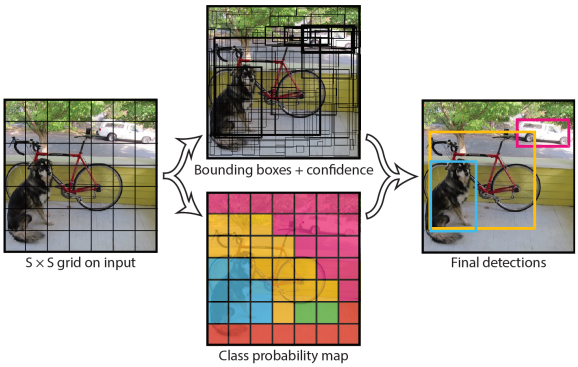


Figura 3.25.- Resumen de YOLOv1

3.2.1.2.- Inferencia

En última capa de la arquitectura de YOLOv1 (ver Figura 3.26) se genera un tensor de salida. Este tensor (que contiene las predicciones) tiene el siguiente formato: $S \times S \times (B \times 5 + C)$, siendo S el tamaño del grid, B el número de BoundingBoxes propuestas por cada celda, estando cada BoundingBox compuesta por el centro de la misma (x, y), el ancho y el alto, y siendo C el número de clases del conjunto de datos. Por ejemplo, en el conjunto de datos Pascal VOC [10] se ha utilizado un grid de 7×7, 2 BoundingBoxes propuestas por cada celda y 20 clases, luego el tensor de salida sería de $7 \times 7 \times 30$.

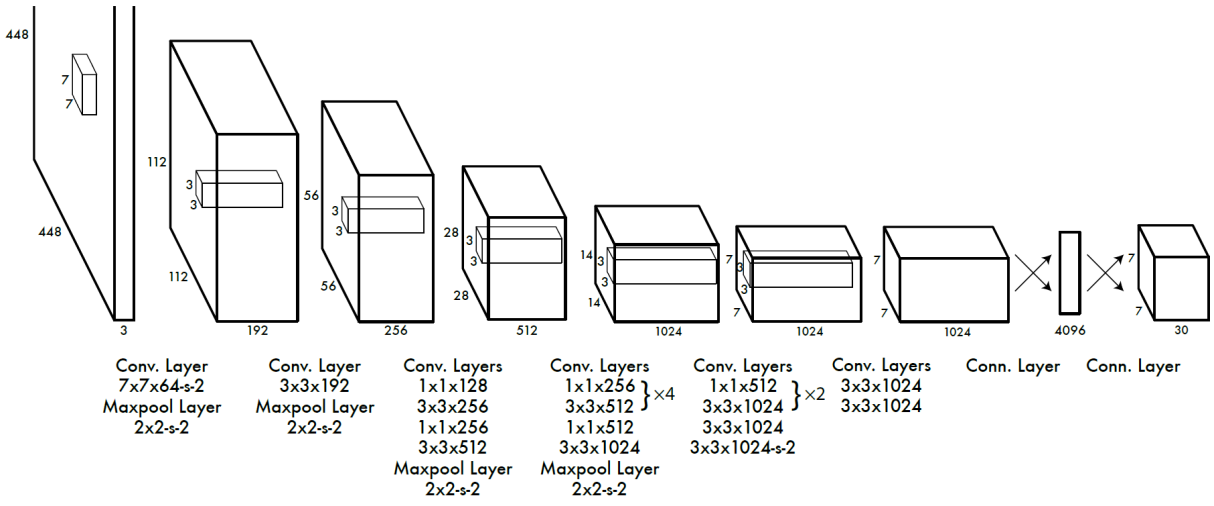


Figura 3.26.- Arquitectura de YOLOv1

El hecho de que cada celda produzca varias predicciones unido a que un objeto puede estar presente en varias celdas puede hacer que se generen demasiadas propuestas de BoundingBoxes (ver Figura 3.23). Solamente, es necesario una BoundingBox para cada objeto, es decir, es necesario eliminar gran parte de las propuestas, y para ello se aplica el algoritmo Non Maximal Suppresion o NMS.

El algoritmo **Non Maximal Suppresion** o **NMS** se describe en la Figura 3.27 y **se aplica para cada clase**. En resumen, **para cada clase**, se selecciona la predicción con más confianza. Si dicha predicción se solapa (IoU) con otra u otras predicciones en más de un cierto umbral (N_t) se eliminan las "otras", es decir, se selecciona la primera, que es la que tiene más confianza, y el resto (que tienen menos confianza) se eliminan. De esta forma solo hay una predicción por objeto. En la Figura 3.27 se muestra el algoritmo completo.

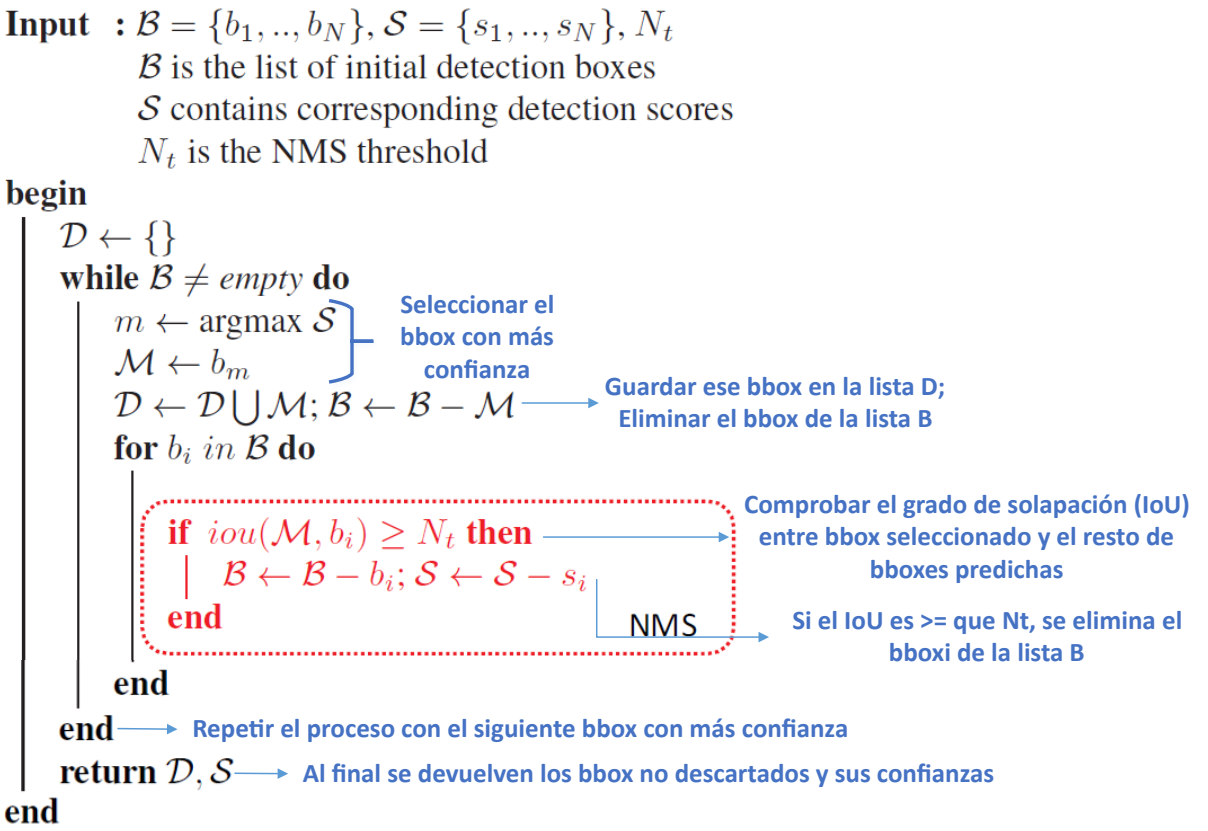


Figura 3.27.- Algoritmo NMS

Para que sea más sencillo de entender se explica con el resultado de aplicar NMS sobre la Figura 3.28. Para esta imagen se han propuesto dos BoundingBoxes, B_A y B_B para la clase Persona y tres para clase Gato, B_C, B_D y B_E. La confianza de cada clase se muestra al lado del nombre de cada BoundingBox

Para la clase Persona se aplica NMS, se establece el umbral de IoU a 0.5 ($N_t = 0.5$) y se selecciona la BoundingBox con más confianza, es decir, B_A. Luego se compara con el resto de BoundingBoxes, en este caso con B_B, y como $IoU(B_A, B_B) \geq N_t$ se elimina B_B.

Como ya se ha aplicado NMS sobre todas las BoundingBoxes de la clase Persona, se debe de aplicar sobre el resto de clases, en este caso Gato. Se selecciona la BoundingBox con más confianza, B_E en este caso. A continuación, se compara B_E, con B_C y B_D. Como $IoU(B_E, B_C) \geq N_t$ y $IoU(B_E, B_D) \geq N_t$ se elimina tanto B_C como B_D.

Como ya no quedan más clases, finaliza el algoritmo NMS, siendo el resultado final el mostrado en la Figura 3.29.

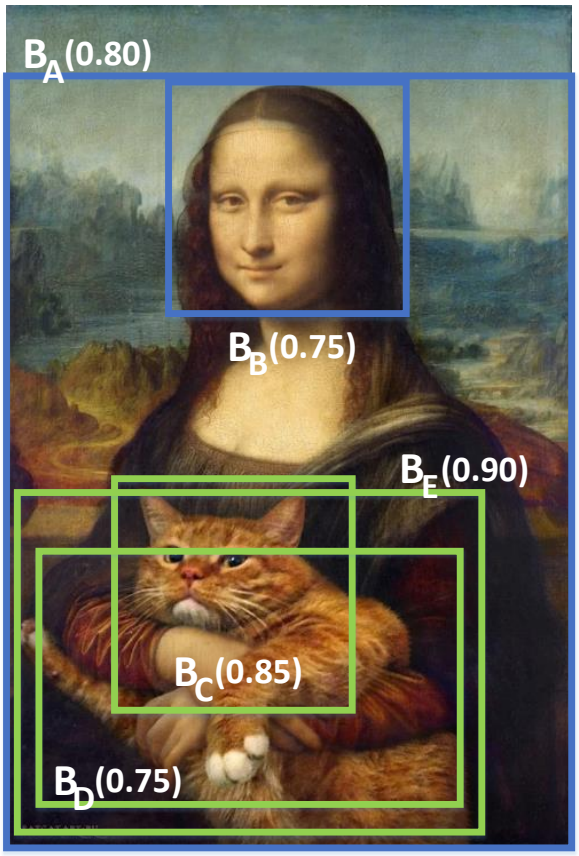


Figura 3.28.- Antes de aplicar NMS

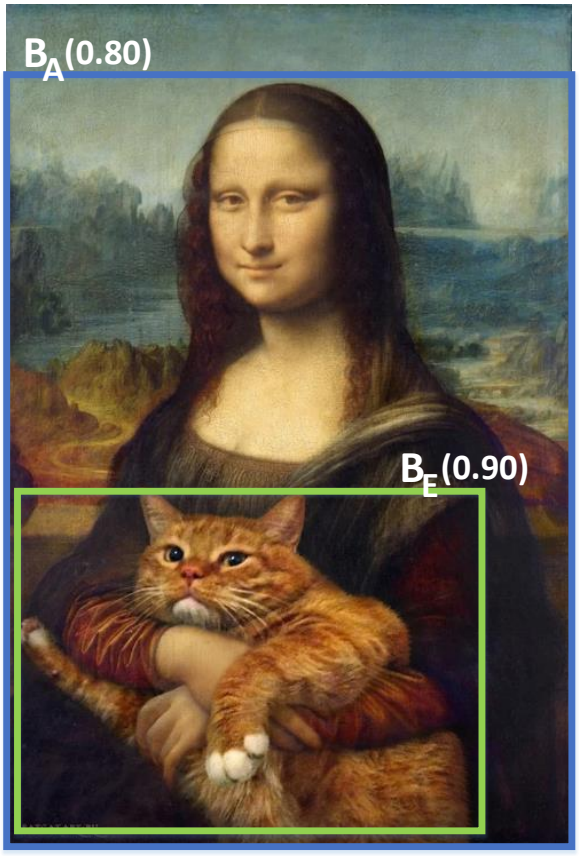


Figura 3.29.- Despúes de aplicar NMS

3.2.1.3.- Función de pérdida

La pérdida o loss no es más que el error cometido por una red neuronal, y el método para calcular la pérdida es la función de pérdida, también llamada función de loss.

La función de pérdida utilizada por esta red se basa en la suma del error cuadrático o sum-squared error (SSE) [8] [11] [12]. La suma del error cuadrático no diferencia entre los errores cometidos entre la localización del objeto (dónde está el objeto) y la clasificación (de que clase es el objeto). Hay que tener en cuenta que hay muchas celdas del grid que pueden no contener ningún objeto. Esto significa que la pérdida será muy importante, y el modelo no convergerá. Para resolver este problema se introducen los parámetros λ_{coord} y λ_{noobj} que se explicarán más adelante, en esta misma sección.

La función de pérdida utilizada por la red YOLOv1 [8] es la siguiente:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left(C_i-\hat{C}_i\right)^2$$

$$+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}\left(C_i-\hat{C}_i\right)^2$$

$$+\sum_{i=0}^{S^2}\mathbb{1}_{ij}^{obj}\sum_{c\in classes}\left(p_i(c)-\hat{p}_i(c)\right)^2$$

A continuación se analiza cada uno de los términos de la función:

- **Primer término:** Este primer término calcula la SSE entre las coordenadas predichas (BoundingBox) y las coordenadas del GroundTruth (que al estar anotadas a mano se supone que son 100 % correctas). Este proceso se realiza en la celdas responsables de realizar detecciones (1_{ij}^{obj}). Las coordenadas de la BoundingBox (x, y) se parametrizan para ser offsets (proporciones) de una celda del grid. De esta manera siempre tendrán un valor entre 0 y 1. El resultado se multiplica por λ_{coord}. El autor utiliza $\lambda_{coord} = 5.0$ [8]. Esta multiplicación refleja que la suma total de todos los términos de error, se sobrepondera.
- **Segundo término:** Este segundo término calcula la SSE de las predicciones de anchura (w) y altura (h). La anchura y la altura se normalizan respecto la anchura y altura de la imagen. De esta manera siempre tendrán un valor entre 0 y 1.

  Hay que tener en cuenta que no es lo mismo cometer pequeñas desviaciones en objetos grandes que en objetos pequeños. Por este motivo, en lugar de utilizar la anchura y la altura directamente, se utiliza su raíz cuadrada, ya que las desviaciones en objetos pequeños son más importantes. Por ejemplo, vamos a suponer que hay dos anotaciones realizadas a mano (GroundTruth) con una altura de 0.75 y 0.35 respectivamente. También, vamos a suponer que las predicciones de la altura son de 0.74 y de 0.34. A simple vista podría parecer que el error cometido es el mismo en ambos casos, es decir, 0.1. Pero esto no es así. En el caso de la primera predicción, el error será de $(\sqrt{0.74}-\sqrt{0.75})^2 = 0.00003356$, mientras que el error cometido en la segunda predicción será de $(\sqrt{0.34}-\sqrt{0.35})^2 = 0.00007247$. A simple vista parecía que el error era de 0.1, pero aplicando la fórmula se penaliza más el error cometido con objetos pequeños, siendo en este ejemplo dos veces superior.
- **Tercer y cuarto término:** En estos dos términos se calcula el error en la confianza predicha por cada celda detectora. Esto se puede apreciar en que en estos términos también se utiliza $\mathbb{1}_{ij}^{obj}$. Para los errores de confianza (C_i) se tienen en cuenta las celdas detectoras, para las cuales la función $\mathbb{1}_{ij}^{obj}$ toma un valor de 1 (tercer término), y las no detectoras, para las cuales la función $\mathbb{1}_{ij}^{noobj}$ toma un valor de 1 (cuarto término), contrariamente a lo visto en los errores de las coordenadas (x, y, w, h) en los que solo se consideran las celdas detectoras. Esto es lógico debido a que una celda detectora tiene un objeto real asociado con el que puede calcular los errores en las coordenadas entre las BoundingBox predichas y el GroundTruth. Para las celdas no detectoras no hay un objeto real asociado y por tanto no se pueden calcular errores en las coordenadas.

  Sin embargo, es necesario calcular el error de confianza para todas las celdas. Se espera un valor de confianza elevado (idealmente 1.0) para las celdas detectoras y un valor muy bajo (idealmente 0.0) para las no detectoras.

En el tercer término, el valor del C_i, lo calcula la red. El valor de C_i se estima para cada predicción como la IoU_{pred}^{truth}. Como previamente se indicó, $\hat{C}_i = Prob(Objeto) * IoU_{pred}^{truth}$, siendo la Prob(Objeto) = 1 y la IoU_{pred}^{truth} el grado de solapamiento entre el BoundingBox y el GroundTruth. Se considera Prob(Objeto) = 1 ya que se sabe con certeza que hay un objeto al estar trabajando con una celda detectora. En el caso del cuarto término, no se sabe el valor de C_i, pues no hay objeto con el que calcular la IoU_{pred}^{truth}. El valor de $\hat{C}_i$ debería de ser cero, debido a que en este caso $Prob(Objeto) = 0$, ya que se sabe con certeza que no hay objeto al estar trabajando con una celda no detectora. Entonces, en este caso $C_i = 0$ sin importar IoU_{pred}^{truth}.

Hay que tener en cuenta (cuarto término) que es muy probable que haya muchas celdas del grid que no sean detectoras (ver Figura 3.30). Esto hace que la puntuación de confianza se decremente, pudiendo llegar a alcanzar valores próximos a cero, y esto a menudo afecta al gradiente de las celdas que sí contienen objetos. Por este motivo, el modelo puede ser inestable y puede que el entrenamiento diverja desde el principio. Por este motivo se subpondera ($\lambda_{noobj} = 0.5$ [8]).

Figura 3.30.- Celda no detectora de objetos

- **Quinto término:** Se calcula la probabilidad de que el objeto pertenezca a una determinada clase. Es decir, este término itera sobre todas las clases en cada celda y calcula la SSE. Este término calcula el error de la clasificación, mientras que los términos primero y segundo calculan el error en la localización.

3.2.1.4.- Limitaciones de YOLOv1

YOLOv1 establece una serie de limitaciones en las predicciones ya que cada celda solo puede predecir dos BoundingBoxes y solo puede tener una clase. Esto limita el número de objetos cercanos que el modelo puede predecir. Por este motivo, el modelo no es muy adecuado para objetos pequeños que pueden aparecer en grupos, como por ejemplo una bandada de pájaros.

Además, como mucho puede detectar 49 objetos por imagen, ya que el grid típicamente es de 7×7, y solo puede detectar un objeto por celda.

3.2.2.- YOLOv2

YOLOv2 [13] nace para solventar algunos de los problemas presentes en YOLOv1. YOLOv1 tiene una serie de defectos si se compara con otros sistemas de detección. Por ejemplo, YOLOv1 tiene una mayor cantidad de errores de localización que Fast R-CNN. Además, presenta un recall más bajo si se compara con las redes basadas en regiones. Para tratar de solventar estos problemas surge YOLOv2, un detector más preciso pero que no disminuye su velocidad.

3.2.2.1.- Principios en los que se basa la red

El principal principio introducido en YOLOv2 son los **AnchorBoxes**.

El objetivo de los detectores es obtener una BoundingBox y la clase de cada objeto [14]. Una BoundingBox no es más que una caja que normalmente se representa mediante cuatro valores: x_{min}, y_{min}, x_{max} y y_{max}. Sin embargo, debido a la gran variedad de escalas y de relaciones de aspecto de los objetos, se ha llegado a la conclusión de que es difícil que la red converja si se pretende calcular estas BoundingBoxes de la manera tradicional (YOLOv1). Para solucionar este problema se propone el uso de los AnchorBoxes [15].

Un AnchorBox es una caja predefinida con un determinado tamaño. Este tamaño se determina utilizando el algoritmo k-means sobre el conjunto de entrenamiento (estos AnchorBoxes **se calculan antes del entrenamiento**) (ver sección 3.2.2.2). De esta manera, se conocen los tamaños de los objetos a detectar y se pueden establecer estos AnchorBoxes. Para saber dónde se colocan estos AnchorBoxes es necesario recordar el concepto de grid. Antiguamente los algoritmos de detección utilizaban una ventana deslizante sobre la imagen y procesaban solo esa parte de la imagen. Como esto era muy ineficiente, se decidió utilizar toda la imagen al mismo tiempo. Como la salida de la red es una matriz de $S \times S$ (S = 13 en YOLOv2) se decidió denominar a esta matriz grid. En otras palabras, los AnchorBoxes se sitúan en las celdas del grid, y comparten el mismo centroide (ver Figura 3.31). Cabe destacar, que los AnchorBoxes son los mismos para todas las celdas y que un AnchorBox puede ocupar varias celdas, ya que, en el caso de los objetos largos, lo más seguro es que estos ocupen varias celdas.

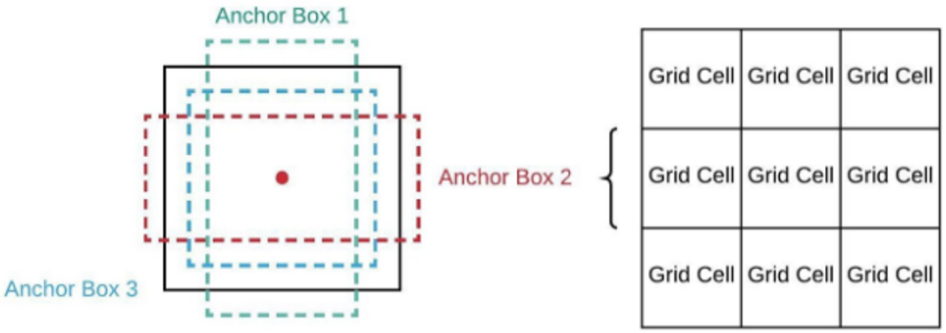


Figura 3.31.- Colocación de los AnchorBoxes en el grid

En el posterior entrenamiento se predicen offsets (ajustes) de estos AnchorBoxes (ver Figura 3.32). Esto funciona porque los AnchorBoxes son similares al GroundTruth (los AnchorBoxes se seleccionan antes del entrenamiento mediante k-means sobre el conjunto de entrenamiento), y gracias a estos AnchorBoxes, el entrenamiento es más sencillo, ya que solo hay que refinar las predicciones iniciales o AnchorBoxes.

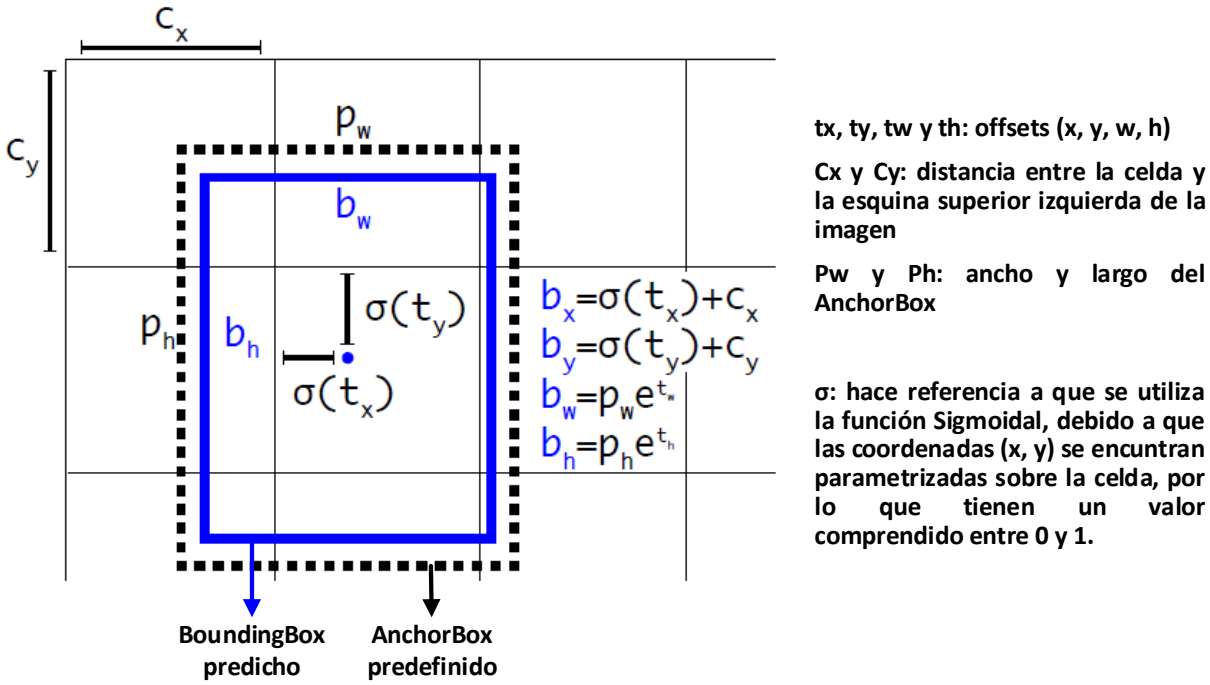


Figura 3.32.- AnchorBox - Cálculo de offsets

Como resumen la secuencia de tareas a realizar es [16]:

1. Calcular los tamaños (en píxeles) de los AnchorBoxes a partir del conjunto de entrenamiento utilizando k-means.
2. Entrenamiento:
   a) Calcular los offsets, y predecir así los BoundingBoxes correspondientes.
   b) Aplicar NMS (Non Maximum Suppression) para seleccionar los mejores BoundingBoxes.

Debido a la utilización de estos AnchorBoxes el tensor de salida tiene la siguiente forma: $S \times S \times B \times (NumAnchorBoxes + C)$. El analista debe de decidir el número de AnchorBoxes a utilizar (B = BoundingBoxes; S = tamaño del grid; C = número de clases).

3.2.2.2.- Estimación de los AnchorBoxes

En otras redes, Fast-RCNN por ejemplo, uno de los problemas es que las dimensiones (largo y ancho) de los AnchorBoxes se seleccionan manualmente [13], lo que significa que su elección puede no ser la más precisa. Aun así, la red puede aprender a ajustar los BoundingBoxes, pero si se seleccionan unos AnchorBoxes mejores, estas predicciones serán más sencillas, y por tanto más precisas.

En lugar de seleccionarlas manualmente, se propone la utilización del método de clustering k-means con el conjunto de entrenamiento. De esta forma se seleccionan automáticamente las dimensiones de estos AnchorBoxes. Ajustándose así, los AnchorBoxes se ajustan más a las diferentes relaciones de aspecto de los datos de entrenamiento. Se supone que los objetos de test tendrán unas relaciones de aspecto similares a los de entrenamiento.

El algoritmo estándar de k-means utiliza la distancia euclídea. Debido a que la distancia euclídea genera mayores errores en los objetos largos que en los pequeños, se ha decidido utilizar el IoU en lugar de la distancia euclídea, ya que el IoU es independiente del tamaño de los objetos.

Para escoger el mejor número de AnchorBoxes se ha de ejecutar k-means con varios valores de k, como se puede apreciar en la Figura 3.33. En esta figura se puede apreciar la evolución del IoU con una distinta cantidad de AnchorBoxes (eje de abscisas) para los conjuntos de datos Pascal VOC [17] y COCO [18].

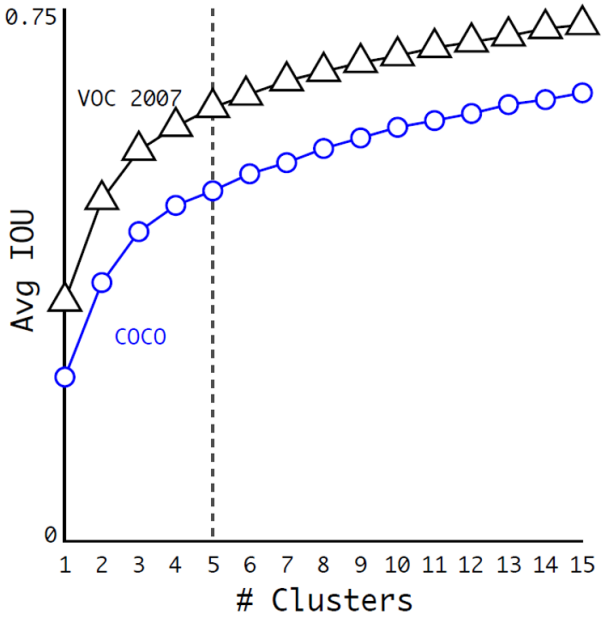


Figura 3.33.- Resultado de aplicar K-means

En este caso [13], se ha decidido que el número de AnchorBoxes sea cinco. Como se puede apreciar en la Figura 3.33, con seis AnchorBoxes se consigue un IoU más alto. Aun así, se ha decidido que el número de AnchorBoxes sea cinco porque la diferencia de IoU con cinco y seis AnchorBoxes no es significativa, y porque cuantos

más AnchorBoxes se utilicen mayor será el coste computacional (tensor de salida = $S \times S \times B \times (NumAnchorBoxes + C)$).

3.2.2.3.- Arquitectura de la red

La gran mayoría de detectores utilizan clasificadores preentrenados con ImageNet. Por ejemplo, AlexNet utiliza imágenes de 256×256. YOLOv1 entrena con 224×224 y aumenta la resolución a 448×448 para la detección.

En YOLOv2 se utiliza una resolución en el entrenamiento de 448×448 para la clasificación, lo que aumenta el mAP un 4 %.

La arquitectura de la red [13] se puede analizar en la Tabla 3.1. La red es la misma que en YOLOv1 (en cuanto a capas se refiere) pero eliminando las capas Fully Connected utilizadas al final de la red.

Num	Tipo	Filtros	Stride	Salida
0	Convolutional	3 x 3 x 32		224 x 224
1	Max pool	2 x 2	2	112 x 112
2	Convolutional	3 x 3 x 64		112 x 112
3	Max pool	2 x 2	2	56 x 56
4	Convolutional	3 x 3 x 128		56 x 56
5	Convolutional	1 x 1 x 64		56 x 56
6	Convolutional	3 x 3 x 128		56 x 56
7	Max pool	2 x 2	2	28 x 28
8	Convolutional	3 x 3 x 256		28 x 28
9	Convolutional	1 x 1 x 128		28 x 28
10	Convolutional	3 x 3 x 256		28 x 28
11	Max pool	2 x 2	2	14 x 14
12	Convolutional	3 x 3 x 512		14 x 14
13	Convolutional	1 x 1 x 256		14 x 14
14	Convolutional	3 x 3 x 512		14 x 14
15	Convolutional	1 x 1 x 256		14 x 14
16	Convolutional	3 x 3 x 512		14 x 14
17	Max pool	2 x 2	2	7 x 7
18	Convolutional	3 x 3 x 1024		7 x 7
19	Convolutional	1 x 1 x 512		7 x 7
20	Convolutional	3 x 3 x 1024		7 x 7
21	Convolutional	1 x 1 x 512		7 x 7
22	Convolutional	3 x 3 x 1024		7 x 7
23	Convolutional	1 x 1 x 1000		7 x 7
24	Avgpool	Global		1000
25	Softmax			

Tabla 3.1.- Arquitectura YOLOv2

En YOLOv2 [19] las predicciones de clase no son a nivel de celda si no que son a nivel de BoundingBox. Cada predicción incluye los siguientes parámetros para la BoundingBox: x, y, w, h, s (score = confianza de la predicción), y probabilidad de pertenecer a cada una de las clases. Por ejemplo, si B = 5 y C = 20, se generarían 5 BoundingBoxes con 25 parámetros, es decir, 125 parámetros por celda (ver Figura 3.34).

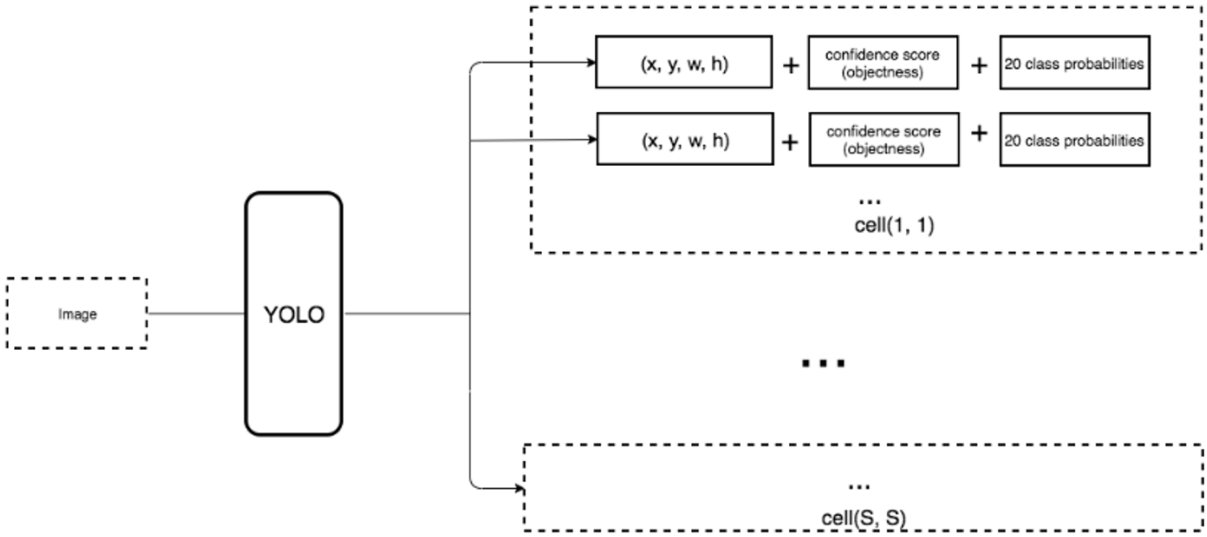


Figura 3.34.- Predicciones por BoundingBox

3.2.3.- YOLOv3

YOLOv3 [20] introduce una serie de cambios de diseño para corregir algunos de los defectos de las versiones anteriores: fallo en la detección de objetos pequeños u objetos que se encuentren muy próximos.

3.2.3.1.- Principios en los que se basa la red

YOLOv3 se sigue basando en los principios empleados en YOLOv1 y YOLOv2. El principio más relevante introducido en YOLOv3 es la **predicción a tres escalas** [20]. YOLOv3 extrae características de cada una de estas escalas utilizando un concepto similar a las redes piramidales de características (feature pyramind networks). Ahora YOLOv3 realiza tres predicciones (una por escala), por este motivo el tensor de salida tiene la siguiente forma: $N \times N \times [3 \times (4 + 1 + C)]$. Siendo N el tamaño del grid en cada escala, tres el número de AnchorBoxes en cada escala, cuatro el número de atributos de las predicciones (x, y, ancho y largo), un valor de confianza y C clases. Hay un tensor por escala, es decir, hay tres tensores.

Otra novedad, es que YOLOv3 permite la clasificación multietiqueta (multilabel) [20], es decir, un objeto puede pertenecer a dos categorías, por ejemplo, perro y animal. Por este motivo, ya no se selecciona la clase con más confianza. Esto se podía hacer cuando las etiquetas eran excluyentes, por ejemplo, perro y coche. En otras palabras, antes un objeto solo podía pertenecer a una clase y ahora puede pertenecer a varias.

En YOLOv3 [21], cada probabilidad de clase se predice utilizando un umbral (threshold). Las clases que tengan una confianza mayor que ese umbral (threshold) se asignan a esa predicción, de esta forma una predicción puede tener asociadas varias clases.

3.2.3.2.- Predicción a tres escalas

Para realizar estas predicciones en tres escalas diferentes [21] [22] (ver Figura 3.35), se realiza un downsampling de la imagen (originalmente de 416×416) de 32, 16 y 8.

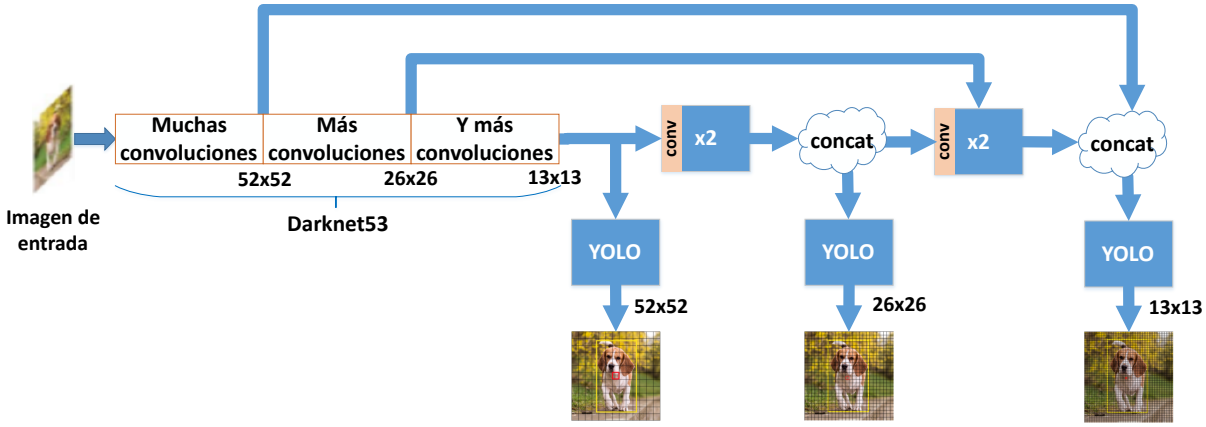


Figura 3.35.- Predicción a tres escalas en YOLOv3 sobre una imagen de 416×416

Yolov3 utiliza Darknet53 como extractor de características. Darknet53 recibe este nombre debido a que se compone por 53 capas convolucionales. Además está inspirado FPN (Feature-Pyramid Network) [23]. FPN es una topología de red en la que el mapa de características disminuye gradualmente en la dimensión espacial, pero más tarde el mapa de características se expande de nuevo y se concatena con mapas de características anteriores con los tamaños correspondientes. Este procedimiento se repite, y cada mapa de características concatenado alimenta a una cabeza de detección separada [22] (ver Figura 3.36).

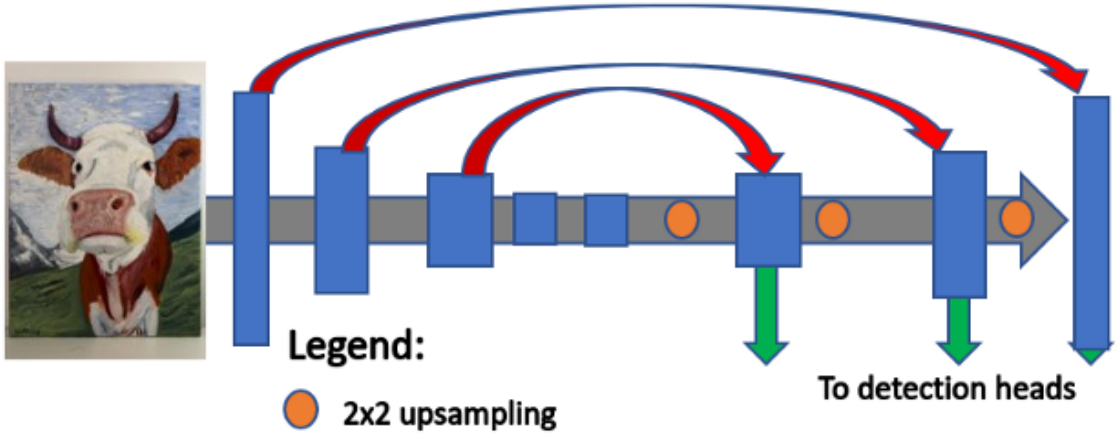


Figura 3.36.- FPN (Feature Pyramid Network)

Volviendo a la Figura 3.35, FPN permite procesar objetos de diferentes tamaños: el bloque de detección 13×13 se encarga de detectar los objetos más grandes, mientras que el bloque de 52×52 se especializa en los objetos pequeños. Es importante destacar que YOLOv3 sigue utilizando AnchorBoxes (ver sección 3.2.2.1) al igual que YOLOv2, pero

en este caso, para YOLOv3 se aplica una modificación: para cada escala se utilizan tres AnchorBoxes, es decir, como hay tres escalas en total se utilizan nueve AnchorBoxes [20].

Gracias a la detección en múltiples escalas, la detección de objetos pequeños es mucho más precisa (ver Figura 3.37), resolviendo así uno de los problemas de las anteriores versiones de YOLO.

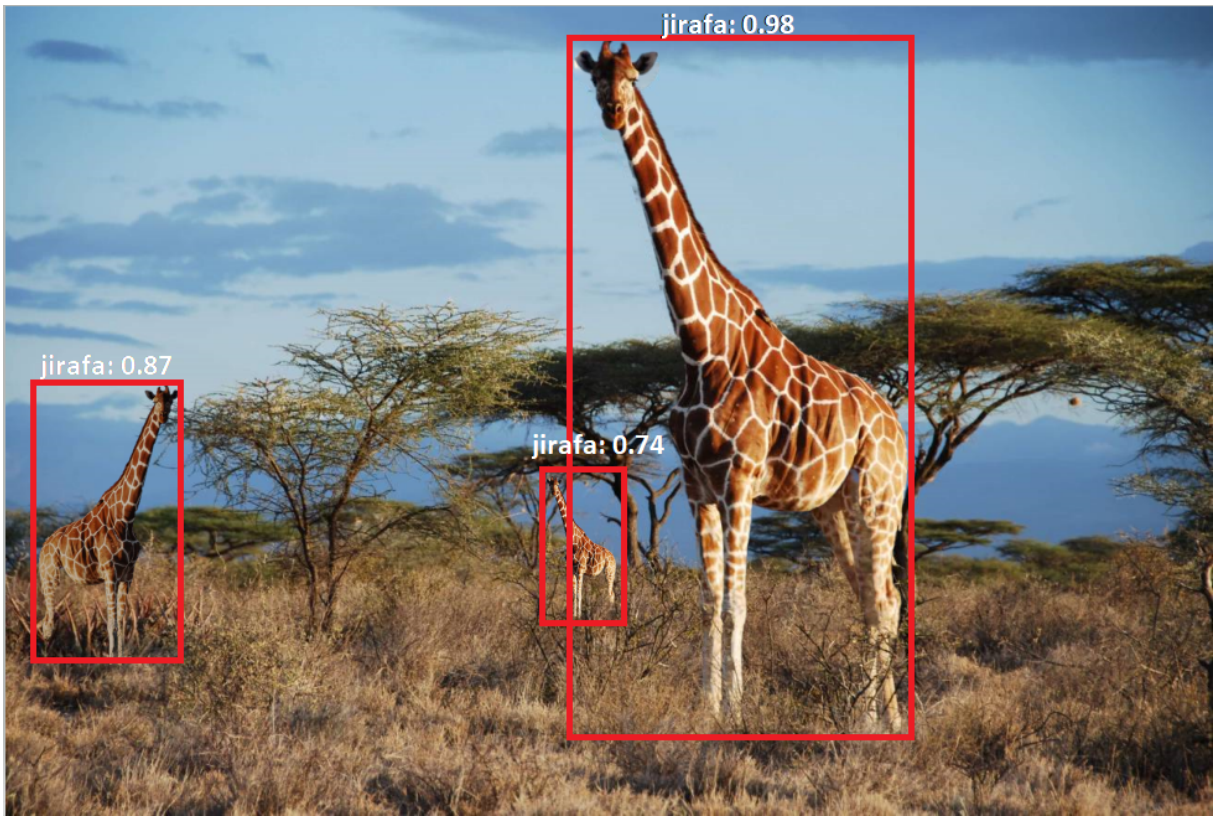


Figura 3.37.- Ejemplo de detección en tres escalas diferentes

3.2.3.3.- Arquitectura

Como se ha explicado en la sección anterior YOLOv3 utiliza Darknet-53 como extractor de características (en YOLOv2 se utiliza Darknet-19). Como se puede ver en la Figura 3.38, se trata de una red puramente convolucional (FCN).

