ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Máster en Visión Artificial
Curso Académico 2018/2019
Trabajo Fin de Máster

Detección Automática de Carteles Publicitarios en Escenas Exteriores

Autor

Ángel Luis Morera Díaz

Tutores

Ángel Sánchez Calle

José Francisco Vélez Serrano

Julio 2019

AGRADECIMIENTOS

Este trabajo ha sido posible gracias al apoyo, el cariño, la energía y el tiempo de muchas personas. Sin este apoyo creo que no hubiera sido capaz de terminar este Máster en Visión Artificial. Por ello quisiera agradecer todo el apoyo y ayuda que me han prestado durante todos estos años.

A mis profesores, por todo lo que me han ayudado cuando se lo he pedido y sin pedírselo, y por todo lo que me han enseñado y transmitido durante todos estos años.

A la primera persona que se lo quiero agradecer es a mi tutor, Ángel Sánchez Calle, por haber confiado en mí para la realización de este trabajo de fin de máster. Te agradezco de corazón toda la ayuda ofrecida a la hora de plantear y desarrollar este trabajo. Ante todas las dificultades surgidas siempre ha estado dispuesto a dedicarme su tiempo y su conocimiento. También quiero agradecer el apoyo a José Francisco Vélez Serrano, muchas gracias por tus sugerencias y correcciones, por estar siempre dispuesto a ayudar y a guiarme en este trabajo.

A toda mi familia, ya que sin su cuidado y su tiempo no hubiese podido dedicarme a estudiar. En especial a mis padres y mis abuelos, por haberme proporcionado la mejor educación y enseñarme que con trabajo, constancia y esfuerzo todo se consigue. También les doy las gracias a mi hermana y a mi hermano, que siempre me han animado con sus palabras.

No me puedo olvidar de mis compañeros de clase y de toda la gente nueva que he conocido, ha sido un placer por mi parte compartir grandes momentos a lo largo de estos dos años con vosotros.

Finalmente, quiero dar las gracias a mis amigos y a todos aquellos que me han preguntado cómo iba y cuando acababa.

Desde aquí, quiero dar las gracias a todos, sin vosotros no lo hubiera logrado. Gracias por hacerme crecer cada día un poco más.

¡Muchas gracias a todos!

Ángel Luis Morera Díaz

Madrid, 4 de julio de 2019

Contenido

Índice de figuras

Índice de tablas

Resumen

El siguiente trabajo aborda el problema de la localización de manera precisa de la posición de los carteles publicitarios en imágenes estáticas. La posibilidad de determinar la posición exacta de un cartel publicitario dentro de una imagen resulta realmente útil e interesante en muchos dominios de la industria, entre los que destacan la Realidad Aumentada, la Publicidad o el Mantenimiento. Para ello, en este trabajo, se han utilizado dos tipos de redes neuronales profundas para la detección de objetos como son *YOLO* y *SSD*, aplicados sobre un único problema, la detección de carteles publicitarios en imágenes. Esto permite evaluar el rendimiento de cada uno de los métodos en igualdad de condiciones.

Este proyecto aborda el problema de detección de carteles publicitarios utilizando el paradigma de *Deep Learning* que proporciona una metodología de aprendizaje automático para determinar las coordenadas exactas que delimitan la región de interés perteneciente al cartel. Se trata de redes neuronales profundas (con múltiples capas) que tienen la capacidad de aprender y realizar ciertas tareas complejas basándose en un entrenamiento inicial. El objetivo será determinar las coordenadas exactas que delimitan el contorno del cartel. Para la obtención de dichas coordenadas se utilizarán diferentes algoritmos de detección basados en redes neuronales como son *YOLO* y *SSD*.

Afrontar un problema de detección y localización como estos de manera manual supone una tarea complicada y engorrosa que implicaría dedicar una enorme cantidad de tiempo y recursos en la detección cuando tan sólo tenemos una cantidad minúscula de imágenes. Por ello se deben buscar soluciones automáticas a estos problemas que realicen la tarea de detección y localización con la mayor brevedad y precisión posible. Gran parte de la dificultad de este problema se debe a la gran variabilidad de la publicidad exterior, así como a las diferentes condiciones climatológicas que puedan afectar a la detección, además del hecho de que los carteles publicitarios, como su propio propósito indica, se encuentran en zonas muy concurridas los que puede provocar oclusiones por vehículos o personas que dificulten su correcta detección.

Debido a que no existe ninguna base de datos acerca de carteles publicitarios en exteriores, también ha sido necesario crear un dataset propio para poder entrenar con él las dos arquitecturas de *Deep Learning*. Las imágenes de carteles publicitarios han sido obtenidas principalmente a través de *Google Images* y de *Google Street View*, y en su mayoría pertenecen a marquesinas de paradas de autobús y diferentes tipos publicidad exterior. Para dar variabilidad al dataset se han capturado carteles a diferentes distancias, condiciones ambientales y de iluminación, así como oclusiones. Antes de proporcionar estas imágenes al sistema, se debe realizar un preproceso con el objetivo de que la precisión del sistema sea la más alta posible. El sistema automático propuesto en este trabajo consiste en dos pasos: extracción de características (aprendizaje) y detección. Durante la fase de aprendizaje se entrenarán dos tipos de redes neuronales diferentes con las imágenes de entrenamiento y se extraerán una serie de patrones que permitirán al sistema aprender correctamente a localizar los carteles. Durante la fase de clasificación, se le proporcionarán a la red imágenes desconocidas para que ésta localice y obtenga de manera automática las coordenadas que delimiten el perímetro del cartel de la manera más precisa posible.

Acrónimos

AR	**A**ugmented **R**eality
BOVW	**B**ag **O**f **V**isual **W**ords
CCA	**C**anonical **C**orrelation **A**nalysis
CNN	**C**onvolutional **N**eural **N**etwork
COCO	**C**ommon **O**bjects in **CO**ntext
DLT	**D**irect **L**inear **T**ransformation
DoG	**D**ifference **o**f **G**aussian
FCN	**F**ully **C**onvolutional **N**etwork
FFT	**F**ast **F**ourier **T**ransform
GPS	**G**lobal **P**ositioning **S**ystem
HMM	**H**idden **M**arkov **M**odel
HIS	**H**ue **S**aturation **I**ntensity
IA	**I**nteligencia **A**rtificial
IoT	**I**nternet **o**f **T**hings
IoU	**I**ntersection **o**ver **U**nion
LSTM	**L**ong **S**hort-**T**erm **M**emory
MLP	**M**ulti-**L**ayer **P**erceptron
OCR	**O**ptical **C**haracters **R**ecognition
PDF	**P**robability **D**ensity **F**unction
RGB	**R**ed **G**reen **B**lue
RMSE	**R**oot **M**ean **S**quare **E**rror
ROI	**R**egion **O**f **I**nterest
RPN	**R**egion **P**roposal **N**etwork
SIFT	**S**cale-**I**nvariant **F**eature **T**ransform
SSD	**S**ingle **S**hot **D**etector
SVM	**S**upport **V**ector **M**achine
TIC	**T**ecnologías de la **I**nformación y **C**omunicación
VC	**V**isión por **C**omputador
WMS	**W**eb **M**ap **S**ervice
YOLO	**Y**ou **O**nly **L**ook **O**nce

Palabras Clave

Detección de Objetos, publicidad exterior, panel publicidad urbana, YOLO, SSD, Ciudad Inteligente, red neuronal de convolución.

Keywords

Object detection, outdoor publicity, urban advertising panel, YOLO, SSD, smart city, convolutional neural network.

1. Introducción

A continuación, se explica brevemente el problema de detección y localización de carteles publicitarios en escenas exteriores, las dificultades que éste presenta así como sus campos de interés.

1.1 Descripción General del Problema

Entre los múltiples formatos publicitarios a los que las empresas, tanto grandes como pequeñas, pueden acceder en la actualidad sin duda destacan por su importancia a pie de calle la publicidad exterior. Este tipo de publicidad es la gran protagonista en multitud de espacios públicos donde puede llegar formar parte del paisaje urbano de la ciudad y llegar a crear una gran repercusión en los viandantes y conductores ya que todo el mundo sale a la calle cada día para ir a trabajar, al supermercado o al gimnasio y no podemos evitar ver los anuncios que se colocan en el mobiliario urbano. Un ejemplo de este tipo de publicidad son los paneles publicitarios y marquesinas, que son un tipo de mobiliario urbano que comúnmente presenta una forma normalizada y un tamaño más reducido que las vallas publicitarias.

La detección de paneles publicitarios en imágenes presenta ventajas importantes tanto en el mundo real como en el mundo virtual, éste último favorecido por los teléfonos inteligentes actuales que están equipados con una gran variedad de sensores integrados como cámaras, GPS o 4G, que permiten acercar al usuario una variedad de aplicaciones de Realidad Aumentada (AR) [10]. En el mundo real, una aplicación útil sería la comprobación automática del estado del mobiliario para facilitar su mantenimiento minimizando la intervención humana. Para ello, antes resultaría necesaria la detección de dichos carteles. Otra aplicación, una vez realizada la detección de los paneles, podría ser el reconocimiento del producto incluido en la publicidad y obtener más información sobre él a través de aplicaciones de Realidad Aumentada. De esta manera, los ciudadanos que usan sus teléfonos inteligentes pueden profundizar mejor y, quizás, disfrutar de los contenidos asociados a los anuncios urbanos. Además, es posible detectar si la información del producto anunciado está actualizada o no. Con respecto a la publicidad en imágenes urbanas de Internet en aplicaciones como *Google Street View*, al detectar paneles en estas imágenes, sería posible reemplazar la publicidad que aparece dentro de un panel por otra propuesta por una compañía.

La detección y localización automática de los paneles publicitarios exteriores en imágenes urbanas reales es una tarea difícil debido a las múltiples condiciones de variabilidad que presentan imágenes que incluyen estos elementos publicitarios. Algunos factores que dificultan la correcta detección de los paneles publicitarios en una escena son, por ejemplo:

- Las condiciones de iluminación (luz solar o luces artificiales) pueden dificultar la detección del panel.
- las condiciones climatológicas adversas, como la oscuridad en la noche, la presencia de niebla o la nieve entre otras.
- la vista en perspectiva o deformaciones del panel o relación de tamaño del panel con respecto al tamaño total de la imagen.

- Oclusiones sobre los paneles publicitarios debido a la presencia de múltiples elementos rodeando el panel como árboles, vehículos y/o viandantes, entre otros factores.
- Presencia de diferentes objetos artificiales del mismo color, forma y tamaño que los carteles publicitarios, como pueden ser carteles de obra, frontales de autobús entre otros.
- Presencia de múltiples carteles en un único fotograma.
- Calidad de las imágenes varía en función de la cámara utilizada para su adquisición.
- Dependiendo del montaje del sistema (sistema fijo, montado en vehículo, …) se pueden capturar imágenes borrosas o capturar el mismo cartel desde diferentes ángulos lo que dificulta su detección.
- Al tratarse de publicidad en el exterior, los carteles pueden encontrarse deteriorados por el paso del tiempo o lluvias torrenciales, granizo entre otras.

La ilustración 1 muestra algunos ejemplos de dificultades, mencionadas anteriormente, que nos podemos encontrar a la hora de detectar y localizar un panel publicitario en un escenario de la vida real.