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

Figura 3.38.- Capas de Darknet-53

El hecho de que YOLOv3 sea un red puramente convolucional (FCN), hace que el tamaño de entrada pueda ser variable [24]. Este hecho puede ser interesante durante la inferencia ya que permite entrenar con un tamaño de imagen y realizar detecciones sobre otro tamaño de imagen.

A priori, esto es una magnifica idea pero durante el entrenamiento no se puede llevar a cabo debido a que lo normal es que se procesen las imágenes por lotes (batch) por lo que es necesario que todas las imágenes tengan el mismo ancho y alto durante la fase de entrenamiento.

Además, durante el entrenamiento y la inferencia se recomienda que el ancho y alto de las imágenes sea múltiplo de 32 para obtener mejores resultados. Esto se debe a que como se ha mencionado anteriormente se utiliza un downsampling de 32, 16 y 8, de manera que si el ancho y alto de la imagen es múltiplo de 32 el resultado siempre será un número entero.

3.2.4.- YOLOv4

Hasta el momento las redes de la familia YOLO, YOLOv1 [8], YOLOv2 [13] y YOLOv3 [20] habían sido desarrolladas principalmente por Joseph Redmon. En esta ocasión, en YOLOv4 se produce un cambio de autor [25].

3.2.4.1.- Principios en los que se basa la red

La gran mayoría de detectores precisos son lentos, lo que impide poder realizar entrenamientos rápidos con grandes conjuntos de datos, como por ejemplo con COCO [18]. YOLOv4 resuelve estos problemas, permitiendo el procesado en tiempo real y utilizando una sola GPU durante el entrenamiento.

Un detector moderno normalmente está compuesto por dos partes [25], una red backbone preentrenada en ImageNet (pesos adaptados para identificar las características más relevantes) y otra parte, llamada cabeza (head) que consume las características extraídas del backbone y las utiliza para predecir clases y BoundingBoxes. Para quienes utilizan una o varias GPU, esta red backbone podría ser VGG, ResNet, ResNeXt or DenseNet. Para quienes utilicen, por el contrario, una CPU podrían utilizar SqueezeNet, MobileNet o ShuffleNet.

En YOLOv4 se ha considerado como posibles backbones CSPResNext50, CSPDarknet53 y EfficienteNet-B3 [26]. CSPResNext50 y CSPDarknet53 se basan en DenseNet que fue diseñada para conectar las capas de una CNN con los siguientes objetivos:

- Solucionar/mejorar el problema del gradiente descendiente.
- Reforzar la propagación de las características.
- Fomentar la reutilización de características.
- Reducir el número de los parámetros de la red.

EfficientNet [27] fue diseñada por Google Brain para estudiar el problema del escalado en las CNNs. Hay muchas decisiones que puedes tomar al escalar las capas convolucionales: tamaño de entrada, tamaño del ancho de escalada, profundidad de escalada y escalar todo lo anterior. Los autores de EffienceNet demuestran que existe un punto óptimo para todas estas cuestiones.

Se ha demostrado que EfficientNet presenta mejores resultados cuando se trata de la clasificación, pero sin embargo en YOLOv4 se ha creído más oportuno utilizar CSPDarknet53, debido a que tiene un mejor rendimiento en la tarea de detección de objetos [25].

Antes de pasar a la cabeza de la red, es necesario hablar del cuello (neck) de la red. El cuello sirve para combinar las características obtenidas en el backbone. En YOLOv4 se utiliza PANet [28] como cuello, y además añade un bloque SPP [29] después de CSPDarknet53 para separar las características más importantes obtenidas del backbone.

Por otro lado, la cabeza suele ser de dos tipos: un detector de objetos de una etapa o de dos etapas. Por ejemplo, un detector de dos etapas sería la familia R-CNN, donde primero se proponen una serie de regiones y luego se procesa cada una de estas regiones. Un ejemplo de un detector de una etapa sería YOLO o SSD. YOLOv4 utiliza a YOLOv3 como cabeza (head).

3.2.4.2.- Modificaciones que no alteran el tiempo de inferencia

Con el objetivo de mejorar la precisión de una red, se pueden llevar a cabo una serie de modificaciones. Estas modificaciones pueden suponer un aumento del coste temporal de la inferencia, pero sin embargo, existen otras modificaciones que no lo aumentan. A estas modificaciones se les conoce como "Bag of freebies" o "Bolsa de regalos" [25].

Estas modificaciones, básicamente, consisten en realizar un aumento de datos, gracias al cual se consigue una mayor variedad de imágenes, aumentanto la robustez de los modelos. Anteriormente, se han analizado dos técnicas de aumento de datos: distorsiones y alteraciones fotométricas (ver sección 3.1), y claramente, y de acuerdo con los propios experimentos realizados durante este proyecto, mejoran los resultados. Las alteraciones fotométricas, consisten en modificar el brillo, el contraste, el tono, la saturación y el ruido de las imágenes. En el caso de las distorsiones fotométricas, se añade de forma aleatoria escalados, recortes, giros y rotaciones. Estos métodos de aumento de datos se basan en ajustar los píxeles, por lo que toda la información de original se mantiene.

Sin embargo, existen otros métodos como por ejemplo **simular la oclusión de objetos**. Este método ha dado buenos resultados tanto en clasificación de imágenes, como en detección de objetos. Por ejemplo, Random Erase y CutOut (técnicas de oclusión de objetos) pueden seleccionar rectángulos aleatoriamente y modificarlos. En la Figura 3.39 se muestra un ejemplo de la técnica Random Erase.

Figura 3.39.- Ejemplo de oclusión de objetos

Funcionalmente, estas técnicas de oclusión de datos sirven para reducir el sobreajuste (ver sección 3.1) [30].

También existen otras técnicas que proponen la utilización de varias imágenes juntas para mejorar el aumento de datos, como ejemplo MixUp. Otra opción, es utilizar CutMix (ver Figura 3.40), que combina recortes de otras imágenes con la imagen original, lo que fuerza al modelo a aprender con muchas más características.

Figura 3.40.- Ejemplo de CutMix

Otra técnica es el aumento de datos en mosaico (Mosaic Data Augmentation) que combina cuatro imágenes en una, forzando al modelo, a identificar objetos en una escala menor que la normal, tal y como se muestra en la Figura 3.41.

Figura 3.41.- Ejemplo de aumento de datos en mosaico

Otra técnica de Aumento de Datos es Self-Adversarial Training (SAT). SAT localiza la parte de la imagen en la que la red más confía durante el entrenamiento, para luego editar dicha parte y forzar, así, a que aprenda a generalizar nuevas características [30].

En la Figura 3.42 se muestra un resumen de las técnicas de aumento de datos analizadas.

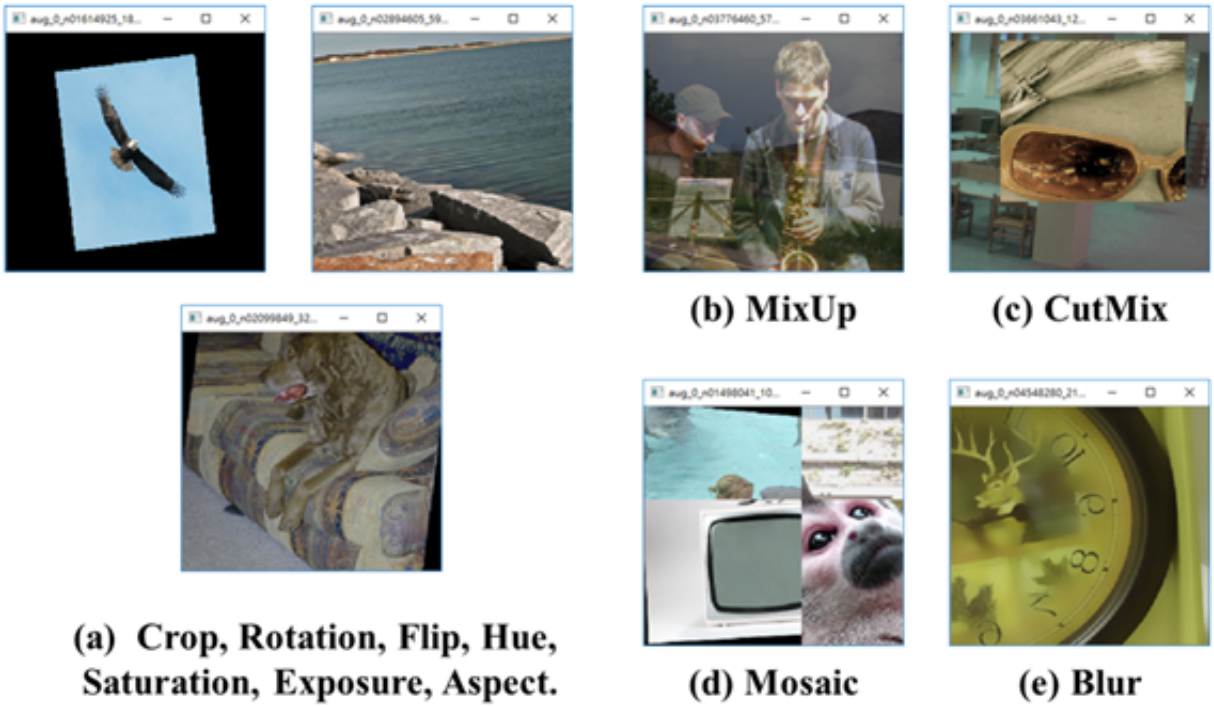


Figura 3.42.- Resumen de las diferentes técnicas de aumento de datos evaluadas en YOLOv4

3.2.4.3.- Arquitectura

El objetivo es encontrar el equilibrio entre la resolución de entrada de la red, el número de capas convolucionales, el número de parámetros (tamaño_$filtro^2$ × $filtros$×canales_imagen) y el número de capas de salida (filtros). De acuerdo con un análisis empírico [25] se demuestra que CSPResNext50 es mejor que CSPDarknet53 en términos de clasificación, pero peor en términos de detección de objetos. Por lo que se utilizará CSPDarknet53 como backbone de la red. CSPDarknet53 se basa en Darknet53 (ver sección 3.2.3.3), pero añade la idea de Cross-Stage-Partial-Networks(CSPNet) [31]. CSPNet separa la entrada de las capas densas (conjunto de múltiples capas convolucionales) en dos partes. La primera parte (ver Figura 3.43 (b)) hace que la entrada fluya hasta la siguiente capa, mientras que la segunda parte hace que la entrada fluya a través de la capa densa [32]. CSPNet reduce el coste temporal en un 20 % y mejora la precisión en el conjunto de datos COCO [31].

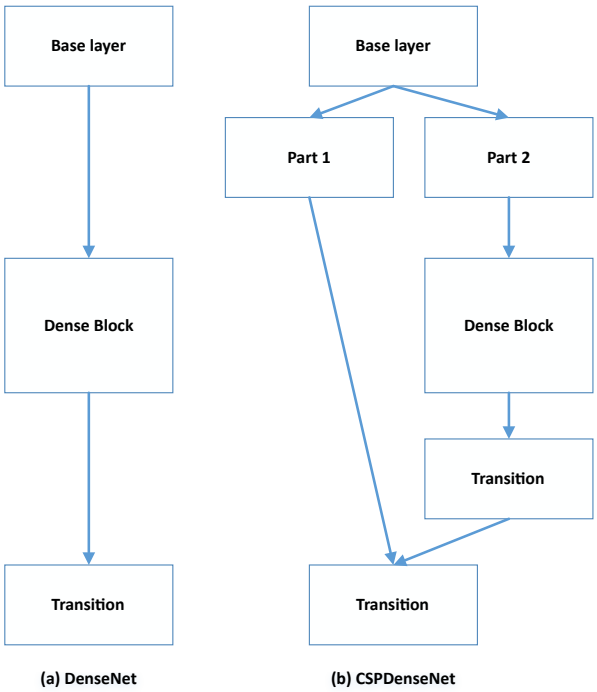


Figura 3.43.- Explicación de CSPNet

Un modelo que es óptimo para la clasificación no siempre es óptimo para detección. A diferencia del clasificador, el detector requiere:

- Mayor tamaño de entrada (resolución): para poder detectar objetos pequeños.
- Más capas.
- Más parámetros.

Hipotéticamente, se puede asumir que un modelo con un tamaño de campo receptivo grande (muchas capas convolucionales 3×3) y un gran número de parámetros debería ser seleccionado como backbone. Esta justificación teórica, junto con una gran cantidad de experimentos, demuestran que CSPDarknet53 es el modelo óptimo como backbone para el detector [25].

Además, se añadió un bloque SPP sobre CSPDarknet53, debido a que separa las características más importantes y no reduce la velocidad de la red. Se utiliza PANet como método de agregación de parámetros de diferentes niveles backbone para diferentes niveles de detector, en lugar de FPN utilizado en YOLOv3.

En resumen, la arquitectura de YOLOv4:

- CSPDarknet53 backbone.
- SSP como módulo adicional.
- PANet como método de agregación (cuello o neck).
- YOLOv3 como cabeza (head).

3.2.5.- YOLOv5

Finalmente, se encuentra YOLOv5 [33]. A diferencia de otras versiones, en diciembre de 2020 no se encuentra disponible un artículo científico sobre la red. Por esta razón, se cita de acuerdo con lo especificado por el propio autor de la red en su repositorio de Github.

Existe una gran controversia entre YOLOv4 y YOLOv5, debido a que ambos utilizan técnicas similares y ambos han sido desarrollados en fechas próximas (23 de abril en el caso de YOLOv4 y 27 de mayo de 2020 en el caso de YOLOv5).

Ambos comparten arquitectura (backbone: CSPDarknet53, cuello (neck): PANet y cabeza (head): YOLOv3) y utilizan aumento de datos en mosaico. Por ejemplo, en la Figura 3.44 se puede apreciar el resultado de aplicar el aumento en mosaico sobre el conjunto de datos DIOR (ver sección 4.1).

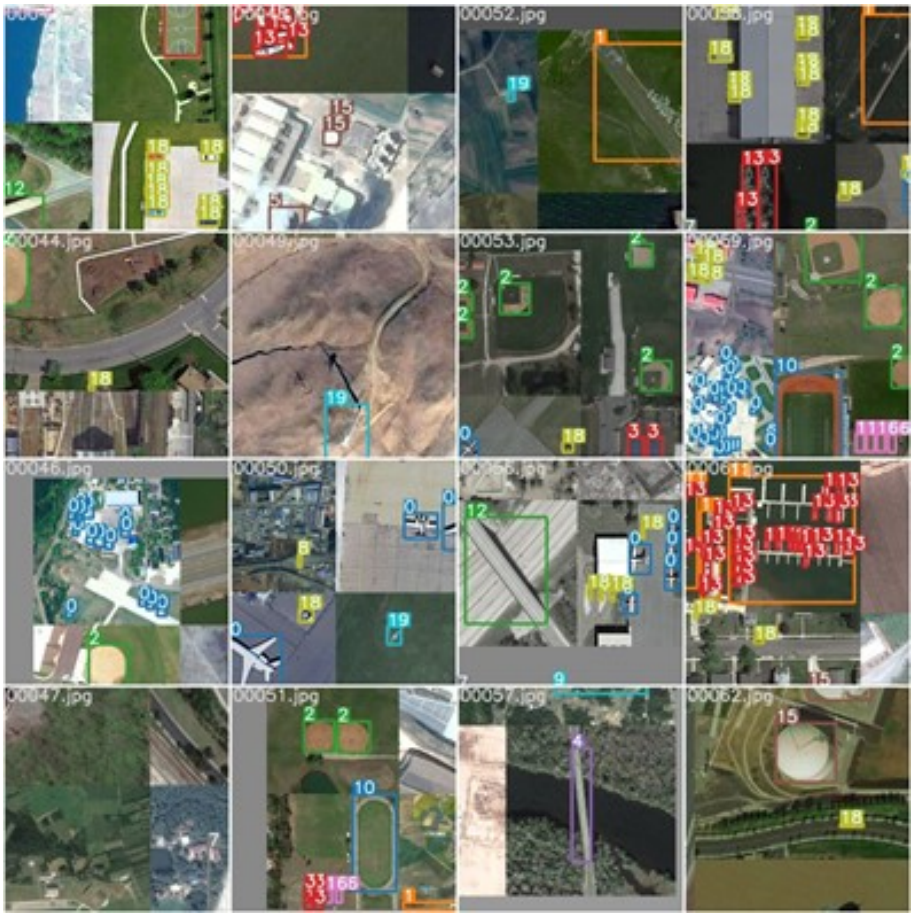


Figura 3.44.- Aumento de datos en mosaico sobre el conjunto de datos DIOR

3.2.5.1.- Principios en los que se basa la red

En este caso, al no existir artículo oficial en diciembre de 2020 y al tratarse de una red muy similar a YOLOv4 se analiza con más detalle la propia implementación que los conceptos teóricos.

Lo primero que cabe destacar es que existen cuatro modelos de YOLOv5. La principal diferencia entre ellos es que unos son más rápidos y sacrifican AP (ver sección 3.3), mientras que otros son más lentos y aumentan el AP, tal y como se puede apreciar en la Figura 3.45 y en la Tabla 3.2. Los cálculos del AP han sido realizados sobre el conjunto de datos COCO [18].

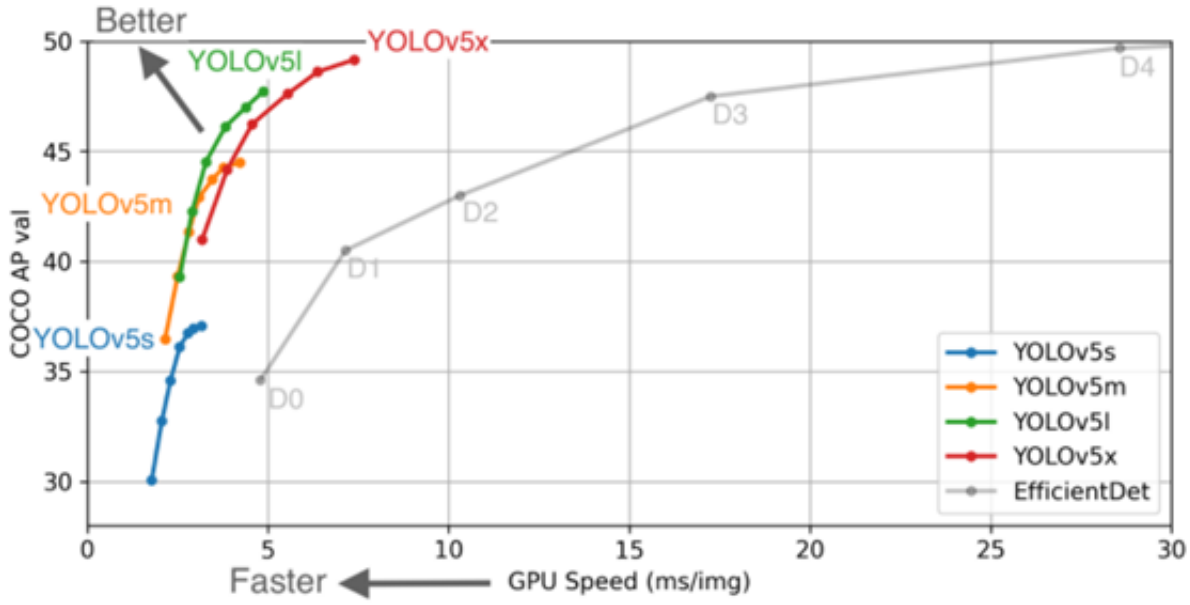


Figura 3.45.- Comparativa de los modelos disponibles en YOLOv5

Model	AP^{val}	AP^{test}	AP_{50}	$Speed_{GPU}$	FPS_{GPU}		Params	FLOPS
YOLOv5s	37.0	37.0	56.2	**2.4ms**	**416**		7.5M	13.2B
YOLOv5m	44.3	44.3	63.2	3.4ms	294		21.8M	39.4B
YOLOv5l	47.7	47.7	66.5	4.4ms	227		47.8M	88.1B
YOLOv5x	**49.2**	**49.2**	**67.7**	6.9ms	145		89.0M	166.4B

Tabla 3.2.- Comparativa de los modelos disponibles en YOLOv5

Una de las principales contribuciones de YOLOv5 es la introducción de esta familia de CNNs al entorno PyTorch. El framework Darknet (en el que se implementan el resto de las versiones de YOLO) está principalmente escrito en C por lo que se tiene un gran control sobre la red implementada, lo que es genial desde el punto de vista de la investigación, pero presenta la desventaja de que es más difícil incorporar nuevas técnicas que hayan demostrado su eficacia.

3.2.5.2.- Auto AnchorBoxes

En la familia de las redes YOLO (a excepción de YOLOv1) las predicciones se calculan refinando los AnchorBoxes, como se ha analizado anteriormente (ver sección 3.31).

En YOLOv5 se siguen utilizando estos AnchorBoxes, y al igual que en otras versiones de la familia YOLO, se utiliza el algoritmo k-means para calcular los mejores AnchorBoxes. La gran mayoría de redes traen por defecto los AnchorBoxes para MS COCO [18]. Lo novedoso de YOLOv5 es que incluye un algoritmo genético para calcular estos AnchorBoxes, además de k-means. Si se dejan los AnchorBoxes de MS COCO, tras unas cuantas generaciones, el algoritmo genético calcula y aplica en la red unos nuevos AnchorBoxes (antes del entrenamiento), adaptados al conjunto de datos en cuestión. Cabe destacar que esto **solo** se aplica, si los AnchorBoxes iniciales no "encajan" con el conjunto utilizado.

3.2.5.3.- Tamaño de entrada de la imagen

Como ya se comentó en la sección dedicada a YOLOv3 (ver sección 3.2.3.3), durante la inferencia es interesante la posibilidad de utilizar el mismo modelo sobre imágenes de diferentes dimensiones. También se comentó la necesidad de que durante el entrenamiento, el tamaño de las imágenes sea el mismo.

Este último hecho, puede suponer un problema si no se disponen de imágenes con las mismas dimensiones. Además hay que añadir el factor de que el tamaño de las imágenes sea múltiplo de 32. Esto se debe a que, como se explicó en el apartado de YOLOv3 (ver sección 3.2.3.2) es necesario utilizar un stride de 32, 16 y 8.

Para solucionar este problema, YOLOv5 incluye una técnica (otras versiones también la incluyen, como por ejemplo YOLOv3) conocida como LetterBox. Al comenzar el entrenamiento o la inferencia, es necesario indicar el tamaño de las imágenes (un único valor para todas las imágenes, y si se indica un valor que no sea múltiplo de 32 automáticamente se calcula el múltiplo de 32 más cercano al valor introducido), y si una imagen no tiene dicho tamaño se redimensiona. Para realizar este redimensionamiento se utiliza la técnica **LetterBox**.

La técnica LetterBox se define como el escalado de la imagen de entrada para que quede dentro del marco de la imagen de salida deseada (LetterBox), conservando la relación de aspecto de la imagen original. Para poder mantener esta relación de aspecto es necesario rellenar (padding) algunos píxeles de la imagen (en la implementación de YOLOv5 se rellena de color constante), tal y como se puede apreciar en la Figura 3.46 y en la Figura 3.47.

Figura 3.46.- Ejemplo de utilizar LetterBox para redimensionar una imagen

Como se puede apreciar en la parte superior de la Figura 3.46, la imagen original tiene unas dimensiones de 1280 x 720 y el tamaño de imagen seleccionado para la inferencia es de de 416. En este ejemplo, se muestran dos opciones: aplicar un LetterBox cuadrado de 416 x 416, lo que a priori puede parecer más lógico, o utilizar un tamaño de 416 en la parte más grande de la imagen (ancho en este caso) y utilizar un múltiplo de 32 en la otra componente (alto en este caso). Inicialmente, se utilizaban recortes cuadrados, pero tras un análisis empírico se llegó a la conclusión de que era más eficiente, optar por la segunda opción [34]. En este caso, el cambio de recortes cuadrados a recortes rectangulares supuso que el tiempo de inferencia se redujese de 1.008 s a 0.632 s, obteniendo exactamente los mismos resultados: una corbata y dos personas.

En la Figura 3.47 se muestra otro ejemplo. En este caso una imagen de 810×1080 en la que se quiere realizar detecciones con tamaño de imagen también de 416. Al igual que en el ejemplo anterior, se puede aplicar un LetterBox cuadrado de 416×416, o en este caso, un LetterBox de 320×416. Esta segunda opción, a diferencia del ejemplo de la Figura 3.46, se establece a 416 el alto, ya que es la componente mayor, y un múltiplo de 32 el ancho. En este caso [34], el tiempo de inferencia con un LetterBox cuadrado de 416×416 es de 0.999 s, mientras que en LetterBox de 320×416 el tiempo es de 0.767 s, obteniendo exactamente los mismos resultados: un bolso, tres personas y un autobús.

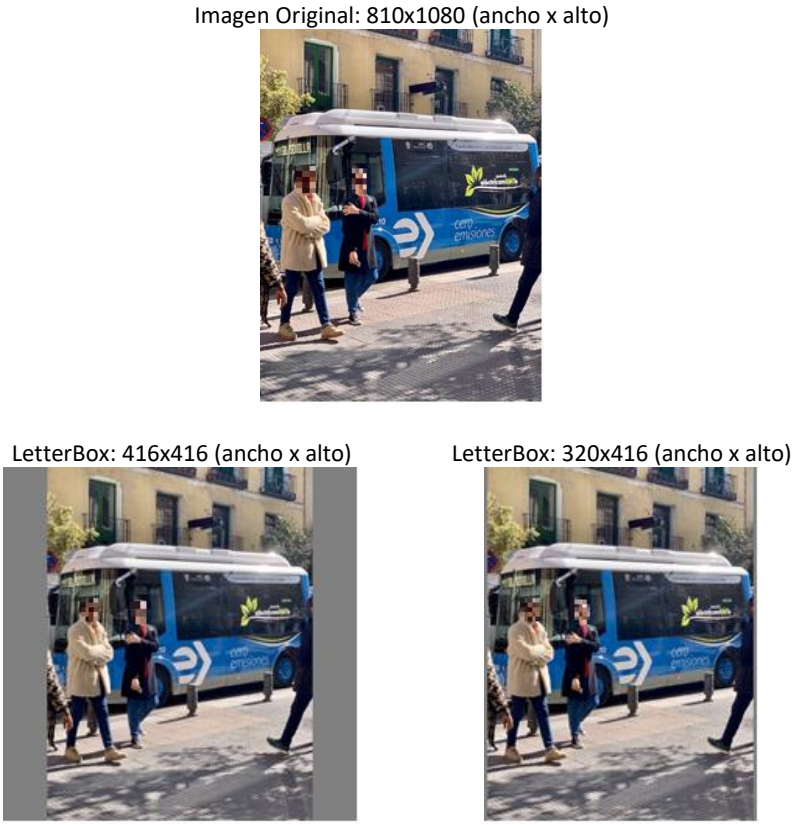


Figura 3.47.- Ejemplo de utilizar LetterBox para redimensionar una imagen

Gracias a esta técnica, da igual cual sea el tamaño de entrada establecido (evidentemente siempre que el redimensionamiento no sea demasiado grande) porque todas las imágenes se dimensionaran a dicho tamaño, manteniendo la relación de aspecto.

3.2.6.- SSD

SSD o Single Shot Multibox Detector [35] es un detector publicado en 2016, y en cuanto a resultados es similar a otras redes. Al igual que en YOLO se trabaja con toda la imagen.

3.2.6.1.- Principios en los que se basa la red

SSD se publicó en 2016, por lo que muchos de los principios en los que se basa ya se han analizado en las secciones de la familia YOLO. A continuación, se comentan sus particularidades.

- **Grid**: al igual que YOLO, SSD trabaja con la imagen completa por lo que aplica un grid sobre la imagen.
- **AnchorBoxes**: denominados DefaultBoxes en el artículo oficial [35], siguen el mismo concepto ya explicado (ver sección 3.2.2.1). La diferencia es que en YOLO (en YOLOv1 no se utilizan) estos AnchorBoxes se seleccionan mediante el algoritmo de clustering K-means, y en SSD se seleccionan manualmente [36].
- **Predicción en múltiples escalas**: SSD al igual que YOLO (a partir de la versión tres) realiza predicciones en múltiples escalas. En este caso SSD, realiza predicciones en dos escalas. En la Figura 3.48 se muestra un ejemplo de este tipo de predicciones. El perro mostrado en la Figura 3.48 encaja con un AnchorBox (el rojo, mostrado en la parte derecha de la Figura 3.48), mientras que el gato encaja con otro AnchorBox (el azul, mostrado en el centro de la Figura 3.48). El perro al ser más grande se detecta utilizando un grid de 4×4, mientras que el gato se detecta utilizando un grid de 8×8.

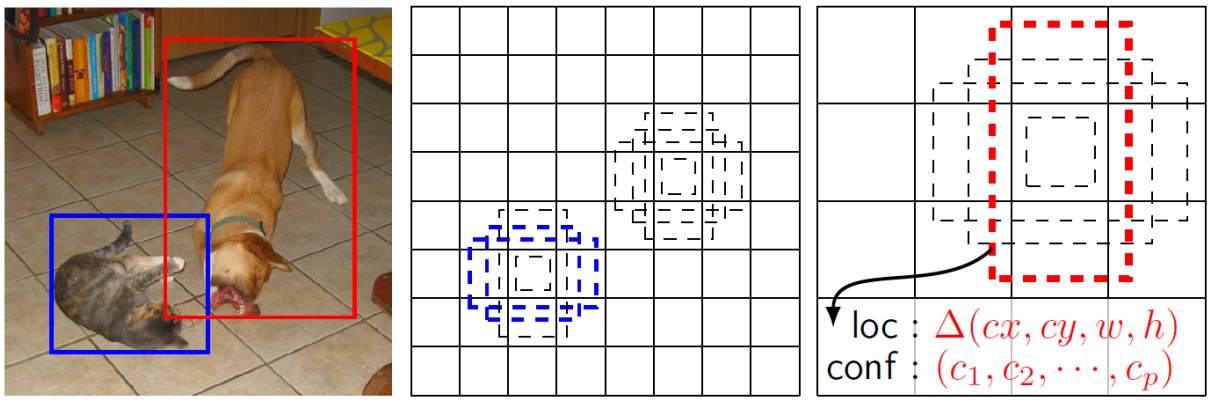


Figura 3.48.- Predicción de SSD en dos escalas

- **Función de pérdida**: en SSD la función de pérdida se divide en dos partes: la perdida de localización y de confianza:
  - **Pérdida de localización**: se calcula como la diferencia entre el GroundTruth y las BoundingBoxes predichas. En SSD solo se penalizan los fallos de los emparejamientos positivos (FP) [36], es decir, si se produce una predicción se espera que sea correcta (TP). Los emparejamientos negativos (que exista GroundTruth pero no se produzca detección) se ignoran (FN). En resumen, se

pretende maximizar el número de TPs, penalizar los FPs y se ignoran los FNs (**ver sección 3.3**).

- **Pérdida de localización**: para cada emparejamiento positivo (TP, FP) se calcula la probabilidad (confianza) de pertenecer a cada una de las clases. La pérdida se calcula en función de las confianzas obtenidas. Es decir, si existe un objeto que es un perro y la confianza obtenida para la clase perro es muy alta no se penalizará (TP), pero sin embargo, si la confianza obtenida para la clase perro es muy baja sí se penalizará (FP).

- **Hard negative mining**: durante el entrenamiento, como es evidente, se producen muchos más predicciones incorrectas que correctas. Este hecho hace que el coste temporal del entrenamiento sea demasiado alto. Para resolver este problema, no se pueden eliminar todas las predicciones negativas, ya que el modelo necesita conocer cómo es una predicción incorrecta, pero si se pueden eliminar una parte. Para seleccionar cuáles, se ordenan de mayor a menor por su pérdida de localización. Una vez ordenadas se seleccionan las que más pérdida tengan (y se eliminan las que menos). De esta forma se establece una relación entre las predicciones incorrectas y correctas de 3 a 1, reduciendo el coste temporal del entrenamiento.
- **Aumento de datos**: anteriormente ya se ha analizado la importancia de utilizar aumento de datos para mejorar la robustez del modelo (ver sección 3.1). Como es de esperar, SSD también utiliza aumento de datos.
- **Non Maximal Suppression**: al igual que en YOLO (ver sección 3.2.1.2) SSD utiliza NMS para eliminar las BoundingBoxes duplicadas.

3.2.6.2.- Arquitectura de la red

En la Figura 3.49 [35] se muestra una comparativa entre la arquitectura de YOLOv1 y SSD. Se realiza esta comparación debido a que estas redes se desarrollaron en el mismo año.

Tal y como se puede ver en la Figura 3.49, y como ya se ha analizado, YOLOv1 utiliza capas Fully Connected mientras que SSD no. Otra conclusión interesante que se puede apreciar en la Figura 3.49 es que SSD consigue un mAP (ver sección 3.3) de 0.743, mientras que YOLOv1 de 0.634 sobre el conjunto de datos Pascal VOC 2007 [17]. Esta diferencia de mAP se debe a que SSD utiliza, entre otras cosas, AnchorBoxes y predicciones en múltiples escalas, y estas innovaciones no llegaron a YOLO hasta versiones más recientes.

Por último, cabe destacar que aunque en la Figura 3.49 se muestra como SSD realiza predicciones a 59FPS y YOLOv1 a 45FPS, el tamaño de las imágenes de entrada no es el mismo. En SSD se utilizan imágenes de 300×300, mientras que en YOLOv1 de 448×448. Para SSD también hay una implementación de 512×512, donde en este caso las predicciones se realizan a 22FPS [35].

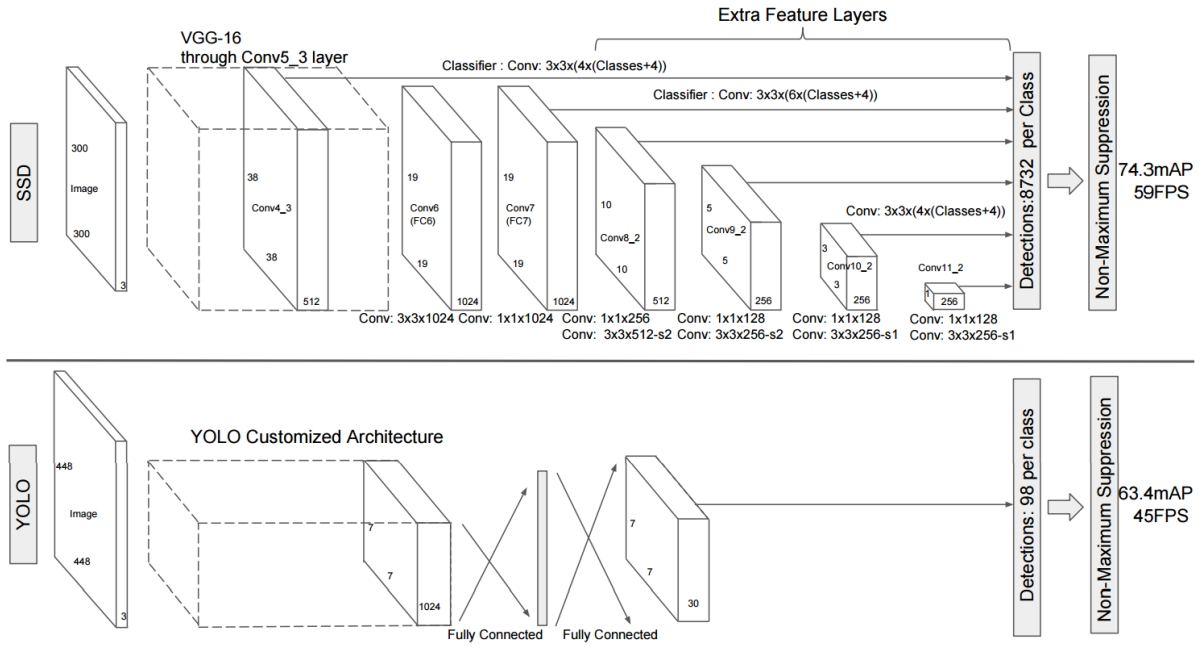


Figura 3.49.- Comparativa de la arquitectura de SSD y YOLOv1

3.3.- Métricas para la evaluación de resultados: Precision, Recall y AP

Para poder analizar correctamente los resultados obtenidos, es necesario definir y utilizar una serie de métricas. En el caso de la detección de objetos la métrica utilizada es el AP o Average Precision [10]. Para poder entender el AP, primero es necesario explicar el contexto de la detección de objetos y definir otras dos métricas: Precision y Recall.

Antes de continuar con las definiciones de Precision y Recall, es necesario tener claro una serie de conceptos [37]:

- **True Positive o TP:** se produce una detección correcta.
- **False Positive o FP:** se produce una detección incorrecta.
- **False Negative o FN:** no se produce una detección y sí se tendría que haber producido.
- **True Negative o TN:** no se produce una detección y realmente no la hay. Los TN no se aplican en la detección de objetos, debido a que en una imagen todo lo que no son objetos (a detectar) sería un TN, por lo que habría infinitos TN.

La forma de determinar si una detección es un True Positive o un False Positive es estableciendo un umbral de IoU, típicamente 0.5. Si una detección se solapa (IoU) con un GroundTruth más de un cierto umbral dicha detección se considera correcta, es decir TP. Por el contrario, si se solapa por debajo de dicho umbral se considera como una detección incorrecta (FP).

A continuación en las Figuras 3.50, 3.51 y 3.52 se pueden observar unos ejemplos para entender mejor los conceptos de TP, FP y FN. La caja verde representa el GroundTruth, mientras que la caja naranja representa predicción o detección realizada.

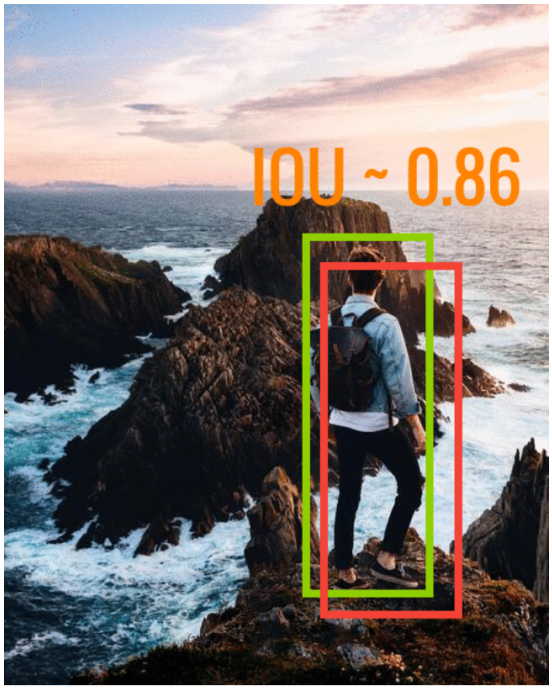


Figura 3.50.- Ejemplo de TP

Figura 3.51.- Ejemplo de FP

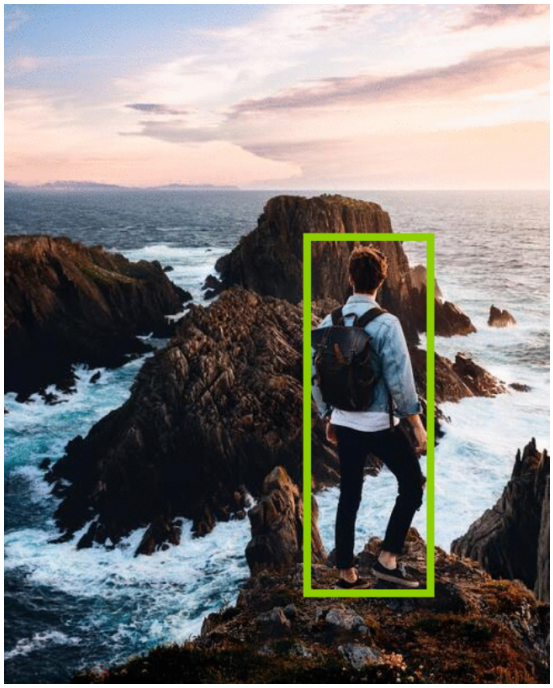


Figura 3.52.- Ejemplo de FN

El algoritmo utilizado para contabilizar los TPs, los FPs y los FNs es el siguiente [38].

Input: Bounding box predictions with confidence scores $\{(b_j, s_j)\}_{j=1}^{M}$ and ground truth boxes $\mathcal{B}$ on image I for a given object class.
Output: Binary results $\{z_j\}_{j=1}^{M}$ of whether or not prediction j is a true positive detection
Let $\mathcal{U} = \mathcal{B}$ be the set of unmatched objects;
Order $\{(b_j, s_j)\}_{j=1}^{M}$ in descending order of s_j;
for j=1 ...M **do**
  Let $\mathcal{C} = \{B_k \in \mathcal{U} \ : \ \text{IOU}(B_k, b_j) \geq \text{thr}(B_k)\}$;
  if $\mathcal{C} \neq \emptyset$ **then**
    Let $k^* = \arg\max_{\{k \ : \ B_k \in \mathcal{C}\}} \text{IOU}(B_k, b_j)$;
    Set $\mathcal{U} = \mathcal{U} \backslash B_{k*}$;
    Set $z_j = 1$ since true positive detection;
  else
    Set $z_j = 0$ since false positive detection;
  end
end

Para cada clase en cada imagen I_i, un algoritmo devuelve una serie de predicciones (b_{ij}, s_{ij}). Estas predicciones están compuestas por la localización o BoundingBox (b_{ij}) y por una confianza (s_{ij}). Estas predicciones se comparan con los datos reales o GroundTruth (B_{ij}). Para cada detección j en la imagen i el algoritmo devuelve $z_{ij} = 1$ en el caso de que la detección predicha coincida con la real o GroundTruth (TP). En el caso de que no coincidan $z_{ij} = 0$ (FP). El criterio para establecer si las dos predicciones coinciden es el del IoU, estableciendo un umbral o threshold. Si el IoU obtenido es mayor o igual que este umbral, entonces la detección predicha coincide con la real, y en caso contrario, no coinciden. Los FNs son los elementos del GroundTruth que no han sido emparejados con una detección.

La definición de **Recall** es la fracción entre el número de objetos detectados por el algoritmo y el número total de objetos existentes en las imágenes, mientras que la definición de **Precision** es la fracción entre las detecciones correctas entre el número total de detecciones realizadas. Las definiciones matemáticas serían las siguientes:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{Todas\ las\ detecciones\ realizadas}$$

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{Todo\ el\ GroundTruth}$$

En otras palabras, la Precision indica cómo de fiables son las detecciones realizadas, mientras que la Recall indica el porcentaje de detecciones realizadas sobre el total de objetos (GroundTruth).

Una vez analizados estos conceptos se procede a comentar la métrica más utilizada en la literatura sobre la detección de objetos: Average Precision (AP) o Precisión Media. El AP consiste en computar la precisión media sobre los diferentes niveles de recall.

Para entender mejor esta métrica se va a utilizar un ejemplo ilustrativo [5]: se supone un conjunto de datos que solo contiene cinco manzanas. En la Tabla 3.3 se recogen todas las predicciones obtenidas para la clase manzanas de todas las imágenes. Estas predicciones se ordenan descendentemente de acuerdo con el nivel de confianza obtenido. La segunda columna indica si la predicción es correcta o no. En este ejemplo se considera una predicción como correcta si el $IoU \geq 0.5$.

#	¿Correcta?	Precision	Recall
1 (Max Conf.)	Sí	1.00 (1/1)	0.20 (1/5)
2	Sí	1.00 (2/2)	0.40 (2/5)
3	No	0.67 (2/3)	0.40 (2/5)
4	No	0.50 (2/4)	0.40 (2/5)
5	No	0.40 (2/5)	0.40 (2/5)
6	Sí	0.50 (3/6)	0.60 (3/5)
7	Sí	0.57 (4/7)	0.80 (4/5)
8	No	0.50 (4/8)	0.80 (4/5)
9	No	0.44 (4/9)	0.80 (4/5)
10 (Min Conf.)	Sí	0.50 (5/10)	1.00 (5/5)

Tabla 3.3.- Ejemplo para ilustrar el cálculo del AP.

Vamos a fijarnos en la fila número 3 del ranking (Tabla 3.3) para calcular la Precision y el Recall.

- Precisión = $2/3 = 0.67$. La precisión es la proporción de TP (en la primera fila hay TP, en la segunda fila hay TP y en la tercera fila no hay TP).
- Recall = $2/5 = 0.4$. El recall es la proporción de TP del total de objetos (GroundTruth).

El Recall crece a medida que bajamos en la Tabla 3.3, mientras que la Precision sigue un patrón de zigzag, es decir, se decrementa con los FP y se incrementa con los TP. Este comportamiento se puede apreciar en la Figura 3.53.

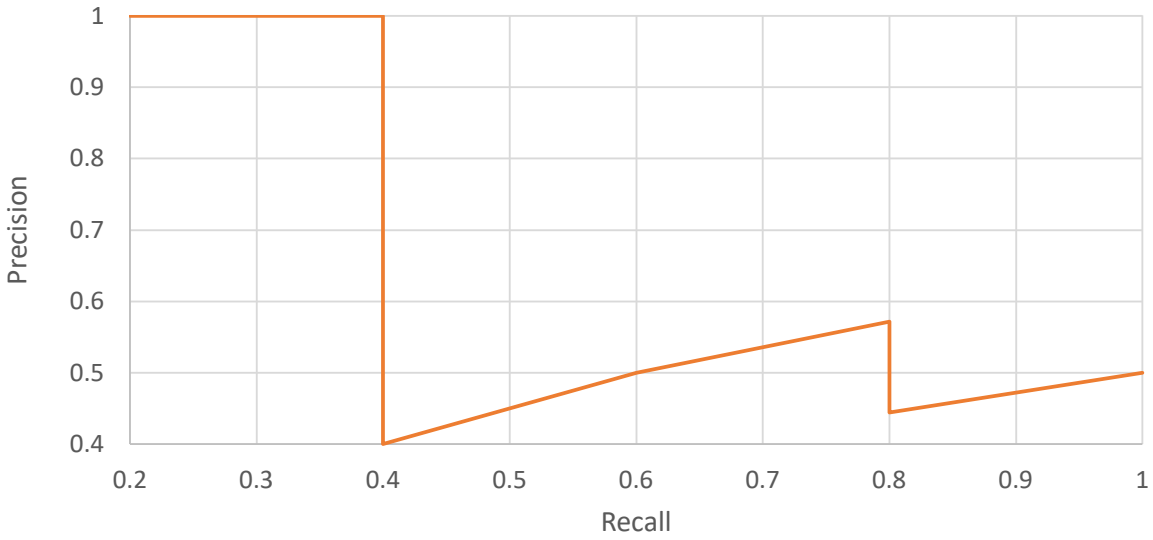


Figura 3.53.- Ejemplo de representar la curva Precision-Recall

Una vez explicados estos conceptos, la definición de Average Precisión (AP) es el área que se encuentra debajo de la curva Precision-Recall. Matemáticamente se puede representar como:

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

Antes de calcular el AP, es muy común suavizar el zigzag (ver Figura 3.54).

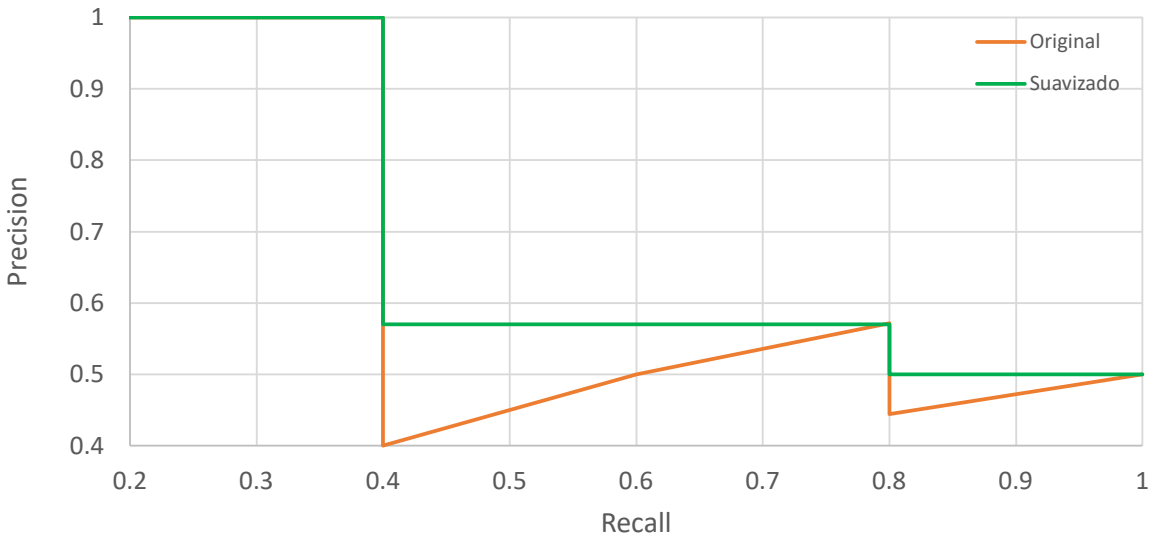


Figura 3.54.- Ejemplo de suavizar la curva Precision-Recall

Gracias a esta modificación, el AP calculado será menos susceptible a pequeñas variaciones. En el conjunto de datos PASCAL VOC [10] se considera una predicción

como correcta si el $IoU \geq 0.5$. Además, si se producen múltiples detecciones sobre el mismo objeto, la primera se cuenta como positivo y el resto como negativos. Volviendo al ejemplo, primero se divide los valores del Recall en once puntos desde 0 a 1 (Figura 3.55), es decir, 0, 0.1, 0.2, . . . , 0.9 y 1.0.

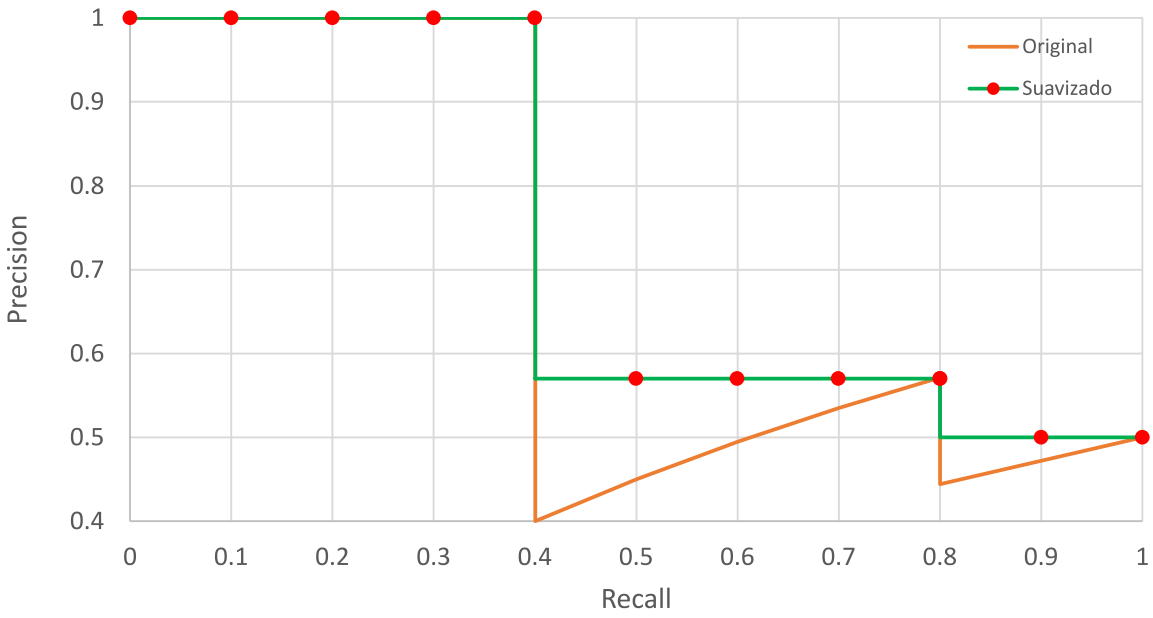


Figura 3.55.- División del Recall en 11 puntos

A continuación, se calcula la precisión media (AP) para cada uno de estos once valores del recall. Matemáticamente [39]: $AP = \frac{1}{11} \sum_{Recall_i} Precision(Recall_i)$.

En el ejemplo (Figura 3.55), el $AP = \frac{5 \times 1.0 + 4 \times 0.57 + 2 \times 0.5}{11} = 0.7527$.

Esta métrica es la utilizada para evaluar y comparar todos los modelos de detección de objetos.

El AP se calcula para cada una de las clases del conjunto de datos, por lo que es muy común utilizar el mAP o Mean Average Precision, calculado como el promedio de los APs de cada una de las clases.

4. Conjuntos de datos utilizados

4.1.- DIOR

4.1.1.- Introducción

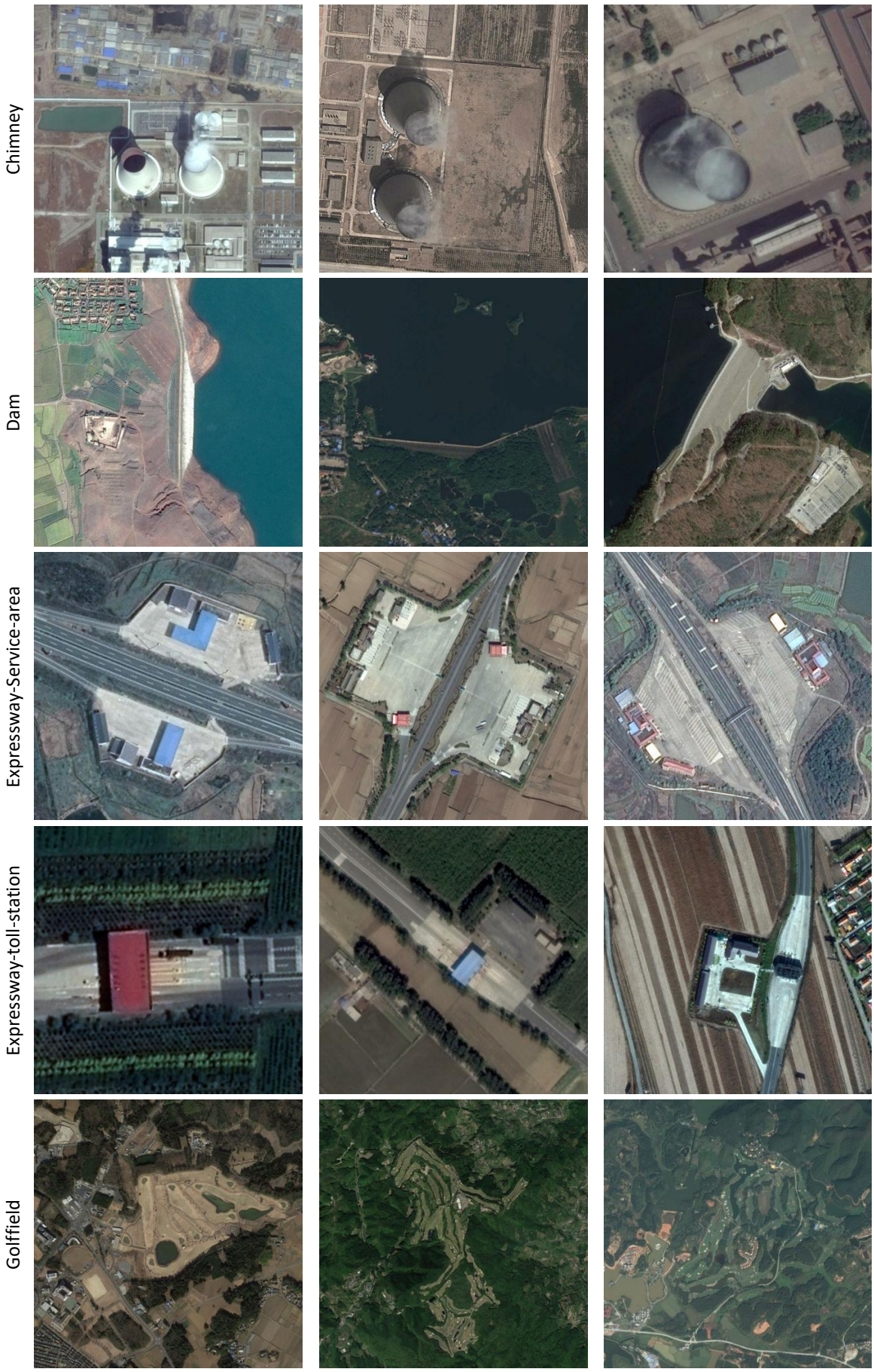
El conjunto de datos DIOR (Detection in Optical Remote Sensing Images) fue creado en 2019 con el objetivo de poner a prueba a los detectores de objetos actuales [40]. Este conjunto de datos cuenta con 23 463 imágenes y 192 472 objetos. Estos objetos son de distinto tipo, ya que el conjunto cuenta con 20 clases diferentes. Además a diferencia de otros conjuntos de datos, cuenta con un gran número de objetos pequeños, difíciles de detectar, así como una gran variabilidad de tamaños de objetos dentro de una misma clase.

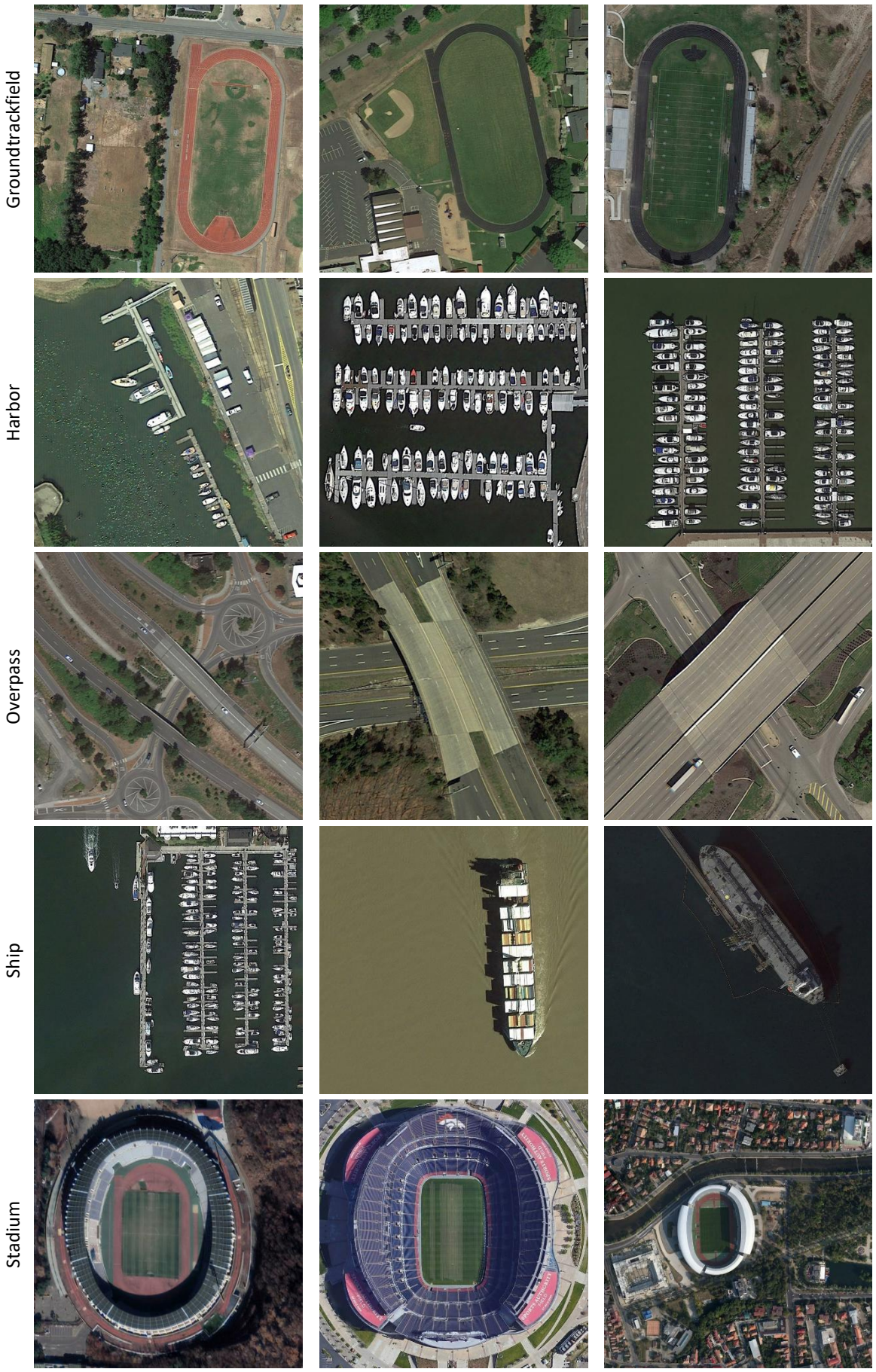
Teniendo en cuenta los conjuntos de datos con **imágenes aéreas**, DIOR es uno de los más grandes y más diversos. Algunos ejemplos de este conjunto se pueden apreciar en la Figura 4.1. A continuación se listan todas las clases.

1. Airplane → Avión
2. Airport → Aeropuerto
3. Baseballfield → Campo de béisbol
4. Basketballcourt → Cancha de baloncesto
5. Bridge → Puente
6. Chimney → Chimenea
7. Dam → Presa
8. Expressway-Service-area → Área de servicio
9. Expressway-toll-station → Estación de peaje
10. Golffield → Campo de golf
11. Groundtrackfield → Pista de atletismo
12. Harbor → Puerto
13. Overpass → Cruce
14. Ship → Barco
15. Stadium → Estadio
16. Storagetank → Tanque de almacenamiento
17. Tenniscourt → Pista de tenis
18. Trainstation → Estación de tren
19. Vehicle → Vehículo
20. Windmill → Molino de viento

El tamaño de las imágenes es de 800 x 800 píxeles y la resolución espacial varía entre 0.5 m y 30 m. Al igual que la mayoría de los conjuntos de datos aéreos las imágenes se toman desde Google Earth, siendo el conjunto más diverso (número de clases), con mayor cantidad de imágenes y con mayor cantidad de objetos anotados de los conjuntos aéreos.


Airplane
Airport
Baseballfield
Basketballcourt
Bridge


Chimney
Dam
Expressway-Service-area
Expressway-toll-station
Golffield


Groundtrackfield
Harbor
Overpass
Ship
Stadium

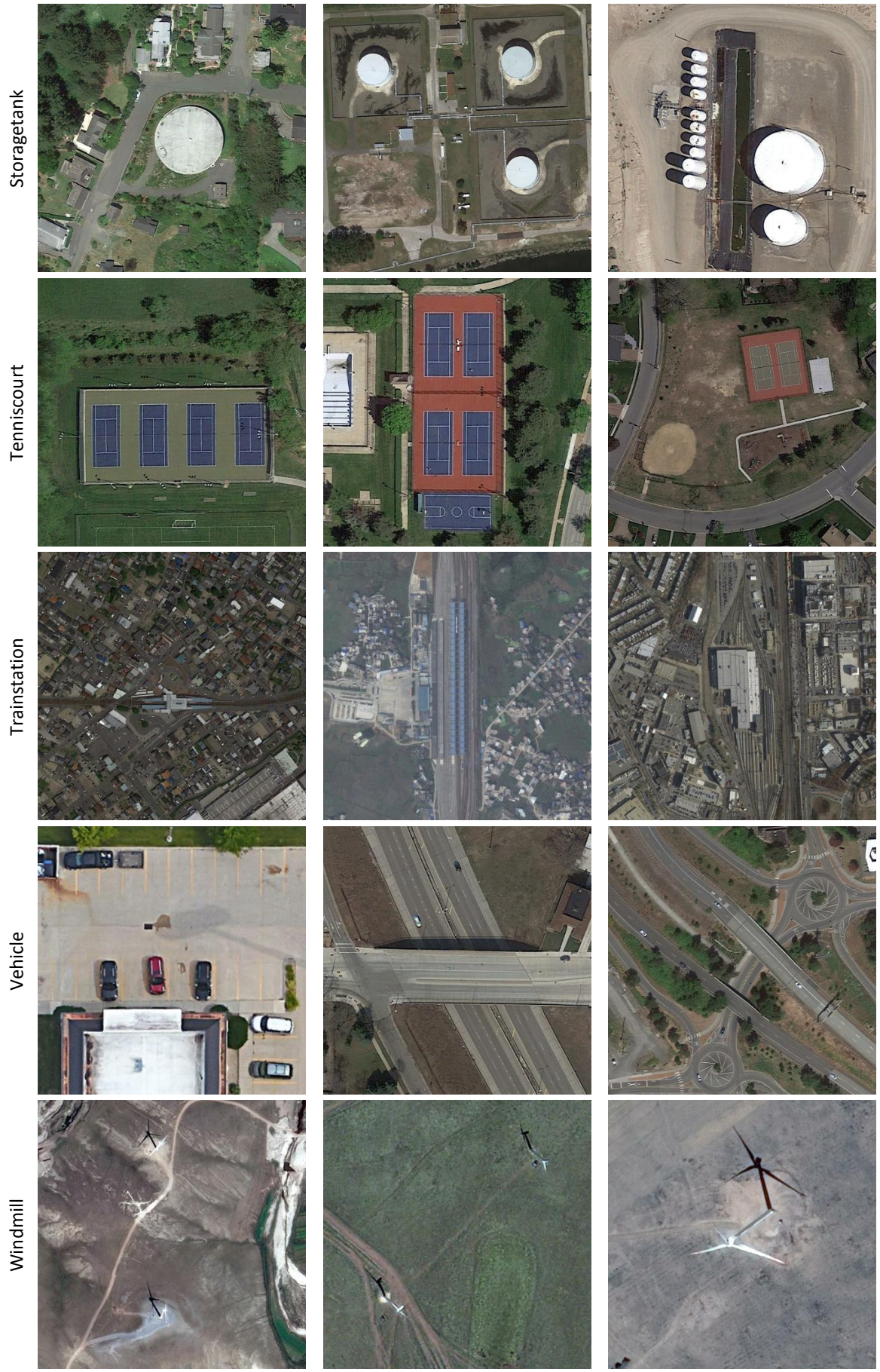


Figura 4.1.- Muestra de cada una de las clases del conjunto DIOR

4.1.2.- Análisis del conjunto de datos

Este conjunto cuenta con un total de 192 472 objetos. En la Figura 4.2 se puede apreciar la cantidad de objetos para cada una de las diferentes clases utilizadas en el conjunto. Las clases Barco (Ship) y Vehículo (Vehicle) tienen muchas más instancias que el resto de clases, ya que representan el 53.39 % del total de objetos anotados.

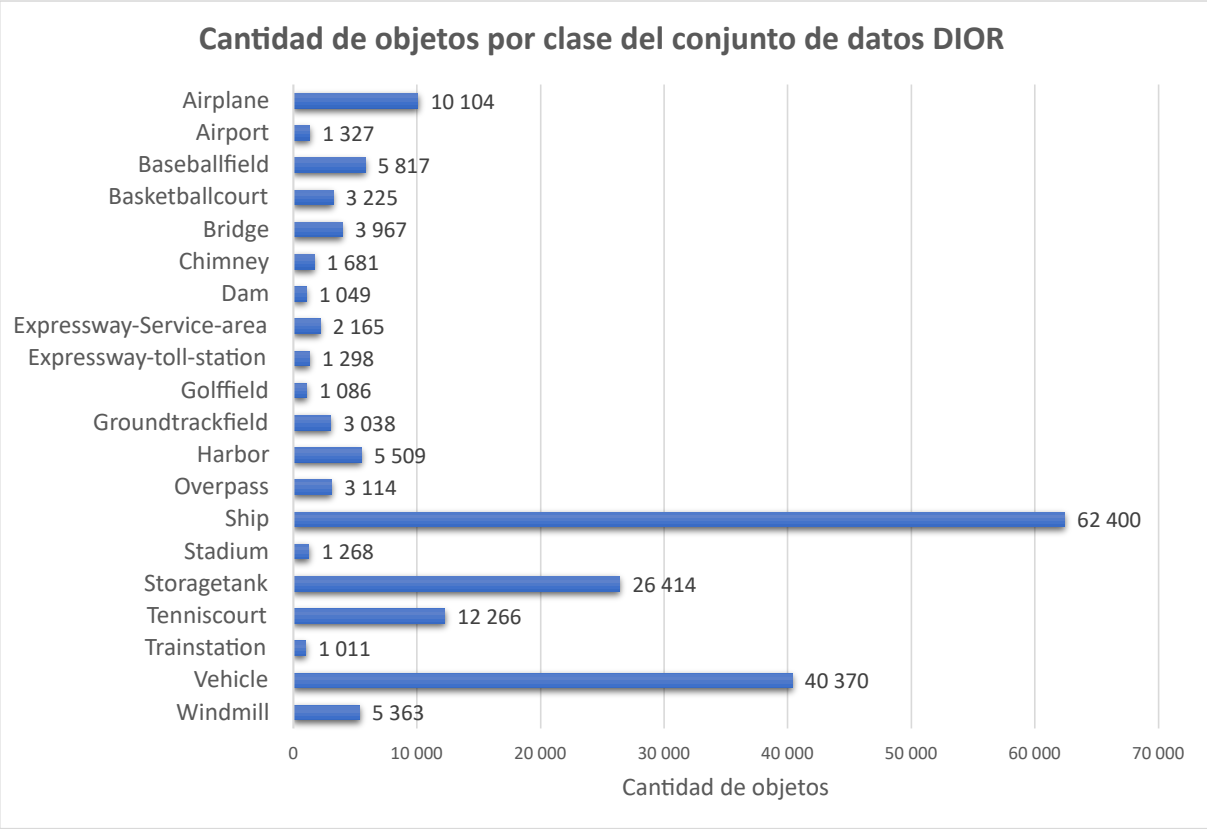


Figura 4.2.- Cantidad de objetos por clase del conjunto de datos DIOR

Otro aspecto importante a tener en cuenta en el análisis del conjunto de datos es la variabilidad de cada clase, entendiendo variabilidad como la diferencia de tamaño entre los objetos de una misma clase. En las Figuras 4.3 y 4.4 se puede apreciar la distribución de la anchura y la altura de cada clase, respectivamente. Por ejemplo, en la Figura 4.3 se puede apreciar como la anchura mínima de la clase Estadio (Stadium) es de 10 píxeles, mientras que la anchura máxima es de 783. Esta diferencia es una muestra de la gran variabilidad de los objetos intraclase. En ese mismo caso, la mediana de la anchura de la clase Estadio (Stadium) es de 233 píxeles. Es importante destacar que tanto en la Figura 4.3 como en la Figura 4.4 se representan en rojo los valores atípicos (outliers) de cada clase. Gracias a esta representación se puede observar, que por ejemplo, en la clase Barco (Ship) hay una gran cantidad de puntos considerados como atípicos. Esto se debe a lo ya mencionado, la gran variabilidad de tamaños de objetos intraclase.

Este análisis sirve para destacar la gran diversidad de los datos dentro de una misma clase. Este hecho, junto con la gran cantidad de clases utilizadas (20 clases) hace que **este**

conjunto de datos sea muy apropiado para determinar los límites de un detector de objetos.

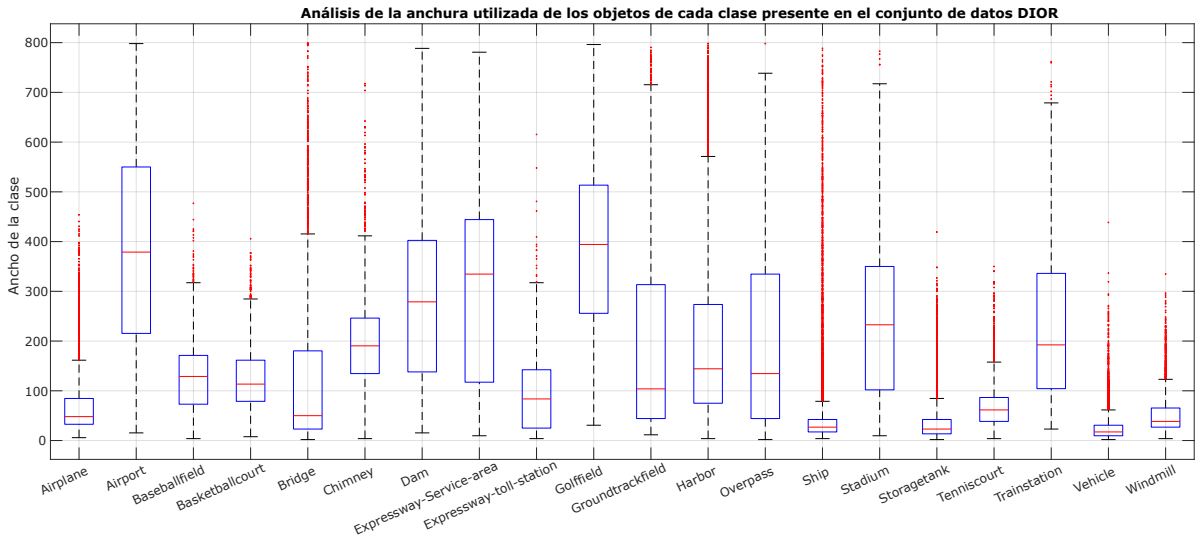


Figura 4.3.- Distribución de la anchura por clases del conjunto de datos DIOR

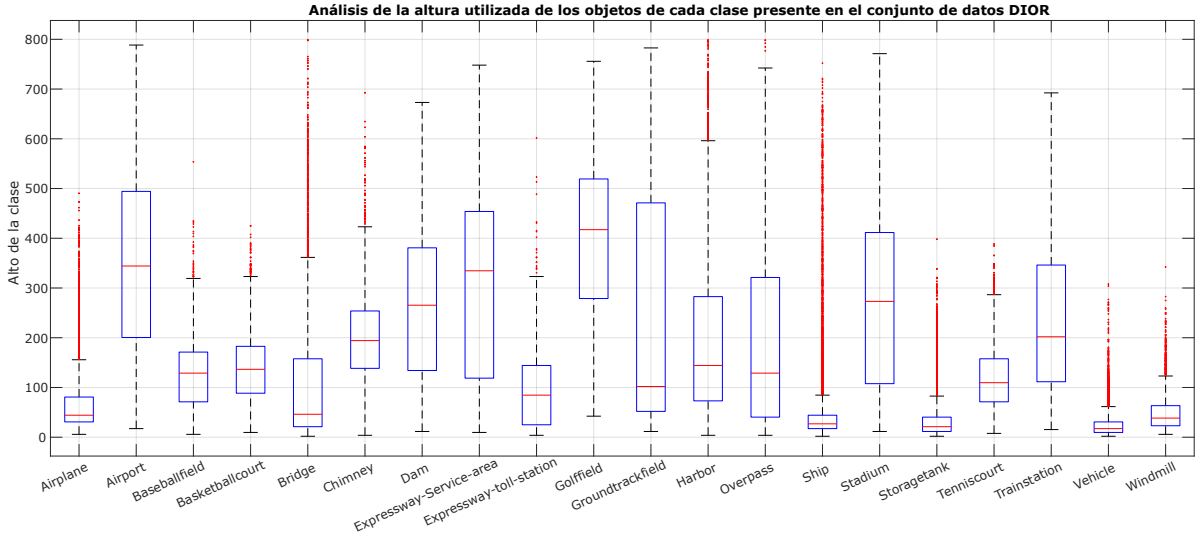


Figura 4.4.- Distribución de la altura por clases del conjunto de datos DIOR

4.1.3.- División del conjunto de datos

Debido a la necesidad de dividir el conjunto de datos en un conjunto de entrenamiento y de test, se divide el conjunto en cuatro subconjuntos, de esta manera se podrá realizar un CrossTesting (ver sección 3.1) durante las pruebas experimentales. En las Tablas 4.1, 4.2, 4.3 y 4.4 se puede apreciar, para cada uno de los cuatros subconjuntos en los que se ha dividido el conjunto DIOR, la cantidad de imágenes, de objetos de cada clase, así como el porcentaje que representan del total de datos. Como la división del conjunto original en los subconjuntos ha sido aleatoria, la cantidad de imágenes y objetos es similar en cada uno de los subconjuntos. Cabe destacar que el total de la columna Nº imágenes, no es la suma de sus filas. Esto se debe a que en una misma imagen puede haber varios objetos de distintas clases, por lo que si en una misma imagen está la clase Puerto y la clase Barco, se suma uno a cada clase, pero realmente solo hay una imagen.

Subconjunto 1								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 1	**Test 1**	**Train 1**	**Test 1**	**Train 1**	**Test 1**	**Train 1**	**Test 1**
airplane	1 020	367	3.70	3.96	7 358	2 746	5.10	5.71
airport	964	346	3.50	3.74	974	353	0.67	0.73
baseballfield	1 832	608	6.65	6.57	4 327	1 490	3.00	3.10
basketballcourt	1 040	329	3.78	3.55	2 474	751	1.71	1.56
bridge	1 630	546	5.92	5.90	2 908	1 059	2.01	2.20
chimney	637	217	2.31	2.34	1 244	437	0.86	0.91
dam	720	266	2.61	2.87	766	283	0.53	0.59
expressway-Service-area	856	269	3.11	2.91	1 641	524	1.14	1.09
expressway-toll-station	901	317	3.27	3.42	954	344	0.66	0.72
golffield	700	246	2.54	2.66	812	274	0.56	0.57
groundtrackfield	1 751	561	6.36	6.06	2 320	718	1.61	1.49
harbor	1 064	410	3.86	4.43	3 945	1 564	2.73	3.25
overpass	1 545	474	5.61	5.12	2 380	734	1.65	1.53
ship	2 008	694	7.29	7.50	46 790	15 610	32.40	32.48
stadium	899	301	3.26	3.25	951	317	0.66	0.66
storagetank	1 253	415	4.55	4.48	20 286	6 128	14.05	12.75
tenniscourt	1 958	624	7.11	6.74	9 376	2 890	6.49	6.01
trainstation	743	251	2.70	2.71	757	254	0.52	0.53
vehicle	4 800	1 621	17.43	17.51	30 121	10 249	20.86	21.32
windmill	1 220	396	4.43	428.00	4 022	1 341	2.79	2.79
Total	**17 597**	**5 866**	**100**	**100**	**144 406**	**48 066**	**100**	**100**

Tabla 4.1.- Subconjunto 1 - DIOR

Subconjunto 2								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 2	**Test 2**	**Train 2**	**Test 2**	**Train 2**	**Test 2**	**Train 2**	**Test 2**
airplane	1 048	339	3.80	3.67	7 879	2 225	5.47	4.61
airport	972	338	3.53	3.66	983	344	0.68	0.71
baseballfield	1 810	630	6.56	6.83	4 326	1 491	3.00	3.09
basketballcourt	1 019	350	3.70	3.79	2 419	806	1.68	1.67
bridge	1 634	542	5.93	5.88	3 006	961	2.09	1.99
chimney	660	194	2.39	2.10	1 280	401	0.89	0.83
dam	762	224	2.76	2.43	811	238	0.56	0.49
expressway-Service-area	842	283	3.05	3.07	1 630	535	1.13	1.11
expressway-toll-station	922	296	3.34	3.21	980	318	0.68	0.66
golffield	720	226	2.61	2.45	816	270	0.57	0.56
groundtrackfield	1 733	579	6.28	6.28	2 280	758	1.58	1.57
harbor	1 110	364	4.03	3.95	4 188	1 321	2.90	2.73
overpass	1 512	507	5.48	5.50	2 357	757	1.63	1.57
ship	2 040	662	7.40	7.18	47 420	14 980	32.89	31.01
stadium	917	283	3.33	3.07	963	305	0.67	0.63
storagetank	1 237	431	4.49	4.67	18 985	7 429	13.17	15.38
tenniscourt	1 899	683	6.89	7.40	8 989	3 277	6.24	6.78
trainstation	736	258	2.67	2.80	750	261	0.52	0.54
vehicle	4 812	1 609	17.45	17.44	30 172	10 198	20.93	21.11
windmill	1 189	427	4.31	4.63	3 937	1 426	2.73	2.95
Total	**17 597**	**5 866**	**100**	**100**	**144 171**	**48 301**	**100**	**100**

Tabla 4.2.- Subconjunto 2 - DIOR

Subconjunto 3								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 3	**Test 3**	**Train 3**	**Test 3**	**Train 3**	**Test 3**	**Train 3**	**Test 3**
airplane	1 041	346	3.77	3.77	7 613	2 491	5.30	5.10
airport	995	315	3.60	3.44	1 011	316	0.70	0.65
baseballfield	1 846	594	6.68	6.48	4 392	1 425	3.06	2.92
basketballcourt	1 028	341	3.72	3.72	2 398	827	1.67	1.69
bridge	1 614	562	5.84	6.13	2 988	979	2.08	2.00
chimney	647	207	2.34	2.26	1 264	417	0.88	0.85
dam	751	235	2.72	2.56	788	261	0.55	0.53
expressway-Service-area	820	305	2.97	3.33	1 579	586	1.10	1.20
expressway-toll-station	924	294	3.34	3.21	985	313	0.69	0.64
golffield	722	224	2.61	2.44	832	254	0.58	0.52
groundtrackfield	1 745	567	6.32	6.18	2 273	765	1.58	1.57
harbor	1 118	356	4.05	3.88	4 183	1 326	2.91	2.71
overpass	1 506	513	5.45	5.59	2 296	818	1.60	1.67
ship	2 013	689	7.29	7.51	45 714	16 686	31.83	34.15
stadium	889	311	3.22	3.39	942	326	0.66	0.67
storagetank	1 265	403	4.58	4.40	19 731	6 683	13.74	13.68
tenniscourt	1 945	637	7.04	6.95	9 339	2 927	6.50	5.99
trainstation	746	248	2.70	2.70	757	254	0.53	0.52
vehicle	4 800	1 621	17.37	17.68	30 548	9 822	21.27	20.10
windmill	1 215	401	4.40	4.37	3 973	1 390	2.77	2.84
Total	**17 597**	**5 866**	**100**	**100**	**143 606**	**48 866**	**100**	**100**

Tabla 4.3.- Subconjunto 3 - DIOR

Subconjunto 4								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 4	**Test 4**	**Train 4**	**Test 4**	**Train 4**	**Test 4**	**Train 4**	**Test 4**
airplane	1 052	335	3.80	3.66	7 462	2 642	5.14	5.59
airport	999	311	3.61	3.40	1 013	314	0.70	0.66
baseballfield	1 832	608	6.63	6.65	4 406	1 411	3.03	2.99
basketballcourt	1 020	349	3.69	3.82	2 384	841	1.64	1.78
bridge	1 650	526	5.97	5.75	2 999	968	2.07	2.05
chimney	618	236	2.23	2.58	1 255	426	0.86	0.90
dam	725	261	2.62	2.85	782	267	0.54	0.57
expressway-Service-area	857	268	3.10	2.93	1 645	520	1.13	1.10
expressway-toll-station	907	311	3.28	3.40	975	323	0.67	0.68
golffield	696	250	2.52	2.73	798	288	0.55	0.61
groundtrackfield	1 707	605	6.17	6.61	2 241	797	1.54	1.69
harbor	1 130	344	4.09	3.76	4 211	1 298	2.90	2.75
overpass	1 494	525	5.40	5.74	2 309	805	1.59	1.70
ship	2 045	657	7.40	7.18	47 272	15 124	32.55	32.02
stadium	895	305	3.24	3.33	948	320	0.65	0.68
storagetank	1 249	419	4.52	4.58	20 240	6 174	13.94	13.07
tenniscourt	1 944	638	7.03	6.98	9 094	3 172	6.26	6.71
trainstation	757	237	2.74	2.59	769	242	0.53	0.51
vehicle	4 851	1 570	17.54	17.16	30 266	10 101	20.84	21.38
windmill	1 224	392	4.43	4.29	4 157	1 206	2.86	2.55
Total	**17 598**	**5 865**	**100**	**100**	**145 226**	**47 239**	**100**	**100**

Tabla 4.4.- Subconjunto 4 - DIOR

Como se puede apreciar en las Tablas 4.1, 4.2, 4.3 y 4.4 la cantidad de imágenes y de objetos por clase de cada subconjunto es equivalente.

4.2.- Standford Aerial Pedestrian Dataset

4.2.1.- Introducción

El conjunto de datos Standford Aerial Pedestrian Dataset (a lo largo del documento se referirá simplemente como conjunto de datos Standford) fue publicado en 2016 con el objetivo de detectar/analizar los diferentes patrones de movimiento de los humanos [41]. Este estudio busca diferenciar el comportamiento de los humanos bajo distintas condiciones: caminando, andando en bicicleta, montando en patinete, conduciendo un coche o conduciendo un autobús. Por este motivo, las clases que componen este conjunto de datos son las siguientes.

1. Biker → Ciclista
2. Bus → Autobús
3. Car → Coche
4. Cart → Cart
5. Pedestrian → Peatón
6. Skater → Patinador

Algunos ejemplos de este conjunto se pueden apreciar en la Figura 4.5. Lamentablemente, no se han podido incluir las clases Cart y Skater, debido a que la calidad de las imágenes en las que aparecen estas clases es muy baja.

Cabe destacar, que este conjunto de datos no está compuesto imágenes, sino por una serie de vídeos. Por este motivo, ha sido necesario crear un extractor de frames para obtener las correspondientes imágenes. Cada uno de los vídeos que componen el conjunto tiene una resolución diferente por lo que se han analizado dos alternativas: redimensionar las imágenes obtenidas a 416×416, reduciendo así el tamaño de los objetos, o aplicar recortes 416×416 sobre estas imágenes. Se ha optado por la segunda opción ya que la calidad de las imágenes no es alta, y reducirla dificultaría la tarea de detección.