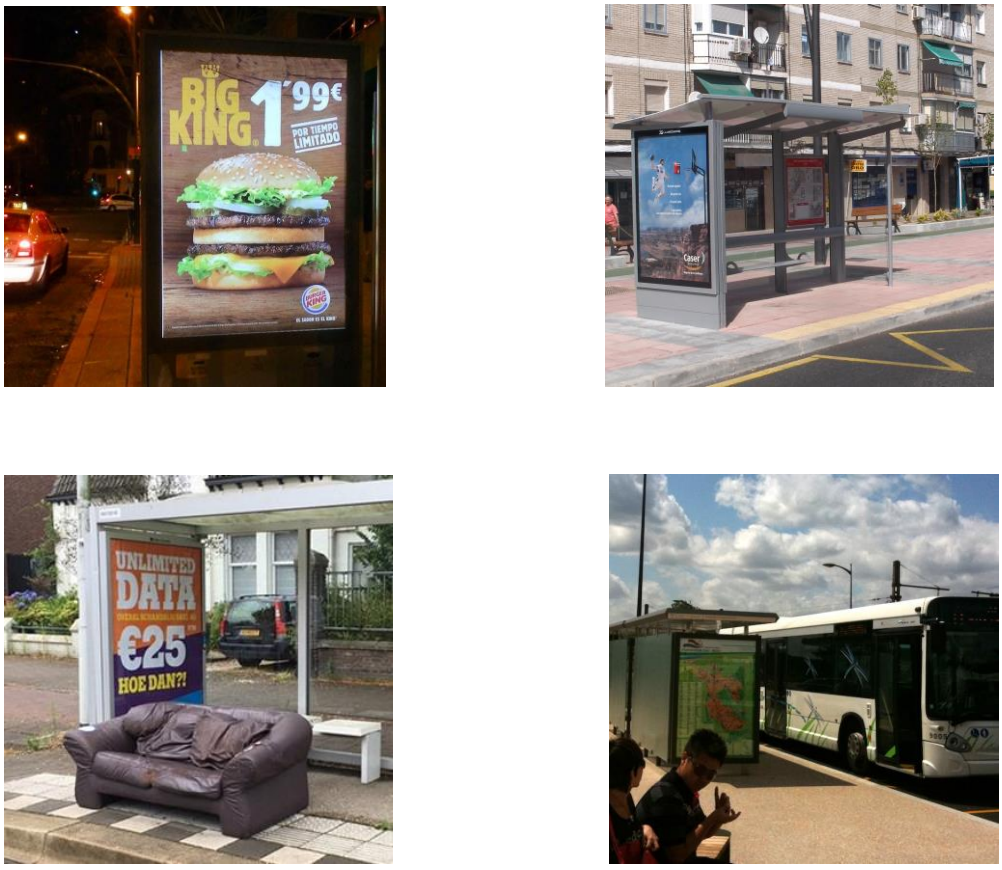

Ilustración 1 Variabilidades de los paneles del conjunto de test

1.2 Descripción específica del problema

En los últimos tiempos, es fascinante ver cuánto se ha avanzado en términos de innovación y como de rápido avanza la tecnología. Dos conceptos en lo que más se ha ido avanzando y trabajando a lo largo de estos últimos años han sido "*Smart City*" y "*Self-Driving*".

A pesar de parecer un concepto actual y emergente, el concepto de "*Smart City*" fue acuñado hace veinte años [25]. Dicho concepto se refiere a un espacio urbano que aplica las Tecnologías de la Información y la Comunicación (TIC) para mejorar la calidad y el rendimiento de los servicios urbanos, como la energía, el transporte y los servicios públicos, a fin de reducir el consumo de recursos y costes generales buscando desarrollo urbano basado en la sostenibilidad. El mantenimiento del espacio urbano en las ciudades inteligentes es una tarea desafiante y que requiere mucho tiempo, ya que este espacio está realmente rodeado por una gran cantidad de Sistemas Inteligentes de *IoT* que pueden capturar y procesar de manera eficiente una gran cantidad de datos. El problema de la monitorización del espacio urbano se ha abordado recientemente utilizando marcos basados en dispositivos de detección de ciudadanos, bajo una filosofía de *crowdsourcing* [5]. Otros trabajos [13], han abordado el problema de la detección de paneles de tráfico con el propósito de hacer un inventario automático de los paneles de tráfico ubicados en una carretera para apoyar el mantenimiento de la carretera y ayudar a los conductores. Además, en trabajos como [32] se estudian sistemas para la detección de cambios estructurales en el espacio público capturados por una cámara monocular a lo largo del tiempo.

Por su parte, el concepto "*Self-Driving*" [19, 26] hace referencia a los vehículos que son capaces de imitar las capacidades humanas de manejo y control, para ello es necesario que puedan percibir el medio que le rodea y navegar en consecuencia. Los sistemas avanzados de control interpretan la información para identificar la ruta apropiada, así como los obstáculos y la señalización relevante. En este aspecto, un paso previo importante, es la detección de las señales de tráfico en la carretera [9, 20, 22], ya que es necesario que el coche tenga en cuenta en cada momento la señalización de la vía. Además de las señales de tráfico, es necesario tener un conocimiento absoluto de todo el entorno [31] lo que rodea al vehículo, como puedan ser la presencia personas, animales u otros vehículos.

Es por ello por lo que en los últimos tiempos se han producido avances significativos en los enfoques de detección de objetos basados en características y clasificadores de aprendizaje automático. Los problemas de detección y reconocimiento visual, aplicados a elementos específicos, en imágenes de exteriores han sido ampliamente estudiados. Por ejemplo, este es el caso de la localización de vehículos [15], la detección de señales de tráfico [1] o las placas de automóviles [21], detección de carteles de tráfico en autopistas [13]. Sin embargo, hasta donde sabemos, no hay trabajos publicados sobre la detección de paneles de anuncios en exteriores. Se han investigado problemas relacionados, como la detección de texto y objetos dentro de imágenes de cartelera segmentadas [30] o la localización de carteles en videos de deportes transmitidos [6]. Otra aplicación relacionada es la inserción de anuncios virtuales en imágenes de calles basadas en la localización de regiones específicas en ellos (por ejemplo, fachadas de edificios) [10]. Por ello, en este trabajo se propone un método robusto para la detección automática de paneles publicitarios urbanos en imágenes de exteriores, sobre todo en marquesinas de paradas de autobús y MUPIs o mupis (Muebles Urbanos para la Presentación de Información). Se trata de un trabajo novedoso ya que son muy pocos los trabajos que han centrado su investigación en este tipo de objetos, a pesar de que en la actualidad la industria de la publicidad exterior ha experimentado un importante crecimiento y gran parte del espacio público se encuentra rodeado por este tipo de objetos.

Las grandes dificultades a la que nos enfrentamos a la hora de tratar con un problema de detección de carteles publicitarios en exteriores son principalmente la variabilidad en la forma y tamaño de los carteles publicitarios, las condiciones climatológicas y de iluminación debido a su localización en el exterior y por último la propia complejidad del entorno.

Para llevar a cabo los diferentes experimentos realizados en este trabajo, debido a que no existen bases de datos públicas acerca de carteles publicitarios, se ha tenido que crear una base de datos propia desde cero recopilando imágenes de carteles publicitarios en marquesinas de autobús y mupis. La base de datos ha sido especialmente diseñada para entrenar y testear detectores de objetos mediante técnicas de aprendizaje automático supervisado, ya que todos los datos se encuentran etiquetados. En total, la base de datos está compuesta por 5700 imágenes que contienen uno o más carteles publicitarios.

Para lograr un aprendizaje óptimo es necesario dotar a la base de datos de variabilidad, es por ello por lo que la base de datos contiene imágenes de carteles a diferentes distancias, de manera que se obtienen diferentes tamaños (grandes, pequeños, medianos…), en diferentes condiciones de iluminación (imágenes nocturnas, diurnas, lluvia, nieve) y en un entorno real, por lo que contiene imágenes de carteles con oclusiones por objetos del entorno. Todas estas imágenes han sido preprocesadas de manera que todas las imágenes tienen un tamaño homogéneo de 512x512 píxeles y además se encuentran etiquetadas. Dichos datos han sido etiquetados en dos formatos diferentes, adecuados para las arquitecturas de *Deep Learning SSD* y *YOLO* entre otras.

Para abordar localización y detección de carteles publicitarios se ha decidido utilizar el paradigma de *Deep Learning*, conjunto de algoritmos de aprendizaje automático que modela abstracciones de alto nivel en datos usando arquitecturas que implementan transformaciones no lineales múltiples. Varias arquitecturas de aprendizaje profundo se han aplicado en problemas de Inteligencia Artificial y Visión por Computador obteniendo resultados de vanguardia en varias tareas lo que nos ha hecho elegir un sistema de redes neuronales de convolución para abordar nuestro problema. Este tipo de redes es especialmente utilizado cuando trabajamos con imágenes y se llaman de convolución ya que en lugar de aplicar la multiplicación de matrices típica lo que hacen es aplicar una operación de convolución sobre los datos de entrada (en nuestro caso, la imagen que contiene el cartel). Las redes neuronales convolucionales consisten en un conjunto de capas con distinta funcionalidad. La estructura de este tipo de redes se divide en dos secciones, en la primera sección se encuentran de forma alterna las capas de convolución y las de reducción, que se encargan de la extracción de características. En la segunda sección se encuentra un perceptrón multicapa que realiza la clasificación de las características extraídas por la componente anterior.

La solución propuesta se basa en la detección de paneles de publicidad urbana al aire libre en imágenes estáticas usando una red neural profunda y robusta que logra una alta precisión en la detección de paneles bajo diferentes condiciones de iluminación, posición y tamaño de los objetivos a detectar. Además, se trata de minimizar el número de falsos para permitir su uso práctico.

1.3 Organización del Documento

En la primera sección del documento se ha realizado una introducción al problema de detección de carteles publicitarios. En primer lugar, se ofrece una visión muy general del problema y las dificultades que éste presenta. Además, se exponen los distintos campos temáticos que se benefician de la detección automática de dichos objetos, así como las dificultades que presentan. Al mismo tiempo se muestra la base de datos utilizada para realizar los diferentes experimentos, así como el número total de datos disponibles para realizar la evaluación de cada experimento. Por último, se explica el paradigma utilizado para resolver el problema: en este caso, *Deep Learning* y la arquitectura utilizada como son las redes neuronales de convolución.

El resto del documento se organiza como sigue. La sección 2 se resume el estado del arte, describiendo los trabajos más actuales sobre la detección de objetos y los avances conseguidos en esta materia. Debido a que no existen artículos previos acerca de la detección de carteles publicitarios, los trabajos mostrados abordaran la detección de objetos lo más similares posibles a los carteles publicitarios, como pueden ser los paneles de las autopistas o las señales de tráfico entre otros. Para ello, se muestran las distintas soluciones automáticas propuestas en estos trabajos, así como los diferentes experimentos realizados para la evaluación de estas soluciones. Por último, se enumeran las distintas bases de datos utilizadas en estos trabajos y se hace un breve resumen de su contenido.

En la sección 3, se explica la solución propuesta para la detección y localización de carteles en imágenes estáticas. En primer lugar, se hace una descripción general de las soluciones propuestas. Posteriormente, se explica el preproceso de los datos antes de que éstos sean utilizados por cada una de las redes en el entrenamiento. Después se explican las diferentes arquitecturas de red utilizadas, así como las distintas herramientas utilizadas en su desarrollo. Por último, explicamos la solución algorítmica al problema tratado.

En la sección 4 se profundiza en los distintos experimentos realizados a la hora de evaluar cada uno de los métodos propuestos como solución al problema de la detección y localización de carteles publicitarios. Para ello, se explicarán las distintas configuraciones de red y los distintos datos utilizados, mostrando los resultados obtenidos para cada una de las soluciones propuestas. Por último, se hace un análisis de los resultados obtenidos explicando los motivos de estos resultados.

Por último, en la sección 5 se muestran las conclusiones del trabajo, las posibles mejoras y trabajos futuros que no se han podido desarrollar durante el presente trabajo.

2. Estado del Arte

En esta sección, se resume la evolución de los problemas de detección de objetos, desde el uso de las técnicas más clásicas hasta los trabajos de *Deep Learning* más actuales.

2.1 Introducción

No son pocos los problemas que resultan interesantes para la Visión por Computador, desde una simple clasificación de imágenes hasta la estimación de una pose facial 3D. Uno de los problemas más interesantes y en los que más se ha avanzado en los últimos años ha sido la detección de objetos, si bien existen muchos otros que pueden ser realmente útiles dependiendo de la tarea que intentamos resolver.