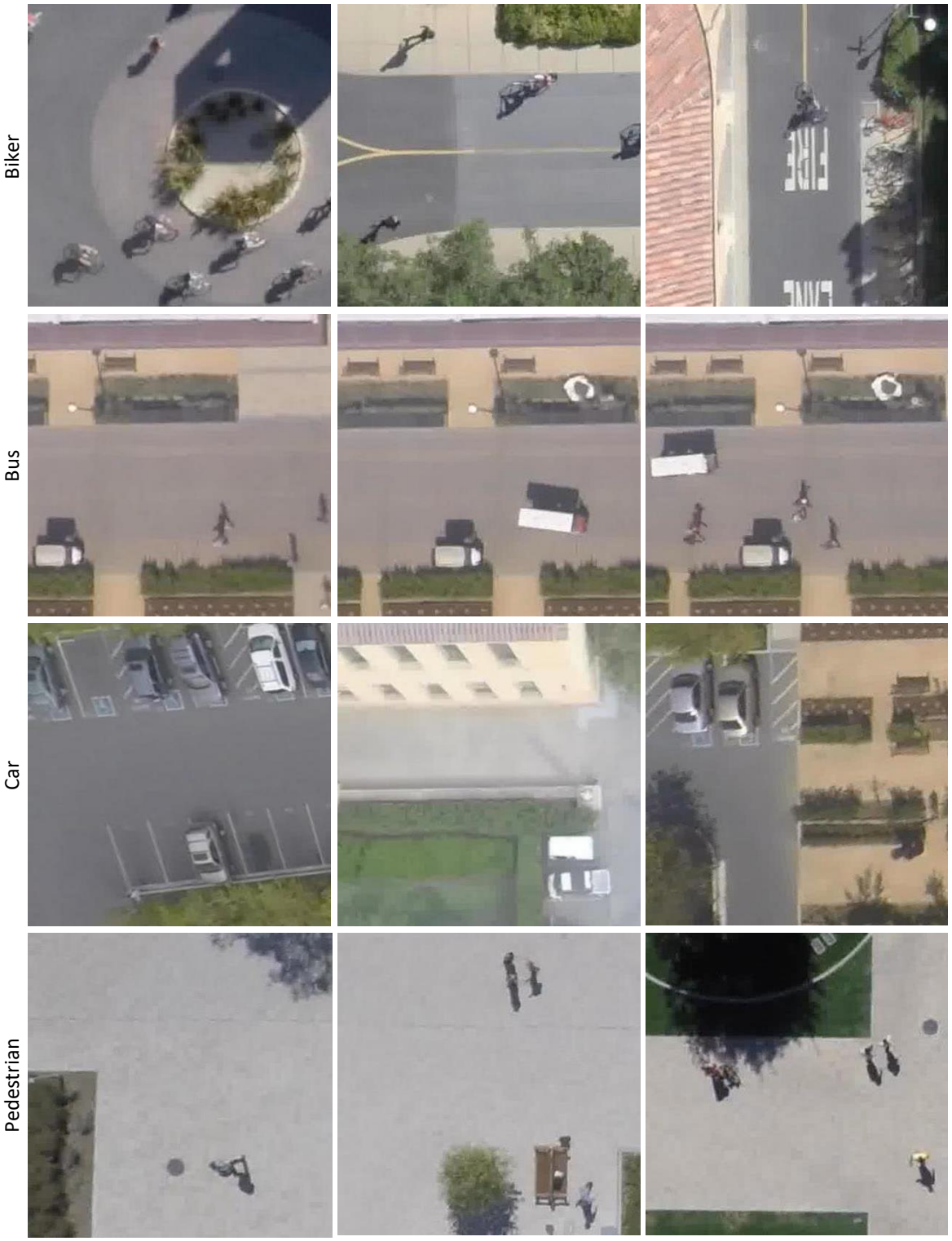


Figura 4.5.- Muestra de cada una de las clases del conjunto Standford

4.2.2.- Análisis del conjunto de datos

Este conjunto cuenta con un total de 41 067 objetos. En la Figura 4.6 se puede apreciar la cantidad de objetos para cada una de las diferentes clases utilizadas en el conjunto. Las clases Ciclista (Biker) y Peatón (Pedestrian) tienen muchas más instancias que el resto de clases, ya que representan el 94.43 % de los datos del total de objetos anotados.

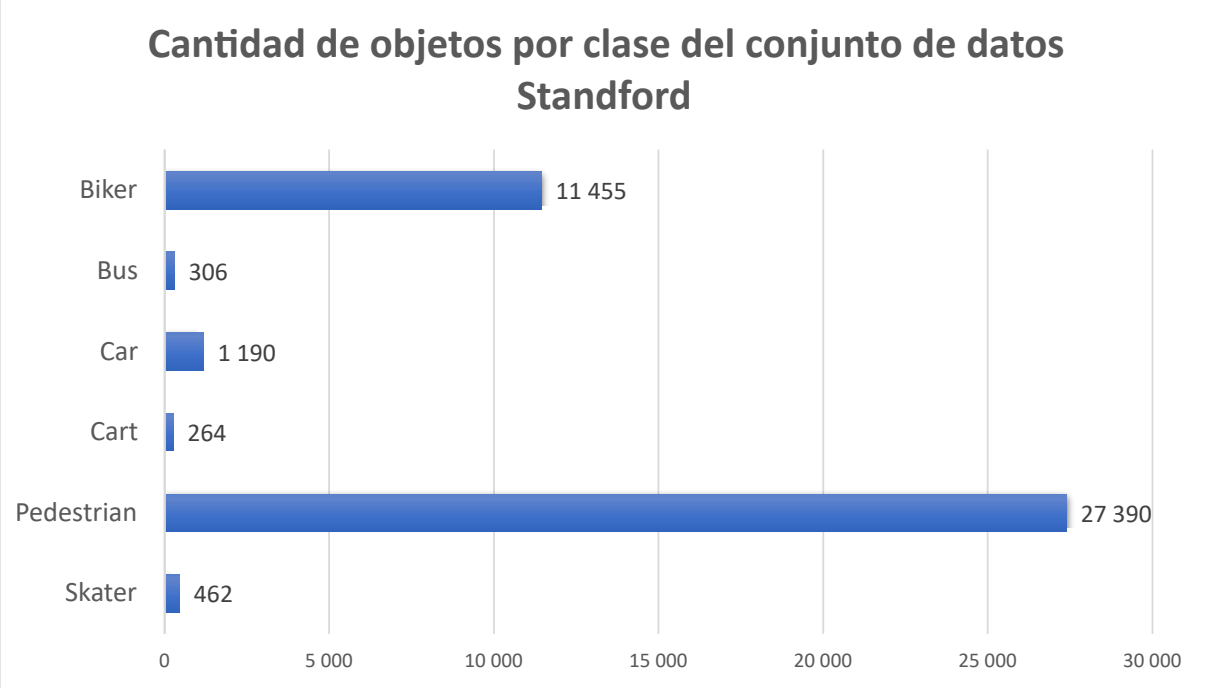


Figura 4.6.- Cantidad de objetos por clase del conjunto de datos Standford

En este caso, y a diferencia de lo que ocurre en DIOR, la variabilidad de los tamaños de los objetos no es significante, como se puede apreciar en las Figuras 4.7 y 4.8. En este caso, **la dificultad a la hora de realizar la detección de objetos viene determinada por la calidad de las imágenes**. Como se puede apreciar en la Figura 4.5 es realmente difícil distinguir algunas clases, como pueden ser Peatón o Ciclista.

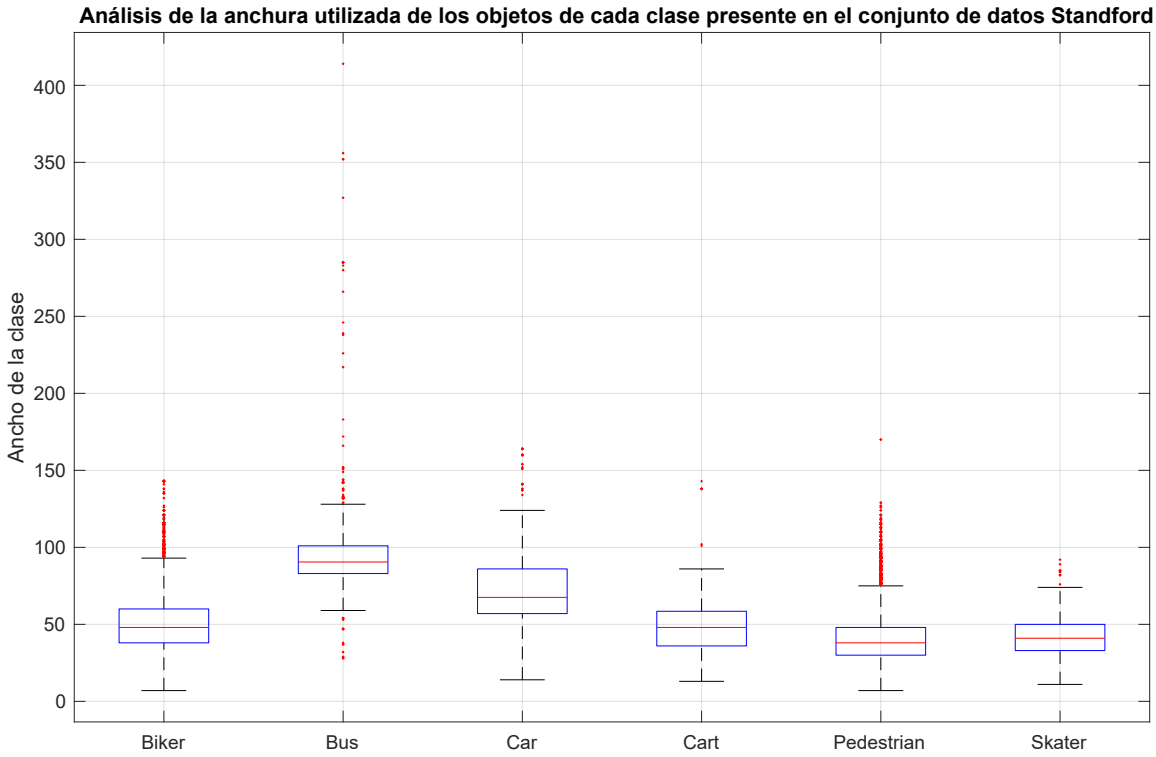


Figura 4.7.- Distribución de la anchura por clases del conjunto de datos Standford

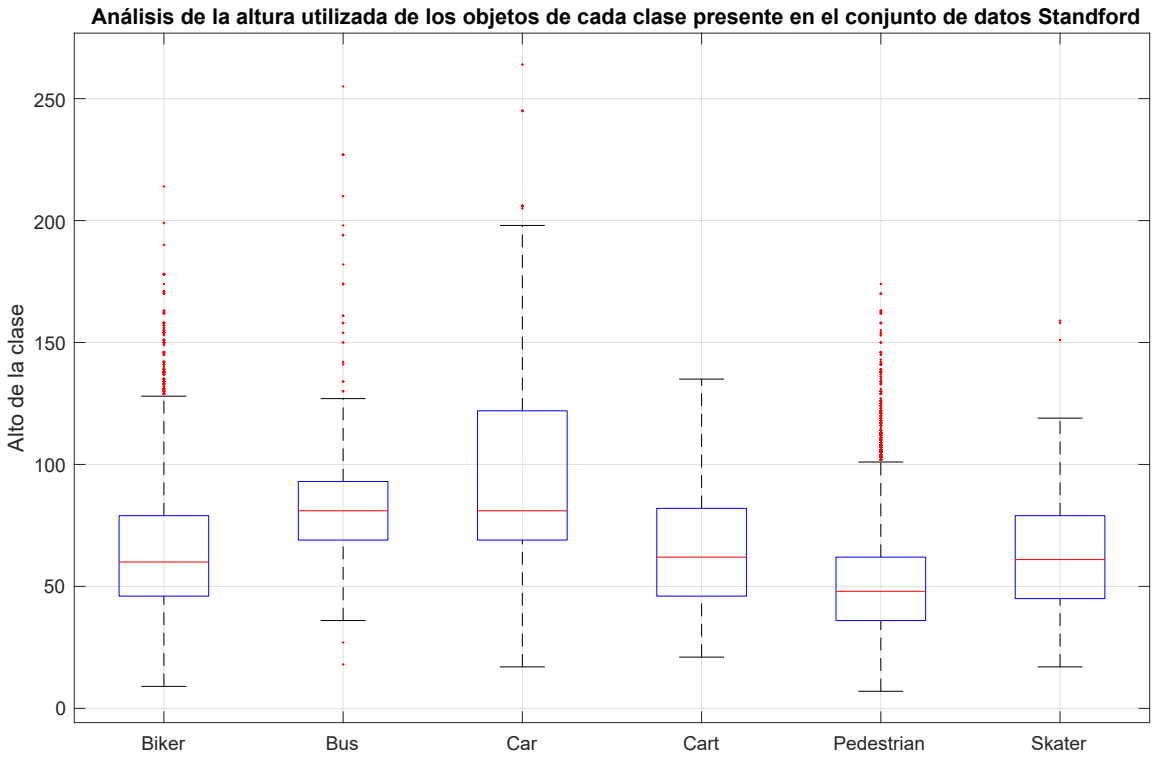


Figura 4.8.- Distribución de la altura por clases del conjunto de datos Standford

4.2.3.- División del conjunto de datos

Al igual que en DIOR (ver sección 4.1.3) se divide el conjunto en cuatro subconjuntos para poder realizar un CrossTesting. En este caso los resultados se pueden apreciar en las Tablas 4.5, 4.6, 4.7 y 4.8. Al igual que en DIOR, y por los mismos motivos, todos los subconjuntos presentan una cantidad de imágenes y de objetos similares.

Subconjunto 1								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 1	**Test 1**	**Train 1**	**Test 1**	**Train 1**	**Test 1**	**Train 1**	**Test 1**
Biker	5 008	1 624	31.17	30.73	8 631	2 824	27.97	27.68
Bus	229	73	1.43	1.38	231	75	0.75	0.74
Car	606	207	3.77	3.92	882	308	2.86	3.02
Cart	176	53	1.10	1.00	203	61	0.66	0.60
Pedestrian	9 739	3 212	60.61	60.79	20 579	6 811	66.68	66.75
Skater	311	115	1.94	2.18	337	125	1.09	1.23
Total	**12 388**	**4 130**	**100**	**100**	**30 863**	**10 204**	**100**	**100**

Tabla 4.5.- Subconjunto 1 - Standford

Subconjunto 2								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 2	**Test 2**	**Train 2**	**Test 2**	**Train 2**	**Test 2**	**Train 2**	**Test 2**
Biker	4 995	1 637	31.19	30.66	8 589	2 866	27.84	28.05
Bus	235	67	1.47	1.25	238	68	0.77	0.67
Car	606	207	3.78	3.88	897	293	2.91	2.87
Cart	163	66	1.02	1.24	189	75	0.61	0.73
Pedestrian	9 689	3 262	60.51	61.09	20 587	6 803	66.73	66.59
Skater	325	101	2.03	1.89	351	111	1.14	1.09
Total	**12 388**	**4 130**	**100**	**100**	**30 851**	**10 216**	**100**	**100**

Tabla 4.6.- Subconjunto 2 - Standford

Subconjunto 3								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 3	**Test 3**	**Train 3**	**Test 3**	**Train 3**	**Test 3**	**Train 3**	**Test 3**
Biker	4 943	1 689	30.92	31.48	8 551	2 904	27.88	27.92
Bus	227	75	1.42	1.40	230	76	0.75	0.73
Car	626	187	3.92	3.49	895	295	2.92	2.84
Cart	169	60	1.06	1.12	192	72	0.63	0.69
Pedestrian	9 697	3 254	60.65	60.65	20 438	6 952	66.65	66.84
Skater	326	100	2.04	1.86	360	102	1.17	0.98
Total	**12 388**	**4 130**	**100**	**100**	**30 666**	**10 401**	**100**	**100**

Tabla 4.7.- Subconjunto 3 - Standford

Subconjunto 4								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 4	**Test 4**	**Train 4**	**Test 4**	**Train 4**	**Test 4**	**Train 4**	**Test 4**
Biker	4 943	1 682	30.92	31.36	8 551	2 861	27.88	27.92
Bus	227	87	1.42	1.62	230	87	0.75	0.85
Car	626	212	3.92	3.95	895	294	2.92	2.87
Cart	169	50	1.06	0.93	192	56	0.63	0.55
Pedestrian	9 697	3 223	60.65	60.09	20 438	6 824	66.65	66.60
Skater	326	110	2.04	2.05	360	124	1.17	1.21
Total	**12 390**	**4 128**	**100**	**100**	**30 666**	**10 246**	**100**	**100**

Tabla 4.8.- Subconjunto 4 - Standford

4.3.- ADAAR

4.3.1.- Introducción

El conjunto de datos ADAAR (Aerial images Dataset for Agricultural Activity Recognition) ha sido facilitado por la empresa Seresco [1]. Este conjunto de datos es el conjunto que se va a utilizar en el servicio final, ya que es el más similar al entorno real. Este conjunto de datos cuenta con 1 838 imágenes y 10 027 objetos. Estos objetos son de distinto tipo pero todos están relacionados con la actividad agraria. A diferencia del resto de conjuntos de datos, **cuenta con pocas imágenes, lo que dificulta el entrenamiento**.

A continuación se listan todas las clases.

1. Abono
2. Animal
3. BolaSilo
4. Comedero
5. FosaPurín
6. Silo

Algunos ejemplos de este conjunto de datos se pueden apreciar en la Figura 4.9. En esta muestra se pueden apreciar todas las clases del conjunto.

Figura 4.9.- Muestra de cada una de las clases del conjunto ADAAR

4.3.2.- Obtención de los datos

A diferencia de los conjuntos DIOR [40] y Standford [41], el conjunto de datos ADAAR ha sido proporcionado por la propia empresa, debido a que es el conjunto más similar con un entorno real, y de esta forma se puede cumplir el objetivo final del proyecto: detectar actividad agraria mediante imágenes aéreas.

La empresa ha proporcionado 55 imágenes de grandes dimensiones (aproximadamente de $10\,000 \times 10\,000$). Como estas imágenes son demasiado grandes para realizar cualquier entrenamiento, se ha procedido a realizar recortes sobre las mismas, con el objetivo de obtener muchas imágenes de 416×416.

Para realizar estos recortes se creado un grid de 416×416 sobre cada imagen, tal y como se puede ver en Figura 4.10. En el caso de que no se pueda hacer un grid exacto se realiza un pequeño solape entre las celdas del grid (ver Figura 4.11).

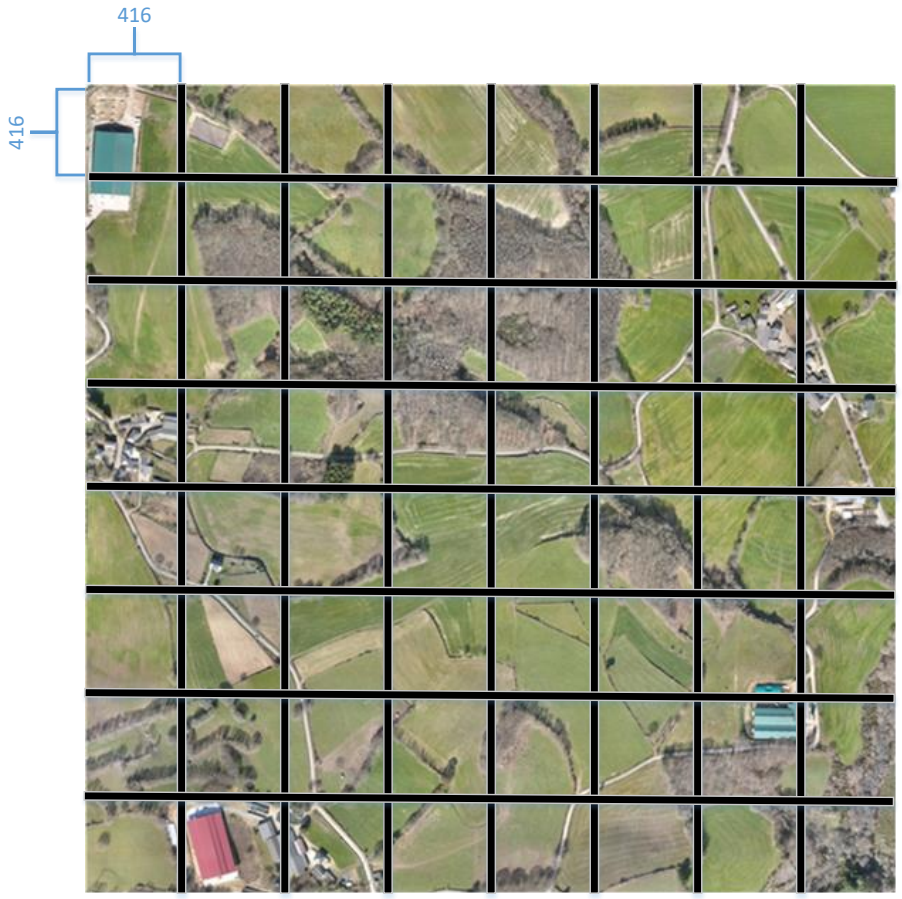


Figura 4.10.- Metodología para generar recortes

Una vez que se crea este grid, se recorre cada una de las celdas. Cada celda es un posible recorte (de 416×416). Si la celda correspondiente tiene al menos un objeto, se realiza el recorte y se adapta la anotación de cada uno de los objetos presentes en dicho recorte. Es necesario adaptar las anotaciones debido a que el GroundTruth disponible hasta el momento es relativo a la imagen completa, y no a su correspondiente recorte. Otro aspecto importante que cabe destaca,r es que al realizar recortes es muy posible que

haya objetos que caigan entre dos celdas, por lo que saldrían cortados. Para solucionar esta situación, se ha decidido que para tener en cuenta **un objeto, éste debe encontrarse en el recorte en al menos un 50 %**, de lo contrario, se obviará. En el caso de que se encuentre en al menos un 50 %, se adaptará su anotación.

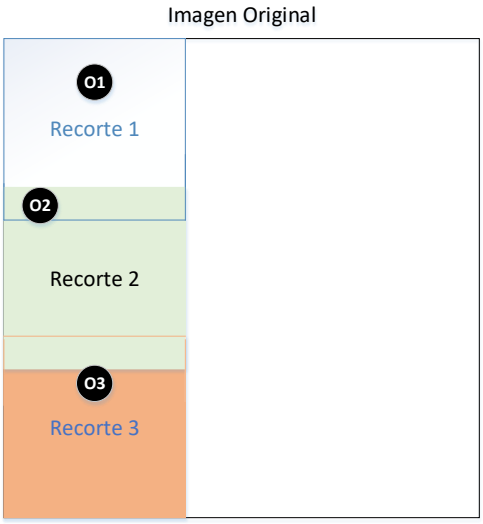


Figura 4.11.- Ejemplo del solape realizado para obtener los recortes

En la Figura 4.11 se puede apreciar visualmente un posible resultado de recortar una imagen original. En este caso, las dimensiones de la imagen original no son divisibles por 416, por lo que se produce un solapamiento entre los tres recortes representados. El objeto 1 (O1) se encontraría en el recorte número 1. El objeto 2 (O2) se encontraría en el recorte 1 y 2. Aunque sea el mismo objeto, hay que tener en cuenta que no saldrá en la misma posición en ambos recortes. Por último, el objeto 3 (O3) está presente en los recortes 2 y 3, pero como en el recorte 2, el objeto está presente en menos de un 50 %, no se incluye en dicho recorte (no se incluye en la anotación correspondiente, el objeto en el recorte si aparece, pero efectivamente no se anota).

Una vez llevado a cabo todo este proceso, se obtienen 526 imágenes y 787 objetos anotados. Esta cantidad de imágenes y objetos es demasiado baja como para poder entrenar un modelo. La solución es la siguiente:

1. Dividir el conjunto de recortes en cuatro subconjuntos (de entrenamiento y test).
2. Sobre los **conjuntos de entrenamiento** se realiza un aumento de datos. Este aumento de datos consiste en lo siguiente: se centra la imagen en cada uno de los objetos ya anotados y se genera un nuevo recorte de 416×416. De esta manera en el nuevo recorte siempre hay un objeto en el centro de la imagen, además del resto de objetos alrededor de dicho objeto central. En la Figura 4.12 se muestra un ejemplo del aumento realizado. El nuevo recorte se centra en uno de los objetos y se tienen en cuenta el resto de los objetos que también se encuentran en el recorte (deben de aparecer al menos al 50 %). Este proceso se repite con todos los objetos obtenidos mediante la técnica del grid (ver Figura 4.10).

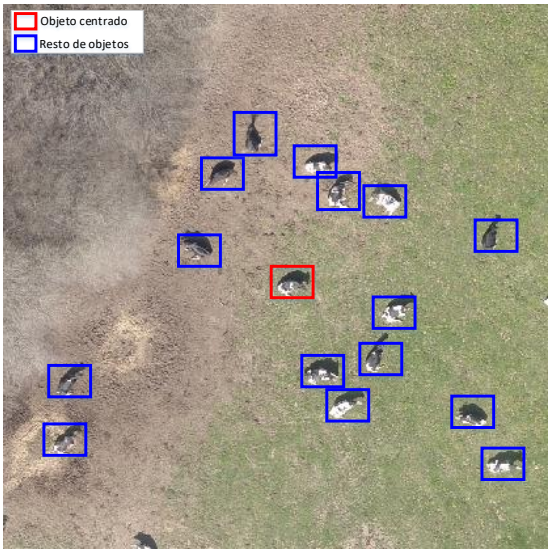


Figura 4.12.- Ejemplo del aumento de datos realizado sobre el conjunto ADAAR para poder optener más datos

En la sección 4.3.5 se muestra con detalle la cantidad de imágenes y objetos resultantes. Es conveniente recordar que este aumento **solo se aplica sobre el conjunto de entrenamiento**, por lo que el test no se ve alterado.

4.3.3.- Análisis del conjunto de datos

Este conjunto cuenta con un total de 10 027 objetos tras haber realizado el aumento de datos manual, explicado en la sección anterior. En la Figura 4.13 se puede apreciar la cantidad de objetos para cada una de las diferentes clases utilizadas en el conjunto. La clase Animal tiene muchas más instancias que el resto de clases, ya que representa el 77.71 % de los datos del total de objetos anotados. Como se aprecia en la Figura 4.13, las clases FosaPurín y Silo presentan muy pocos objetos, se podrían considerar despreciables ya que con esa cantidad de objetos no es factible obtener unos resultados fiables.

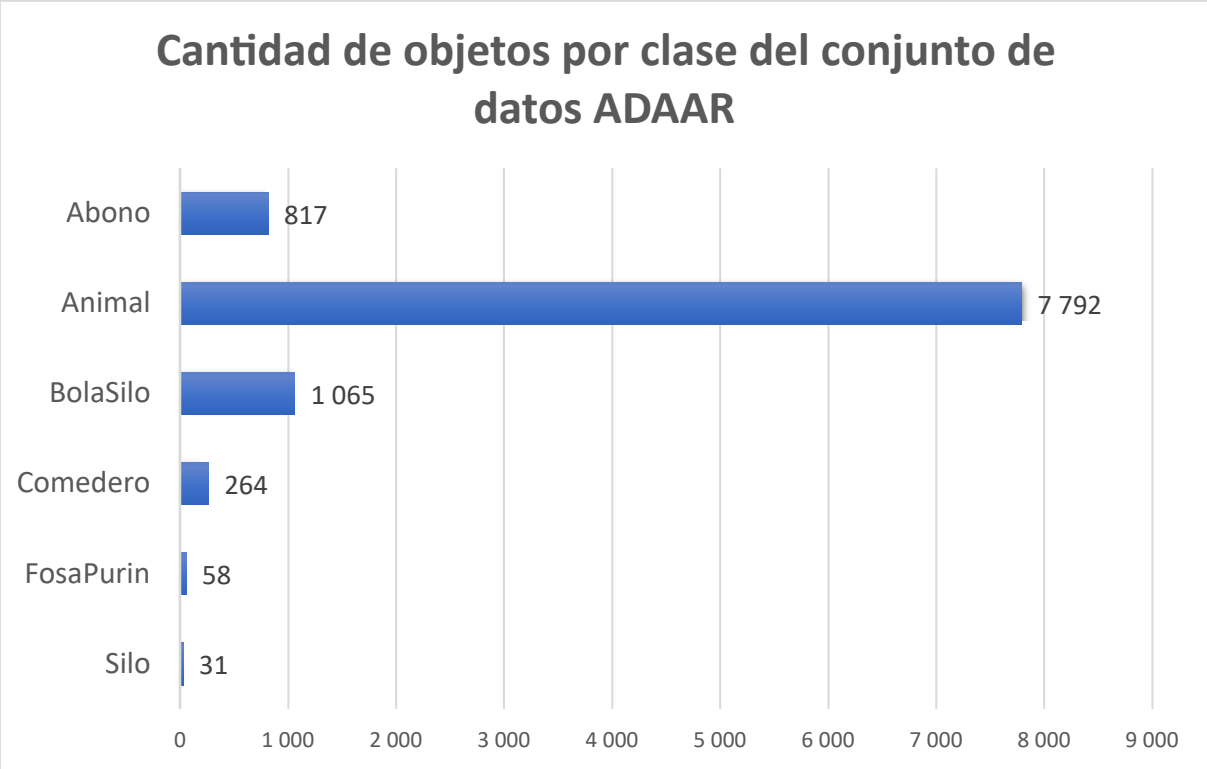


Figura 4.13.- Cantidad de objetos por clase del conjunto de datos ADAAR

En el caso del conjunto de datos DIOR, la dificultad se encuentra en la gran variabilidad de tamaños de objeto intraclase, en el caso de Standford, la dificultad se encuentra en la baja calidad de las imágenes y en este caso, el conjunto de datos ADAAR, **la dificultad viene dada por la baja cantidad de objetos** presentes en el conjunto. En las Figuras 4.14 y 4.15 se puede apreciar cómo no existe una gran variabilidad intraclase. La única excepción podría ser la clase BolaSilo, que cuenta con una gran cantidad de valores atípicos (outliers). Esto se debe a que, como se puede observar en la Figura 4.9, hay ocasiones en la que se anotan las BolasSilo de forma independiente (como por ejemplo en la imagen de la izquierda), y en otras ocasiones, se anotan varias BolasSilo de forma conjunta (como en la imagen central y de la derecha).

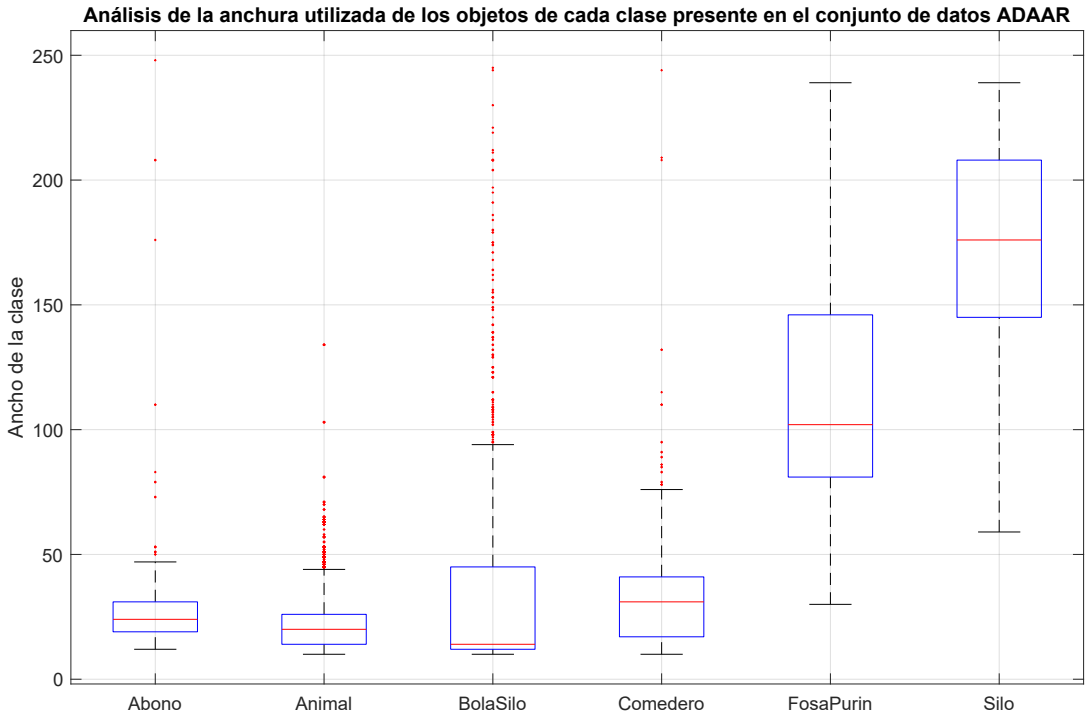


Figura 4.14.- Distribución de la anchura por clases del conjunto de datos ADAAR

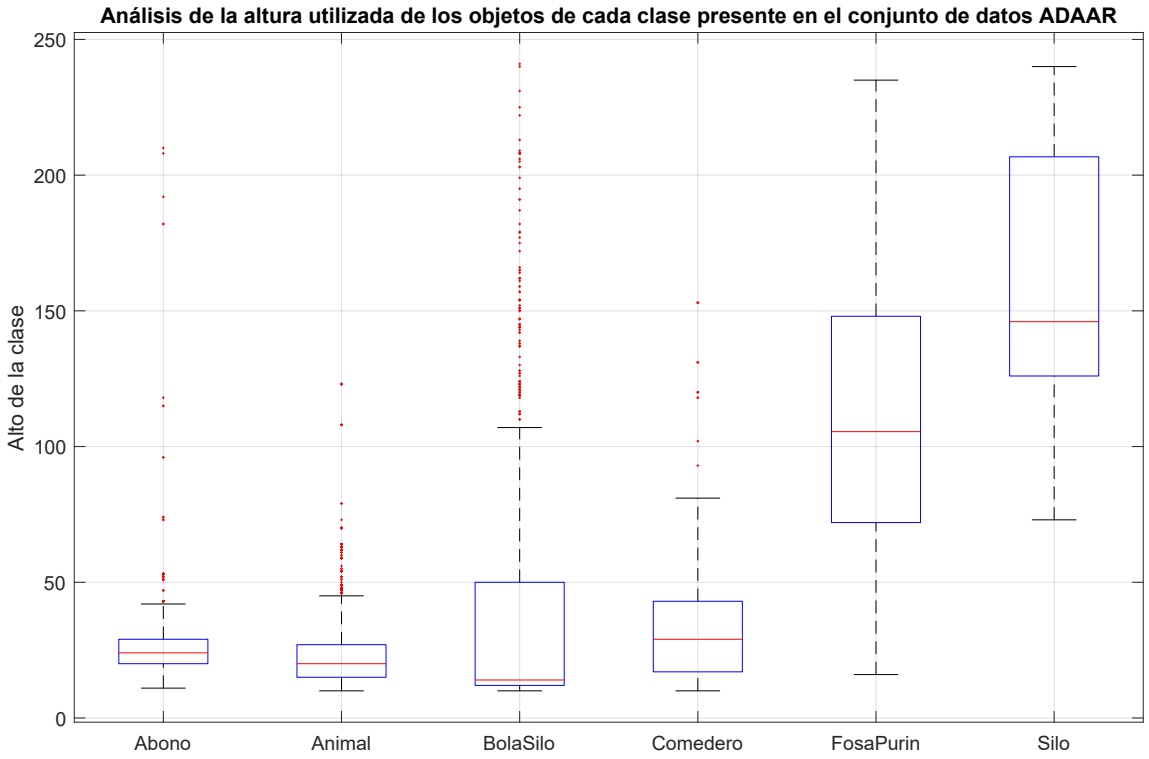


Figura 4.15.- Distribución de la altura por clases del conjunto de datos ADAAR

4.3.4.- División del conjunto de datos sin aumento de datos realizado manualmente

En las Tablas 4.9, 4.10, 4.11 y 4.12 se muestra la cantidad de imágenes y de objetos obtenidos para cada uno de los subconjuntos. Como es evidente, la cantidad de imágenes y de objetos es demasiado baja como para poder entrenar un modelo, por lo que se debe de optar por realizar el aumento de datos manual, antes de realizar el entrenamiento.

Subconjunto 1								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 1	**Test 1**	**Train 1**	**Test 1**	**Train 1**	**Test 1**	**Train 1**	**Test 1**
Abono	15	8	3.66	6.02	51	18	5.99	5.68
Animal	212	66	51.71	49.62	568	215	66.75	67.82
BolaSilo	102	33	24.88	24.81	141	51	16.57	16.09
Comedero	57	16	13.90	12.03	65	23	7.64	7.26
FosaPurin	18	7	4.39	5.26	19	7	2.23	2.21
Silo	6	3	1.46	2.26	7	3	0.82	0.95
Total	**394**	**130**	**100**	**100**	**851**	**317**	**100**	**100**

Tabla 4.9.- Subconjunto 1 sin aumento - ADAAR

Subconjunto 2								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 2	**Test 2**	**Train 2**	**Test 2**	**Train 2**	**Test 2**	**Train 2**	**Test 2**
Abono	18	5	4.51	3.47	56	13	6.17	4.98
Animal	206	72	51.63	50.00	609	174	67.14	66.67
BolaSilo	95	40	23.81	27.78	146	46	16.10	17.63
Comedero	53	20	13.28	13.89	67	21	7.39	8.05
FosaPurin	20	5	5.01	3.47	21	5	2.32	1.92
Silo	7	2	1.75	1.39	8	2	0.88	0.77
Total	**386**	**138**	**100**	**100**	**907**	**261**	**100**	**100**

Tabla 4.10.- Subconjunto 2 sin aumento - ADAAR

Subconjunto 3								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 3	**Test 3**	**Train 3**	**Test 3**	**Train 3**	**Test 3**	**Train 3**	**Test 3**
Abono	19	4	4.74	2.82	46	23	5.54	6.80
Animal	205	73	51.12	51.41	559	224	67.35	66.27
BolaSilo	102	33	25.44	23.24	137	55	16.51	16.27
Comedero	48	25	11.97	17.61	60	28	7.23	8.28
FosaPurin	20	5	4.99	3.52	20	6	2.41	1.78
Silo	7	2	1.75	1.41	8	2	0.96	0.59
Total	**388**	**136**	**100**	**100**	**830**	**338**	**100**	**100**

Tabla 4.11.- Subconjunto 3 sin aumento - ADAAR

Subconjunto 4								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 4	**Test 4**	**Train 4**	**Test 4**	**Train 4**	**Test 4**	**Train 4**	**Test 4**
Abono	17	6	4.06	4.84	54	15	5.90	5.95
Animal	211	67	50.36	54.03	613	170	66.92	67.46
BolaSilo	106	29	25.30	23.39	152	40	16.59	15.87
Comedero	61	12	14.56	9.68	72	16	7.86	6.35
FosaPurin	17	8	4.06	6.45	18	8	1.97	3.17
Silo	7	2	1.67	1.61	7	3	0.76	1.19
Total	**404**	**120**	**100**	**100**	**916**	**252**	**100**	**100**

Tabla 4.12.- Subconjunto 4 sin aumento - ADAAR

4.3.5.- División del conjunto de datos con aumento de datos realizado manualmente

En este caso ocurre lo mismo que con los conjuntos DIOR y Standford (ver secciones 4.1.3 y 4.2.3), es necesario dividir el conjunto en cuatro subconjuntos para poder realizar el CrossTesting. En las Tablas 4.13, 4.14, 4.15 y 4.16 se detalla la división de estos subconjuntos tras haber realizado el aumento de datos manual, explicado en la sección 4.3.2. Al igual que en el resto de casos, la cantidad de imágenes y de objetos de la misma clase en cada subconjunto es similar.

Subconjunto 1								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 1	**Test 1**	**Train 1**	**Test 1**	**Train 1**	**Test 1**	**Train 1**	**Test 1**
Abono	91	8	5.07	6.02	799	18	8.23	5.68
Animal	1 073	66	60.37	49.62	7 577	215	78.03	67.82
BolaSilo	361	33	20.12	24.81	1 014	51	10.44	16.09
Comedero	187	16	10.42	12.03	241	23	2.48	7.26
FosaPurin	50	7	2.79	5.26	51	7	0.53	2.21
Silo	22	3	1.23	2.26	28	3	0.29	0.95
Total	**1 708**	**130**	**100**	**100**	**9 710**	**317**	**100**	**99**

Tabla 4.13.- Subconjunto 1 con aumento - ADAAR

Subconjunto 2								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 2	**Test 2**	**Train 2**	**Test 2**	**Train 2**	**Test 2**	**Train 2**	**Test 2**
Abono	94	5	5.28	3.47	804	13	8.23	4.98
Animal	1 079	72	60.58	50.00	7 627	174	78.07	66.67
BolaSilo	352	40	19.76	27.78	1 016	46	10.40	17.63
Comedero	183	20	10.28	13.89	243	21	2.49	8.05
FosaPurin	50	5	2.81	3.47	50	5	0.51	1.92
Silo	23	2	1.29	1.39	29	2	0.30	0.77
Total	**1 698**	**138**	**100**	**100**	**9 769**	**261**	**100**	**100**

Tabla 4.14.- Subconjunto 2 con aumento - ADAAR

Subconjunto 3								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 3	**Test 3**	**Train 3**	**Test 3**	**Train 3**	**Test 3**	**Train 3**	**Test 3**
Abono	95	4	5.32	2.82	794	23	8.19	6.80
Animal	1 079	73	60.45	51.41	7 581	224	78.16	66.27
BolaSilo	359	33	20.11	23.24	1 008	55	10.39	16.27
Comedero	178	25	9.97	17.61	236	28	2.43	8.28
FosaPurin	51	5	2.86	3.52	51	6	0.53	1.78
Silo	23	2	1.29	1.41	29	2	0.30	0.59
Total	**1 702**	**136**	**100**	**100**	**9 699**	**338**	**100**	**100**

Tabla 4.15.- Subconjunto 3 con aumento - ADAAR

Subconjunto 4								
Clase	**Nº imágenes**		**% imágenes**		**Nº objetos**		**% objetos**	
	Train 4	**Test 4**	**Train 4**	**Test 4**	**Train 4**	**Test 4**	**Train 4**	**Test 4**
Abono	93	6	5.15	4.83	802	15	8.20	5.95
Animal	1 085	67	60.17	54.03	7 629	170	78.01	67.46
BolaSilo	363	29	20.13	23.39	1 023	40	10.46	15.87
Comedero	191	12	10.59	9.67	248	16	2.53	6.34
FosaPurin	48	8	2.66	6.45	49	8	0.50	3.17
Silo	23	2	1.27	1.61	28	3	0.29	1.19
Total	**1 718**	**120**	**100**	**100**	**9 779**	**252**	**100**	**100**

Tabla 4.16.- Subconjunto 4 con aumento - ADAAR

En la Figura 4.16 se analiza la diferencia de objetos tras haber realizado el aumento de datos manual (ver sección 4.3.2). Como se puede apreciar, aunque todas las clases aumentan significativamente su número de objetos, la más beneficiada es la clase Animal. Esto se debe, a que tal y como se explicó en la sección 4.3.2, al centrar el nuevo recorte en un determinado objeto también se tienen en cuenta el resto de los objetos que lo rodean. En el caso de los animales, vacas generalmente, suelen encontrarse en rebaños, es decir juntos, y por este motivo esta clase es la más beneficiada.

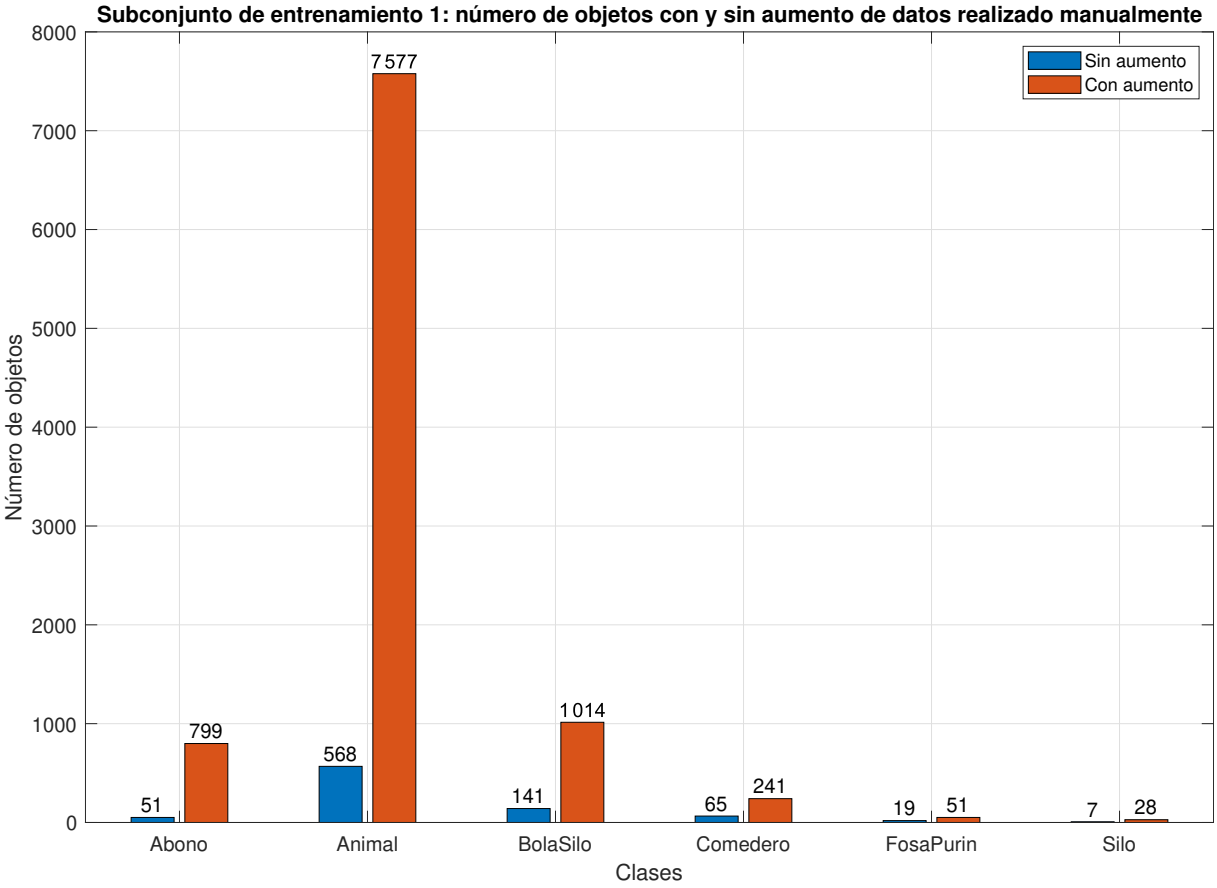


Figura 4.16.- Comparativa del número de objetos del subconjunto 1 de entrenamiento habiendo realizado aumento de datos manual y sin haberlo realizado

5. Evaluación de redes para la detección de objetos

5.1.- Diseño experimental

Tras a haber analizado diferentes redes (ver sección 3.2) y tras haber presentado los diferentes conjuntos de datos utilizados en este proyecto (ver capítulo 4), se propone el diseño experimental mostrado en la Figura 5.1. Para cada conjunto (DIOR, Standford y ADAAR) se va a evaluar con las siguientes redes: YOLOv2, YOLOv5 y CustomVision. El conjunto ADAAR también se evalúa con SSD. Para evaluar cada uno de los detectores de objetos es necesario establecer la configuración de la red y del entrenamiento. La configuración de la red consiste en seleccionar los siguientes valores:

- **Tamaño de entrada**: tamaño de las imágenes que se van utilizar en el entrenamiento. 416×416 en el caso de YOLOv2, YOLOv5 y CustomVision, y 300×300 en el caso de SSD.
- **Número de clases**: seis en el caso de Standford y ADAAR, y veinte en el caso de DIOR.
- **Número de AnchorBoxes**: cinco con YOLOv2, nueve con YOLOv5, seis con SSD y un valor desconocido en CustomVision.
- **Backbone** o FeatureExtractionNetwork: dependiendo del detector utilizado se disponen de distintos backbones. En cada experimento se indica el backbone utilizado.
- **Función de activación**: en el caso de utilizar Darknet53 como backbone la función de activación es Leaky Relu. En el caso de utilizar otro backbone, resnet50 por ejemplo, la función de activación es Relu.

También es necesario configurar los hiperparámetros de entrenamiento:

- **Solucionador**: o solver en inglés, se dispone de SGDM y Adam.
- **Epochs**: número de veces que el conjunto de datos al completo se utiliza en el entrenamiento.
- **BatchSize**: para que un modelo sea robusto es necesario disponer de un número de imágenes elevado. Por este motivo no es posible almacenar todas las imágenes en memoria. Debido a este hecho, es necesario procesar los datos (imágenes) por lotes (batches). El bachSize indica el número de imágenes utilizadas en cada lote.
- **LearningRate**: parámetro que determina la velocidad con la que se ajustan los pesos de la red.
- **Aumento de datos**: en la sección 3.1 se analiza la importancia de utilizar aumento de datos. Durante el entrenamiento existe la posibilidad de no utilizarlo si se considera oportuno. En el caso de utilizarlo, es necesario seleccionar cuáles de los diferentes aumentos se desean utilizar.
- **Momentum**: valor para indicar cuanto contribuye la iteración anterior con la iteración actual en lo que se refiere al descenso del gradiente estocástico.

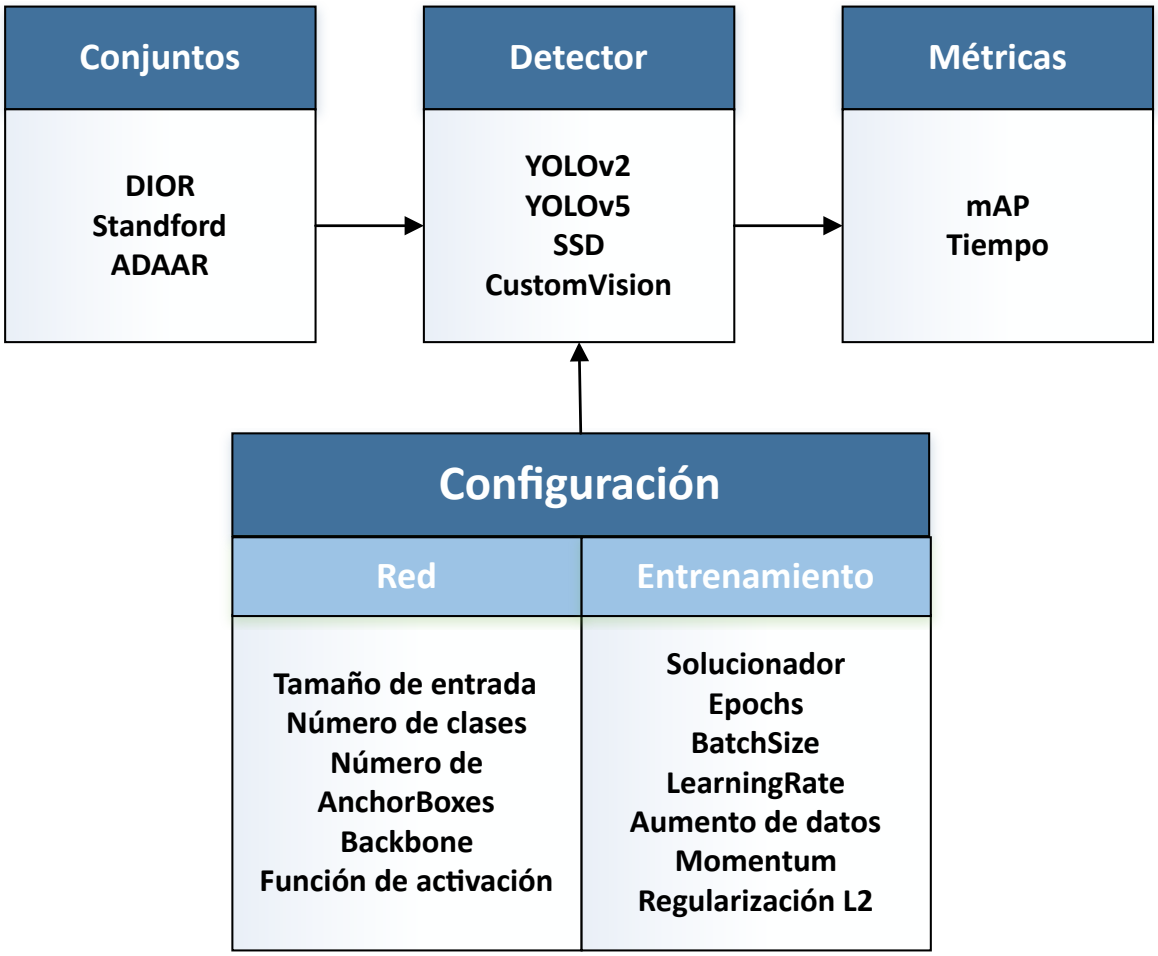


Figura 5.1.- Diseño experimental

- **Regularización L2**: técnica para reducir la complejidad de un modelo. Para ello se penaliza la función de pérdida. Gracias a esto se consigue reducir el sobreajuste. A medida que aumenta la cantidad de características, el modelo se hace más complejo y trata de ajustarse más a todos los parámetros (sobreajuste u overfitting). Para reducir el sobreajuste se pueden penalizar los pesos, haciéndolos muy pequeños, próximos a cero. Esto ayuda a simplificar el modelo. La regularización se basa en la premisa que cuando más pequeños sean los pesos, más sencillo será el modelo y por lo tanto se evitará el sobreajuste.

Se decide comenzar con **YOLOv2** debido a que existe una implementación en Matlab [42]. Gracias a esta implementación, se pueden realizar pruebas rápidas, ya que al utilizar Matlab no es necesario instalar ninguna dependencia, y de esta forma es posible adquirir un conocimiento que se podrá utilizar en el futuro. A continuación, se utiliza **YOLOv5**. Se decide utilizar debido a que es la red más moderna, y uso es sencillo. Además, promete unos resultados muy satisfactorios. Posteriormente, se utiliza **SSD** pero solo con el conjunto ADAAR, ya que es el que mejor se adapta al objetivo del proyecto: **reconocimiento de actividad agraria**. Por último, y con el fin de analizar los servicios ofrecidos en Azure, se decide utilizar **CustomVision** con todos los conjuntos.

Una vez finalizados los experimentos, se analizará el mAP y el tiempo para determinar cuál ha sido el mejor experimento. En el caso de que varios experimentos tengan el mismo mAP se seleccionará el que se haya completado en menos tiempo.

La forma de plantear un entrenamiento es la siguiente: se parte de la configuración base establecida por los autores de la implementación en cuestión. A continuación, se modifica uno de los parámetros de entrenamiento. Si el experimento mejora al anterior, se mantiene fijo dicho parámetro y se varia otro. Si no mejora al anterior, se utiliza el parámetro anterior. De esta forma se consigue el mejor experimento posible para cada uno de los detectores.

YOLOv2 se ha utilizado sobre la **Máquina 1**, mientras que YOLOv5 y SSD se ha utilizado sobre la **Máquina 2** (ver sección 7.1). CustomVision funciona como una caja negra donde se desconoce el hardware utilizado y los parámetros de entrenamiento.

5.2.- Resultados obtenidos con el conjunto de datos DIOR

El primer conjunto de datos utilizado ha sido DIOR (ver sección 4.1). Se ha utilizado este conjunto debido a que se puede utilizar como benchmark de un detector de objetos, debido a la presencia de 20 clases, la gran cantidad de objetos anotados, así como la gran variabilidad intraclase en cuanto al tamaño de los objetos.

Sobre este conjunto se han utilizado las siguientes CNNs: YOLOv2, YOLOv5 y Azure CustomVision. A continuación se muestran los resultados.

5.2.1.- YOLOv2

En este caso se ha utilizado la implementación de YOLOv2 disponible en Matlab [42]. Tras realizar un gran número de experimentos (ver Anexo A), la configuración que mejor resultados ha generado se recoge en la Tabla 5.1. En este caso se ha utilizado 30 epochs y un BatchSize de 20 imágenes. Este BatchSize es el más alto que se puede configurar con esta implementación de la red y el hardware utilizado, ya que si se utiliza un BatchSize superior se produce un error por falta de memoria en la GPU. Como solucionador/optimizador de la red se utiliza Adam, en lugar de SGDM, por lo que tampoco se utiliza Momentum. El entrenamiento de este experimento se completa en aproximadamente diez horas y cuarenta minutos.

Una vez realizados todos los experimentos disponibles en el **Anexo A**, se llega a la conclusión de que el Aumento de datos es indispensable. Se han probado como backbones Resnet50, Resnet101, Darknet19, Darknet53 y Movilenetv2. Claramente, Resnet50 es el que mejores resultados genera. Además Adam genera mejores resultados que SGDM.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
InputSize	[416 416 3]
Número de Clases	20
Número de AnchorBoxes	5
FeatureExtractionNetwork (BackBone)	Resnet50
FeatureLayer	ReLU
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	30
BatchSize	20
LearningRate	0.0001
Data Augmentation	Sí
Momentum	No
L2 (Weight decay)	0.0005
Duración del entrenamiento	10:42:06

Tabla 5.1.- Configuración del mejor experimento - YOLOv2 - DIOR

Una vez finalizado el experimento se calcula el AP de cada clase utilizando para ello el conjunto de test. En la Figura 5.2 se detalla el resultado obtenido para cada clase. Como se puede observar, existen clases, como pueden ser Chimney (Chimenea) o Basketballcourt (Cancha de baloncesto), en las que se obtienen unos resultados muy satisfactorios. Por otro lado existen otras clases, como pueden ser Bridge (Puente) o Vehicle (Vehículo) en las que los resultados no son muy buenos.

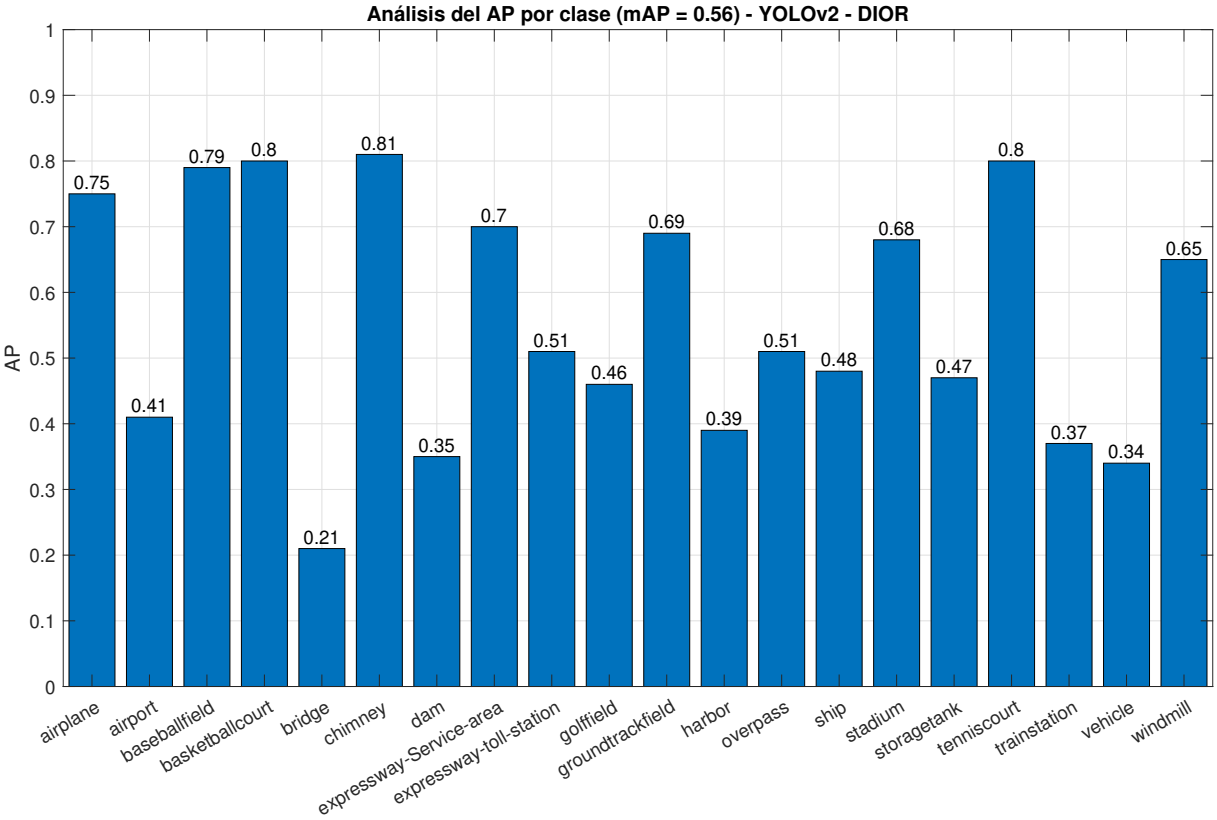


Figura 5.2.- Resultado del mejor experimento - YOLOv2 - DIOR

Este experimento se corresponde con el experimento **E007-01** disponible en el Anexo A y ha sido el mejor de todos los experimentos realizados con YOLOv2 sobre el conjunto de datos DIOR, obteniendo un mAP de 0.56. Cabe destacar que **el mAP es la media de los APs** de cada una de las clases.

5.2.2.- YOLOv5

En este caso se ha utilizado la implementación oficial de YOLOv5 [33]. Tras realizar un gran número de experimentos (ver Anexo A), la configuración que mejores resultados ha generado se recoge en la Tabla 5.2. En esta ocasión, al utilizar una implementación mucho más moderna y por lo tanto más optimizada, se puede utilizar un BatchSize 32 imágenes y 300 epochs sin que afecte al tiempo de entrenamiento. En este caso, el hardware utilizado solo varía en el sistema operativo utilizado (ver sección 7.1).

En esta implementación solo dispone de CSPDarknet53 como backbone. También es mucho más eficiente que la de YOLOv2 debido a que en un tiempo equivalente se pueden multiplicar por 100 el número de epochs del entrenamiento.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
Modelo	YOLOv5M
InputSize	[416 416 3]
Número de Clases	20
FeatureExtractionNetwork	CSPDarknet53
FeatureLayer	Leaky ReLU
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	300
BatchSize	32
LearningRate	0.001
Data Augmentation	Sí
Momentum	0.937
L2 (Weight decay)	0.0005
Shuffle	No disponible
Duración del entrenamiento	13:18:00

Tabla 5.2.- Configuración del mejor experimento - YOLOv5 - DIOR

Una vez finalizado el experimento se calcula el AP de cada clase utilizando para ello el conjunto de test. En la Figura 5.3 se detalla el resultado obtenido para cada clase. Como se puede observar, en esta ocasión los resultados de todas las clases son muy satisfactorios produciéndose, quizás, dos excepciones: Bridge (Puente) y Vehicle (Vehículo), aunque en ambos casos mejoran los resultados obtenidos con YOLOv2 (ver Figura 5.2).

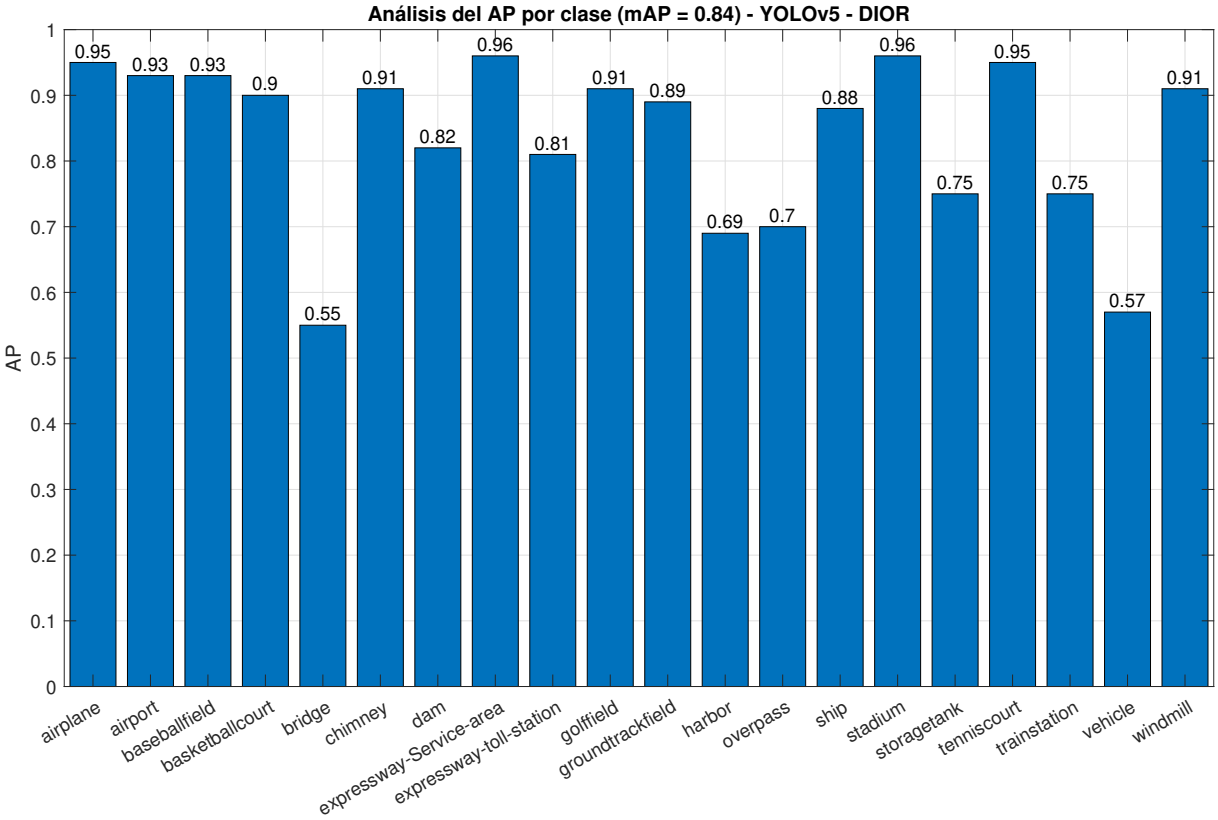


Figura 5.3.- Resultado del mejor experimento - YOLOv5 - DIOR

Este experimento se corresponde con el experimento **E011-04** disponible en el Anexo A y ha sido el mejor de todos los experimentos realizados con YOLOv5 sobre el conjunto de datos DIOR, obteniendo un mAP de 0.84.

5.2.3.- CustomVision

En este caso se ha utilizado el servicio Azure CustomVision [43]. Tras realizar dos experimentos (ver Anexo A) la mejor configuración se puede apreciar en la Tabla 5.3. En este caso, al **funcionar CustomVision como una caja negra** donde solo se puede seleccionar la duración del experimento, el número de experimentos es muy limitado.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
InputSize	[416 416 3]
Número de Clases	20
FeatureExtractionNetwork	Desconocido
FeatureLayer	Desconocido
Parámetros de entrenamiento	
Solucionador de red (solver network)	Desconocido
Epochs	Desconocido
BatchSize	Desconocido
LearningRate	Desconocido
Data Augmentation	Desconocido
Momentum	Desconocido
L2	Desconocido
Duración del entrenamiento fijada	01:00:00

Tabla 5.3.- Configuración del mejor experimento - CustomVision - DIOR

En el caso de CustomVision los resultados se pueden observar en la Figura 5.4. Los resultados son muy dispares entre las diferentes clases, produciéndose muy buenos resultados en clases como pueden ser Airplane (Avión) o Windmill (Molino de Viento), y resultados muy malos como en la clase Expressway-toll-station (Estaciones de peaje).

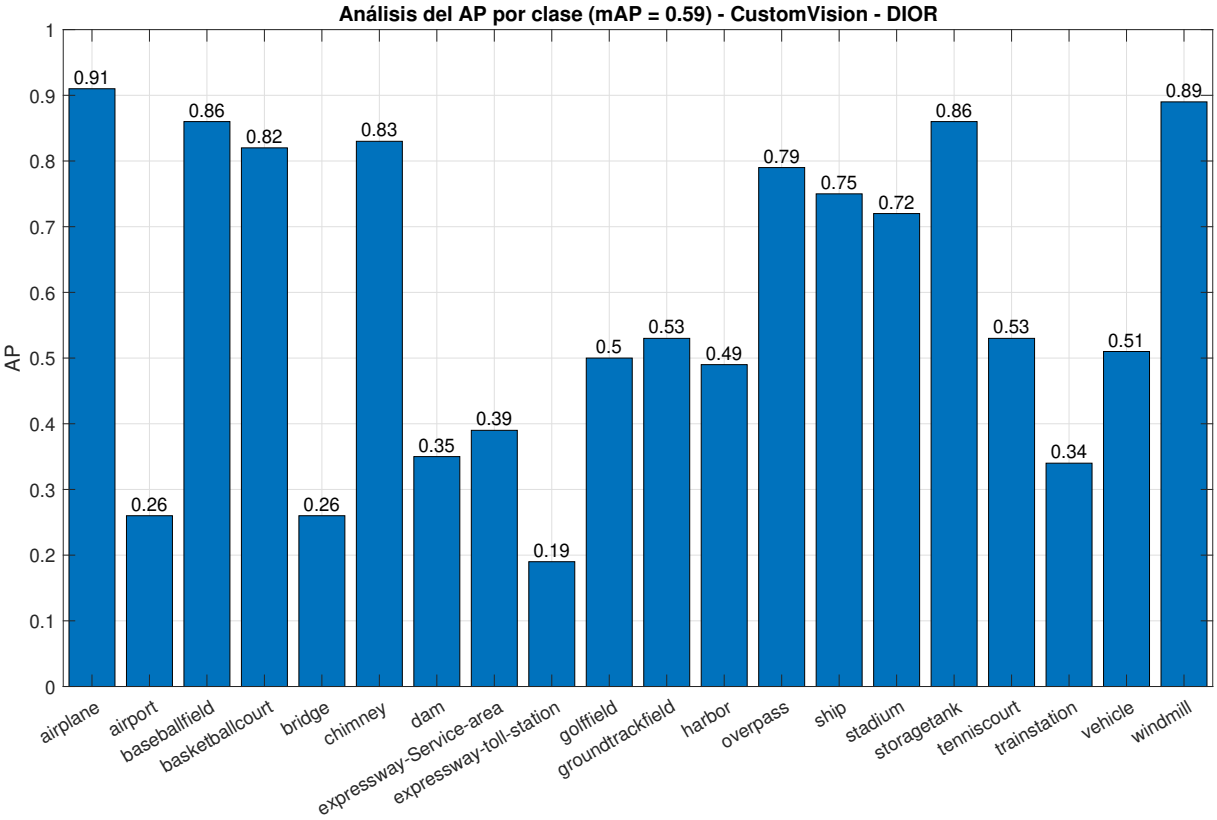


Figura 5.4.- Resultado del mejor experimento - CustomVision - DIOR

Este experimento se corresponde con el experimento **E002-01** disponible en el Anexo A y ha sido el mejor de todos los experimentos realizados con CustomVision sobre el conjunto de datos DIOR, obteniendo un mAP de 0.59.

5.2.4.- Comparativa de los resultados obtenidos con YOLOv2, YOLOv5 y CustomVision sobre el conjunto de datos DIOR

En la Figura 5.5 se puede analizar los resultados obtenidos con las diferentes redes utilizadas sobre los conjuntos de datos DIOR. Claramente, observando los mAPs obtenidos, YOLOv5 es la que mejores resultados ofrece con un mAP de 0.84. Entre Azure CustomVision y YOLOv2 no se aprecian diferencias significativas, obtenido un mAP de 0.59 y 0.56, respectivamente.

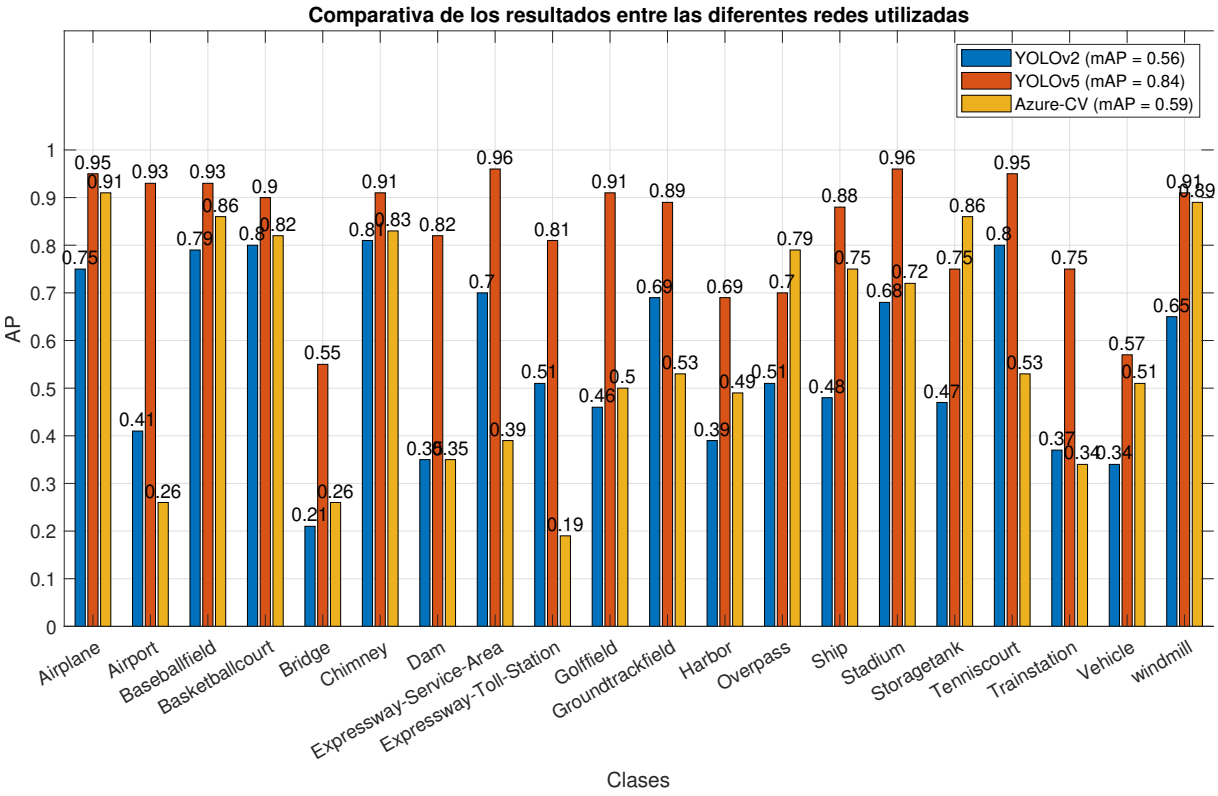


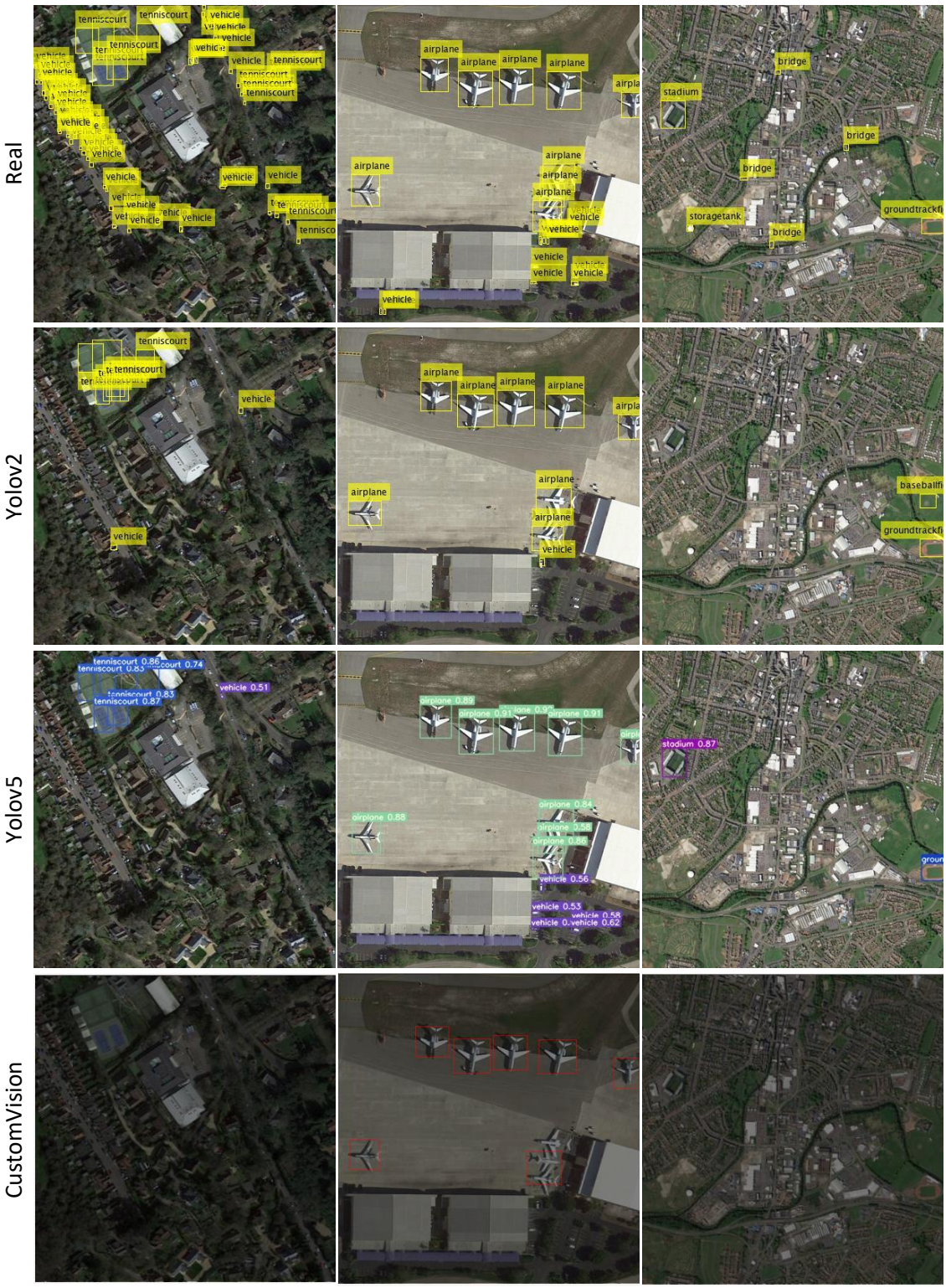
Figura 5.5.- Comparativa de resultados - DIOR

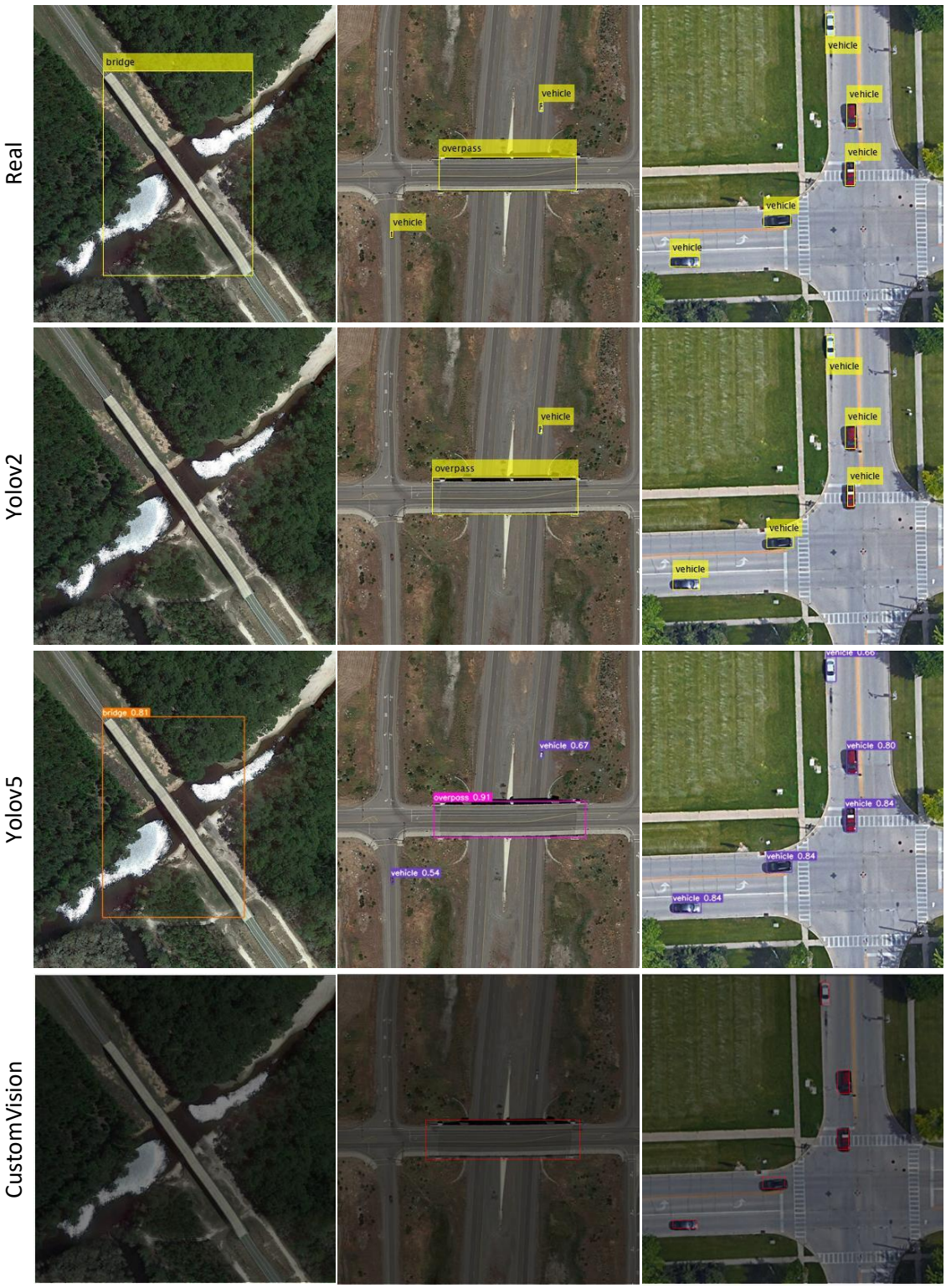
Tras analizar estos resultados se llega a la conclusión de que CustomVision y YOLOv2 no realizan predicciones correctas sobre objetos demasiado grandes. Por ejemplo, las clases Airport (Aeropuerto), Dam (Presa), Expressway-Service-Area (Área de servicios), Golffield (campo de golf) o Trainstation (estación de tren) no se detectan correctamente ni por YOLOv2 ni por CustomVision, produciéndose correctas detecciones en el caso de YOLOv5.

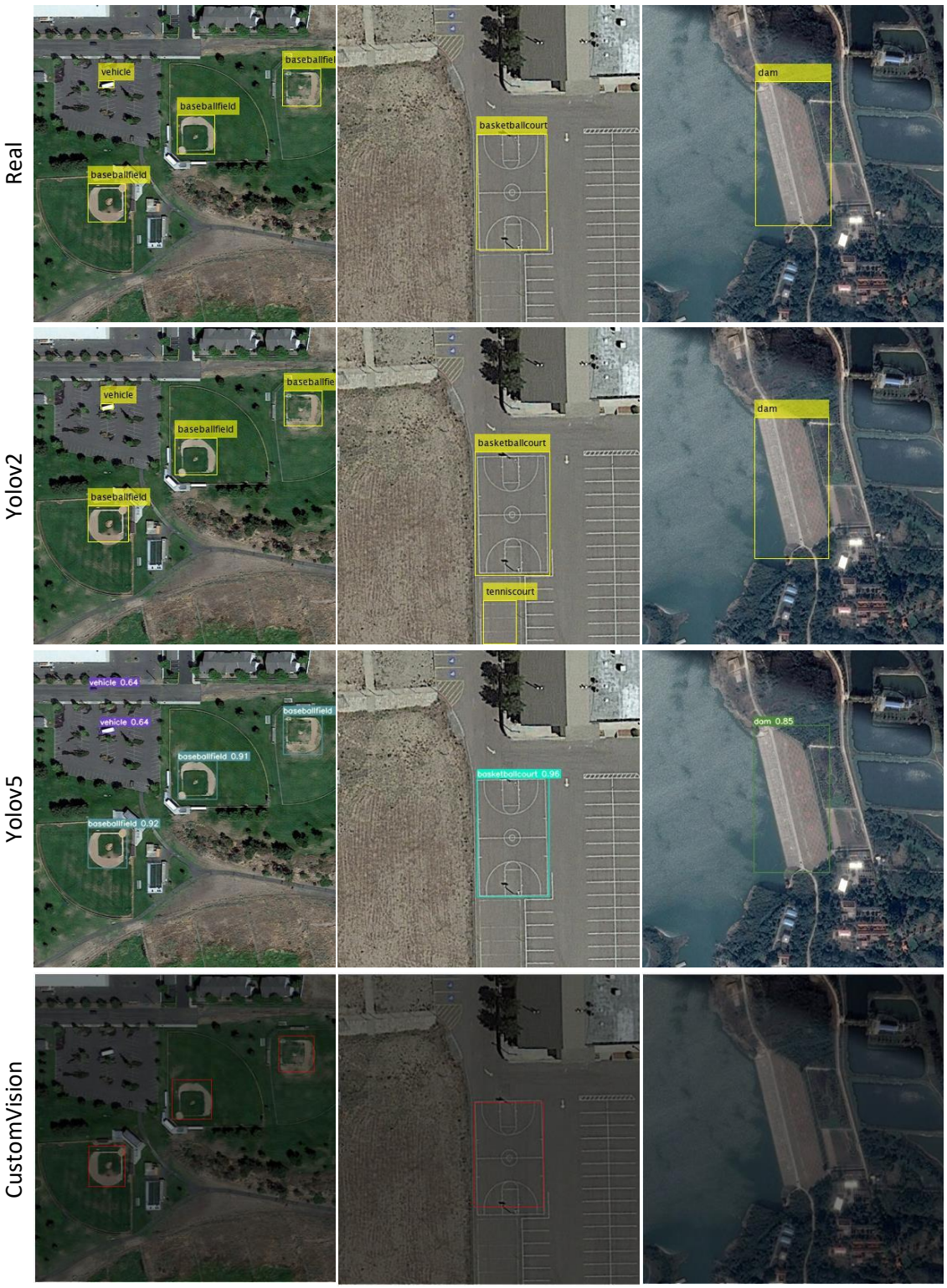
En la Figura 5.6 (nótese que esta Figura ocupa varias páginas) se muestra el resultado de aplicar las diferentes redes sobre una serie de imágenes de test. La forma de leer la Figura 5.6 es la siguiente: en la primera fila (Real) se encuentra la imagen con las anotaciones del GroundTruth, en la segunda fila (YOLOv2) se encuentran las predicciones obtenidas por la red YOLOv2, en la tercera fila (YOLOv5) se encuentran las predicciones obtenidas por la red YOLOv5, y finalmente, en la última fila (CustomVision), se encuentran las predicciones obtenidas por CustomVision. **Todas las predicciones se han generado con una confianza mínima de 0.5.**

Viendo las imágenes se llega a la misma conclusión que analizando la Figura 5.5, donde se encuentra el AP de cada una de las clases del conjunto DIOR: YOLOv5 produce las mejores detecciones de entre todas las redes.