A la hora de elegir un sistema de aprendizaje automático para resolver un problema es imprescindible tener en cuenta una serie de aspectos que resultarán decisivos para la resolución del problema. Considerando estos aspectos, podemos etiquetar los sistemas en varias categorías. En primer lugar, es necesario saber el tipo de datos que tenemos disponibles para entrenar el sistema de aprendizaje automático. En función de esta información, podemos distinguir entre las categorías de aprendizaje supervisado, aprendizaje semi-supervisado y aprendizaje no supervisado. Se habla de aprendizaje supervisado cuando todos los datos disponibles se encuentran correctamente etiquetados y a partir de estos datos es capaz de aprender y resolver el problema sobre imágenes totalmente desconocidas. Por su parte, el aprendizaje no supervisado utiliza conjuntos de datos no etiquetados para agruparlos en categorías que se crean en función de las características comunes en cada uno de los grupos. Por último, el aprendizaje semi-supervisado es una mezcla de los dos anteriores, se tienen datos etiquetados y datos sin etiquetar, de manera que se intenta relacionar los datos sin etiquetar con algunos de los grupos etiquetados. En segundo lugar, hay que tener en cuenta el tipo de solución que mejor se adapta a nuestro problema. En función de este aspecto podemos diferenciar entre Clasificación, Localización, Detección de Objetos y Segmentación Semántica. El problema de Clasificación probablemente sea el más conocido en Visión por Computador y consiste en asociar una imagen en una de diversas categorías diferentes. En los últimos años, los modelos de clasificación han conseguido en algún caso superar el rendimiento humano [39]. El problema de Localización es similar a la clasificación, salvo que ahora además de predecir la clase, la localización también encuentra la ubicación exacta de un único objeto dentro de la imagen. Por su parte, el problema de Detección de Objetos consiste en encontrar y clasificar un número variable de objetos en una imagen. La diferencia importante es la parte "variable". En contraste con problemas como la clasificación, la salida de la detección de objetos es variable en longitud, ya que la cantidad de objetos detectados puede cambiar de una imagen a otra. Por último, la segmentación de instancias va un paso más allá, ya que no solo encuentra cada objeto dentro de la imagen, sino que para cada objeto se obtiene una máscara con los píxeles del objeto. La segmentación de instancias asocia a cada píxel de la imagen una etiqueta de clase correspondiente al objeto que está representando. De manera que para objeto de la imagen se obtiene una máscara formada por los pixeles de dicha clase.

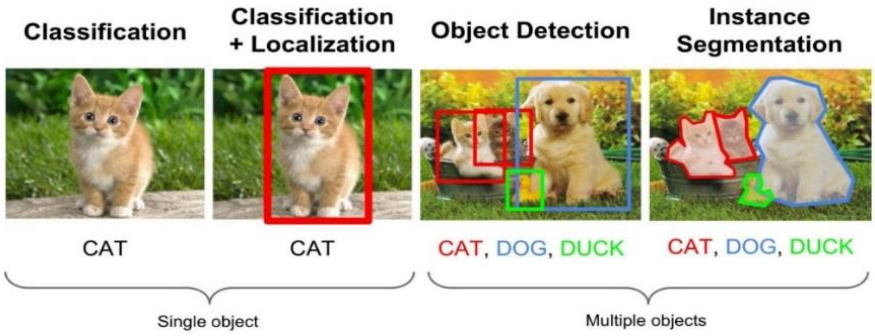

Ilustración 2 Tipos de Problemas de Deep Learning

En este trabajo, nos centramos en la detección de objetos ya que, desde hace varios años, son muchos los trabajos e investigaciones que han dedicado su tiempo a la búsqueda de soluciones automáticas para los problemas de detección y localización de objetos en imágenes. El objetivo principal de este problema consiste en obtener las coordenadas exactas que determinan la región de interés que contiene al objeto dentro de la imagen, en este caso, los carteles publicitarios.

Generalmente, el funcionamiento de los sistemas que realizan la detección automática de objetos se divide en dos fases: extracción de características y predicción, si bien suele ser muy habitual realizar una fase previa de preproceso de la imagen. La etapa de preproceso tiene como objetivo mejorar y realzar las propiedades de la imagen para facilitar las operaciones de segmentación y extracción de las características, así como etiquetación de los datos y la homogenización del tamaño de las imágenes. En este caso, para la detección y localización automática de carteles publicitarios, únicamente ha sido necesario etiquetar y homogeneizar el tamaño de las imágenes. Por su parte, el objetivo de la extracción de características consiste en obtener modelos que caractericen los diferentes tipos de carteles publicitarios y generar regiones de interés que facilitan el proceso de aprendizaje necesario para la tarea de predicción. Por último, durante la etapa de predicción, se utilizan los mapas de características y las regiones propuestas durante la fase anterior para obtener un "*Bounding Box*" o "*ROI*" así como la probabilidad de que el objeto pertenezca a cada una de las categorías. Sí bien en este caso, el objeto solamente puede pertenecer a una única categoría, es decir cartel publicitario. La mayoría de las soluciones automáticas propuestas hacen uso de algoritmos de aprendizaje automático para realizar la tarea de extracción de características y predicción, si bien en unos casos, cada una de estas tareas se realiza por separado y en otros se realiza de manera conjunta.

2.2 Trabajos Previos

Muchos han sido los autores que desde hace tiempo se han interesado en buscar soluciones automáticas a los problemas de detección de objetos motivados mayormente por la enorme utilidad de este problema en importantes ramas de la industria como son la conducción autónoma, las "*Smart Cities*" o la publicidad entre otras. Miguel Ángel García y otros (2003) [1] propusieron un sistema en tiempo real para la detección automática de señales de tráfico en base a la forma y color de la señal. Desde entonces, muchos han sido los autores que ha abordado el problema aplicando diferentes técnicas y algoritmos para lograr mejoras significativas en la tarea de detección de objetos. A continuación, se presentan por orden cronológico distintas investigaciones sobre la detección de objetos desde que se publicara

la primera solución hasta nuestros días, donde Murhaf Hossari y otros (2018) [33] abordan el problema para la detección de vallas publicitarias en videos de escenas exteriores usando técnicas de *Deep Learning*.

Los mencionados, Miguel Ángel García y otros [1] plantearon un algoritmo para la detección de señales de tráfico en imágenes estáticas. Dicho algoritmo se basaba en la forma y color de las señales de tráfico y funcionaba en dos fases. En primer lugar, se realizan proyecciones horizontales y verticales de los bordes para obtener las posibles regiones candidatas que contengan al objeto. En segundo lugar, se validan las regiones candidatas a partir de técnicas de umbralización de la imagen en base a los colores azul y rojo, además se realiza un análisis de la forma y se aplica *template matching* para obtener el tipo de señal. Dicho trabajo se trataba de un comienzo en su investigación de un sistema de detección y reconocimiento de señales de tráfico en tiempo real, por lo que no se presentaron resultados de la evaluación del sistema.

En 2004, Frank Aldershoff y Theo Gevers [2] trataron el problema de localización y seguimiento de vallas publicitarias en partidos de fútbol emitidos por *streaming* con el objetivo de reemplazar la publicidad real por una publicidad virtual orientada a diferentes audiencias. Para ello, el sistema propuesto utilizaba características fotométricas invariantes basadas en el espacio de color *RGB* normalizado, ya que son menos susceptibles a cambios de iluminación y representaba las vallas publicitarias como una función de densidad de probabilidad (*PDF*) en el espacio de características *RG*. Se utilizaba un histograma para aproximar el *pdf* y construir el modelo. Para encontrar el objetivo se realiza una búsqueda región a región comparando el histograma de la región con el histograma meta mediante el coeficiente de *Bhattacharyya*, de manera que un objeto se localiza cuando la distancia entre los histogramas es mínima. Para la evaluación del sistema propuesto se procesó un breve videoclip de un partido de fútbol. Durante aproximadamente 10 segundos de vídeo se realizó seguimiento visual de una valla publicitaria que se movía de izquierda a derecha y hacia atrás, y también cambiaba de tamaño debido a la panorámica y zoom de la cámara. Además, durante ese tiempo, el contenido publicitario fue remplazado por otro. Durante el período en que se realizó el seguimiento de la cartelera, los jugadores ocasionalmente obstruyeron la valla, pero no afectó el proceso de seguimiento. Además, se pudo observar que nunca se perdía el rastro de la valla y la región de seguimiento se adaptaba adecuadamente al zoom.

En 2006, Vazquez-Reina y otros [3], estudiaron un enfoque para la detección y extracción de texto en paneles de señalización vial. El algoritmo propuesto funciona en dos fases, la primera se centraba en la detección del panel mientras que la segunda realizaba la extracción del texto contenido en el panel. El algoritmo de detección del panel funciona en tres fases. La primera fase consiste en segmentar la imagen original. La principal dificultad de esta fase son los cambios de intensidad, por ello se utiliza el espacio de color *HSI*, ya que los componentes Hue y Saturación son lo suficientemente invariantes a dichos cambios. El fondo de la mayoría de los paneles de tráfico es azul o blanco, por lo que se realiza una umbralización por color (azul o blanco) con el objetivo de crear una máscara donde los píxeles pertenecientes al panel se marcan como objeto mientras que el resto se etiqueta como fondo. Luego, se realiza un etiquetado de las componentes conexas y se analizan todas las regiones candidatas mediante un proceso de selección, donde algunos son

descartados. La segunda fase consiste en clasificar la forma de cada región. Para ello, se aplica la transformada rápida de *Fourier* (*FFT*) a cada región y se compara su firma con una firma objetivo de referencia. Este método es invariante a rotaciones del objeto, deformaciones o distorsiones en la proyección de la cámara. El tercer y último paso de la detección del objeto consiste en reorientar la región. Para ello se obtienen los cuatro vértices de la imagen original a partir de la firma de la región, ya que los vértices se corresponden con los picos máximos de la firma. A partir del algoritmo de Transformación Lineal Directa (*DLT*), se puede calcular la matriz de transformación homogénea y reorientar el panel. Para la evaluación del sistema, se probaron diferentes configuraciones con diferentes parámetros. Algunos de los parámetros modificados para realizar las pruebas fueron los umbrales de segmentación por luminancia, umbrales de área, de tamaño y relación de aspecto. Después de realizar pruebas con varios paneles de señalización, finalmente se eligió la configuración que ofrecía mejores resultados, si bien es cierto no se mostraron dichos resultados en este trabajo.

En 2008, Lamberto Ballan y otros [7] propusieron un sistema semiautomático para la detección de logos en videos deportivos. La aparición de marcas comerciales en los videos deportivos a menudo se caracteriza por deformaciones en perspectiva, movimiento borroso y oclusiones. Para obtener una técnica de emparejamiento robusta, se utilizan puntos obtenidos por *DoG* y descriptores *SIFT* como representación compacta de los aspectos importantes y la textura local en las marcas comerciales. Para ello se detectan los puntos *SIFT* y se seleccionan los fotogramas midiendo su calidad visual en función de la borrosidad, la cantidad de bordes y la cantidad de puntos. La selección de los fotogramas se realiza mediante un clasificador *SVM*, utilizando un núcleo *RBF*. Las marcas comerciales se representan como una bolsa de puntos de *SIFT* y cada marca se representa mediante una o más instancias gráficas. Una marca registrada está representada por N puntos *SIFT* detectados en la imagen y cada fotograma de un video se representa de manera similar a una bolsa de puntos. La detección y recuperación de marcas registradas se realiza comparando el conjunto de características locales que representan la marca registrada con las características locales detectadas en el fotograma del video. Los resultados del emparejamiento se realizan comparando el número de puntos *SIFT* en un fotograma con la marca registrada de referencia. La determinación de si un marco contiene una marca comercial se realiza mediante el umbral de la puntuación de coincidencia normalizada. Para localizar la marca registrada en el cuadro original y aproximar su área, calculamos una estimación robusta de la nube de puntos característicos. Dadas las ubicaciones de puntos de características se calcula la estimación robusta del centroide y los puntos emparejados con poca influencia se descartan. Para la evaluación del sistema utilizaron videos de diferentes eventos deportivos como Volley, MotoGP y fútbol. Dichos vídeos contienen marcas en diferentes condiciones de iluminación, con oclusiones y deformaciones por la perspectiva. Los resultados obtenidos mostraron que, en la mayoría de los casos, se lograba una tasa de precisión de alrededor del 85% con un *recall* de alrededor del 80%.

También en 2008, Alok Watve y Shamik Sural [6] plantearon un método para la detección y análisis de vallas publicitarias en retrasmisiones de partidos de futbol por televisión. Para ello, se identificaban los límites de un marco y el tipo de cada marco. Los fotogramas de cada marco se segmentaban para localizar posibles regiones de interés, es decir, ubicaciones dentro del fotograma donde potencialmente hay una valla publicitaria. Finalmente, se usaba

una combinación de características locales y globales para detectar carteles individuales comparando el cartel con un conjunto de plantillas de referencia.

En 2009, Fabien Moutarde y otros [8] diseñaron un sistema para la detección de los límites de velocidad a partir de las señales de tráfico de la carretera. El sistema propuesto funciona en tres fases. En la primera fase, se detectan las posibles señales de tráfico en la imagen. En la mayoría de los trabajos, estas detecciones se basan en la forma y color de las señales buscadas, pero en este caso se utilizan imágenes en escala de grises, lo que mejora la robustez en condiciones adversas de iluminación. En este trabajo, la detección está basada en la forma de las señales, por ello se utiliza una transformada de Hough circular especialmente adaptada para la aplicación en Europa y una detección de rectángulo especialmente diseñada para la detección de señales de límite de velocidad de Estados Unidos. Los falsos positivos en esta etapa no son un problema, ya que serán filtrados eficientemente en la siguiente etapa. La fase de reconocimiento segmenta los caracteres dentro de las posibles señales y aplica un reconocimiento óptico de dígitos basado en una red neuronal *MLP*. El algoritmo de segmentación de caracteres debe ser eficiente y robusto a las variaciones de orientación. Por ello, se realiza una binarización de la señal en base a un umbral adaptable y se aplica un etiquetado de componente conexas. La red neuronal es un perceptrón multicapa (*MLP*) con 10 salidas (1 para cada valor de dígito) entrenado en bases de datos especialmente construidas de dígitos extraídos del límite de velocidad. Dependiendo de todas las salidas de la red neuronal, se calcula una medida de confianza que se asigna a las señales detectadas y reconocidas. La señal se valida si su confianza supera un umbral de validación, generalmente si la señal se identifica con una confianza razonable en al menos 2 ó 3 fotogramas consecutivos. Para la evaluación del sistema se utilizaron 150 minutos de vídeo, totalmente distintos a los utilizados para el entrenamiento. La tasa de detección alcanzada por el sistema fue del 89%. En total se analizaron 281 señales de tráfico, de las cuales 250 fueron correctamente detectadas, validadas y clasificadas, 29 no pudieron ser validadas y 2 fueron correctamente detectadas y validadas, pero no clasificadas. La mayoría del 11% de las detecciones no correctas se debía al ruido o la oclusión que impedían la segmentación de los dígitos.

En 2011, Álvaro González y otros [11] propusieron un sistema de inspección automático para tareas de conservación de las señales de tráfico, a bordo de un vehículo, que realiza tareas de inspección a velocidades normales de conducción. El sistema se basaba en el principio de retrorreflexión de la luz, ya que parte de la luz infrarroja que entra en contacto con los paneles es reflejada y capturada por un sistema estereoscópico formado por dos cámaras de alta resolución. El proceso de inspección se puede dividir en dos etapas con diferentes tareas. La primera etapa consiste en la adquisición y registro de las secuencias estereoscópicas de entrada. En esta segunda etapa, las secuencias grabadas anteriormente se procesan y como resultado se obtiene un informe del panel analizado. El algoritmo de procesamiento de imágenes actúa en diferentes fases. El primer paso consiste en detectar la ubicación precisa de las señales y paneles en las imágenes. Para ello, se realiza un análisis de las formas aplicando la transformada de *Hough* a los contornos de una imagen de bordes obtenidas aplicando el método de *Canny*. Para cada detección, se obtiene una referencia global combinando la posición global del vehículo y la posición relativa entre el vehículo y panel, que se calcula combinando el sensor de visión estéreo con el odómetro. Cada panel detectado se analiza y clasifica en una de las categorías en base a su luminancia y forma y

se lleva a cabo un proceso de segmentación para separar sus elementos básicos. Finalmente, el sistema genera un informe que tiene la información sobre cada panel. Como resultado de los experimentos realizados, el sistema detección era capaz de alcanzar una precisión en la detección de los paneles de un 99,52%.

En 2011, Yu Huang y otros [10] abordaron el problema de inserción de anuncios virtuales para Realidad Aumentada en fachadas de una escena real específica obtenida de *Google Street View*. Para ello, en primer lugar, extraían las líneas rectas de la imagen y se identificaban los puntos de fuga. A partir de dos conjuntos de líneas paralelas correspondientes a los respectivos puntos de fuga, se detectaba una estructura plana rectangular que satisfacía la verificación de la esquina y la verificación de la dirección dominante para la inserción del anuncio. Para obtener un efecto real en la inserción de anuncios en vídeo, se creó un ciclo de detección y seguimiento para el registro dinámico. Ese filtrado de movimiento está diseñado para reducir las vibraciones en la alineación de realidad virtual. Finalmente, se ajustaba la armonización del color para que no diferir en exceso la parte real con la virtual. Para la evaluación del sistema se capturaron algunos vídeos en HD (1440x1080) con una videocámara. Los resultados demostraron que la alineación real-virtual y la agudeza visual se realizaba de manera satisfactoria incluso con un ligero temblor de la mano, sin embargo se podía observar que la oclusión por árboles no se pudo resolver. Además, los resultados no se obtienen en tiempo real.

En 2012 Álvaro González y otros [12] desarrollaron un método para reconocer el texto de los paneles de tráfico de las imágenes de *Google Street View*. Para el reconocimiento de palabras usaron modelos ocultos de Markov (*HMM*) y los servicios de mapas web (*WMS*) para aumentar la efectividad del algoritmo de reconocimiento. En primer lugar, se obtuvieron las imágenes del servicio de *Street View* de Google. Después se aplicó un detector de texto sobre la imagen completa. Sin embargo, a veces el método de detección de texto no localizaba todo el texto contenido en los paneles de tráfico debido a diferentes razones, como el tamaño pequeño del texto o la distorsión de la imagen. Por ello, se complementó con un algoritmo de detección de panel que se aplicaba después del método de detección de texto. En la posición donde se encontraban los paneles en la imagen, se aplicaba restauración de caracteres utilizando *MSER* para localizar todos los caracteres que no eran detectados durante la primera etapa de detección. Finalmente, se aplicó el reconocimiento de caracteres basado en *HMM* que calculaba el modelo más probable y utilizaba un diccionario de palabras, que incluía todas las palabras que el sistema podía reconocer y suelen aparecer en los paneles de carreteras. Para aumentar la efectividad del algoritmo de reconocimiento, se utilizó un Servicio de mapas web (*WMS*) para reducir el tamaño del diccionario a un área geográfica limitada. El servicio utilizado es Cartociudad, una base de datos oficial de la red vial española. Para evaluar el sistema se utilizaron 214 paneles de los cuales 86 son paneles sobre la carretera y 128 son paneles de tráfico ubicados en el lado derecho de la carretera. Las tasas de detección y reconocimiento obtenidas se muestran en relación a diferentes rangos de distancias para observar cómo afecta la distancia al panel. Cuando el panel se encuentra a una distancia corta, se detectan el 92% de las palabras, el 67,21% son reconocidas. El 73,35% de los números son detectados, pero solo el 64,13% son reconocidos. El 63,08% de los símbolos son detectados, de los cuales el 80% son reconocidos. A media distancia, el 53,87% de las palabras son detectadas y el 41,94% son reconocidas. El 20,88% de los números son detectados y el 40% reconocidos. El

47% de los símbolos son detectados y el 75,53% reconocidos. A larga distancia, el 73,73% de las palabras son detectadas y el 15,29% reconocidas, el 3,77% de los números son detectados y el 9,68% reconocidos, el 20,76% de los símbolos son detectados y el 84,15% reconocidos.

En 2013, Álvaro González y otros [13] presentaron una mejora de su sistema para reconocer el texto de los paneles de tráfico en las imágenes de *Google Street View*. Su objetivo era generalizar el método propuesto en [12], para ello se detectan los paneles de tráfico y después se reconoce el texto contenido en el panel. En primer lugar, crearon su propio dataset de paneles de tráfico a partir de imágenes de *Google Street View*, que contenía tanto ejemplos positivos como negativos y estaba separado en dos conjuntos disjuntos, entrenamiento y test. Para detectar los paneles contenidos en la imagen utilizaron un algoritmo de detección basado en segmentación por color y la técnica *Bag Of Visual Words* (*BOVW*), que convierte un conjunto de descriptores locales de gran dimensionalidad en un vector individual de dimensionalidad fija para cada una de las imágenes. En primer lugar, se aplicó el algoritmo de segmentación y después se extrajeron características con diferentes técnicas (*SIFT*, *Hue Histogram*, *TCH*) para entrenar dos clasificadores, un *SVM* y *Naïve Bayes* para la detección de los carteles. Una vez que todos los paneles han sido detectados, se aplica el algoritmo de reconocimiento de texto propuesto en [12] para detectar el contenido. Los resultados experimentales se mostraron en función de dos tipos de imágenes, los paneles laterales y los paneles de encima de la carretera y los diferentes tipos de descriptores utilizados. Para los paneles laterales se obtuvo una precisión en la detección de 60,80% con *SIFT*, 84,97% con *HUE Histogram* y 94,68% con *TCH*. Por su parte, para la detección de paneles superiores se lograron unas tasas de precisión de 79,38% con *SIFT*, 84,97% con *HUE Histogram* y 96,38% con *TCH*.

En 2016 Tejendra Panchal y otros [21] abordaron el problema de la localización y reconocimiento de matrículas en video. Para ello se propone un algoritmo que funciona en dos fases, una de detección de la matrícula y otra para el reconocimiento de los dígitos de la matrícula. En la primera fase, se obtienen las imágenes de la cámara y se realiza un preprocesamiento que consiste en el ajuste de la imagen ya que las imágenes pueden aparecer algo rotadas. Una vez preprocesada la imagen se aplica un detector de esquinas de Harris para extraer las características en la imagen. Una vez extraídos los puntos de interés, se aplica un enfoque de la ventana deslizante para encontrar la región más probable donde se encuentra la matrícula. En la segunda fase, una vez detectada ya la matrícula, se realiza la segmentación. Para ello, en primer lugar, se aplica un método de super-resolución para hacer la región de la matrícula sea más grande y facilitar la tarea de segmentación. Después, se realiza un umbralizado adaptativo para un tamaño de ventana 10x10 y se aplican las operaciones morfológicas de erosión y dilatación para eliminar posible ruido en la imagen. Por último, se utiliza el algoritmo *CCA* para encontrar las componentes conexas pertenecientes a la matrícula. Para evaluar la eficiencia y la solidez del método propuesto, se utilizó una base de datos de 65 imágenes de automóviles en diversas condiciones de iluminación, de las cuales 35 eran imágenes sencillas y los 30 restantes eran más complejas. Los resultados se muestran para la detección y segmentación de matrículas. De las 35 imágenes simples, el 100% fueron detectadas y el 97,14% fueron segmentadas correctamente. Por su parte, de las 30 imágenes complejas, el 93,33% fueron detectadas y

el 90% segmentadas correctamente. Por lo tanto, la precisión media del sistema alcanzó un 93,84%.

En 2016 Mariusz Bojarski y otros [19] investigaron acerca de un sistema de conducción autónoma mediante técnicas de "*Deep Learning*" a partir de video, entrenando una *CNN* para mapear los píxeles de una imagen frontal directamente a la dirección del coche. Dicho sistema aprende todo el proceso de procesamiento necesario para conducir un automóvil con tráfico real con unos datos mínimos de entrenamiento. Estos datos únicamente comprenden el ángulo de giro del volante y no incluye otros aspectos como las líneas de las carreteras o las señales. La motivación principal para este trabajo es evitar la necesidad de reconocer características específicas designadas por el hombre, como marcas de carril, barandillas u otros autos. Para la adquisición de los datos se montaron 3 cámaras en un vehículo y se captura el ángulo de dirección aplicado por un conductor humano. Los datos de entrenamiento contienen imágenes individuales obtenidas del vídeo, combinadas con el comando de dirección correspondiente. Entrenar con datos solo del conductor humano no es suficiente. La red debe aprender a recuperarse de los errores. De lo contrario, el coche se desviará lentamente de la carretera. Por ello, a los datos obtenidos se les aplicaron técnicas de *data augmentation*, giros y rotaciones aleatorias para enseñar a la red cómo recuperarse de una posición u orientación deficiente. Las imágenes se le pasan a la *CNN* que calcula un giro de dirección. El giro propuesto se compara con el giro referencia para esa imagen y los pesos de la *CNN* se van ajustando a la salida deseada mediante *backpropagation*. La imagen de entrada se divide en planos *YUV* y se pasa a la red. La red consta de 9 capas, incluida una capa de normalización, 5 capas convolucionales y 3 capas totalmente conectadas. La primera capa de la red realiza la normalización de la imagen. Las capas convolucionales se diseñaron para realizar la extracción de características y se eligieron empíricamente mientras que las capas *fully-connected* están diseñadas para funcionar como un controlador para la dirección. Antes de probar la *CNN* entrenada en carretera, primero se evalúa su rendimiento en un simulador. Una vez obtenidos buenos resultados en la simulación, se realizaron las pruebas en carretera. Como métrica de la evaluación, se calcula el porcentaje de tiempo que el coche conduce de manera autónoma sin intervención humana. Las intervenciones ocurren cuando el vehículo autónomo se desvía de la línea central más de un metro. Los resultados experimentales en carretera demuestran que el vehículo es autónomo el 98% del tiempo.