Por último, cabe destacar el formato de las anotaciones de cada red. El Groundtruth y YOLOv2 se han anotado utilizando Matlab, por este motivo tienen un aspecto similar. En el caso de YOLOv5 se anota cada clase de un color diferente, y a continuación del nombre de la clase se muestra la confianza de dicha predicción. Finalmente, en CustomVision al utilizarse la interfaz web para generar las predicciones no se indica la clase de la predicción.


Real
Yolov2
Yolov5
CustomVision


Real
Yolov2
Yolov5
CustomVision
bridge
vehicle
overpass
bridge 0.81
vehicle 0.67
overpass 0.91
vehicle 0.54
vehicle 0.80
vehicle 0.84


Real
Yolov2
Yolov5
CustomVision
vehicle
baseballfield
basketballcourt
dam
tenniscourt
vehicle 0.64
baseballfield 0.91
baseballfield 0.92
basketballcourt 0.96
dam 0.85


Real
Yolov2
Yolov5
CustomVision
chimney
vehicle
vehicle
vehicle
stadium
groundtrackfield
trainstation
chimney 0.67
stadium 0.68

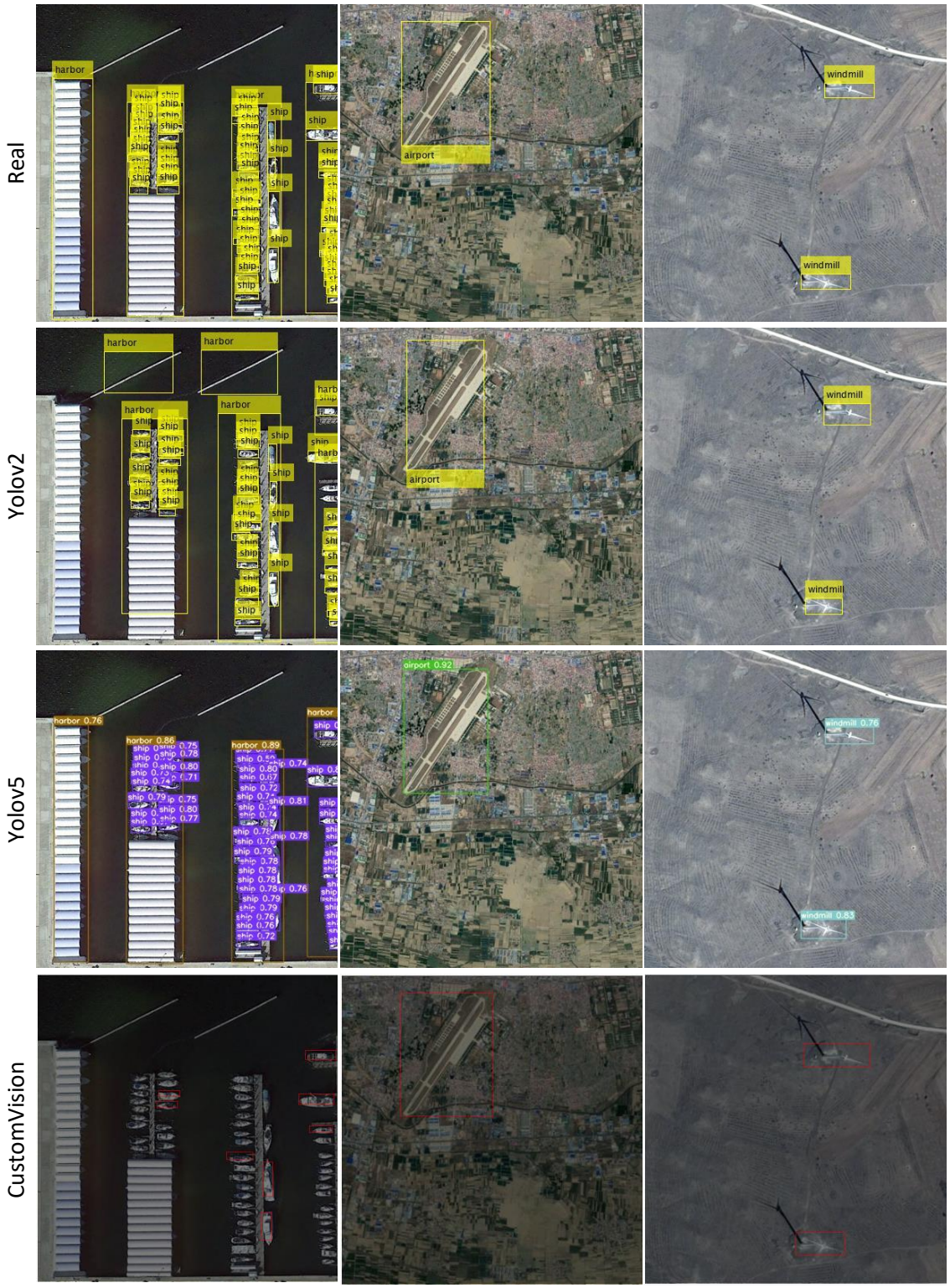


Figura 5.6.- Muestra de las detecciones realizadas - DIOR

5.3.- Resultados obtenidos con el conjunto de datos Standford

El segundo conjunto de datos utilizado ha sido Stanford (ver sección 4.2). **Se ha utilizado este conjunto a petición de la empresa Seresco**, debido a que sería interesante poder detectar actividad humana en un campo de cultivo, ya que esto sería una muestra de que ha habido actividad agrícola.

Sobre este conjunto, también se han utilizado YOLOv2, YOLOv5 y Azure CustomVision. A continuación se muestran los resultados.

5.3.1.- YOLOv2

En este caso, y al igual que con DIOR, se ha utilizado la implementación de YOLOv2 disponible en Matlab [42]. Tras realizar un gran número de experimentos (ver Anexo B) la configuración con la que se han obtenido mejores resultados se puede apreciar en la Tabla 5.4. En esta ocasión se utiliza un BatchSize de 16 imágenes durante 30 epochs, siendo el tiempo de entrenamiento de aproximadamente siete horas.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
InputSize	[416 416 3]
Número de Clases	6
Número de AnchorBoxes	5
FeatureExtractionNetwork (BackBone)	Resnet50
FeatureLayer	ReLU
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	30
BatchSize	16
LearningRate	0.0001
Data Augmentation	Sí
Momentum	No
Duración del entrenamiento	07:17:23

Tabla 5.4.- Configuración del mejor experimento - YOLOv2 - Standford

Al igual que con DIOR, una vez acabado el entrenamiento se calcula el AP sobre el conjunto de test para cada una de las clases. Los resultados se muestran en la Figura 5.7. En este caso, y como se podía esperar tras ver las imágenes de muestra de la Figura 4.5, a excepción de la clase Car (Coche) los resultados no son muy buenos. Las clases Cart y Skater (Patinador) tienen unos resultados muy lejos de lo esperado. Estos resultados se deben a dos motivos: la baja calidad de las imágenes, y por ende de los objetos, y la baja cantidad de objetos de las clases Cart y Skater (Patinador) (ver Figura 4.6).

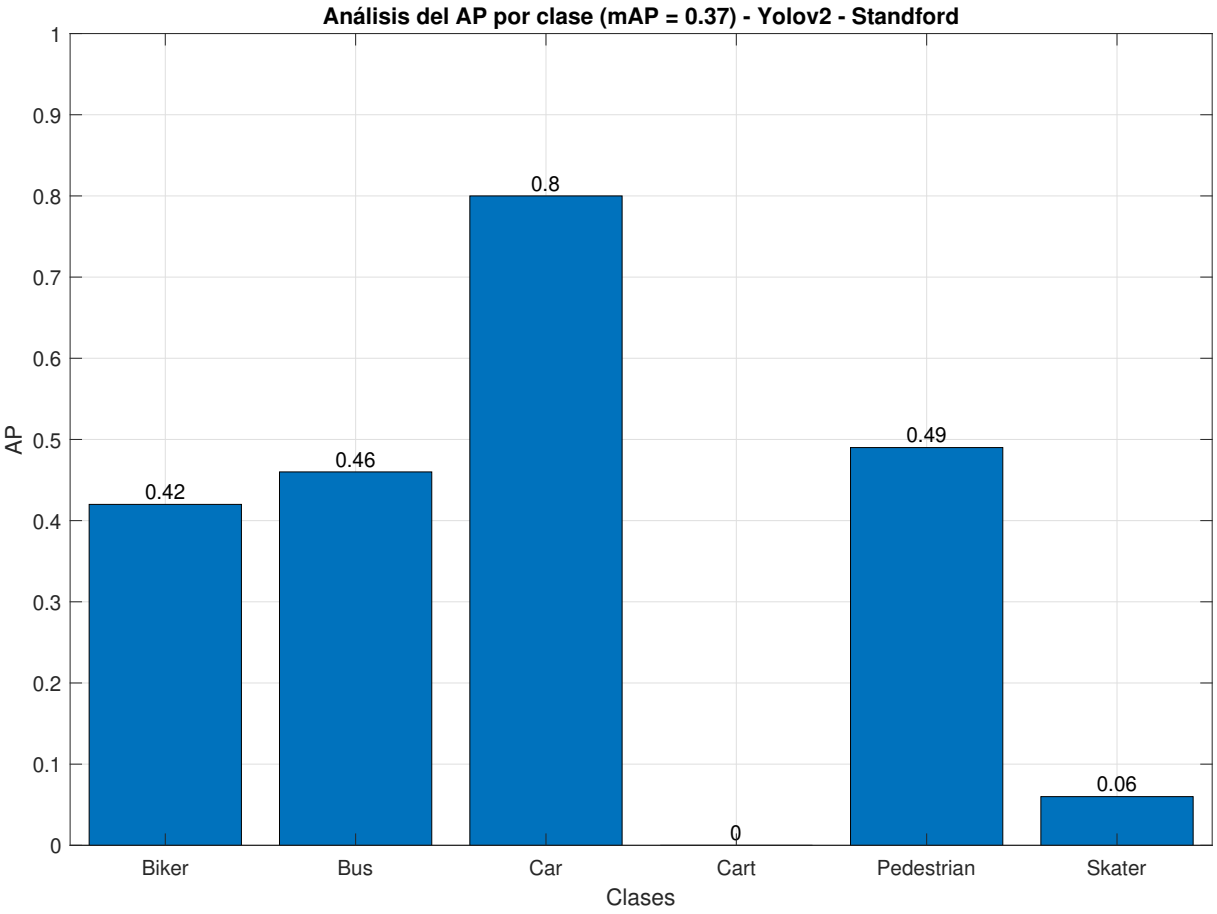


Figura 5.7.- Resultado del mejor experimento - YOLOv2 - Standford

Este experimento se corresponde con el experimento **E006-11** disponible en el Anexo B (sección de recortes) y ha sido el mejor de todos los experimentos realizados con YOLOv2 sobre el conjunto de datos Standford, obteniendo un mAP de 0.37.

5.3.2.- YOLOv5

En este caso se ha utilizado la implementación oficial de YOLOv5 [33]. Tras realizar un gran número de experimentos (ver Anexo B) la configuración con la que se han obtenido mejores resultados se puede apreciar en la Tabla 5.5. En esta ocasión se utiliza un BatchSize de 32 imágenes durante 200 epochs, siendo el tiempo de entrenamiento de aproximadamente siete horas.

En este caso, se decide establecer 200 epochs, y no 300 como con DIOR debido a que el conjunto es más pequeño, y con 300 epochs los resultados no mejoran a los obtenidos con 200.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
Modelo	YOLOv5L
InputSize	[416 416 3]
Número de Clases	6
FeatureExtractionNetwork (BackBone)	CSPDarknet53
FeatureLayer	Leaky ReLU
Parámetros de entrenamiento	
Solucionador de red (solver network)	SGDM
Epochs	200
BatchSize	32
LearningRate	0.001
Data Augmentation	Sí
Momentum	0.937
L2 (Weight decay)	0.0005
Shuffle	No disponible
Duración del entrenamiento	06:49:00

Tabla 5.5.- Configuración del mejor experimento - YOLOv5 - Standford

En la Figura 5.8 se pueden observar los resultados obtenidos. Como era de esperar las clases Cart y Skater (Patinador) tienen un AP prácticamente nulo. En el resto de clases sí se produce una mejora significativa con respecto a los resultados obtenidos con YOLOv2 (ver Figura 5.7), mejorando un 12 % en el caso de clase Car (Coche) y un 20 % aproximadamente en el resto de clases.

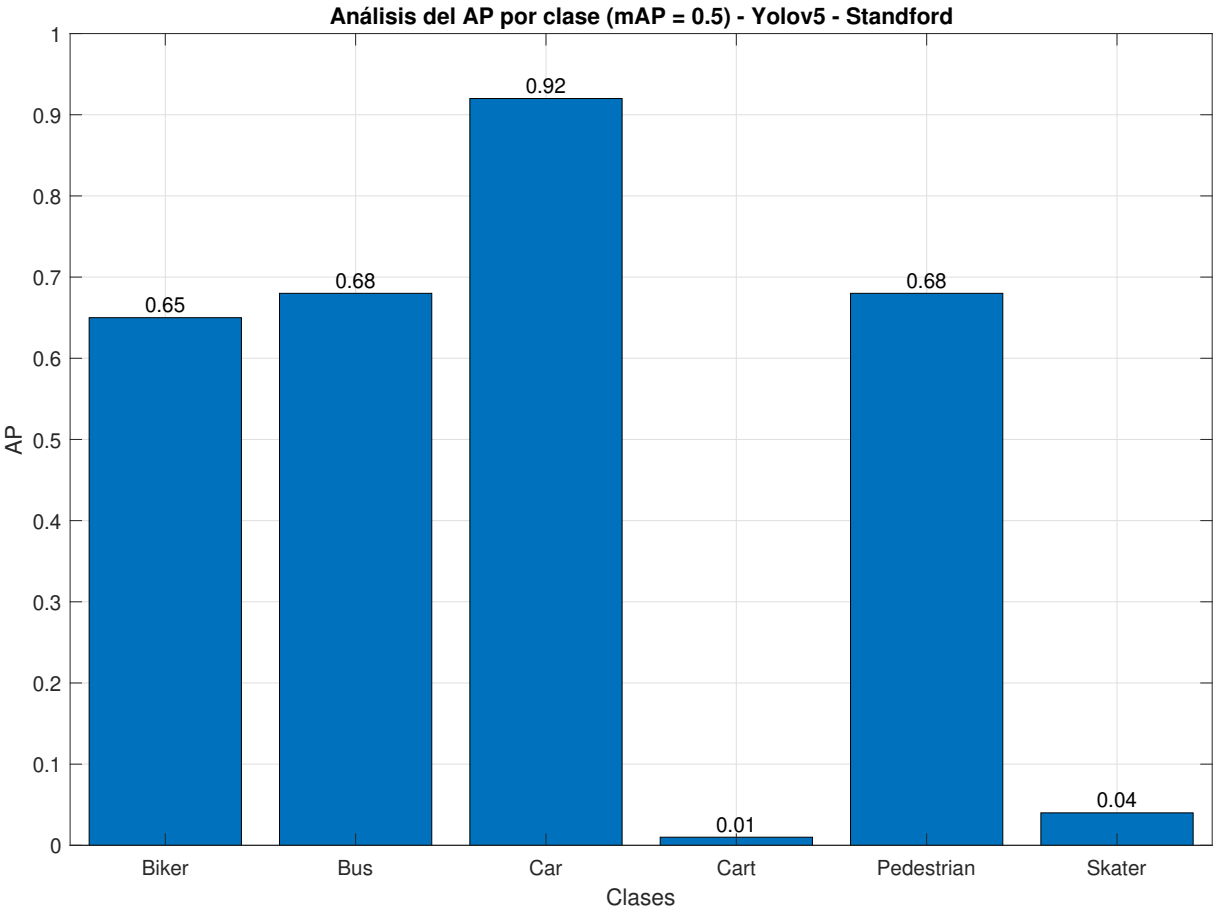


Figura 5.8.- Resultado del mejor experimento - YOLOv5 - Standford

Este experimento se corresponde con el experimento **E006-02** disponible en el Anexo B y ha sido el mejor de todos los experimentos realizados con YOLOv5 sobre el conjunto de datos Standford, obteniendo un mAP de 0.5.

5.3.3.- CustomVision

En este caso se ha utilizado el servicio Azure Custom Vision [43]. Tras realizar dos experimentos (ver Anexo B) la mejor configuración se puede apreciar en la Tabla 5.6. En este caso, al **funcionar CustomVision como una caja** negra donde solo se puede seleccionar la duración del experimento, el número de experimentos es muy limitado.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
InputSize	[416 416 3]
Número de Clases	20
FeatureExtractionNetwork	Desconocido
FeatureLayer	Desconocido
Parámetros de entrenamiento	
Solucionador de red (solver network)	Desconocido
Epochs	Desconocido
BatchSize	Desconocido
LearningRate	Desconocido
Data Augmentation	Desconocido
Momentum	Desconocido
L2	Desconocido
Duración del entrenamiento fijada	01:00:00

Tabla 5.6.- Configuración del mejor experimento - CustomVision - Standford

Los resultados se pueden apreciar en la Figura 5.9. Los resultados de las clases Cart y Skater (Patinador) siguen siendo nulos. En el resto de clases, los resultados superan a los obtenidos con YOLOv2 aunque son inferiores que los obtenidos con YOLOv5.

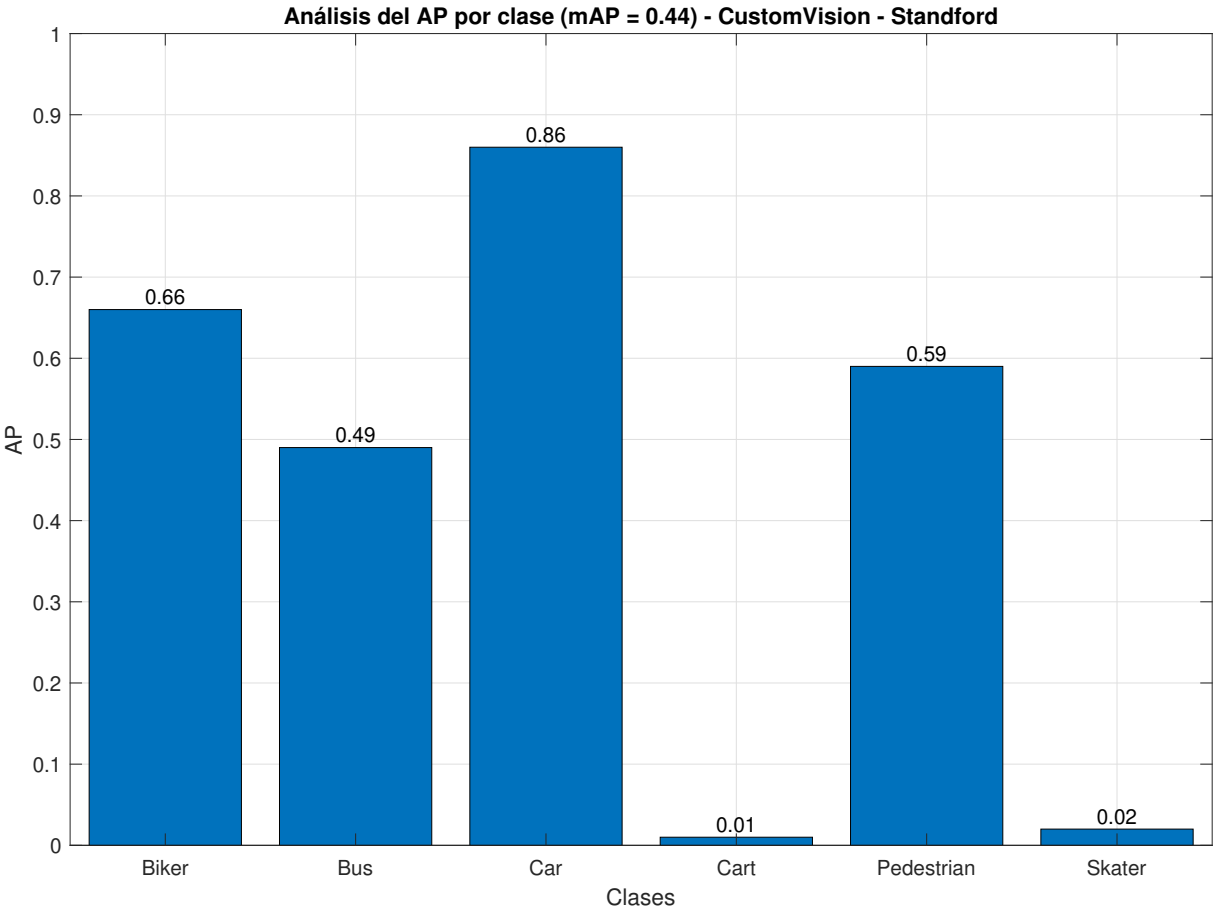


Figura 5.9.- Resultado del mejor experimento - CustomVision - Standford

Este experimento se corresponde con el experimento **E002-01** disponible en el Anexo B y ha sido el mejor de todos los experimentos realizados con CustomVision sobre el conjunto de datos Standford, obteniendo un mAP de 0.44.

5.3.4.- Comparativa de los resultados obtenidos con YOLOv2, YOLOv5 y CustomVision sobre el conjunto de datos Standford

En la Figura 5.10 se puede apreciar una comparativa de los resultados obtenidos tras aplicar las distintas redes utilizadas sobre el conjunto Standford. Los mAPs obtenidos, ordenados de mayor a menor, son los siguientes: YOLOv5 = 0.5, CustomVision = 0.44 y YOLOv2 = 0.36.

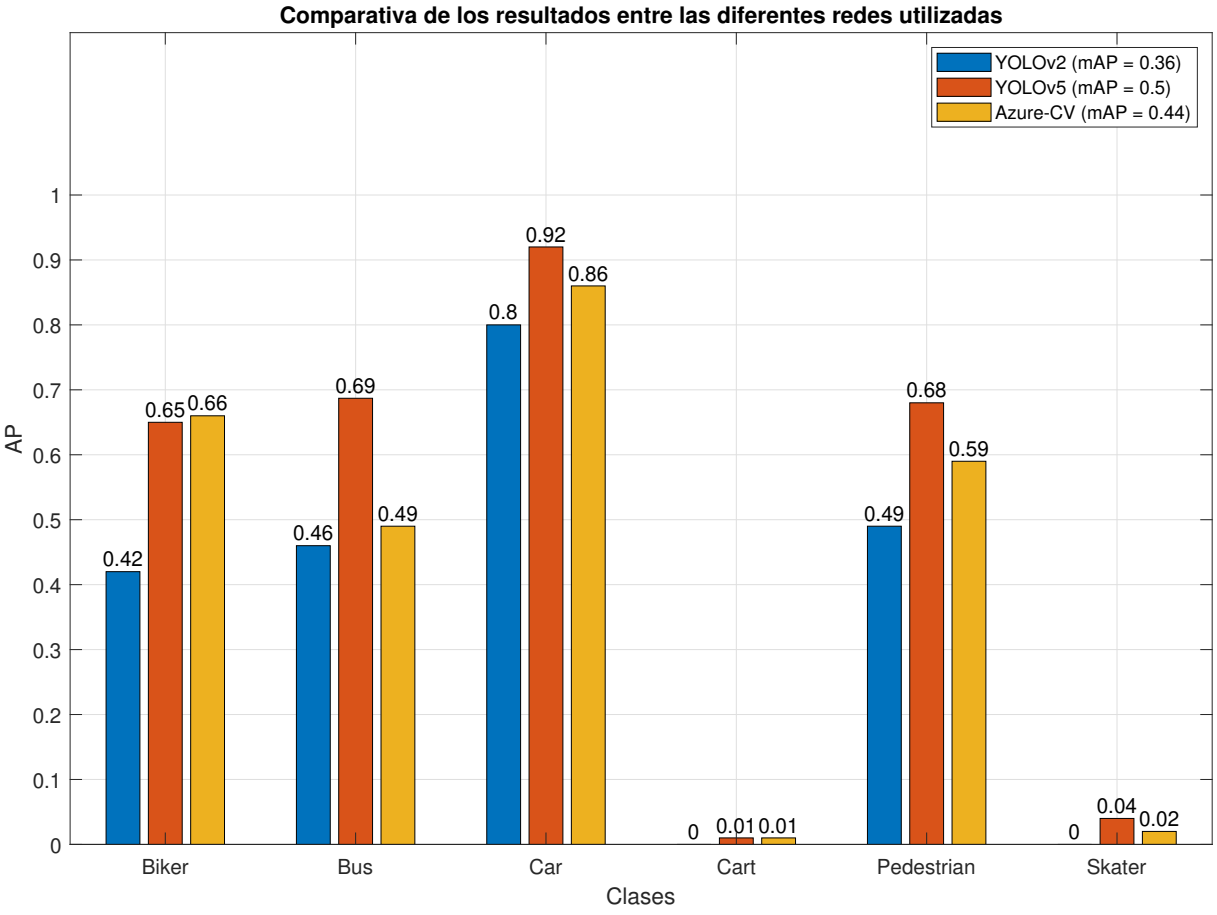
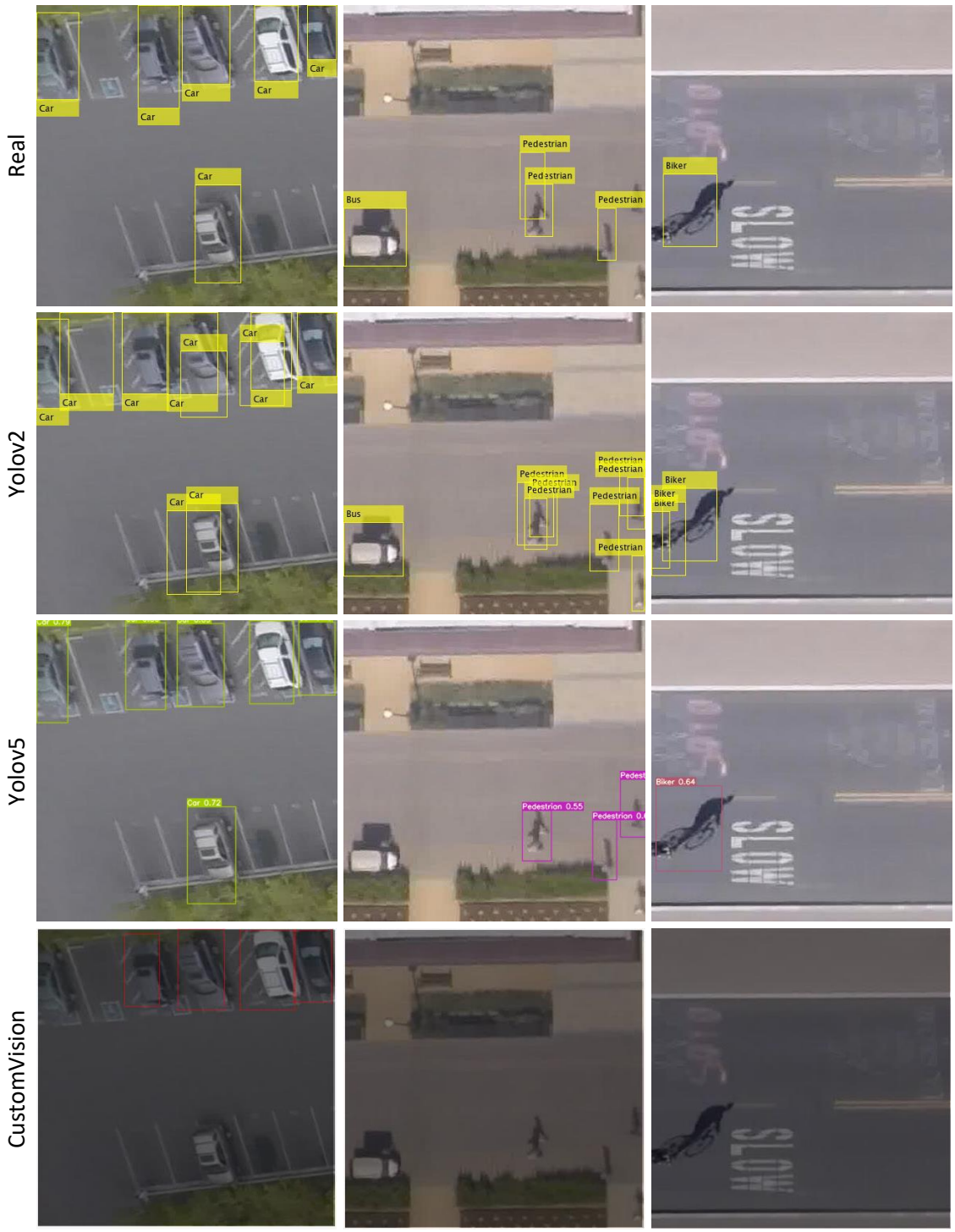
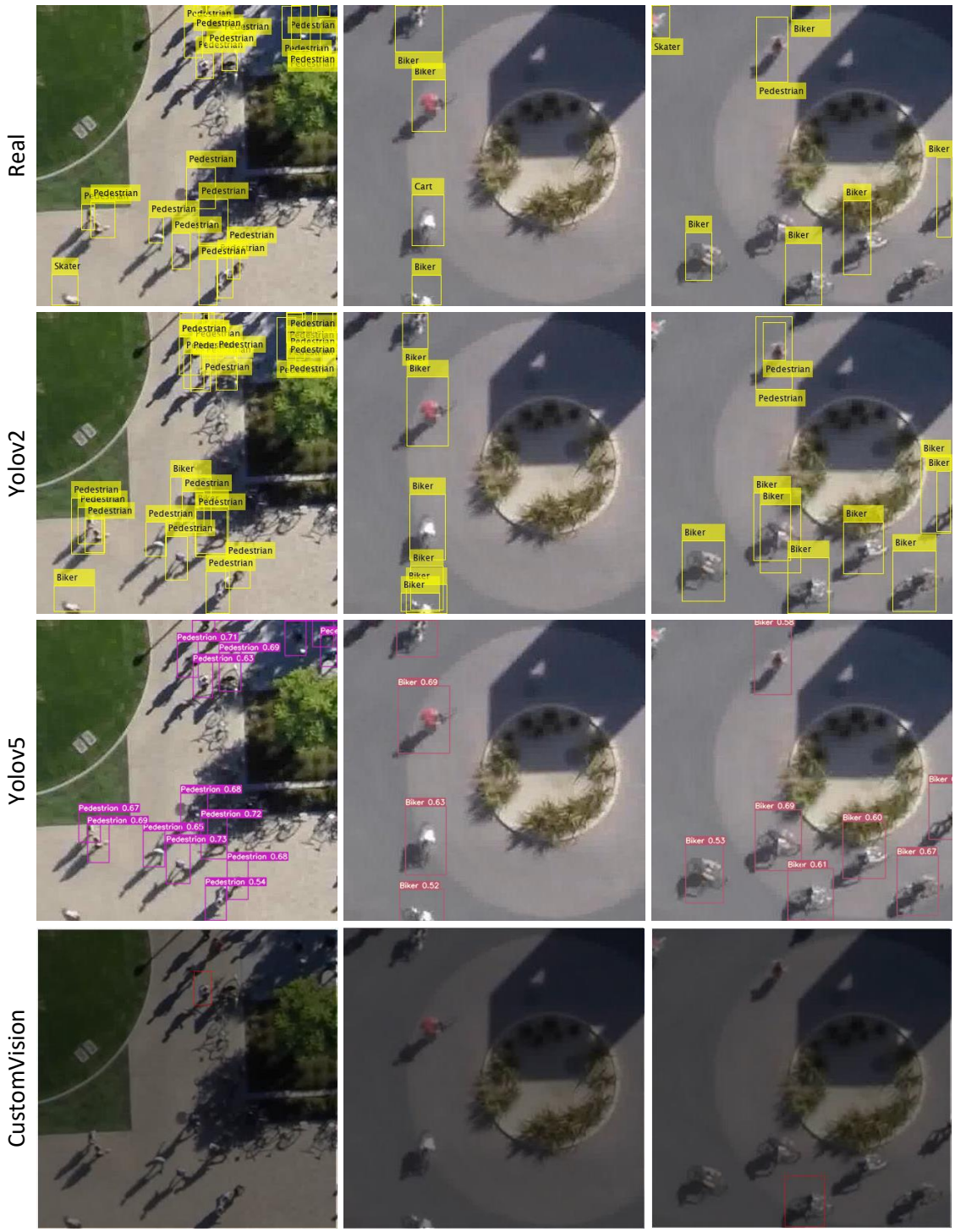


Figura 5.10.- Comparativa de resultados - Standford

Tras analizar los resultados, y al igual que en DIOR (ver sección 5.3), YOLOv5 ofrece los mejores resultados en todas las clases menos en Biker (Ciclista), donde CustomVision le supera en un 1 %.

En la Figura 5.11 (nótese que esta Figura ocupa varias páginas) se muestra el resultado de utilizar las diferentes redes sobre una serie de imágenes de test. La forma de leer la Figura 5.11 es la misma que la Figura 5.6, es decir en la primera fila (Real) se encuentra la imagen con las anotaciones del GroundTruth, en la segunda fila (YOLOv2) se encuentran las predicciones obtenidas por la red YOLOv2, en la tercera fila (YOLOv5) se encuentran las predicciones obtenidas por la red YOLOv5, y finalmente, en la última fila (CustomVision) se encuentran las predicciones obtenidas por CustomVision. **Todas las predicciones se han generado con una confianza mínima de 0.5.**


Real
Yolov2
Yolov5
CustomVision


Real
Yolov2
Yolov5
CustomVision

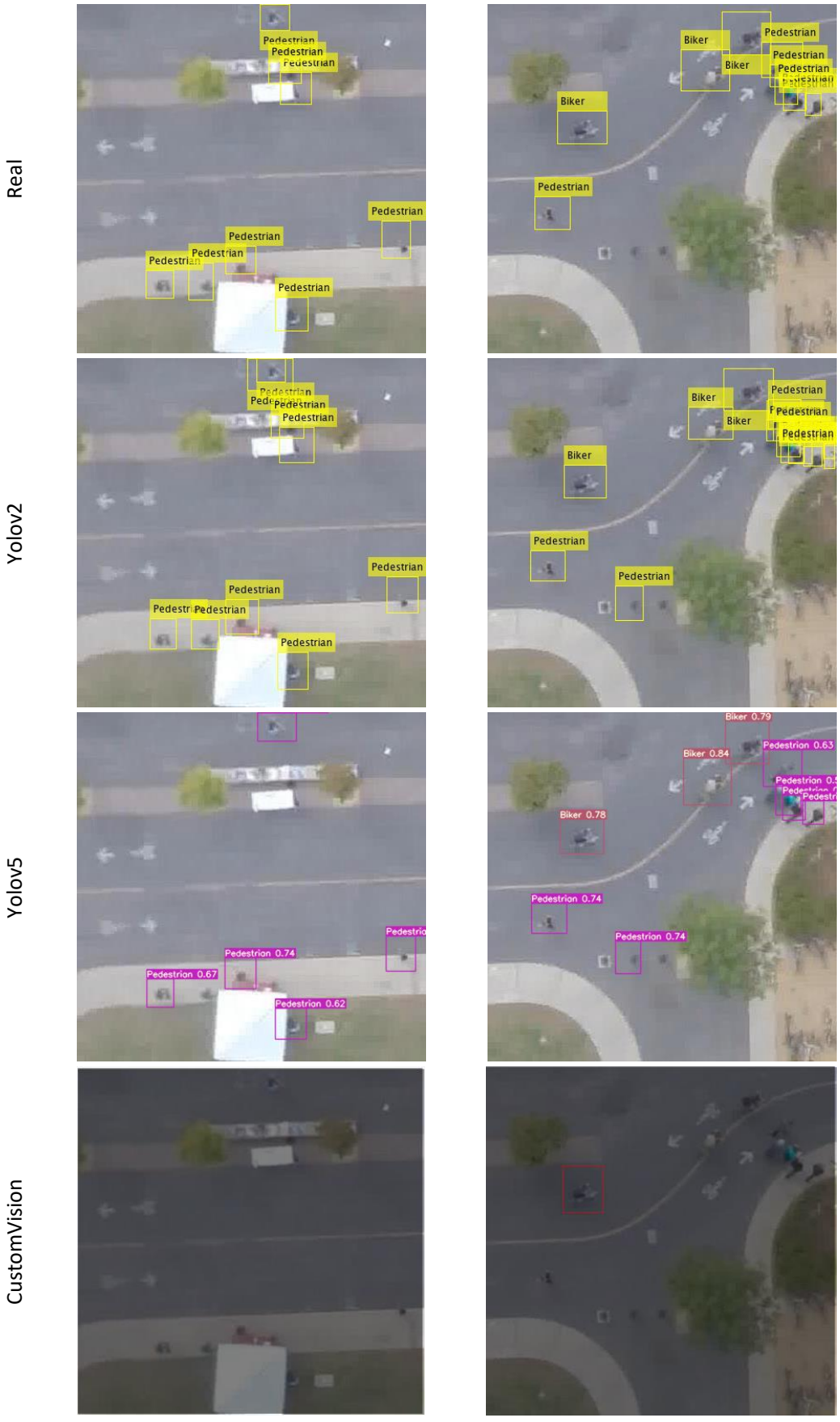


Figura 5.11.- Muestra de las detecciones realizadas - Standford

5.4.- Resultados obtenidos con el conjunto de datos ADAAR

El tercer y último conjunto de datos utilizado ha sido ADAAR (ver sección 4.3). Este conjunto es el propuesto para poder llevar a cabo la detección de actividad agrícola mediante imágenes aéreas.

Sobre este conjunto, también se han utilizado YOLOv2, YOLOv5 y Azure CustomVision, y además se ha utilizado SSD. A continuación, se muestran los resultados.

5.4.1.- YOLOv2

En este caso, y al igual que con DIOR y Standford, se ha utilizado la implementación de YOLOv2 disponible en Matlab [42]. Tras realizar un gran número de experimentos (**ver Anexo C**) la configuración con la que se han obtenido mejores resultados se puede apreciar en la Tabla 5.7. En esta ocasión se utiliza un BatchSize de 16 imágenes durante 30 epochs, siendo el tiempo de entrenamiento de aproximadamente una hora.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
InputSize	[416 416 3]
Número de Clases	6
Número de AnchorBoxes	5
FeatureExtractionNetwork	Resnet50
FeatureLayer	ReLU
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	30
BatchSize	16
LearningRate	0.0001
Data Augmentation	Sí
Momentum	0.9
L2 Regularization	0.0001
Duración del entrenamiento	01:16:04

Tabla 5.7.- Configuración del mejor experimento - YOLOv2 - ADAAR

En la Figura 5.12 se detalla el AP de cada una de las clases, obtenido tras realizar el experimento. Hay que tener en cuenta que estos resultados pueden ser un poco engañosos debido a que las clases FosaPurín y Silo tienen 7 y 3 instancias, respectivamente (ver Tabla 4.13).

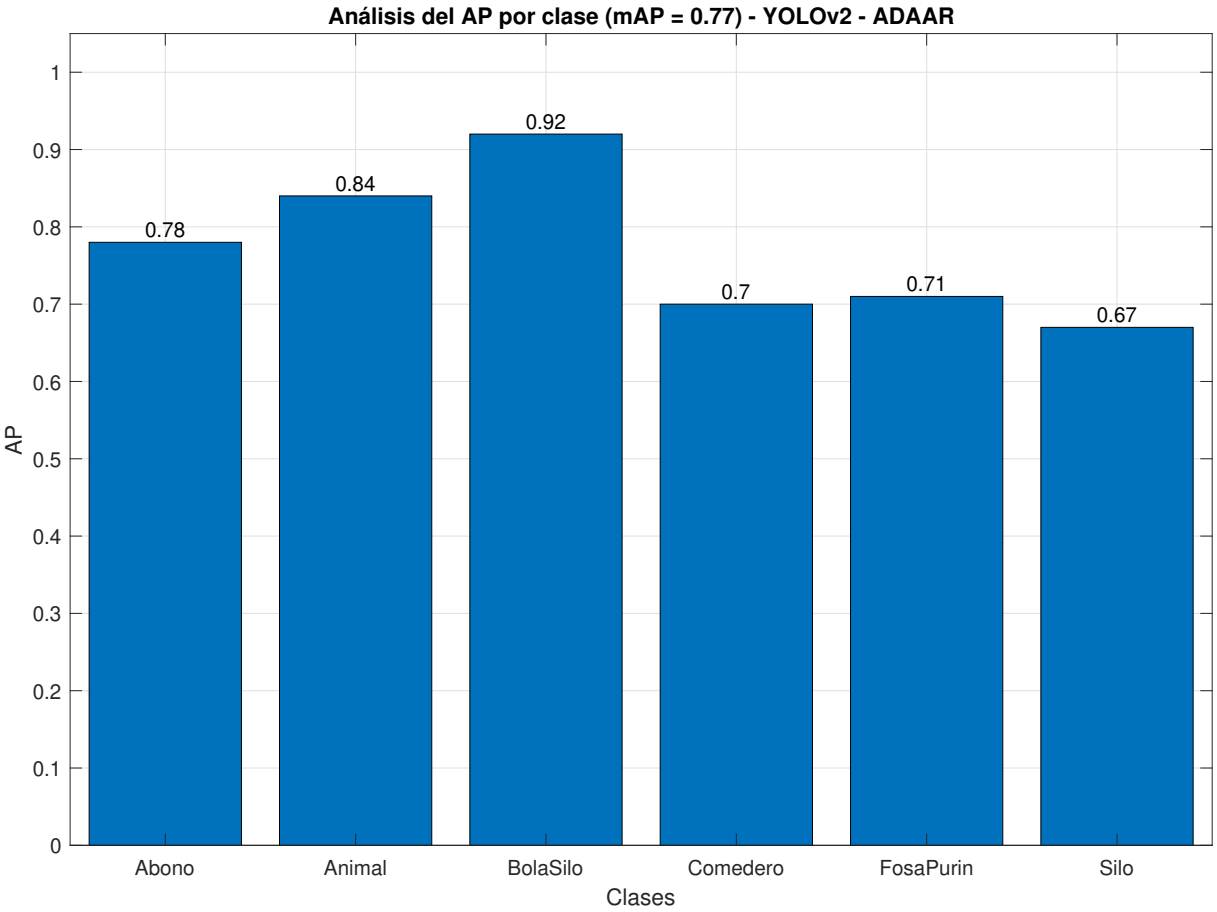


Figura 5.12.- Resultado del mejor experimento - YOLOv2 - ADAAR

Este experimento se corresponde con el experimento **E017-01** disponible en el Anexo C y ha sido el mejor de todos los experimentos realizados con YOLOv2 sobre el conjunto de datos ADAAR, obteniendo un mAP de 0.77.

5.4.2.- YOLOv5

En este caso se ha utilizado la implementación oficial de YOLOv5 [33]. Tras realizar un gran número de experimentos (**ver Anexo C**) la configuración con la que se han obtenido mejores resultados se puede apreciar en la Tabla 5.8. En esta ocasión se utiliza un BatchSize de 32 imágenes durante 500 epochs, siendo el tiempo de entrenamiento de aproximadamente una hora.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
Modelo	YOLOv5S
InputSize	[416 416 3]
Número de Clases	6
FeatureExtractionNetwork	CSPDarknet53
FeatureLayer	Leaky ReLU
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	500
BatchSize	32
LearningRate	0.001
Data Augmentation	Sí
Momentum	No
L2 (Weight decay)	0.0005
Shuffle	No disponible
Duración del entrenamiento	01:04:00

Tabla 5.8.- Configuración del mejor experimento - YOLOv5 - ADAAR

En la Figura 5.13 se puede apreciar el resultado del experimento. Los resultados son muy satisfactorios ya que la clase con peores resultados es Comedero, con un AP de 0.8. Al igual que con YOLOv2, cabe destacar que las clases FosaPurín y Silo tienen 7 y 3 instancias, respectivamente (ver Tabla 4.13).

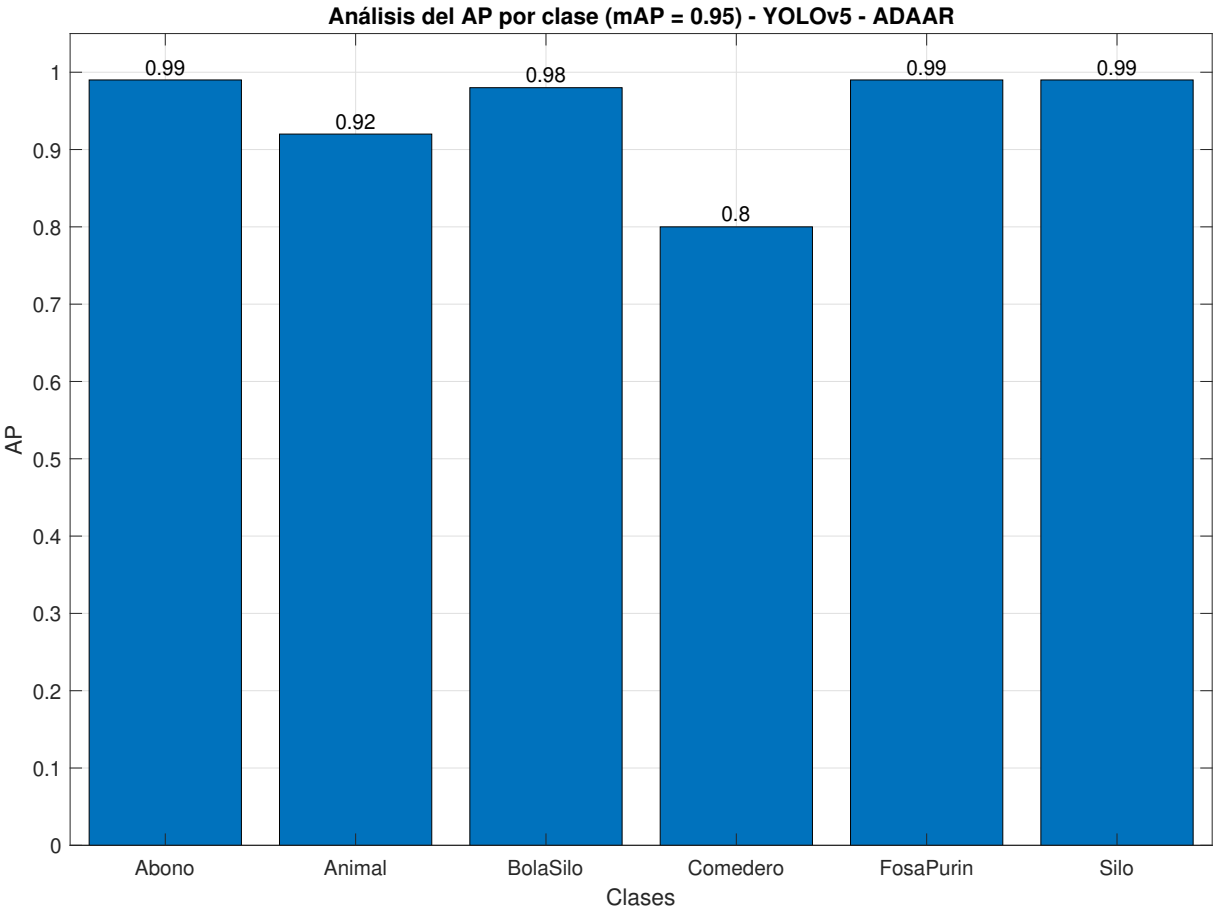


Figura 5.13.- Resultado del mejor experimento - YOLOv5 - ADAAR

Este experimento se corresponde con el experimento **E004-01** disponible en el Anexo C y ha sido el mejor de todos los experimentos realizados con YOLOv2 sobre el conjunto de datos ADAAR, obteniendo un mAP de 0.95.

5.4.3.- CustomVision

En este caso se ha utilizado el servicio Azure Custom Vision [43]. Tras realizar dos experimentos (**ver Anexo C**) la mejor configuración se puede apreciar en la Tabla 5.9. En este caso, al funcionar CustomVision como una caja negra donde solo se puede seleccionar la duración del experimento, el número de experimentos es muy limitado.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
InputSize	[416 416 3]
Número de Clases	6
FeatureExtractionNetwork (Backbone)	Desconocido
FeatureLayer	Desconocido
Parámetros de entrenamiento	
Solucionador de red (solver network)	Desconocido
Epochs	Desconocido
BatchSize	Desconocido
LearningRate	Desconocido
Data Augmentation	Desconocido
Momentum	Desconocido
L2	Desconocido
Duración del entrenamiento fijada	01:00:00

Tabla 5.9.- Configuración del mejor experimento - CustomVision - ADAAR

En la Figura 5.14 se puede observar los resultados obtenidos para cada una de las clases tras la realización del experimento. Como siempre es necesario recordar que las clases FosaPurín y Silo tienen 7 y 3 instancias, respectivamente (ver Tabla 4.13).

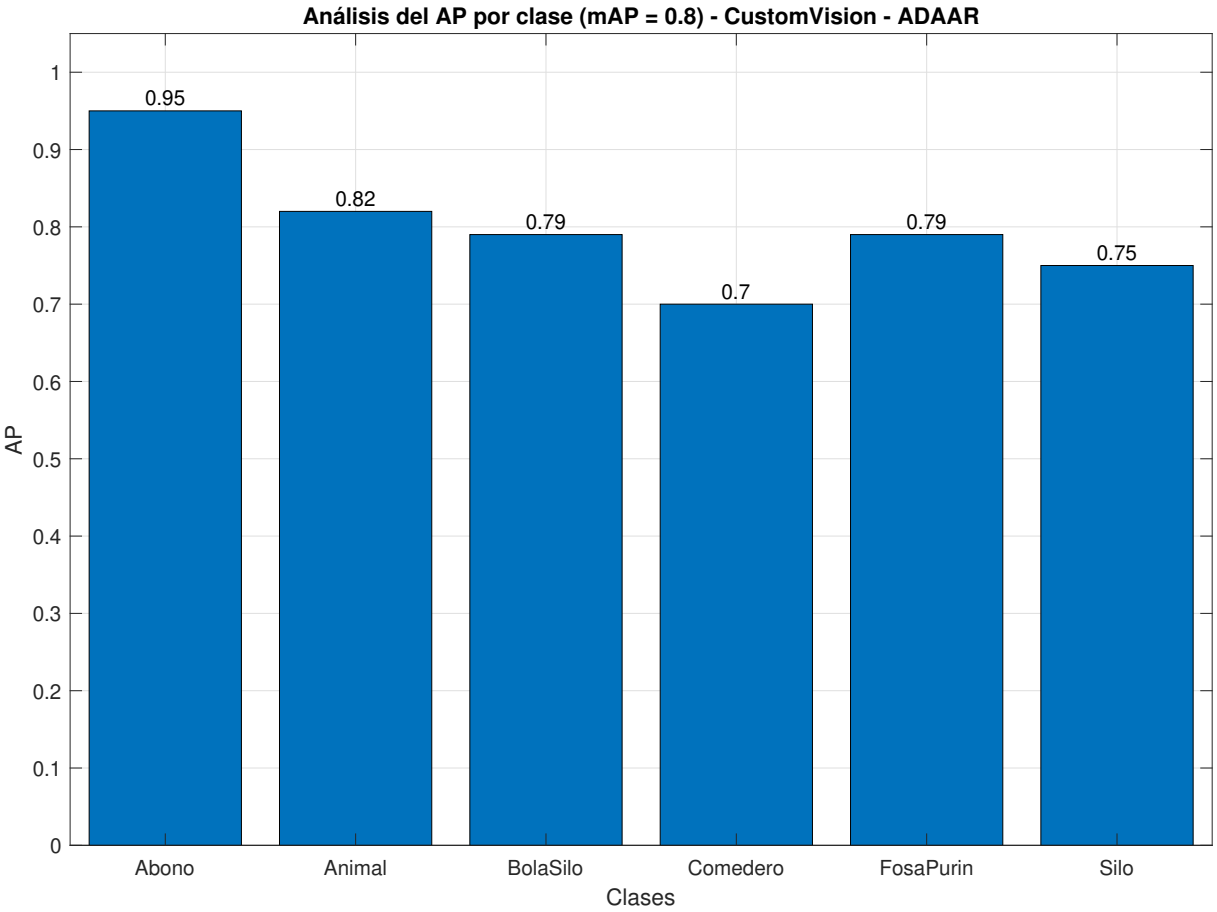


Figura 5.14.- Resultado del mejor experimento - CustomVision - ADAAR

Este experimento se corresponde con el experimento **E002-01** disponible en el Anexo C y ha sido el mejor de todos los experimentos realizados con CustomVision sobre el conjunto de datos ADAAR, obteniendo un mAP de 0.8.

5.4.4.- SSD

En este caso se ha utilizado la implementación de SSD desarrollada por NVIDIA [44]. Esta implementación de SSD tiene un tamaño fijo de entrada de imagen de 300×300. En el resto de redes se utiliza una entrada de 416×416. Debido a este motivo, es necesario volver a realizar recortes sobre las imágenes originales.

Tras realizar un gran número de experimentos (**ver Anexo C**) la configuración con la que se han obtenido mejor resultados se puede apreciar en la Tabla 5.10. En esta configuración se utiliza un BatchSize de 32 imágenes durante 40 epochs, siendo el tiempo de entrenamiento de aproximadamente de cuatro horas.

En el caso de esta implementación de SSD, se dispone como backbones Resnet50, Resnet101, Resnet34 y Resnet18. Tras los experimentos disponibles en el **Anexo C** se llega a la conclusión que para esta implementación y este conjunto el que mejor resultados ofrece es Resnet18.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1
Conjunto de test	Test1
Parámetros de la red	
InputSize	[300 300 3]
Número de Clases	6
FeatureExtractionNetwork	Resnet18
FeatureLayer	ReLU
Parámetros de entrenamiento	
Solucionador de red (solver network)	SGDM
Epochs	40
BatchSize	32
LearningRate	0.001
Data Augmentation	Sí
Momentum	0.9
L2	0.0005
Duración del entrenamiento	04:23:00

Tabla 5.10.- Configuración del mejor experimento - SSD - ADAAR

Los resultados obtenidos se pueden apreciar en la Figura 5.15. Al igual que en el resto de los experimentos, es necesario recordar que las clases FosaPurín y Silo tienen 7 y 3 instancias, respectivamente (ver Tabla 4.13).

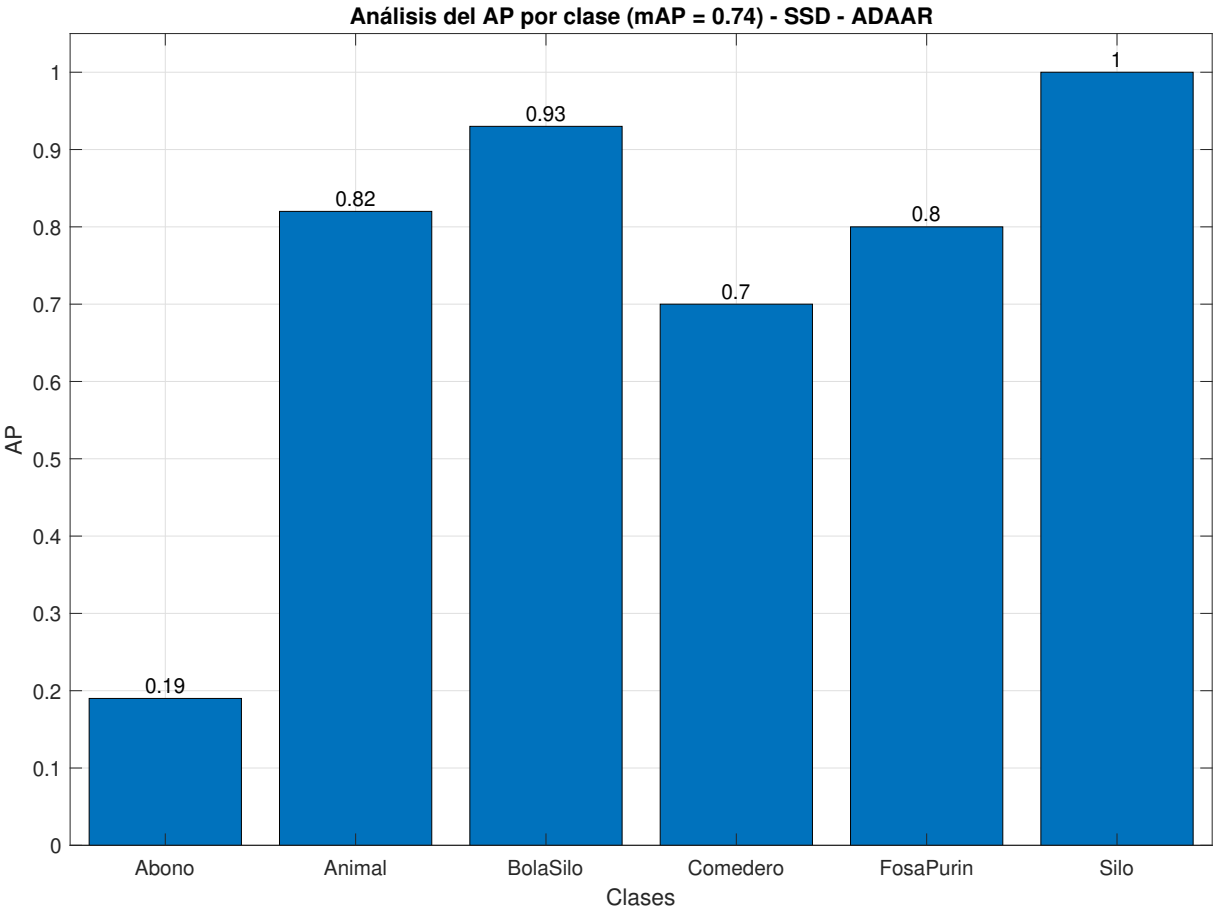


Figura 5.15.- Resultado del mejor experimento - SSD - ADAAR

Este experimento se corresponde con el experimento **E004-01** disponible en el Anexo C y ha sido el mejor de todos los experimentos realizados con SSD sobre el conjunto de datos ADAAR, obteniendo un mAP de 0.74.

5.4.5.- Comparativa de los resultados obtenidos con YOLOv2, YOLOv5, CustomVision y SSD sobre el conjunto de datos ADAAR

En la Figura 5.16 se puede apreciar como YOLOv5 tiene un mAP superior al resto de redes. Entre YOLOv2, CustomVision y SSD los resultados son muy similares. Como ya se ha explicado las clases FosaPurín y Silo tienen muy pocas instancias (ver Tabla 4.13), por lo que acertar o fallar en estas clases puede significar que el mAP aumente o disminuya considerablemente.

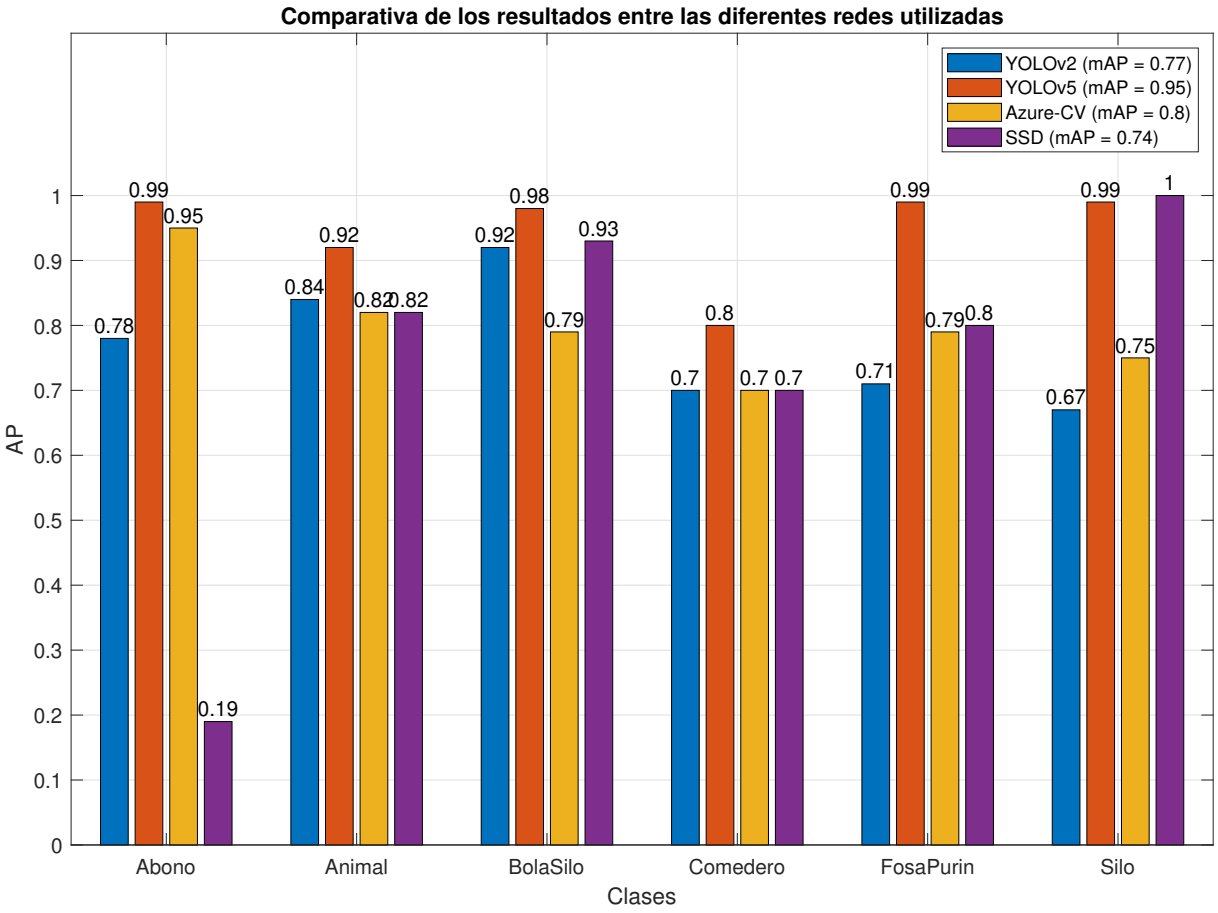


Figura 5.16.- Comparativa de resultados - ADAAR

En la Figura 5.17 (nótese que esta Figura ocupa varias páginas) se muestra el resultado de aplicar las diferentes redes sobre una serie de imágenes de test. La forma de leer la Figura 5.17 es la misma que la Figura 5.6 y la Figura 5.11, es decir en la primera fila (Real) se encuentra la imagen con las anotaciones del GroundTruth, en la segunda fila (YOLOv2) se encuentran las predicciones obtenidas por la red YOLOv2, en la tercera fila (YOLOv5) se encuentran las predicciones obtenidas por la red YOLOv5, y finalmente, en la última fila (CustomVision) se encuentran las predicciones obtenidas por CustomVision. **Todas las predicciones se han generado con una confianza mínima de 0.5.**

El formato de las anotaciones es idéntico al formato explicado en sección 5.3, donde se analizan los resultados del conjunto de datos DIOR.


Real
Yolov2
Yolov5
CustomVision
animal
bolaSilo
fosaPurin
Animal 0.89
Animal 0.90
Animal 0.91
Animal 0.92
Animal 0.93
Animal 0.91
BolaSilo 0.92
FosoPurin 0.93


Real
Yolov2
Yolov5
CustomVision
animal
silo
bolaSilo
Animal 0.91
Animal 0.92
Silo 0.87
BolaSilo 0.88

Figura 5.17.- Muestra de las detecciones realizadas - ADAAR

6. Implementación y despliegue de un servicio de detección de objetos

6.1.- Descripción

El objetivo final del proyecto es disponer un servicio para poder detectar actividad agrícola. Por este motivo se decide crear un **prototipo**. Considerando los resultados vistos en el capítulo 5 se ha decidido utilizar **YOLOv5** como red predictora de objetos, ya que presenta los mejores resultados para todos los conjuntos de datos.

A parte de realizar las detecciones sobre las imágenes seleccionadas, el único requisito es la generación de un archivo **.geojson** con cada una de las imágenes. Este archivo contendrá las coordenadas del polígono que indican donde se encuentra cada objeto detectado, la clase de dicho objeto (Abono, Animal, ...) y la confianza de dicha predicción.

En la Figura 6.1 se puede observar una imagen devuelta por el servicio implementado. Debido a que esta imagen es de $10\,000 \times 10\,000$ es necesario realizar zoom en una parte de la imagen. En la Figura 6.2 se puede apreciar cómo se han realizado las detecciones. En el fragmento de código 6.1 se encuentra una parte de la estructura del fichero .geojson asociado a imagen C8_UAV.tif.

Figura 6.1.- Imagen devuelta por el servicio implementado

Figura 6.2.- Zoom sobre la imagen devuelta por el servicio

Código 6.1.- Formato del .geojson asociado a cada imagen

```
{
    "type": "FeatureCollection",
    "name": "C8_UAV.tif",
    "_comment": "Los puntos de las BoundingBoxes se encuentran en el siguiente
        formato: x_izq y_sup ancho alto",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        [571178.65, 4772754.3],
                        [571180.65, 4772754.3],
                        [571180.65, 4772752.3],
                        [571178.65, 4772752.3],
                        [571178.65, 4772754.3]
                    ]
                ]
            },
            "properties": {
                "class": "Animal",
                "confidence": 0.94,
                "boundingBox": "586 3607 20 20"
            }
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "Polygon",
```

```
            "coordinates": [
                [
                    [571173.55, 4772765.2],
                    [571175.4500000001, 4772765.2],
                    [571175.4500000001, 4772763.2],
                    [571173.55, 4772763.2],
                    [571173.55, 4772765.2]
                ]
            ]
        },
        "properties": {
            "class": "Animal",
            "confidence": 0.94,
            "boundingBox": "535 3498 19 20"
        }
    }
  ]
}
```


Una vez accedido al servicio mediante un navegador se mostrará la interfaz de la Figura 6.3. En la interfaz se puede seleccionar el modelo que se desea utilizar (radiobuttons), la confianza mínima que deben tener las predicciones (slider) y las imágenes sobre las que se desea realizar las predicciones (botón "Elegir archivos"). Para cada uno de los conjuntos de datos utilizados se han seleccionado los modelos obtenidos en el capítulo 5. Adicionalmente, en el caso del conjunto Standford se ha creado un modelo solo para dos clases: Human (Humano) y Vehicle (Vehículo). Para ello se ha unificado las clases originales de la siguiente forma:

- Human: Biker, Pedestrian y Skater.
- Vehicle: Bus, Car y Cart.

Figura 6.3.- Interfaz del servicio implementado

Como se ha comentado anteriormente, el requisito establecido es la generación de un fichero .geojson asociado a cada imagen en la que se desee realizar la detección. Este .geojson tendrá unas coordenadas asociadas. Las redes determinan la localización

de los objetos mediante píxeles, por lo que para realizar la transformación píxel → coordenada, es necesario que las imágenes sobre las que se realizan las predicciones estén georreferenciadas. Este requisito solo se aplica a las imágenes del conjunto de datos ADAAR, ya que estas imágenes han sido proporcionadas por la empresa y este era su objetivo. Las imágenes DIOR y Standford, no se encuentran georreferenciadas ya que el objetivo de los creadores de estos conjuntos no era generar un .geojson.

Debido a esta casuística, se ha decidido que en el caso de utilizar los modelos de DIOR y Standford no se genere un fichero .geojson sino que se genere un fichero .txt con la información que proporcionaría la red de una manera predeterminada (con algún ajuste para que sea más sencillo su comprensión). Cada objeto detectado se presenta en una línea (del fichero .txt) diferente y está compuesto por la clase del objeto, la confianza de la predicción, la esquina superior izquierda de la BoundingBox predicha (x, y), el ancho y el alto. La esquina superior izquierda (x,y), el ancho y el alto se expresan en píxeles y se encuentran normalizados a las dimensiones de la imagen. En el fragmento de código 6.2 se puede apreciar las predicciones realizas sobre una imagen (de test) del conjunto DIOR. En ella se han predicho cinco aviones (airplane) con unas confianzas de 0.85, 0.85, 0.85, 0.86 y 0.87 respectivamente. En este caso las imágenes son de 416 × 416 por lo que las dimensiones de la BoundingBox se han dividido entre 416.

Código 6.2.- Formato original del predicciones

```
airplane 0.85 0.30889421701431 0.296875 0.10336538404226 0.10817307978868
airplane 0.85 0.832932710647 0.40264421701431 0.09855769574642 0.10817307978868
airplane 0.85 0.7115384340286 0.7932692170143 0.10096153616905 0.11057692021131
airplane 0.86 0.703125 0.4771634638309 0.09855769574642 0.10336538404226
airplane 0.87 0.4795673191547 0.6899038553237 0.09855769574642 0.09615384787321
```

Es importante destacar que el objetivo de la empresa no es utilizar imágenes de 416 × 416, sino que pretenden utilizar imágenes del rango 10 000 × 10 000. Este hecho implica que una imagen puede ocupar toda la memoria de la GPU y por este motivo no se pueda proporcionar el servicio. La solución establecida es que primero se intenta utilizar la GPU y realizar las predicciones, si es posible. En caso de que no lo sea, se procede a utilizar la CPU. Funcionalmente no existe diferencia entre utilizar la GPU y la CPU, la única diferencia es temporal.

6.2.- Despliegue

La gran mayoría de redes se encuentran escritas en Python, utilizando frameworks como Keras/Tensorflow o Pytorch. Debido a que se encuentran escritas en Python lo más sencillo es utilizar un framework para el desarrollo de aplicaciones web en Python. En este caso se ha optado por utilizar Flask, así como Javascript y Bootstrap 4.

Para acceder al servicio, se hace uso de peticiones REST como se puede ver en la Figura 6.4. En este ejemplo se utiliza la herramienta Postman (herramienta de desarrollo de APIs con capacidad de hacer peticiones HTTP [45]). En esta petición se solicita al servicio las predicciones sobre la imagen 00011.jpg que tengan una confianza mínima de 0.6, utilizando el modelo generado para el conjunto DIOR.

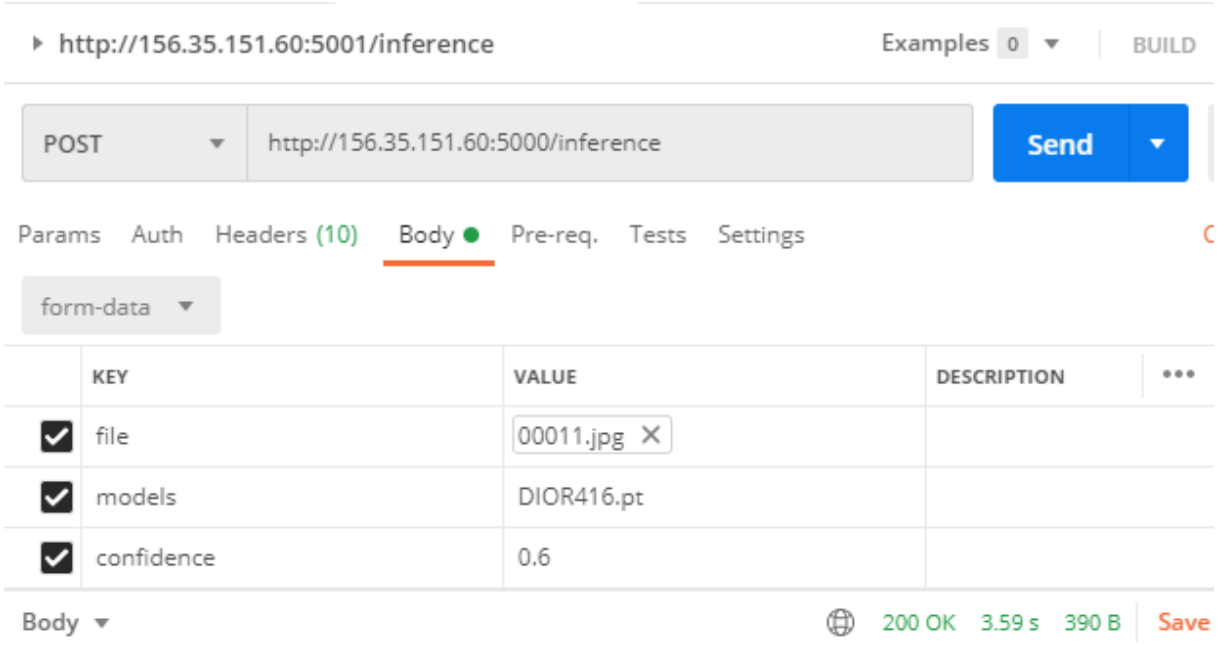


Figura 6.4.- Petición REST al servicio mediante Postman

Finalmente, una vez comprobado que todo funciona correctamente se ha Dockerizado el servicio con el fin de facilitar futuros despliegues. En los fragmentos de código 6.3, 6.4 y 6.5 se puede observar el contenido del Dockerfile, del requirements.txt y del docker.compse.yml, respectivamente. Gracias a esta Dockerización es posible realizar el despliegue del servicio mediante un solo comando: docker-compose up. Evidentemente, **es necesario que previamente se haya instalado docker** (y **nvidia-docker2** si se desea utilizar la GPU durante la inferencia).

Código 6.3.- Contenido del Dockerfile

```
FROM nvidia/cuda:11.0-base

RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y
    --no-install-recommends \
  tzdata \
  python3.8 \
  python3-pip \
  gdal-bin \
  libgdal-dev \
  python3-gdal \
  'ffmpeg' \
  'libsm6' \
  'libxext6' -y

ADD . /code
WORKDIR /code

RUN pip3 install -r requirements.txt
CMD ["python3", "app.py"]
```

Código 6.4.- Contenido del requirements.txt

```
Cython==0.29.21
matplotlib==3.3.1
numpy==1.19.1
opencv-python==4.4.0.44
pillow==7.0.0
PyYAML==5.3.1
scipy==1.5.2
torch==1.6.0
torchvision==0.7.0
tqdm==4.48.0
flask==1.1.1
geojson-rewind==0.2.0
```

Código 6.5.- Contenido del docker.compose.yml

```
version: '2.3'
services:
  web:
    runtime: nvidia # Utilizar solo en caso de disponer de una GPU y querer utilizarla
    build: .
    ports:
     - "5000:5000"
```

6.3.- Análisis de diferentes infraestructuras para desplegar el servicio

Una vez creado el servicio es necesario seleccionar la infraestructura de despliegue. Para poder tomar la mejor decisión es necesario analizar el Sistema Bajo Test (SUT) mostrado en la Figura 6.5.

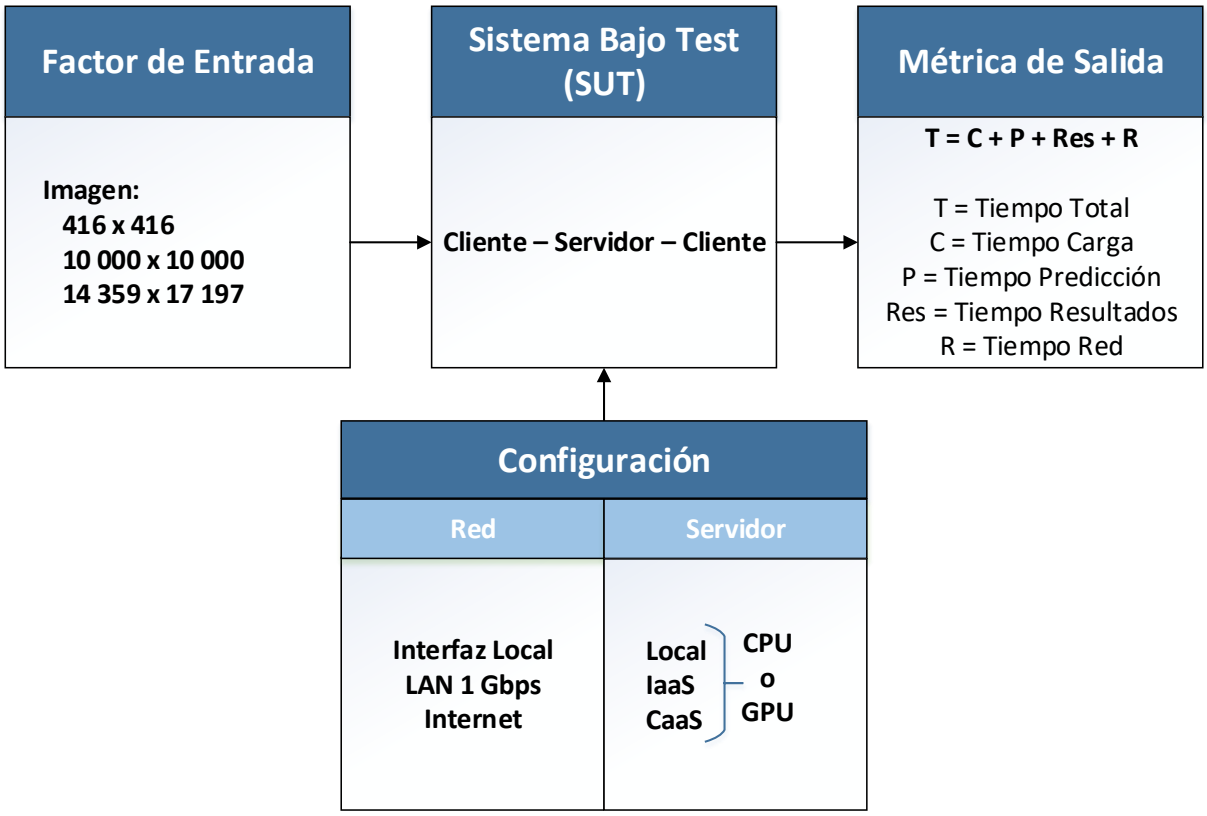


Figura 6.5.- Sistema Bajo Test - Implementación del servicio

El Sistema Bajo Test (SUT) se evalúa con tres niveles de factor de entrada: imagen de 416×416, imagen de $10\,000 \times 10\,000$ e imagen de $14\,359 \times 17\,197$. El cliente siempre será la máquina 2 del laboratorio (ver sección 7.1), mientras que el servidor requiere una configuración específica. Por tanto, en la configuración del sistema se consideran dos factores: red y servidor.

- **Red**: los experimentos se realizan sobre una interfaz local, es decir utilizando la misma máquina como cliente y servidor (A-A), utilizando una LAN de 1 Gbps (A-B) o mediante internet. Para comprobar la velocidad real entre el cliente y el servidor se ha utilizado el programa **iperf**, obteniendo los valores mostrados en la Tabla 6.1.

Cliente	Red	Servidor	Velocidad
Máquina 2	Localhost	Máquina 2	63.7 Gbps
Máquina 2	LAN 1000 Mbps	Máquina 1	938 Mbps
Máquina 2	LAN 1000 Mbps - Internet	Azure-WE IaaS: NC6	131 Mbps

Tabla 6.1.- Velocidad entre el cliente y el servidor

- **Servidor**: para poder evaluar correctamente el despliegue más eficiente es necesario disponer de varias infraestructuras:
  - **Local** → Máquina 2: Intel Core i7-9700K, 64 GB de RAM y NVIDIA GeForce 2080 Ti.
  - **IaaS** → Standard NC6: 6 vCPU, 56 GB de RAM y NVIDIA K80.
  - **CaaS**: 1-4 vCPU y 16 GB RAM.

Una vez configurado el servidor, es necesario disponer de una métrica para poder comparar las distintas implementaciones. La métrica seleccionada es el **tiempo total** desde que el cliente realiza una petición hasta que obtiene la respuesta. El tiempo total se descompone en:

- **Tiempo de Carga**: antes de realizar las predicciones necesarias, es necesario cargar el modelo creado, así como cargar la imagen en memoria, acceder a la GPU, etc. Este tiempo se denomina tiempo de carga.
- **Tiempo de Predicción**: tiempo necesario para generar las predicciones sobre la imagen de entrada.
- **Tiempo de Resultados**: tiempo necesario para procesar los resultados de la imagen y generar una salida previamente establecida, en este caso la generación del archivo geojson.
- **Tiempo de Red**: dado que la imagen se transmite por red es necesario tener en cuenta este tiempo. Para calcularlo se utiliza la siguiente fórmula:

$$T.\,Red = T.\,Total - T.\,Carga - T.\,Predicción - T.\,Resultados$$

Para comparar las configuraciones se utiliza la métrica Latency Ratio. Esta métrica se calcula de la siguiente forma:

$$Latency\ Ratio = \frac{Tiempo\,Total\ del\ Experimento}{Tiempo\,Total\ del\ mejor\ Experimento}$$

Es decir, si un experimento tiene un Latency Ratio de 1.2 y otro de 2.3, el primer experimento es más rápido que el segundo. Para realizar esta evaluación se realizan una serie de peticiones http al servicio. Para evitar posibles experimentos atípicos, **en cada configuración se realizan 10 peticiones**, con las mismas características, y se **promedian** los tiempos.

6.3.1.- Evaluación del sistema con imágenes de 416 x 416

En la Tablas 6.2 y 6.7 se muestran los resultados de evaluar el servicio bajo diferentes configuraciones: sobre una imagen de 416 × 416 y sobre un lote (batch) de 100 imágenes de las mismas dimensiones.

ID	Servidor	Red (Mbps)	CPU o GPU	T. Carga (seg)	T. Predicción (seg)	T. Resultados (seg)	T. Red (seg)	T. Total (seg)	Latency Ratio	Coste / hora
1	Local: A-A	Local	GPU	2.49	0.01	0	0.02	2.52	2.45	0.08 €
2	Local: A-A	Local	CPU	0.93	0.08	0	0.04	1.05	1.01	0.06 €
3	Local: A-B	1000	GPU	2.44	0.01	0	0.05	2.50	2.43	0.08 €
4	Local: A-B	1000	CPU	0.93	0.08	0	0.21	1.22	1.18	0.06 €
5	IaaS: NC6	1000	GPU	3.27	0.05	0	0.25	3.57	3.46	0.98 €
6	IaaS: NC6	1000	CPU	1.54	0.13	0	0.24	1.91	1.85	0.98 €
7	CaaS: 1 vCPU	1000	CPU	2.05	0.54	0	0.34	2.93	2.85	0.11 €
8	CaaS: 2 vCPU	1000	CPU	1.71	0.29	0	0.33	2.32	2.25	0.15 €
9	CaaS: 3 vCPU	1000	CPU	1.73	0.32	0	0.33	2.37	2.30	0.19 €
10	CaaS: 4 vCPU	1000	CPU	1.82	0.33	0	0.32	2.48	2.41	0.23 €

Tabla 6.2.- Resultados de la evaluación del SUT con una imagen de 416 × 416

ID	Servidor	Red (Mbps)	CPU o GPU	T. Carga (seg)	T. Predicción (seg)	T. Resultados (seg)	T. Red (seg)	T. Total (seg)	Latency Ratio	Coste / hora
11	Local: A-A	Local	GPU	2.85	0.90	0	0.27	4.02	1.01	0.08 €
12	Local: A-A	Local	CPU	0.95	7.51	0	0.28	8.74	2.19	0.06 €
13	Local: A-B	1 000	GPU	2.82	0.84	0	0.33	3.99	1.00	0.08 €
14	Local: A-B	1 000	CPU	0.88	7.49	0	0.34	8.70	2.18	0.06 €
15	IaaS: NC6	1 000	GPU	3.97	3.89	0	2.62	10.48	2.63	0.98 €
16	IaaS: NC6	1 000	CPU	1.56	13.70	0	2.44	17.70	4.44	0.98 €
17	CaaS: 1 vCPU	1 000	CPU	3.99	58.02	0	2.58	64.59	16.20	0.11 €
18	CaaS: 2 vCPU	1 000	CPU	1.16	32.02	0	2.32	35.50	8.90	0.15 €
19	CaaS: 3 vCPU	1 000	CPU	0.73	30.68	0	2.32	33.73	8.46	0.19 €
20	CaaS: 4 vCPU	1 000	CPU	1.24	32.54	0	2.87	36.65	9.19	0.23 €

Tabla 6.3.- Resultados de la evaluación del SUT con cien imágenes de 416 × 416

Para realizar esta evaluación se han utilizado imágenes del conjunto de datos DIOR, por lo que, al no estar estas imágenes georreferenciadas no se puede generar el archivo geojson, es decir el tiempo de resultados siempre es cero.

A continuación, en la Figuras 6.6 y 6.7 se pueden analizar mejor los resultados obtenidos. Como ya se ha mencionado el cliente siempre se encuentra en el laboratorio. Para cada uno de los experimentos se detalla el tiempo de carga, de predicción, de obtención de resultados y de red. Todos estos tiempos apilados suman el tiempo total.

Además, en la parte superior de cada una de las barras se muestra el Latency Ratio correspondiente (recordar que el experimento más rápido tiene una Latency Ratio de 1x).