En 2016 Seokwoo Jung y otros [23] plantearon un sistema de ayuda a la conducción para reconocer señales de tráfico captadas por una cámara frontal montada en el vehículo, por ejemplo, para advertir al conductor si supera la velocidad máxima permitida para la vía por la que circula. Para ello, entrenaron una *CNN LeNet-5* para el reconocimiento de 6 tipos diferentes de señales de tráfico. Antes de reconocer la señal, fue necesario una fase de detección. En la fase de detección, se aplicaron algoritmos de segmentación basados en color y transformada de *Hough* para extraer regiones candidatas de señales de tráfico. La prueba de reconocimiento se realiza en la carretera del campus *KAIST* y se observó un rendimiento casi en tiempo real detectando con precisión todas las señales del trayecto, en total 16 señales detectadas.

En 2017 Yi Wei y otros [27] participaron en el *IEEE Smart World 2017 NVIDIA AI City Challenge*, concurso de I+D donde uno de los problemas a resolver, *AI City Track 1*, consiste

en el desarrollo de un sistema de detección y clasificación de objetos de la calle o tráfico a partir de vídeo. Para ello, construyeron un detector de objetos para la localización y clasificación de vehículos y personas. Dicho detector consistía en la combinación de dos modelos de detección de objetos de vanguardia, como son *R-CNN* [16] y *ResNet* [17], tanto para la detección de objetos como para la extracción de características. Específicamente, extrajeron características de la última capa del bloque "*conv4*" en *Resnet101* y luego dichas características eran usadas por el detector *Faster R-CNN*. Para la implementación, utilizaron la *API* de detección de objetos *Tensorflow* [29] partiendo de un modelo pre-entrenado con los datasets *COCO* y *UADETRAC*. De esta manera, incluyeron más muestras de entrenamiento para las clases raras (incluidas las motocicletas y bicicletas para peatones) en el conjunto de datos *AIC*. Una vez que se obtiene el modelo de detección de objetos inicial, éste se ajusta con precisión al conjunto de datos *AIC* para mejorar la clasificación utilizando el descenso de gradiente estocástico como el algoritmo de optimización con un momentum de 0.9 durante el entrenamiento. La velocidad de aprendizaje se reduce a la mitad a medida que crece la iteración para asegurar una parada temprana. Para mejorar aún más la capacidad discriminativa del detector, se seleccionan manualmente muestras negativas. Este enfoque puede aumentar la capacidad de clasificación para distinguir objetos del fondo. Para la evaluación del sistema, se entrenaron 3 modelos, uno para conjunto de entrenamiento del *AIC* y se utilizó como métrica *Mean Average Precision* (*mAP*). En *AIC480 challenge* este método alcanzó la segunda posición mientras que en *AIC540* y *AIC1080* obtuvieron el tercer puesto.

En 2017 Lu Chi y Yadong Mu [28] propusieron un modelo basado en *Deep Learning* que mapeaba directamente imágenes de entrada a comandos de dirección de un vehículo (ángulos de rueda, frenado o aceleración) a partir de vídeos de conducción reales. En cada acción, el modelo debe tener en cuenta tanto el estado actual como los estados anteriores que memoriza. Esto se logra utilizando redes recurrentes *LSTM* y unidades *LSTM-CONV* en diferentes capas de la red. Para el entrenamiento, además de los fotogramas se debe tener en cuenta la velocidad, el par y el ángulo de la rueda. Durante dicho proceso, se utiliza el error cuadrático entre los datos predichos por la red y los datos de referencia obtenidos durante la conducción real para ir ajustando los pesos mediante *backpropagation*. La red propuesta funciona en dos etapas, una de extracción de características y otra de predicción. Por ello la red es tratada conceptualmente como dos subredes que se encargan de realizar cada una de las etapas del proceso. Los fotogramas de vídeo de entrada se introducen primero en una subred de extracción de características, generando un vector de características de longitud fija que representa el entorno visual y el estado interno de un vehículo, y posteriormente dicho vector se envía a la subred de predicción. La subred de extracción de características utiliza convoluciones espacio temporales *st-conv* en lugar de convoluciones clásicas, lo que permite codificar la información temporal en base a varios fotogramas consecutivos. Dicha información se codifica en un vector de dimensión 128 que agrega información de múltiples escalas a través de un *skip* de manera que cada una de las respuestas en las capas de *st-conv* se alimenta a una *Fully-Connected*. La entrada y la salida de todas las capas de *st-conv* son tensores 4-D, donde las dos primeras dimensiones trazan las posiciones espaciales en una imagen, los tercera indexa diferentes canales de características y la cuarta corresponde a los fotogramas del video. La subred de predicción fusiona varios tipos de información temporal en múltiples capas de la red. En total hay tres

unidades *LSTM* en la subred, una de ellas se encuentra en el núcleo de ésta, cuya entrada es el vector de características de 128-d que se extrae de la subred anterior junto con las acciones de dirección y estado del vehículo. La función objetivo final es el promedio de pérdida en cada fotograma de los tres tipos de predicciones de la red (ángulo, torsión y velocidad). Para evaluar el sistema propuesto, se utilizó un subconjunto del dataset lanzado en 2016 por la empresa *Udacity* en su proyecto de conducción autónoma. Dado que el objetivo era predecir el ángulo de la rueda, el rendimiento del modelo se basó en términos de *RMSE* de los ángulos de la rueda. Los resultados experimentales produjeron un *RMSE* de 0.0637, mejorando los resultados de las demás alternativas.

En 2017 Kaiming He y otros [31] presentaron un sistema, llamado *mask R-CNN* para la segmentación de objetos en imágenes. Dicho enfoque era capaz de detectar los objetos en una imagen y simultáneamente generar una máscara de segmentación para cada instancia. La red utilizada era una extensión de *Faster R-CNN*, que agregaba una pequeña *FCN*, aplicada a cada *ROI* para predecir la máscara del objeto en paralelo con la rama ya existente para la predicción de la clase y la *ROI* del objeto. *Mask R-CNN* funciona en dos etapas. La primera etapa utiliza una *RPN* para proponer *ROIs* de posibles objetos candidatos mientras que en la segunda etapa se extraen características de cada *ROI* candidata utilizando *RoIPool* y se realiza la clasificación y obtención de los *bounding boxes*. Paralelamente, durante esta segunda etapa también se genera la máscara de segmentación para cada *ROI*. Para cada *ROI*, la red genera una máscara para cada una de las posibles clases y se selecciona la máscara de salida en función de la clase predicha por la rama de clasificación. La máscara $m \times m$ se crea utilizando una *FCN*, lo que permite que la máscara mantenga la distribución espacial $m \times m$ sin convertirlo en una representación vectorial sin dimensiones espaciales. Lo que destacó de este trabajo es la capa *RoIAlign*. Debido al uso de *RoIPool* para la extracción de características se introducían desalineaciones entre la *ROI* y las características extraídas, lo que producía un efecto negativo en la predicción de las máscaras. La capa *RoIAlign* permite alinear de nuevo las características extraídas con la *ROI* de entrada utilizando una interpolación bilineal. Para evaluar el sistema, se entrenó la red utilizando 80k imágenes de entrenamiento y 35k imágenes para validación, todas ellas pertenecientes al dataset de *COCO* [14]. Como resultado de la evaluación, el sistema propuesto logró un 35,7% de precisión, superando los resultados de *MNC* y *FCIS*, ganadores de los desafíos de segmentación de *COCO* 2015 y 2016, respectivamente.

En 2018 Tripidok Intasuwan y otros [30] diseñaron un sistema que conectaba automáticamente un panel publicitario con el sitio web del producto al instante para obtener más detalles o información de contacto. Para evaluar el sistema, tuvieron que recopilar un conjunto de datos formado por 500 imágenes de carteles, así como el enlace al sitio web oficial del producto. El método propuesto funciona en tres fases. La primera fase consiste en la detección del objeto contenido en la imagen, para ello el *pipeline* de detección de objetos utilizaba la *API* de *Cloud Vision* [34]. Dicha *API* utiliza *Fast R-CNN* [18] y *SSD* [24] para obtener los *bounding boxes* de los objetos en la imagen junto a sus puntuaciones de confianza. La segunda etapa consiste en la detección y reconocimiento del texto del anuncio, para ello se utilizó *Tesseract OCR* [4] que obtiene las palabras contenidas en la imagen junto a sus puntuaciones de confianza. Durante la tercera etapa, se utilizan las salidas de las dos etapas anteriores para encontrar el sitio web correspondiente a la imagen de entrada. El sistema utilizó el *bounding box* con mayor confianza para buscar con *API*

Google Custom Search el ranking del sitio web basándose exclusivamente en información visual. De forma similar a la búsqueda visual, el sistema envía la etiqueta textual a la *API* para obtener el ranking del sitio web que se basa únicamente en información textual. Finalmente, la clasificación resulta de la combinación lineal ponderada de la búsqueda visual y textual. Para evaluar el modelo de detección de objetos propuesto se utilizó el *dataset MS COCO* [14]. Para ello, se entrenó un modelo con 1000 imágenes para detectar objetos de 4 categorías diferentes (gato, naranja, televisión y oso de peluche). Los bounding boxes con IoU mayor a 0.5 se consideran detecciones válidas. La precisión media conseguida por el sistema de detección para cada una de las categorías fue del 73%. Por su parte, el sistema de reconocimiento de texto alcanzó una precisión entre 65%-71% dependiendo del tamaño de la valla publicitaria.

En 2018 Murhaf Hossari y otros [33] proponen una arquitectura de *Deep Learning* llamada *ADNet* para la detección de anuncios publicitarios en vídeos. El modelo de *ADNet*, se inspiró en la arquitectura *VGG19* que utiliza filtros de convolución 3x3 con una profundidad de hasta 19 capas y se entrenó con un conjunto de datos que constaba tanto de ejemplos positivos como negativos. *ADNet* utiliza los pesos pre-entrenados de *imagenet* de la red *VGG*. Durante el entrenamiento, solamente se modifican los pesos de las 5 primeras capas de la red *VGG* pre-entrenada. Además, se eliminaron las últimas 3 capas del modelo *VGG19* para añadir tres capas *Fully-Connected*. La dos primeras capas *FC* tienen 1024 neuronas con función de activación *relu* mientras que la última capa tiene dos neuronas y utiliza una función de activación *softmax*. También utiliza un factor de *Dropout* de 0.5 para evitar sobreajuste. Para evaluar el rendimiento, se entrenó el modelo de *ADNet* con 18945 imágenes, durante 50 épocas utilizando el descenso de gradiente estocástico como optimizador. Además, se utilizó un *learning rate* 0,0001, un *batch size* de 16 y la función de pérdida *categorical cross-entropy*. Como resultado de la evaluación, el modelo alcanzó una precisión del 94% calculando dicha precisión como (TP + TN) / (TP + TN + FP + FN). Hay que tener en cuenta que la evaluación del sistema se realizó con un *dataset* en el que el tamaño de los carteles publicitarios era superior al 10% del tamaño de la imagen. Además, en dichas imágenes no aparecían carteles con oclusiones y los carteles no podían sobresalir de los límites de la imagen, por lo que el cartel debía aparecer completamente en la imagen. Además, cada imagen solo contenía un cartel. Todos estos factores favorecen a alcanzar una alta precisión en la detección y localización de los carteles publicitarios en las imágenes.

2.3 Bases de Datos

Para evaluar un sistema de detección de objetos es necesario disponer de una gran cantidad datos, en este caso imágenes, ya que dichos sistemas necesitan aprender correctamente las características de los objetos con el objetivo de localizarlos y detectarlos de manera adecuada en las imágenes. En ese sentido, se hacen imprescindibles las Bases de Datos que almacenan grandes cantidades de datos pertenecientes a un mismo contexto para su posterior uso. El objetivo de este trabajo es lograr un sistema de detección que sea capaz de detectar y localizar carteles publicitarios en imágenes estáticas. Debido a la aún poca investigación dedicada a la detección de carteles publicitarios, hasta hace relativamente pocos meses no había ninguna base de datos dedicada específicamente a la detección de carteles publicitarios, ya que son muy pocos los trabajos que centran su investigación en este tipo de problemas. Por estos motivos, en el estado del arte se ha intentado mostrar diferentes trabajos centrados en la detección de objetos de aspecto similar a los paneles de

publicidad como pueden ser los paneles de trafico de las carreteras o simplemente las señales de tráfico, ya que además para este tipo de objetos sí que existen gran variedad de bases de datos.

Además, en muchos casos, debido a la no existencia de *datasets* específicos para un problema, se ha optado por crear un *dataset* propio recopilando datos a partir de otros *datasets* o simplemente obteniendo imágenes a través de *Google Street View*. Varios ejemplos de ello son [12, 13], que utilizaron *Google Street View* para recopilar un conjunto de imágenes de paneles de trafico de las carreteras españolas para evaluar su sistema de reconocimiento de texto en paneles de tráfico. También el trabajo [33] se creó su propio *dataset* de paneles publicitarios seleccionando imágenes de diferentes *datasets* como *COCO* [14] o *Mapillary Vistas* [35].

En este trabajo se ha optado por crear un *dataset* propio a partir de imágenes de *Google Images* así como de *Google Street View*. Las imágenes recopiladas presentan diferentes condiciones de iluminación, así como climatológicas para proporcionar variabilidad al *dataset*. Además, los carteles han sido capturados desde diferentes perspectivas y a diferentes distancias, de manera que se consiguen diferentes tamaños de cartel. Además, al tratarse de publicidad, es muy importante que se detecten los carteles incluso cuando se produzcan oclusiones, por los que algunos carteles aparecen ocultados por objetos del entorno como vehículos, árboles o peatones. Posteriormente se describirá el *dataset* con más detalle.

A continuación, se describen muy brevemente cada una de las bases de datos utilizadas por los trabajos de detección de objetos mencionados en el estado del arte, así como su uso en cada uno de los trabajos mencionados.

2.3.1 Cartociudad

Cartociudad [37] es una base de datos oficial de la red vial española que incluye todo tipo de rutas, autopistas, vías urbanas y calles. Esta base de datos es apoyada por el gobierno español y por diferentes instituciones públicas estatales españolas, además, es actualizada cada poco tiempo con las coordenadas *GPS* (latitud y longitud) de donde se tomaron cada una de las imágenes. El servicio de geocodificación inversa permite, a través de una simple petición al servidor de Cartociudad con la posición *GPS* como entrada, obtener un documento en formato XML que indica diferentes características del punto que se solicita, como por ejemplo el nombre de la carretera o calle, código postal, la provincia entre otros.

En trabajos como [12, 13] utilizan las imágenes de los paneles de carreteras con sus respectivas coordenadas de latitud y longitud para realizar una petición a Cartociudad y obtener el nombre de la provincia donde se encuentra el panel de tráfico.

2.3.2 Udacity Driving Dataset

En 2016 la empresa *Udacity* lanzó un proyecto de conducción autónoma de código abierto y organizó una serie de competiciones y desafíos para resolverlo. En consecuencia, liberó un conjunto de datos conocido como *Udacity Driving Dataset* que contiene 223 GB de videos y datos etiquetados con más de 70 minutos de conducción real en Montain View en varios días alternos con diferentes condiciones climatológicas, días soleados y días cubiertos. Junto con los fotogramas capturados por las cámaras se proporcionan la latitud, longitud, marcha,

freno, aceleración, ángulos de dirección y velocidad. Esta información es gratuita para cualquier persona, en cualquier parte del mundo.

En trabajos como [28] se utiliza dicho *dataset* para entrenar un sistema de *Deep Learning* que mapeaba un conjunto de imágenes de entrada a los correspondientes comandos de dirección de un vehículo.

2.3.3 AI City Dataset

Desde 2017, *NVIDIA* ha lanzado anualmente una serie de competiciones y desafíos en el área de la visión por computador para ayudar a abordar problemas de actualidad y fomentar la investigación y el desarrollo de técnicas de *Deep Learning*. En consecuencia, han creado el dataset *NVIDIA AI City*, formado por más de 80 horas de vídeos de tráfico con diferentes resoluciones, en diferentes condiciones de iluminación y con sus respectivas etiquetas.

Los videos han sido grabados en diferentes condiciones ambientales y de iluminación, que van desde el día a la noche. Las etiquetas para los conjuntos de datos incluyen: automóvil, SUV, camión pequeño, camión mediano, camión grande, peatón, autobús, furgoneta, grupo de personas, bicicleta, motocicleta, señal de tráfico verde, señal de tráfico roja, señal de tráfico amarilla. Para la tarea de detección de objetos, el conjunto de datos se divide en 3 subconjuntos según la resolución de vídeo. El conjunto de datos *AIC480* contiene vídeos con una resolución de 720x480 píxeles, *AIC1080* contiene vídeos con una resolución de 1920x1080 píxeles y *AIC540* se obtiene mediante el submuestreo de los fotogramas de *AIC1080*.

En trabajos como [27] utilizan dicha base de datos para diseñar y evaluar un sistema de detección de objetos de vehículos y personas.

2.3.4 MS COCO

En el año 2015, *Microsoft* publicó un *dataset* de acceso público con el objetivo de hacer avanzar el estado del arte respecto al problema de reconocimiento de objetos en el contexto de comprensión de la escena. Para ello, se recopilaron imágenes de escenas complejas que contienen objetos comunes en su contexto natural. Los objetos se etiquetaron utilizando segmentaciones por instancia para ayudar a la localización no precisa de los objetos. Dicho conjunto de datos está formado por 328 mil imágenes diferentes que contienen un total de 2.5 millones de instancias etiquetadas que pertenecen a 91 clases de objetos diferentes. La creación de nuestro conjunto de datos se basó en una amplia interfaz de colaboración entre usuarios y trabajadores. El *dataset* está pensado para la detección de categorías, localización de instancias y segmentación de instancias.

El trabajo [27] realizó *fine-tuning* a partir de un modelo pre-entrenado con el *dataset* de *COCO* para construir un detector de vehículos y personas. En [30], se utilizó el *dataset* para entrenar y evaluar un detector de objetos contenidos en carteles publicitarios. En [31] se utilizó para entrenar y evaluar una red de segmentación a nivel de píxel como *Mask R-CNN* y en [33] utilizaron algunas imágenes del *dataset COCO* para construir su propio *dataset* de detección de paneles publicitarios.

2.3.5 Mapillary Vistas Dataset

Mapillary Vistas es un conjunto de datos de imágenes a nivel de calle para la comprensión de escenas de todo el mundo y potenciar la movilidad y el transporte autónomo a escala

global. El *dataset* está orientado tanto para detección de objetos como para segmentación, si bien para el propósito de este trabajo nos interesa solamente la detección de objetos. En ese sentido, *Mapillary Vistas* hace hincapié en el reconocimiento de instancias individuales, tanto de objetos estáticos de la calle (letreros, postes, etc) como de elementos dinámicos (automóviles, peatones, ciclistas). El objetivo principal es la detección de objetos críticos para agentes que actúan de forma autónoma como automóviles o robots de transporte. Recientemente, en el 2018, *Mapillary Research* se unió a las populares tareas de reconocimiento de *COCO* con el conjunto de datos de *Mapillary Vistas*.

El dataset cuenta con más de 25000 imágenes de alta resolución, de las cuales 18000 se utilizan para entrenamiento, 2000 para validación y 5000 para test. Estas imágenes pertenecen a 152 categorías de objetos diferentes, de las cuales 100 clases están preparadas para la segmentación de objetos. Para capturar las imágenes, se han utilizado una amplia gama de sensores en diferentes condiciones de iluminación (sol, lluvia, nieve, niebla, neblina) y en diferentes momentos del día (amanecer, luz del día, anochecer e incluso de noche). Además, cada una de las imágenes han sido tomadas desde diferentes puntos de vista (desde carretera, aceras y fuera de carretera) y en diferentes partes del mundo.

En trabajos como [33, 36] se ha utilizado este *dataset* para seleccionar algunas imágenes donde aparecían carteles publicitarios para crear su propio *dataset* específico de carteles publicitarios.

2.3.6 ALOS Dataset

Recientemente, en abril de 2019, se ha anunciado la publicación de un *dataset* de carteles publicitarios [36] a gran escala de carteles publicitarios capturados en escenas al aire libre. El *dataset* se ha llamado *ALOS*, que significa *Advert Localization in Outdoor Scenes*. El conjunto de datos *ALOS* contiene imágenes del *dataset Mapillary Vistas*, que almacena imágenes con licencia que se pueden filtrar para incluir solo las imágenes de una categoría específica, en este caso *Billboard*.

El conjunto de datos contiene imágenes seleccionadas que deben cumplir una serie de requisitos, como tener buen contraste, texto legible y buena resolución de imagen. La forma de la cartelera debe ser un polígono convexo de 4 lados y la distorsión por perspectiva debe ser mínima, siendo 15 grados la máxima distorsión permitida. Por otra parte, la cartelera no debe estar ocluida en gran medida, el cartel debe ser visible sin esfuerzo humano. La cantidad de oclusión no debe ser superior al 10% de la cartelera. Basándose en dichas restricciones, se ha construido el *dataset* que contiene 8065 imágenes. La cartelera más pequeña cubre el 0,13% del área de la imagen; mientras que la cartelera más grande captura el 77.08% de su área de imagen. La mayoría de las imágenes tienen una sola cartelera. Finalmente, también se incluyen imágenes en este conjunto de datos, cuyos carteles están parcialmente ocluidos.

2.4 Base de Datos del Proyecto

Actualmente, con el auge de las técnicas de aprendizaje automático se hace necesario el uso de grandes bases de datos que almacenen miles o millones de imágenes para poder entrenar sistemas basados en técnicas de *Deep Learning*. En algunos casos, como por ejemplo en este trabajo, se hace complejo abordar un problema de detección de carteles publicitarios con una técnica de este tipo debido a la falta de bases de datos sobre este tipo

de objetos. Para poder usar diferentes arquitecturas de *Deep Learning* en la resolución de dicho problema, ha sido necesario crear una base de datos propia de carteles publicitarios en escenas exteriores. Es importante, además de la cantidad de imágenes, dotar al *dataset* de gran variabilidad en las mismas para que el sistema aprenda correctamente a discriminar en las imágenes entre objetos que son carteles y objetos que no lo son.

La base de datos creada está formada por aproximadamente 5800 imágenes que contienen 6380 carteles etiquetados, ya que cada imagen contiene al menos un cartel publicitario con sus correspondientes etiquetas en formato *XML*. Todos los carteles contenidos en las imágenes han sido etiquetados con la herramienta *VGG Image Annotator*, que permite etiquetar diferentes regiones de la imagen a partir de diferentes formas poligonales. De esta manera, se etiquetan los carteles seleccionando cada una de las esquinas del cartel para obtener sus coordenadas (X,Y). Esta manera de etiquetar los datos permite adaptar dicha información para diferentes arquitecturas de red como son *SSD* o *YOLO* entre otras. Todas las imágenes contenidas en el dataset han sido recopiladas de dos fuentes diferentes, *Google Images* y *Google Street View*. A diferencia de otros *datasets*, en el que los carteles deben ocupar más de un 10% del área total de la imagen, a la hora de recopilar las imágenes no se ha aplicado ninguna restricción por tamaño de cartel. Además, tampoco se han descartado imágenes con oclusiones o cuyo cartel esté parcialmente fuera de la imagen como si hacen otras bases de datos. Para dotar a nuestro conjunto de datos con la variabilidad necesaria se han tenido en cuenta los siguientes aspectos:

- ***Área total de cartel:*** con el objetivo de que la red aprenda a discriminar entre carteles de diferentes tamaños y a diferentes distancias, se construido el *dataset* con carteles cuya área total va desde el 2% hasta el 90% de la imagen.
- ***Condiciones de iluminación:*** al tratarse de carteles exteriores, es importante que el *dataset* contenga imágenes en diferentes condiciones climatológicas y de iluminación. Para ello, se han recopilado imágenes nocturnas, diurnas, en días soleados, con lluvia, con niebla o con nieve entre otras.
- ***Oclusiones:*** Al situarse en un espacio público, es muy habitual que se produzcan oclusiones por viandantes, vehículos u otros objetos, por lo que hay que recolectar imágenes que cumplan dichas condiciones para que la red aprenda a discriminar en situaciones como éstas. Para ello, el *dataset* contiene imágenes de carteles ocluidos por viandantes, vehículos y diferentes tipos de mobiliario público.
- ***Perspectiva de imagen:*** Dependiendo del sistema de adquisición de las imágenes (que podría ser una cámara fija o bien algo móvil), es posible que un mismo cartel sea capturado desde diferentes perspectivas por lo que el *dataset* contiene imágenes de carteles en diferentes perspectivas.
- ***Presencia objetos similares:*** en la vía pública los carteles pueden compartir espacio con objetos muy similares como, carteles de obra, de aparcamiento o el frontal de un autobús, que resultan ser muy similares en forma a los cartel publicitarios. Por ello, nuestro conjunto de datos de datos contiene imágenes con objetos similares a un cartel publicitario, como por ejemplo carteles de obra, señales de tráfico y autobuses de manera que durante el entrenamiento se pueda aprender a discriminar carteles frente a este tipo de objetos parecidos.