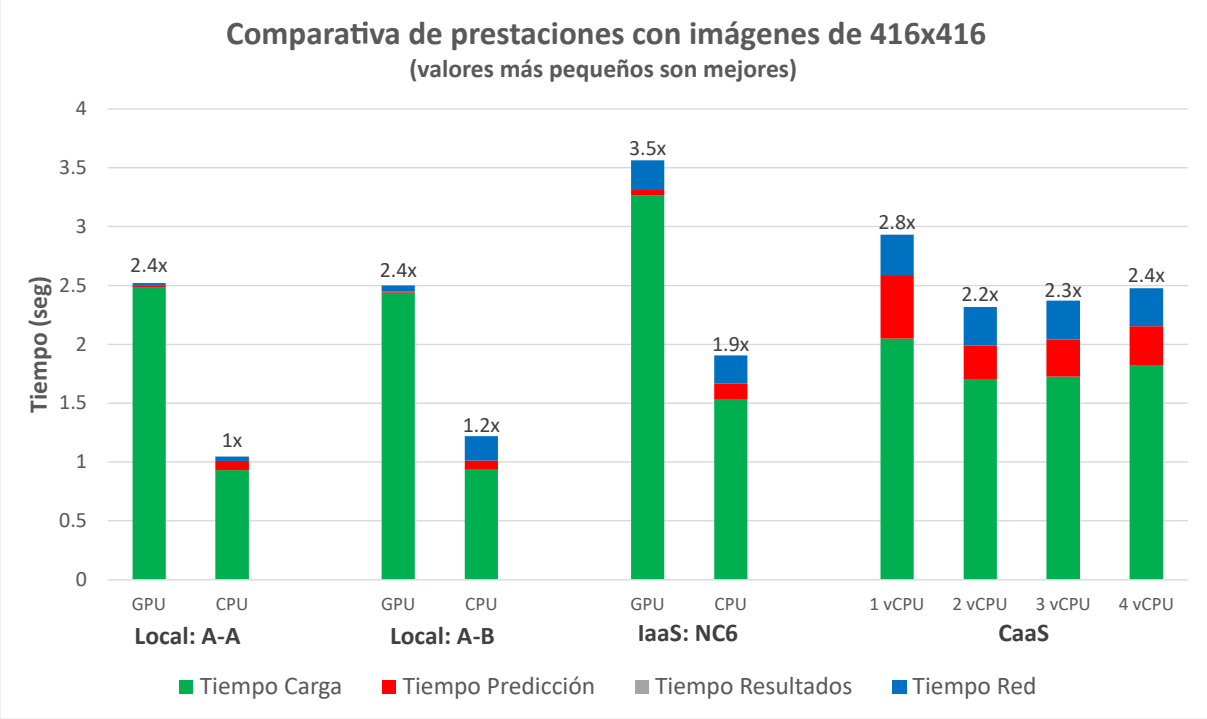


Figura 6.6.- Comparativa de prestaciones del SUT con una imagen de 416 × 416

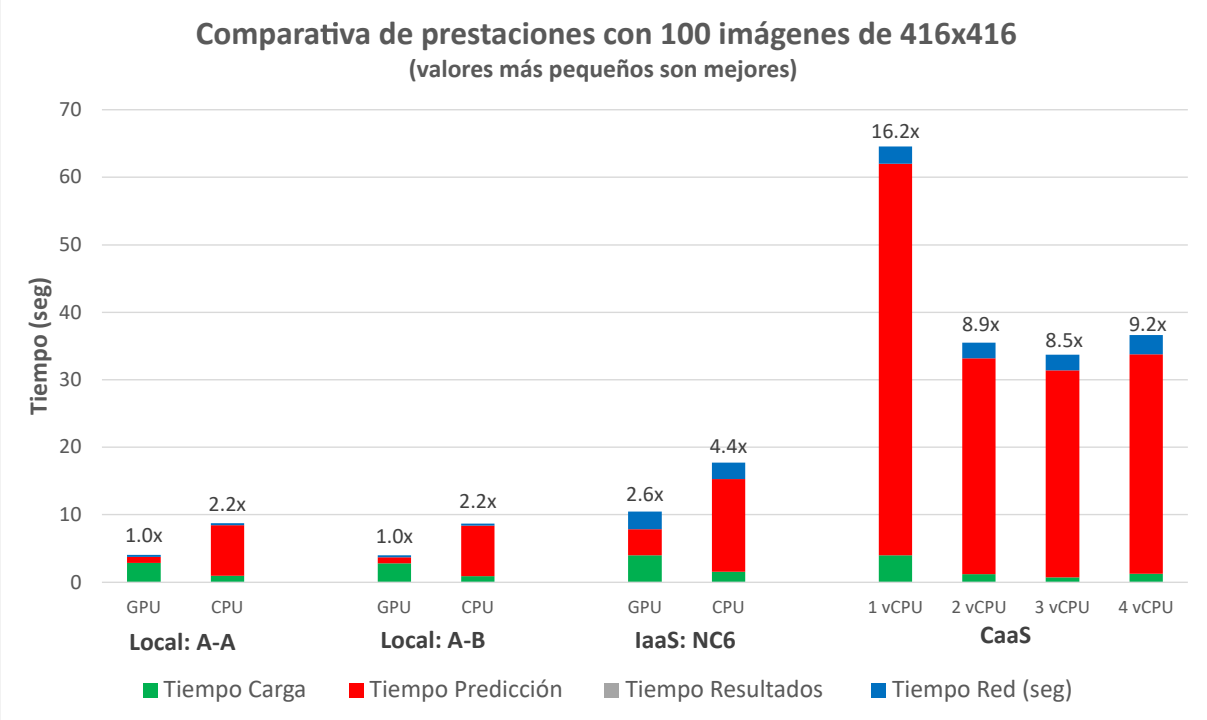


Figura 6.7.- Comparativa de prestaciones del SUT con cien imágenes de 416 × 416

El tiempo de red es despreciable en cada uno de los experimentos, como se puede apreciar en las Tablas 6.2 y 6.3 y en las Figuras 6.6 y 6.7. Esto se debe a que la imagen al ser de 416×416 pesa 34 KB. En tiempos de predicción influye el uso de CPU o GPU. Este hecho no encaja con el tiempo total en el caso de utilizar una sola imagen (Tabla 6.2 y Figura 6.6), ya que los tiempos totales con GPU son superiores a los de CPU. Esto se debe al tiempo de carga, ya que el de la GPU es muy superior al de la CPU.

Si en lugar de procesar las imágenes individualmente, se procesan por lotes (Tabla 6.7 y Figura 6.7), los tiempos se comportan de forma diferente, ya que, como el modelo solo se carga una vez, y la GPU es más rápida que la CPU, al utilizar cien imágenes el tiempo de carga tiene menos influencia en el tiempo total. En este caso, el tiempo más relevante es el de predicción, por lo que utilizar la GPU acelera el servicio.

En ambos casos, procesar una imagen o cien, la configuración más rápida es en la que se utiliza el servidor disponible en el laboratorio (Local A-A y Local A-B). Esto se debe a que la máquina utilizada en el laboratorio (máquina 2, ver sección 7.1) presenta unas mejores prestaciones que las infraestructuras disponibles en Azure (al menos con la subscripción adquirida). Además, hay que tener en cuenta que el ancho de banda en el laboratorio es muy superior al ancho de banda que hay desde el cliente (laboratorio) hasta el servidor en Azure (en este caso el servidor se aloja en los Países Bajos) (ver Tabla 6.1).

En la última configuración, se despliega el servidor en un entorno CaaS. Esta infraestructura, no es compatible con la imagen Docker mostrada en la sección anterior, debido a que no soporta CUDA 11. Por este motivo, se ha decidido realizar la evaluación utilizando solo la CPU. Como se puede apreciar, en las Figuras 6.6 y 6.7 los resultados son claramente inferiores a los obtenidos con el IaaS. En este caso, se utiliza la instancia NC6, que dispone de seis vCPU, mientras que en el CaaS se dispone de hasta cuatro.

6.3.2.- Evaluación del sistema con imágenes de 10 000 x 10 000

En la Tabla 6.4 y en la Figura 6.8 se muestran los resultados de evaluar el servicio bajo diferentes configuraciones sobre una imagen de 10 000 × 10 000. Los resultados son muy diferentes a los obtenidos en los experimentos con imágenes de 416 × 416.

ID	Servidor	Red (Mbps)	CPU o GPU	T. Carga (seg)	T. Predicción (seg)	T. Resultados (seg)	T. Red (seg)	T. Total (seg)	Latency Ratio	Coste / hora
21	Local: A-A	Local	GPU	7.26	0.40	1.00	13.03	21.69	1.0	0.08 €
22	Local: A-A	Local	CPU	5.53	28.06	1.00	13.10	47.69	2.2	0.06 €
23	Local: A-B	1000	GPU	7.29	0.40	1.00	17.45	26.14	1.2	0.08 €
24	Local: A-B	1000	CPU	5.54	27.80	1.00	17.46	51.80	2.4	0.06 €
25	IaaS: NC6	1000	GPU	15.07	4.40	1.85	61.52	82.84	3.8	0.98 €
26	IaaS: NC6	1000	CPU	9.04	33.89	1.88	53.52	98.34	4.5	0.98 €
27	CaaS: 1 vCPU	1000	CPU	10.47	134.31	1.93	49.37	196.08	9.1	0.11 €
28	CaaS: 2 vCPU	1000	CPU	10.01	75.89	2.00	55.63	143.52	6.6	0.15 €
29	CaaS: 3 vCPU	1000	CPU	10.03	82.82	1.99	56.86	151.70	7.0	0.19 €
30	CaaS: 4 vCPU	1000	CPU	10.84	86.81	2.10	53.57	153.32	7.1	0.23 €

Tabla 6.4.- Resultados de la evaluación del SUT con una imagen de 10 000 × 10 000

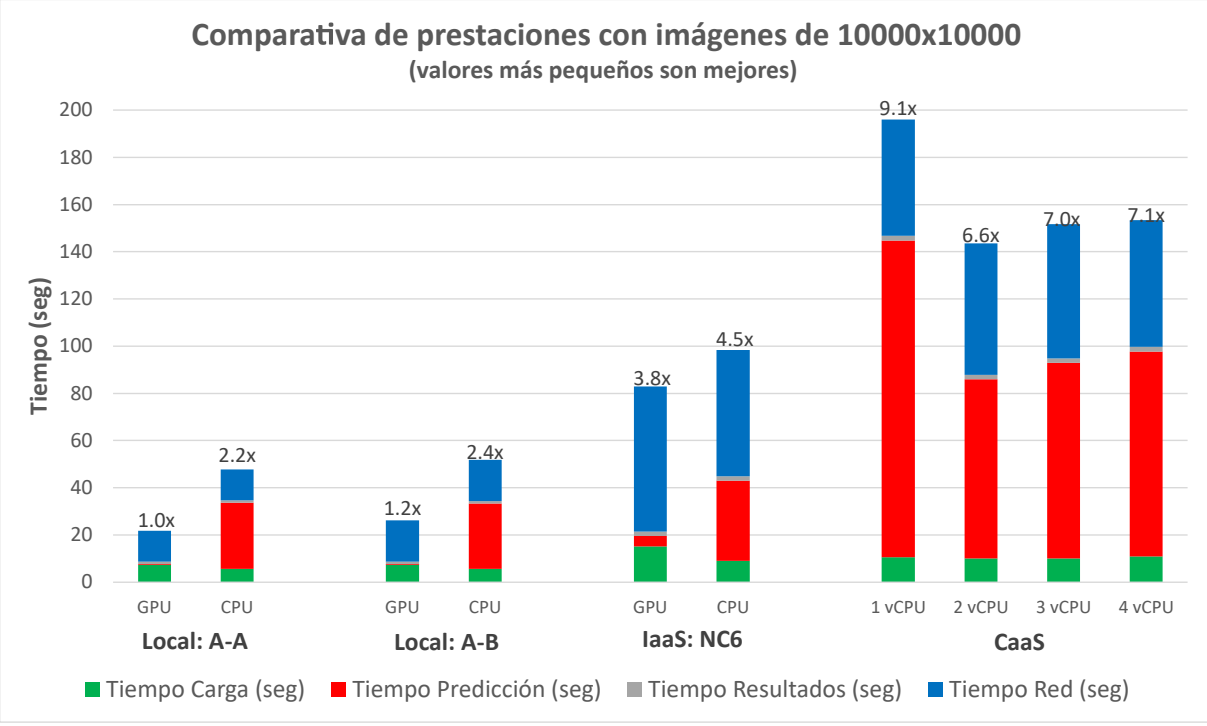


Figura 6.8.- Comparativa de prestaciones del SUT con una imagen de 10 000 × 10 000

En este caso, tal y como se puede apreciar en la Figura 6.8, los tiempos de carga no son tan relevantes. Como es evidente con estas imágenes tan grandes (aproximadamente

400 MB) los tiempos más influyentes son los de predicción y red. Al ser estos tiempos los más relevantes, a diferencia que con imágenes de 416×416, **el uso de GPU o de CPU es más influyente**.

Es interesante destacar que **los tiempos de red varían mucho** dependiendo de la velocidad entre el cliente y el servidor (ver Tabla 6.1). Por ejemplo, los experimentos 23 y 25 son equivalentes, pero en el segundo la velocidad entre el cliente y el servidor es de 131 Mbps, mientras que en el primero es de 1 Gbps. Como se puede apreciar en la Figura 6.8, el tiempo de red (barra azul) del tercer experimento (experimento 23 en la Tabla 6.4) es muy inferior al del quinto experimento (experimento 26 en la Tabla 6.4).

6.3.3.- Evaluación del sistema con imágenes de 14 539 x 17 197

Por último, se analizan en la Tabla 6.5 y en la Figura 6.9 los resultados de evaluar el servicio sobre una imagen de 14 359 × 17 197.

ID	Servidor	Red (Mbps)	CPU o GPU	T. Carga (seg)	T. Predicción (seg)	T. Resultados (seg)	T. Red (seg)	T. Total (seg)	Latency Ratio	Coste / hora
31	Local: A-A	Local	GPU	-	-	-	-	-	-	0.08 €
32	Local: A-A	Local	CPU	13.92	67.99	5.24	36.19	123.35	1.0	0.06 €
33	Local: A-B	1000	GPU	-	-	-	-	-	-	0.08 €
34	Local: A-B	1000	CPU	13.89	67.47	5.23	40.04	126.63	1.0	0.06 €
35	IaaS: NC6	1000	GPU	-	-	-	-	-	-	0.98 €
36	IaaS: NC6	1000	CPU	22.38	82.13	8.16	133.25	245.92	1.9	0.98 €
37	CaaS: 1 vCPU	1000	CPU	-	-	-	-	-	-	0.11 €
38	CaaS: 2 vCPU	1000	CPU	-	-	-	-	-	-	0.15 €
39	CaaS: 3 vCPU	1000	CPU	-	-	-	-	-	-	0.19 €
40	CaaS: 4 vCPU	1000	CPU	-	-	-	-	-	-	0.23 €

Tabla 6.5.- Resultados de la evaluación del SUT con una imagen de 14 359 × 17 197

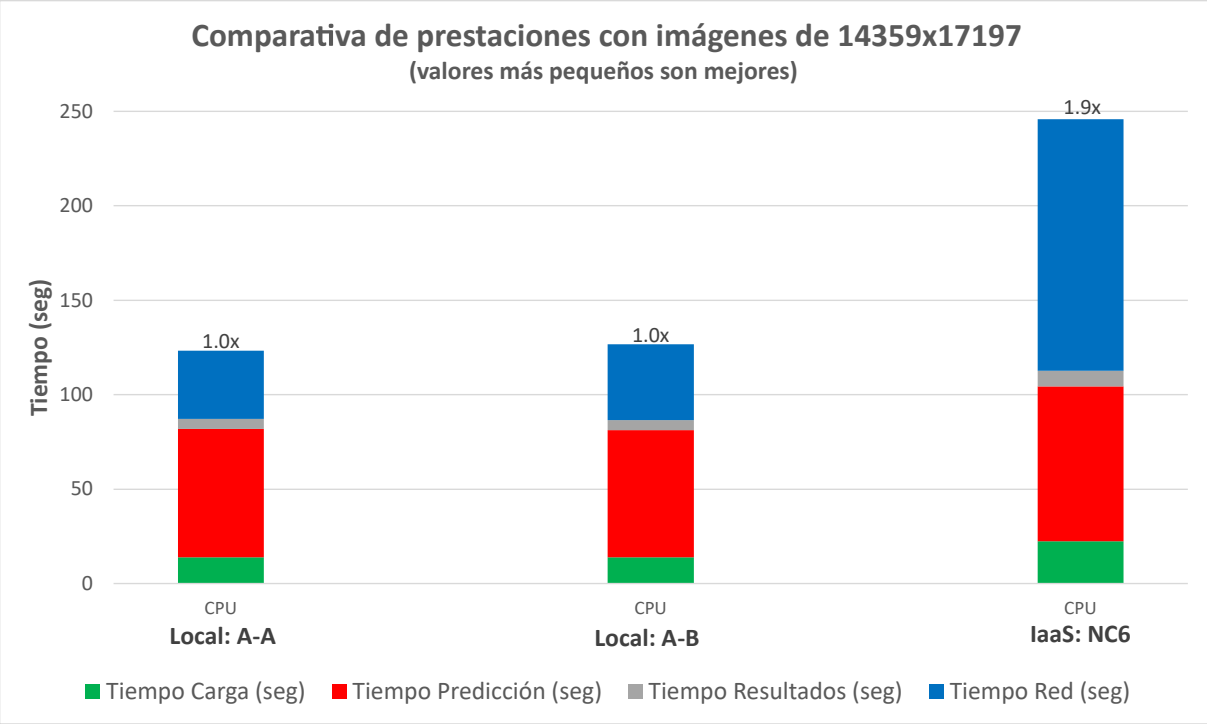


Figura 6.9.- Comparativa de prestaciones del SUT con una imagen de 14 359 × 17 197

En este caso al ser la imagen demasiado grande no se puede utilizar la GPU debido a que la imagen no entra en memoria, por lo que se han realizado pruebas con la CPU.

Tampoco se puede evaluar con una infraestructura CaaS ya que la memoria máxima es de 16 GB.

Los resultados son similares los obtenidos con imágenes de $10\,000 \times 10\,000$. Lo más destacado es que en el caso del IaaS, es más influyente el tiempo de red que el de predicción. Esto se debe a que cuanto más grande sea la imagen a procesar más influye el tiempo de red.

Tanto en el caso de anterior, donde se procesa una imagen de $10\,000 \times 10\,000$, como en este, el tiempo de resultados (generación del archivo geojson) no es relevante, ya que no representa un porcentaje muy alto del tiempo total.

6.3.4.- Comparativa de las prestaciones con respecto al coste

En las Figuras 6.10, 6.11, 6.12 y 6.13 se muestra una comparativa de las prestaciones (tiempo total) de cada una de las infraestructuras probadas con respecto al coste por hora de dicha infraestructura.

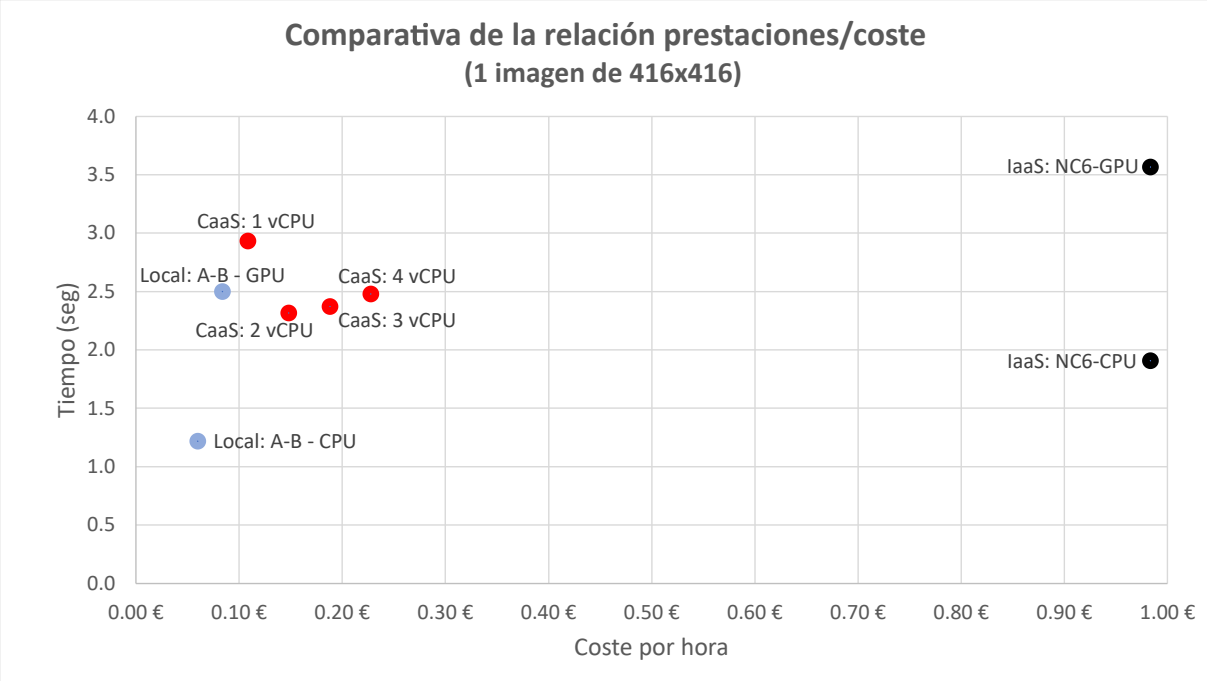


Figura 6.10.- Comparativa de las prestaciones con respecto al coste utilizando una imagen de 416×416

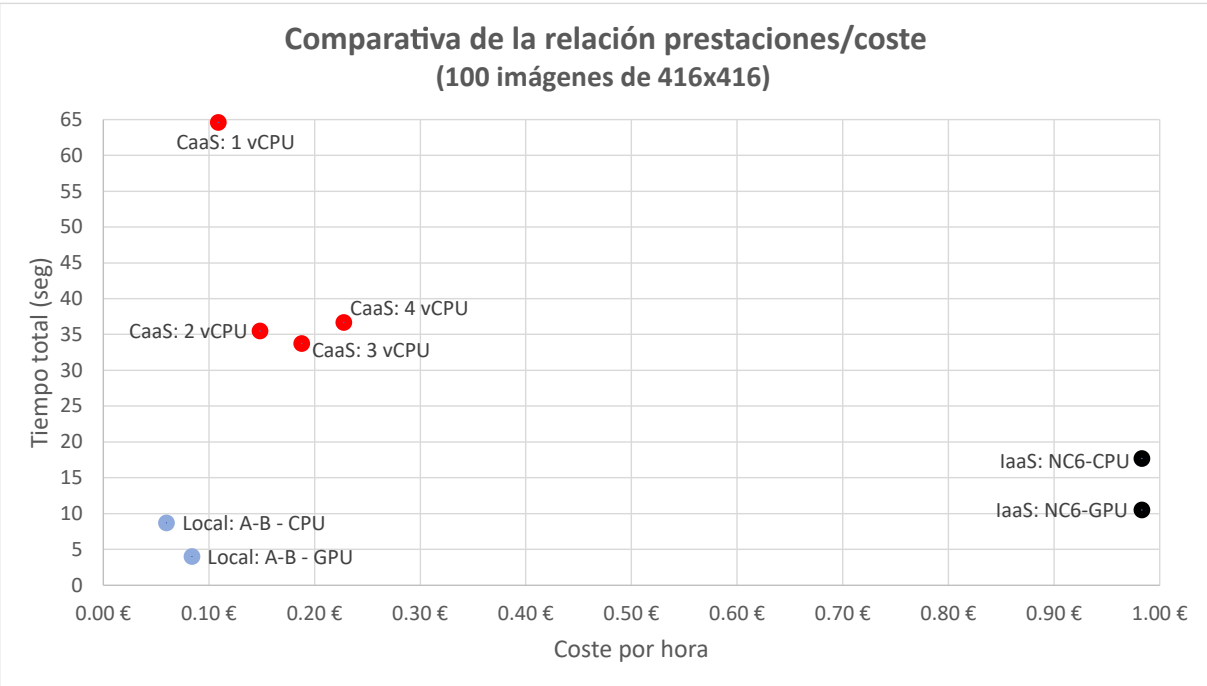


Figura 6.11.- Comparativa de las prestaciones con respecto al coste utilizando cien imágenes de 416×416

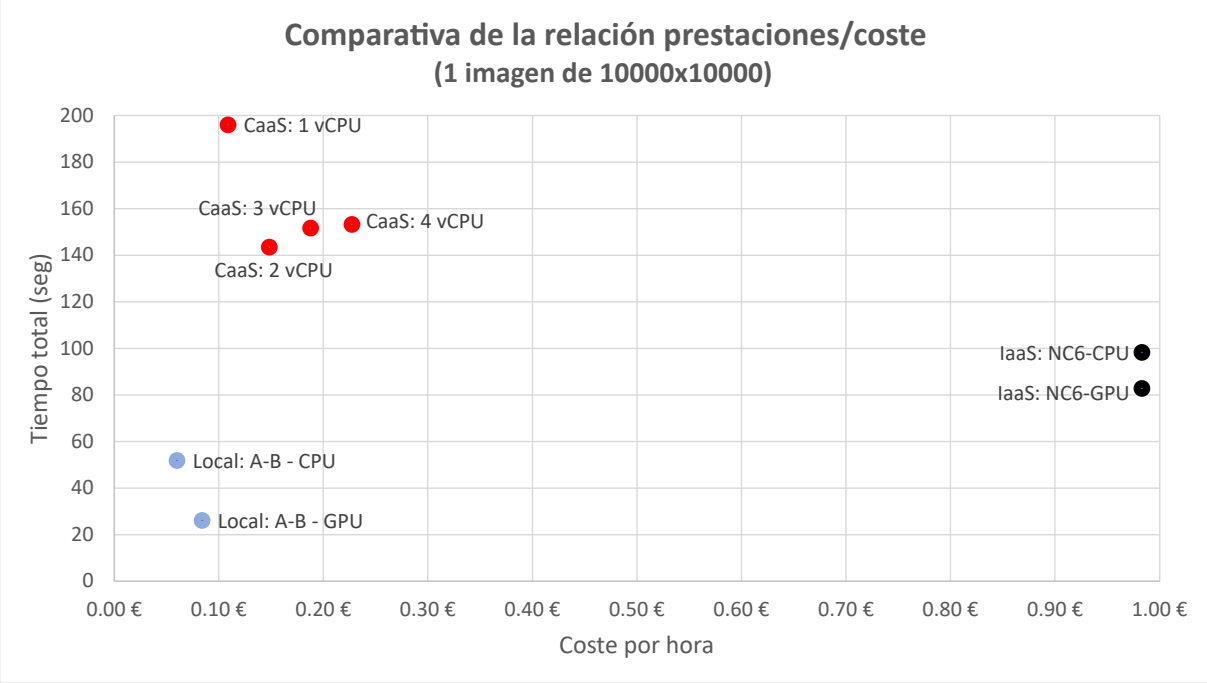


Figura 6.12.- Comparativa de las prestaciones con respecto al coste utilizando una imagen de $10\,000 \times 10\,000$

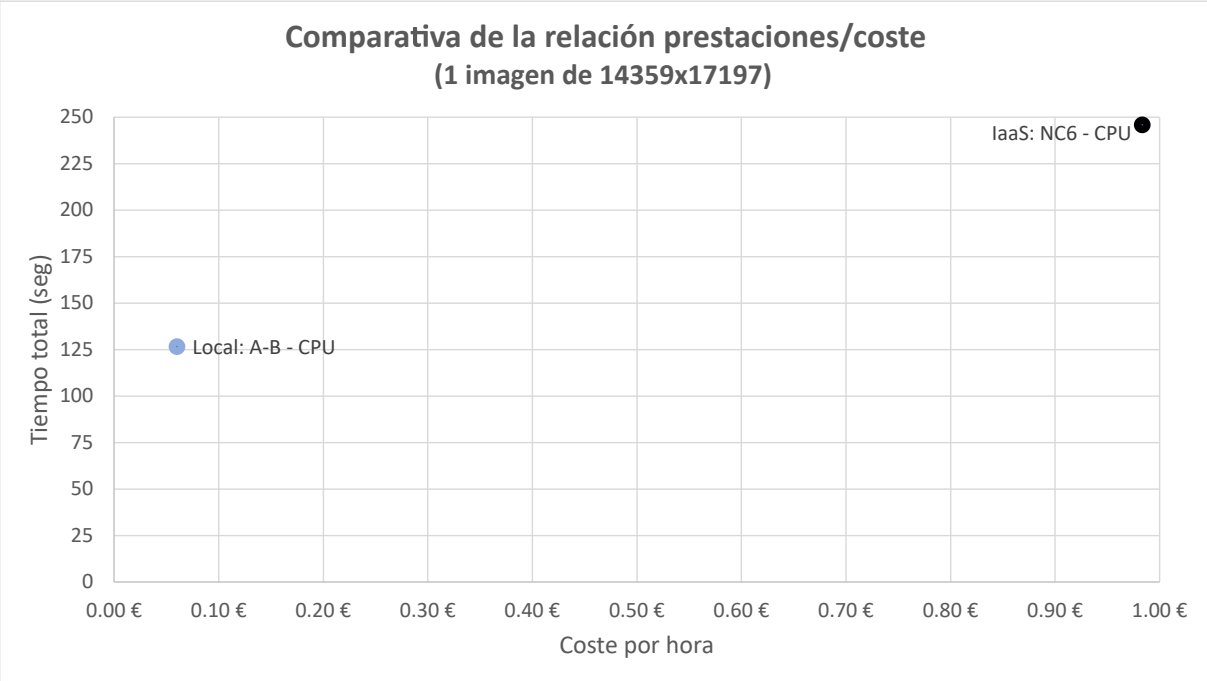


Figura 6.13.- Comparativa de las prestaciones con respecto al coste utilizando una imagen de $14\,359 \times 17\,197$

Para el cálculo de los costes se supone que el servicio se encuentra realizando peticiones continuamente durante una hora.

Cabe destacar que el coste de la infraestructura **local** se calcula de la siguiente forma:

$$\text{Coste total por hora} = \text{Inversión inicial por hora} + \text{Consumo eléctrico por hora}$$

$$\text{Inversión Inicial por hora con GPU (amortizada a 5 años)} = \frac{3\,219.66€}{5 \text{ años} \times 365 \text{ días} \times 24 \text{ horas}} = 0.07€/h$$

$$\text{Inversión Inicial por hora con CPU (amortizada a 5 años)} = \frac{3\,219.66€ - 1\,199.00€}{5 \text{ años} \times 365 \text{ días} \times 24 \text{ horas}} = 0.05€/h$$

$$\text{Consumo eléctrico por hora} = \text{Consumo de la máquina} \times \text{precio/kW h} = 0.074kW \times 0.14€/\text{kW h} = 0.01€/h$$

$$\text{Coste total por hora con GPU} = 0.07€/h + 0.01€/h = 0.08€/h$$

$$\text{Coste total por hora con CPU} = 0.05€/h + 0.01€/h = 0.06€/h$$

En las Figuras 6.10, 6.11, 6.12 y 6.13, cuanto más cerca del eje X se encuentre un punto mejores prestaciones ofrecerá dicha infraestructura, mientras que cuanto más cerca del eje Y menor será el coste. En todas las configuraciones la infraestructura que ofrece mejores prestaciones y un menor coste es la local, a excepción de la local con GPU al utilizar una imagen de 416×416 (ver Figura 6.10). Como ya se comentó esto se debe al tiempo de carga de la GPU. A continuación, por prestaciones la siguiente mejor configuración es la IaaS. Hay que tener en cuenta que también es la infraestructura más cara de todas. Por último, se encuentran las configuraciones que utilizan CaaS, que aunque ofrecen peores prestaciones suponen un coste inferior al del IaaS. Es necesario destacar, que aunque existe diferencia de prestaciones entre una infraestructura IaaS y CaaS esta no es tan grande, como se podría pensar. Esto se debe principalmente a que al utilizar imágenes grandes los tiempos de red influyen mucho sobre el tiempo total.

7. Hardware utilizado

Para poder desarrollar este proyecto se han adquirido dos máquinas, una con Sistema Operativo Windows y otra Linux, además de una subscripción a Azure. A continuación se especifican los componentes de las máquinas adquiridas.

7.1.- Máquinas adquiridas

En la Tabla 7.1 se detallan los componentes de las máquinas adquiridas. Los componentes más destacados son el procesador i7 9700K, los 64 GB de RAM y la GPU NVIDIA Geforce 2080 Ti.

Se han adquirido dos máquinas con los componentes mostrados en la Tabla 7.1. A la **Máquina 1** se le ha instalado **Windows 10 Pro**, mientras que a la **Máquina 2** se le ha instalado **Ubuntu 20.04**.

El coste de cada una de las máquinas asciende a 3 219.66€.

Componente	Modelo	Unidades
Procesador	Intel Core i7 9700K	1
Placa Base	Gigabyte Aorus Z390	1
RAM	16 GB DDR4 3000 Vengeance Corsair	4
Disco SSD	1TB SSD NVMe M.2 S970 EVO PLUS Samsung	1
Disco SSD	4 TB SSD S860 EVO Samsung	1
Disco HDD	Western Digital GOLD de 14 TB	1
GPU	GeForce RTX 2080 Ti TURBO 11GB	1
Caja	ATX F200 ELEMENTS COOLBOX	1
Ventilador	Plannar 120 Coolbox	1
Fuente de alimentación	Corsair HX1200 1200W Platinum	1
Unidad de disco óptico	DVD RW 24D5MT SATA ASUS	1
Ratón	Logitech M110 Silent	1
Teclado	Logitech K120	1
Monitor	Philips 272B8QJEB	1

Tabla 7.1.- Desglose de los componentes utilizados en las máquinas adquiridas

7.2.- Azure

En las Tablas 7.2 y 7.3 se recogen los costes computables a la utilización de Azure CustomVision y al despliegue de las instancias de máquinas virtuales utilizadas. En CustomVision se computan tres costes:

- **Almacenamiento**: para poder entrenar un modelo es necesario cargar las imágenes del conjunto de entrenamiento. El coste que supone es de 0.59€ por 1 000 imágenes. Este coste se calcula al mes.
- **Entrenamiento**: el coste por hora de entrenamiento es de 16.87€.
- **Detección**: el coste por realizar predicciones sobre 1 000 imágenes es de 1.69€.

A la hora de crear máquinas virtuales los costes se computan de forma distinta. Para poder almacenar datos es necesario disponer de una cuenta de almacenamiento. El coste mensual de 10 000 imágenes es de 0.05€. Una cuenta de almacenamiento debe de estar almacenada en disco. En este caso se ha optado por un disco estándar HDD S6. El coste mensual de este disco es de 2.54€. Las máquinas virtuales vienen predefinidas en lo que se denominan instancias. Una instancia no es más que un conjunto de componentes, por ejemplo cantidad de núcleos, memoria, etc. En este proyecto se utilizan dos tipos instancias: NC6 y NC12. Sus componentes y sus costes se recogen en la Tabla 7.3. Por último, cabe destacar que el sistema operativo de la máquina virtual se debe de instalar en un disco aparte. En este caso se ha optado por un SSD P6, cuyo precio mensual es de 9.47€.

Concepto	Descripción	Coste por unidad
Almacenamiento	Coste por 1 000 imágenes/mes	0.59 €
Entrenamiento	Coste por hora	16.87 €
Detección	Coste por 1 000 imágenes	1.69 €

Tabla 7.2.- Coste de CustomVision - Azure

Concepto	Descripción	Coste por unidad
Cuenta de Almacenamiento	Coste por 10 000 imágenes/mes	0.05 €
Estándar HDD S6	Coste mensual por disco (hasta 64 GiB)	2.54 €
Premium SDD P6	Coste mensual por disco (hasta 64 GiB)	9.47 €
Máquina GPU NC6	Coste por hora (por tenerla encendida). 6 núcleos, 56 GiB RAM, NVIDIA K80	0.98 €
Máquina GPU NC12	Coste por hora (por tenerla encendida). 12 núcleos, 112 GiB RAM, 2 x NVIDIA K80	1.97 €

Tabla 7.3.- Coste de las MV - Azure

8. Conclusiones

Durante el desarrollo de este trabajo se han analizado diferentes Redes Neuronales Convolucionales especializadas en la detección de objetos, así como distintos conjuntos de datos. Todo ello con un objetivo: **reconocimiento de actividad agrícola mediante el uso de imágenes aéreas**.

Tras haber realizado este análisis, mediante una serie de pruebas experimentales, se llega a la conclusión de que la mejor red para poder implementar un servicio de reconocimiento de actividad agrícola es **YOLOv5**, entrenando el modelo con el conjunto de datos **ADAAR**. En este caso el mAP conseguido es de 0.95, llegando en algunas clases 0.99. Esto implica que las detecciones son muy precisas y por lo tanto muy fiables.

Es importante destacar que el conjunto ADAAR dispone de menos instancias que los otros conjuntos. En un caso ideal, para crear un modelo mucho más robusto se necesitarían más imágenes y objetos. Lamentablemente, dado que el conjunto ha sido proporcionado por la empresa colaboradora no ha sido posible ampliarlo. Aun así, al utilizar otros conjuntos con más instancias (DIOR y Standford) se concluye que YOLOv5 es la red detectora (de las utilizadas) que mejores resultados ofrece.

El despliegue de este servicio se puede llevar a cabo mediante diferentes tipos de infraestructuras. En este trabajo se han analizado las siguientes: **local, IaaS y CaaS**. Tras estudiar las diferentes posibilidades, se llega a la conclusión de que no existe una solución que supere a las demás. Si bien es cierto que existen infraestructuras en las que el tiempo del servicio es inferior a otras, estas infraestructuras (con mejores prestaciones) suponen un coste mayor. Por este motivo, la mejor opción de cara a poner en marcha un servicio de estas de estas características dependerá del escenario de uso y de los requisitos específicos del servicio. Si se desea un servicio más rápido se puede realizar el despliegue en una infraestructura con mejores prestaciones, lo que supone un precio mayor, y si por el contrario el tiempo del servicio no es relevante, será más conveniente realizar el despliegue en una infraestructura con peores prestaciones pero con un precio inferior.

Este trabajo sirve para conocer donde se encuentran los límites actuales de esta tecnología, pudiéndose utilizar como benchmark en futuros proyectos.

Finalmente, cabe destacar que se han cumplido los objetivos. Se ha conseguido crear un modelo capaz de reconocer la actividad agrícola para la cual ha sido entrenado y se han evaluado las diferentes posibilidades de despliegue, tanto en un entorno local como en la nube.

Referencias

[1] SERESCO, "Líder en soluciones de software para empresas." https://www.seresco.es/. Accedido el 28 de diciembre de 2020.

[2] C. S. Wiki, "Bounding boxes." https://computersciencewiki.org/index.php/Bounding_boxes. Accedido el 8 de mayo de 2020.

[3] C. Elkan, "Evaluating classifiers," *San Diego: University of California*, 2012.

[4] A. Gandhi, "Data augmentation — how to use deep learning when you have limited data — part 2." https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/, 2018. Accedido el 28 de agosto de 2020.

[5] J. Hui, "Map (mean average precision) for object detection." https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173, Abril 2019. Accedido el 21 de mayo de 2020.

[6] "Qué es overfitting y underfitting y cómo solucionarlo." https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/, Diciembre 2017. Accedido el 28 de agosto de 2020.

[7] J. I. Bagnato, "¿cómo funcionan las convolutional neural networks? visión por ordenador." https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/, Noviembre 2018. Accedido el 15 de abril de 2020.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[9] J. Redmon, "You only look once: Unified, real-time object detection." https://www.youtube.com/watch?v=NM6lrxy0bxs&feature=youtu.be, Septiembre 2016. Accedido el 28 de agosto de 2020.

[10] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[11] S.-H. Tsang, "Review: Yolov1 — you only look once (object detection)." https://towardsdatascience.com/yolov1-you-only-look-once-object-detection-e1f3ffec8a89, Marzo 2019. Accedido el 5 de mayo de 2020.

[12] Anirudh, "Yolo object detection: Understanding the you only look once paper." https://hackerstreak.com/yolo-made-simple-interpreting-the-you-only-look-once-paper/, Febrero 2020. Accedido el 20 de mayo de 2020.

[13] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[14] E. Y. Li, "Dive really deep into yolo v3: A beginner's guide." https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e, Marzo 2010. Accedido el 11 de mayo de 2020.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[16] "Anchor boxes — dive into deep learning." https://d2l.ai/chapter_computer-vision/anchor.html. Accedido el 14 de mayo de 2020.

[17] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge 2007 (voc2007) results," 2007.

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[19] J. Hui, "Real-time object detection with yolo, yolov2 and now yolov3." https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088, Agosto 2019. Accedido el 13 de mayo de 2020.

[20] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[21] A. Kathuria, "What's new in yolo v3?." https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b, Abril 2018. Accedido el 15 de junio de 2020.

[22] U. Almog, "Yolo v3 explained." https://towardsdatascience.com/yolo-v3-explained-ff5b850390f, Octubre 2020. Accedido el 22 de diciembre de 2020.

[23] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[24] R. Balsys, "Yolo v3 theory explained." https://pylessons.medium.com/yolo-v3-theory-explained-33100f6d193, Noviembre 2020. Accedido el 22 de diciembre de 2020.

[25] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[26] J. Solawetz, "Breaking down yolov4." https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/, Junio 2020. Accedido el 07 de septiembre de 2020.

[27] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[28] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768, 2018.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[30] J. Solawetz, "Data augmentation in yolov4." https://blog.roboflow.com/yolov4-data-augmentation/, Mayo 2020. Accedido el 07 de septiembre de 2020.

[31] C.-Y. Wang, H.-Y. Mark Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 390–391, 2020.

[32] D. Shah, "Yolov4 — version 3: Proposed workflow." https://medium.com/visionwizard/yolov4-version-3-proposed-workflow-e4fa175b902, Junio 2020. Accedido el 15 de noviembre de 2020.

[33] G. Jocher, "ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements." https://github.com/ultralytics/yolov5, Octubre 2020.

[34] G. Jocher, "Rectangular inference." https://github.com/ultralytics/yolov3/issues/232, Abril 2019. Accedido el 11 de oct de 2020.

[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[36] J. Hui, "Ssd object detection: Single shot multibox detector for real-time processing." https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06, Marzo 2018. Accedido el 23 de mayo de 2020.

[37] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242, 2020.

[38] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[39] T. C. Arlen, "Understanding the map evaluation metric for object detection." https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3, Febrero 2020. Accedido el 22 de mayo de 2020.

[40] K. Li, G. Wan, G. Cheng, L. Meng, and J. Han, "Object detection in optical remote sensing images: A survey and a new benchmark," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 159, pp. 296–307, 2020.

[41] A. Robicquet, A. Alahi, A. Sadeghian, B. Anenberg, J. Doherty, E. Wu, and S. Savarese, "Forecasting social navigation in crowded complex scenes," *arXiv preprint arXiv:1601.00998*, 2016.

[42] Matlab, "Implementación de yolov2/matlab." https://es.mathworks.com/help/deeplearning/ug/object-detection-using-yolo-v2.html?s_tid=blogs_rc_6. Accedido el 14 de diciembre de 2020.

[43] Microsoft, "Azure custom vision." https://www.customvision.ai/. Accedido el 07 de diciembre de 2020.

[44] NVIDIA, "Implementación de ssd/pytorch." https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Detection/SSD#quick-start-guide, 2018. Accedido el 14 de diciembre de 2020.

[45] Postman, "The collaboration platform for api development." https://www.postman.com/. Accedido el 11 de noviembre de 2020.