En una primera fase inicial, se seleccionaron manualmente 1800 imágenes de carteles publicitarios en exteriores. Una vez etiquetados cada uno de los carteles contenidos en las imágenes, se realizó un análisis en función del tamaño de los carteles. Debido a que el *dataset* estaba desequilibrado, ya que la mayoría de los carteles ocupaba un área total menor al 10% de la imagen, se aplicó *Data Augmentation* con el objetivo de equilibrar el *dataset* a la par que aumentamos el tamaño de la muestra. Para tal efecto, se aplicaron algunas operaciones geométricas a las imágenes; en particular, rotaciones de -5 grados a 5 grados, y diferentes escalas zoom sobre las imágenes de entre 10%-20%. Aplicando dichas transformaciones se ha conseguido un conjunto de datos de aproximadamente 5.800 imágenes para entrenamiento y validación. Además, como muchas arquitecturas de *Deep Learning* requieren un tamaño de entrada fijo, todas las imágenes que conforman el *dataset* han sido re-escaladas a un tamaño fijo de 512x512 píxeles. Este re-escalado mantiene el *aspect ratio* de las imágenes ya que de lo contrario se estarían deformando los objetos que aparecen en ella.

El *dataset* creado se puede separar en función de diversos factores, por ejemplo, se pueden separar los datos en datos empleados para entrenamiento, validación y test o bien separar los datos por tamaño de cartel. Un aspecto importante a tener en cuenta es que el conjunto de datos de test debe ser completamente independiente del resto, ya que la red no puede conocer en ningún momento las imágenes de test a lo largo de su proceso de aprendizaje.

Al separar el *dataset* en entrenamiento, validación y test, hay que tener en cuenta que las 5800 imágenes se han empleado para entrenamiento y validación, utilizando aproximadamente el 95% de las imágenes para entrenamiento y el 5% restante para validación. Por su parte, el conjunto de test es un conjunto de datos independiente formado por 132 imágenes. A continuación, se puede observar una tabla que muestra los datos utilizado en cada subconjunto:

Subconjunto	**Entrenamiento**	**Validación**	**Test**
Nº de imágenes	**5510**	**290**	**132**

Tabla 1 Distribución de las imágenes del dataset

Al separar los datos por tamaño, podemos observar que el mayor número de carteles ocupa un área total de la imagen menor al 10%, los cuales no son considerados en otros *dataset*. Además, se ha intentado que el número de carteles se encuentren más o menos equilibrado por tamaño, es decir que haya el mismo número de carteles de un tamaño que de otro, si bien esto es algo bastante difícil de conseguir. En total, entre las 5800 imágenes de entrenamiento y validación, hay etiquetados 6380 carteles, los cuales se muestran separados por tamaño a continuación:

Tamaño	0 - 10%	10 - 20%	20 - 30%	30 - 40%	40 - 50%	50 - 60%	60 - 70%	70 - 80%	80 - 90%
Nº de carteles	**1300**	**580**	**800**	**610**	**600**	**920**	**920**	**580**	**70**

Tabla 2 Número de carteles distribuidos por tamaño

3. Solución Propuesta

A continuación, se explica detalladamente cada una de las soluciones de *Deep Learning* propuestas para la resolución del problema de detección de carteles publicitarios en imágenes estáticas.

3.1 Descripción General

En la actualidad, a la hora de abordar un problema de detección de objetos, a todo el mundo le viene a la mente el paradigma de *Deep Learning* y las redes neuronales, que tan de moda están en nuestros días. En este trabajo, basado en la detección y localización de carteles publicitarios en imágenes estáticas, también se ha decido utilizar una solución basada en *Deep Learning*. Para ello, se proponen dos arquitecturas de red diferentes para la resolución de un mismo problema utilizando la misma base de datos, lo que permite realizar una comparativa entre ambos enfoques en idénticas condiciones.

La elección de este tipo de solución viene motivada por el hecho de que recientemente otros autores ya han utilizado arquitecturas de *Deep Learning*, como redes neuronales o redes de convolución, en problemas de Visión por Computador y detección automática de objetos obteniendo resultados sorprendentemente buenos. Además, en el estado del arte se puede observar cómo han ido evolucionando el tipo de soluciones a lo largo de los años, desde el uso de técnicas de visión más clásicas a principios de la década del 2000 hasta el uso de técnicas de *Deep learning* en la actualidad.

Antes de que las imágenes con carteles publicitarios sean procesadas por la red neuronal durante el entrenamiento, el único preproceso necesario sobre dichas imágenes es que todas ellas tengan exactamente el mismo tamaño. Como durante la creación de la base de datos, todos los datos adquiridos han sido re-escalados al mismo tamaño, entonces no se requerirá ninguna fase de preprocesado de los datos durante la fase de aprendizaje del sistema. A continuación, se explica minuciosamente cada una de las partes de un sistema de detección y localización de objetos mientras que en el siguiente apartado se explicará en detalle en que consiste cada uno de los algoritmos utilizados para ello.

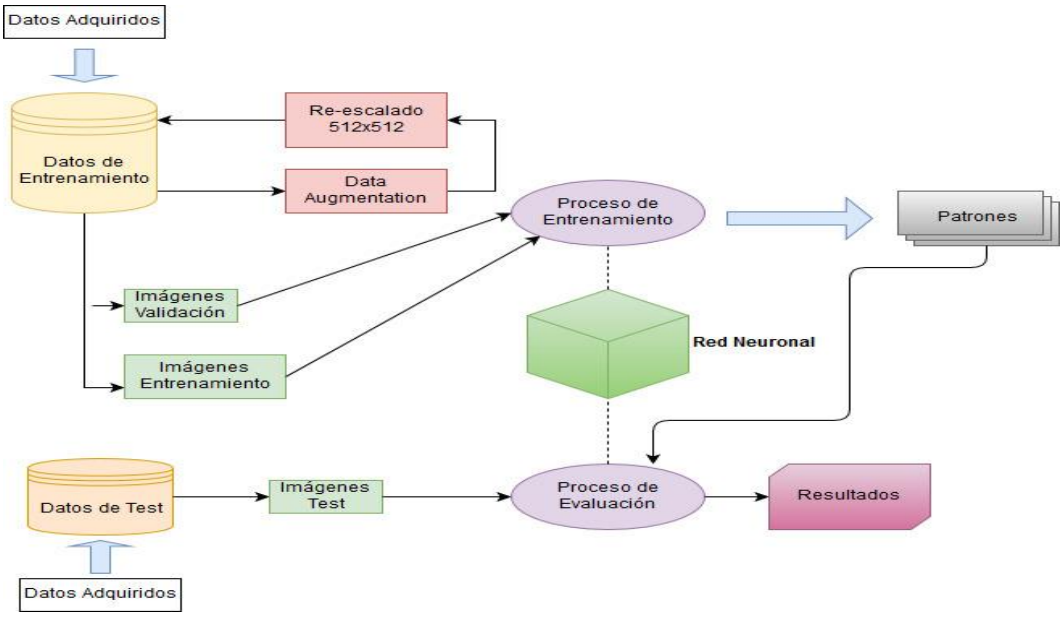

Ilustración 3 Funcionamiento general del sistema

Los modelos computacionales de *Deep Learning* imitan las características del sistema neuronal humano, de manera que en el interior del sistema existen multitud de unidades de proceso que se especializan durante la fase de aprendizaje en la detección de determinadas características ocultas en los datos de entrada. Es decir que los algoritmos son capaces de aprender de manera autónoma obteniendo ellos mismos las conclusiones acerca de la estructura embebida en los datos. Todas las maneras de definir *Deep Learning* tienen dos cosas en común:

- Múltiples capas de procesamiento no lineal
- Aprendizaje de características en cada una de las capas, formando estas una jerarquía desde un nivel de abstracción de más bajo a más alto (en las capas más profundas).

La principal diferencia entre los algoritmos de *Deep Learnin*g respecto a otros tipos de algoritmos de aprendizaje menos profundo reside en el número de capas, así como en el número de transformaciones aplicadas a la señal mientras se propaga la información desde las capas más externas hasta la capa de salida. Cada una de estas capas contiene una serie pesos y umbrales que se van modificando a lo largo del entrenamiento hasta aprender unos valores óptimos para la tarea objetivo.

Las redes neuronales están formadas por un conjunto de elementos, llamados neuronas, que se encuentran conectados entre sí y que trabajan conjuntamente con el objetivo de extraer una serie de patrones característicos de las imágenes de entrada. Poco a poco, durante el proceso de aprendizaje, las neuronas van creando y reforzando dichas conexiones para "aprender" el conjunto de características más relevantes de las imágenes de entrenamiento.

La salida de la red dependerá del tipo de problema que se necesite abordar. Por ejemplo, la salida para un problema de clasificación será una lista de valores que representan la probabilidad para cada una de las clases. Suponiendo que tenemos un problema de clasificación con tres clases diferentes, la salida [0.001, 0.998, 0.001] indicaría que el objeto de la imagen pertenece a la clase 2, ya que es el de mayor confianza.

Por su parte, la salida de los problemas de detección de objetos es un poco más compleja, ya que además de la clase es necesario indicar la localización del objeto dentro de la imagen. La salida este tipo de problemas viene representado por una estructura de salida como la siguiente $Y = [P_C, B_X, B_Y, B_{w,}\ B_H,\ C_1, C_2,\ C_3]^T$ donde:

- $\boldsymbol{P_C}$ indica si se detecta algún objeto, 1 significa que se detecta y 0 indica que se ignorará el resto de la salida.
- $\boldsymbol{B_X}$ hace referencia a la coordenada X del centro del objeto.
- $\boldsymbol{B_Y}$ hace referencia a la coordenada Y del centro del objeto.
- $\boldsymbol{B_H}$ hace referencia a la altura del *Bounding Box*.
- $\boldsymbol{B_W}$ hace referencia a la altura del *Bounding Box*.
- $\boldsymbol{C_1}..\boldsymbol{C_X}$ son los valores de confianza para cada una de las clases.

En este trabajo, para resolver el problema de la detección de carteles publicitarios, se van a utilizar dos arquitecturas de *Deep Learning* que permiten trabajar en tiempo real, como son *Single Shot Detector* y *You Only Look Once*, más conocidas como *SSD* y *YOLO*

respectivamente. Una de las principales características de estas redes es el uso de capas de convolución para la extracción de características de las imágenes. Lo que diferencia a las redes neuronales de convolución es que suponen explícitamente que las entradas son imágenes y que aplican una operación llamada convolución. Esta operación recibe como entrada una imagen y aplica sobre ella un *kernel* para obtener un mapa de características de la imagen eliminando gran cantidad de información irrelevante. A medida que avanza el entrenamiento, el *Kernel* irá mejorando en el filtrado de una imagen para obtener información relevante. Este proceso es automático y se denomina aprendizaje de características. Este proceso generaliza automáticamente para cada tarea y tan sólo hay que entrenar la red para encontrar nuevos *kernels* que sean relevantes para una nueva tarea.

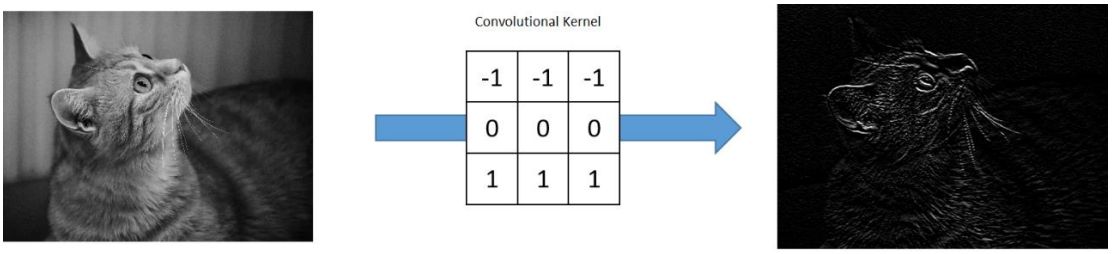

Ilustración 4 Aplicación de kernel de convolución a una imagen

Una red neuronal de convolución (*CNN)* reduce gradualmente el tamaño del mapa de características y aumenta la profundidad a medida que la información se propaga hacia las capas más profundas. Las capas más profundas cubren campos receptivos más grandes y construyen representaciones más abstractas, mientras que las capas poco profundas cubren campos receptivos más pequeños. A la hora de utilizar esta información, se usan las capas poco profundas para predecir objetos pequeños y las capas más profundas para predecir objetos grandes. A continuación, se muestra detalladamente la arquitectura y funcionamiento de ambos tipos de redes utilizadas en este proyecto:

3.1.1 Single Shot Multibox Detector (SSD)

Single Shot Multibox Detector es un algoritmo de *Deep Learning* puramente convolucional usado para la detección de objetos en tiempo real. A partir de su nombre, se puede inferir el funcionamiento general de este tipo de arquitectura. *Single Shot* significa que las tareas de localización y detección se realizan en una sola pasada de la red. Por su parte, *Multibox* se refiere a una técnica de regresión para obtener los *bounding box* de las imágenes mientras que *Detector* indica que la red es un detector de objetos capaz de clasificar los objetos detectados.

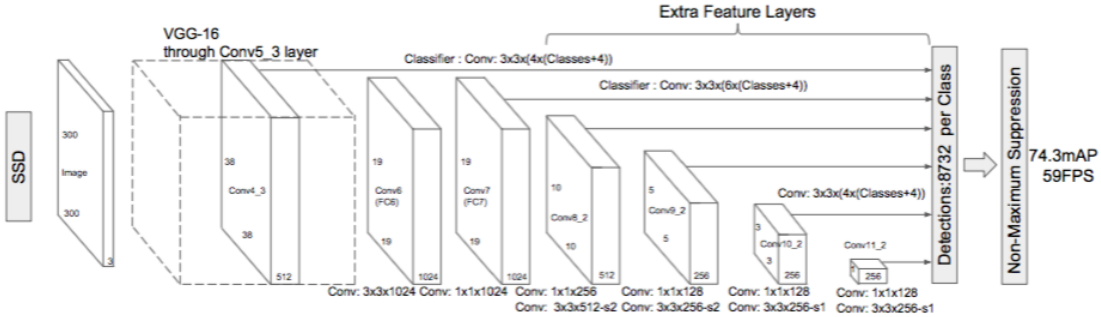

Ilustración 5 Arquitectura de red Single Shot Detector

La arquitectura *SSD* está basada en la red *VGG-16*, si bien se sustituyen las capas "*fully-connected"* de *VGG* por un conjunto de capas convolucionales que permiten extraer características a diferentes escalas y disminuir progresivamente el tamaño de entrada en cada subsiguiente capa. La razón principal para usar *VGG-16* como red base se debe a la idea de *Transfer Learning* y el uso de una red pre-entrenada en un gran *dataset* para mejorar los resultados. El gran rendimiento de *VGG* en tareas de clasificación hace de esta red una buena base de partida para la arquitectura *SSD*.

Como se puede observar en la ilustración 5, la primera capa de convolución genera un conjunto de mapas de características de tamaño 38x38 y una profundidad de 512. Cada punto en el mapa de características cubre una parte de la imagen original y cada uno de los 512 canales son las características de cada punto. Este mapa de características se utiliza para hacer una clasificación de imágenes y predecir la etiqueta. Además, aplicando regresión se pueden obtener los *bounding boxes* para los objetos pequeños contenidos en la imagen. El segundo conjunto de mapas de características, de tamaño de 19x19, se utiliza para la detección de objetos ligeramente más grandes, ya que los puntos de las características cubren campos receptivos más grandes. En la última capa, solo hay un punto en el conjunto de mapas de características que es utilizado para la detección de objetos grandes. Las convoluciones 1x1 ayudan en la reducción de la profundidad, ya que el número de mapas de características disminuye manteniendo el ancho y alto de mapa de éste.

Para cada punto del mapa de características se calculan una serie de *bounding boxes* predeterminados que son conocidos como *anchors*. Es recomendable elegir un conjunto variado *anchors* a diferentes escalas y con diferente ratio de aspecto con el objetivo de detectar el mayor número de objetos posibles. Durante el entrenamiento, para cada punto del mapa de características, se generan una serie de *anchors* que se utilizan para hacer coincidir con la *ROI* del *groundtruth* y calcular las etiquetas y *bounding boxes* en la imagen. El criterio utilizado hacer coincidir un anchor con la *ROI* del *groundtruth* es *Intersection Over Union (IoU)*. A mayor superposición entre anchor y *groundtruth*, mayor será el valor del *IoU*. Originalmente estos *anchors* se seleccionan de manera que el *IoU* sea mayor a 0.5, un valor de 0.5 aún no es lo suficientemente bueno, pero sin embargo proporciona un punto de inicio sólido para el algoritmo de regresión que calcula los *bounding boxes*. En la última capa de la red, como resultado de la clasificación se obtiene una salida de tamaño *4 x (nº clases + 1 fondo)* para cada punto del mapa de características. Este número 4 hace referencia al número de *anchors* utilizado.

Durante la fase de entrenamiento se intenta hacer *matching* entre los *anchors* y las *ROI* del *groundtruth*. Cuando múltiples *anchors* coinciden con la *ROI* del *groundtruth* se considera como un objetivo positivo. Sin embargo, la mayoría de *anchors* tendrá un *matching* con un *IoU* bajo, considerado como objetivos negativos, de manera que el conjunto de datos este desequilibrado. Para hacer que el conjunto de datos sea más equilibrado se utiliza la técnica conocida como *negative hard mining*. La idea de esta técnica es, en lugar de utilizar todas las predicciones, mantener una proporción de ejemplos negativos a positivos de alrededor de 3:1. De esta manera, se conservan ejemplos negativos que son necesarios para que la red aprenda y se le indique explícitamente lo que constituye una detección incorrecta.

Cuando la red recibe una imagen, todos los *anchors* tendrán un conjunto de *bounding boxes* y etiquetas asociadas. A medida que se incrementa el número de *matching* entre los *anchors*

y una *ROI* de *groundtruth*, también aumenta el número de objetivos positivos en la imagen. Durante la inferencia, en la imagen se pueden producir múltiples previsiones que predicen un mismo objeto. Para eliminar estos duplicados se utiliza el algoritmo *Non Max Supression* (*NMS*). *NMS* descarta los *bounding boxes* con menor probabilidad, consideradas ruidosas, y mantiene los que tienen un *IoU* alto.

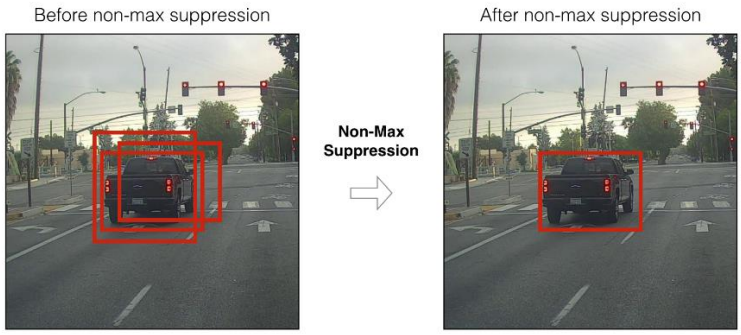

Ilustración 6 Algoritmo Non Max Supression

Al tratarse de un problema de detección de objetos es necesario una función de pérdida o *loss* adecuada tanto para la tarea de clasificación como la de predicción de los *bounding boxes*. Para ello, *SSD* utiliza como función de perdida una combinación de dos criterios, la perdida de clasificación y la perdida de regresión:

- **Perdida de clasificación:** mide el nivel de confianza en las predicciones de cada *bounding box*. Para el cálculo de dicho valor se utiliza *Categorical Cross-Entropy*.
- **Perdida de regresión:** mide la distancia entre los *bounding box* predichos por la red respecto a los *bounding box* reales del *groundtruth*. Para ello, se utiliza la medida *L2-Norm*.

3.1.2 You Only Look Once (YOLO)

You Only Look Once es otro tipo de arquitectura de *Deep Learning*, que al igual que *SSD*, es capaz de realizar detección de objetos en tiempo real. Como su propio nombre indica realiza las predicciones sobre las imágenes de entrada en una sola pasada, lo que hace que la ejecución de este modelo de *Deep Learning* sea muy rápida. Este tipo de arquitectura presenta algunos conceptos novedosos que son exclusivos del algoritmo *YOLO*, pero otros se comparten con otros modelos de *Deep Learning* orientados a la detección de objetos.

Uno de estos conceptos únicos de *YOLO* son las rejillas. El concepto rejilla consiste en dividir las imágenes en una cuadrícula de celdas de tamaño *NxN*, por ejemplo, 19x19. Para cada celda de la cuadrícula, se calculará un vector de salida Y con el siguiente formato $Y = [P_C, B_X, B_Y, B_{w,}\ B_H,\ C_1, C_2,\ C_3]^T$ explicado anteriormente.

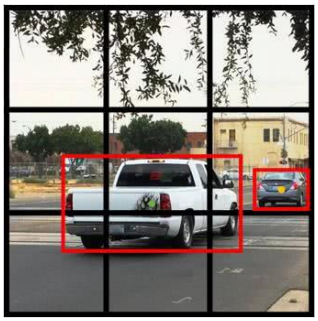

Ilustración 7 Imagen dividida en rejilla 3x3

En la ilustración 7 se muestra una imagen de ejemplo dividida en una cuadrícula de tamaño 3x3. Habitualmente, el tamaño de la cuadrícula es mayor, 17x17 ó 19x19, ya que cuantas más celdas tengamos, mayor será la precisión de la detección. En este ejemplo, aparecen dos detecciones, y para cada *bounding box* se calcula el centro de la detección, lo que permite asociar la detección con la correspondiente celda. Como resultado, se obtiene un vector de salida para cada una de las celdas en la que se encuentra dividida la imagen.

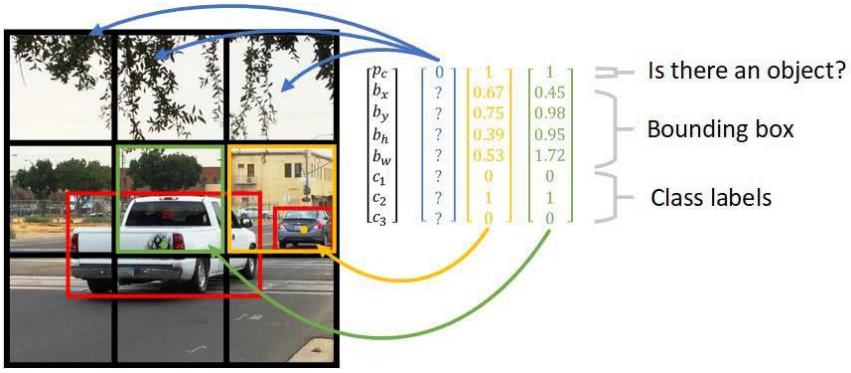

Ilustración 8 Vectores de las detecciones en la imagen

En color azul, se muestran los vectores para las celdas que no contienen ninguna detección, en dichos vectores el valor P_C es 0, lo que indica que el resto de los valores del vector no se tendrán en cuenta ya que no poseen ninguna detección. Por su parte, en color amarillo y verde se muestran las salidas para las celdas que si contienen detecciones. En ellas se muestra el valor P_C a 1, y el resto de los valores indicaran la posición del objeto dentro de la imagen, así como la clase a la que pertenece el objeto.

En este ejemplo solamente hay 3 clases, pero se podrían tener más clases, lo que implicaría que el tamaño del vector de salida aumentaría y sería 5 + Nº de clases. Además, teniendo en cuenta que la imagen se divide en una cuadrícula *NxN*, la salida de destino que combina todas las celdas de la cuadrícula tendría un tamaño de *N x N x (5 + Nº de clases).*

Este tipo de enfoque tiene una limitación, y es que al separar la imagen únicamente en celdas, cuando hay dos objetos en la misma celda es difícil determinar para que objeto se va a calcular el vector de salida, ya que por cada celda únicamente se tiene un solo vector. Para solucionar dicho problema se utiliza el concepto de *anchors*, también utilizada por la arquitectura *SSD* explicada anteriormente.

Los *anchors* permiten al algoritmo *YOLO* detectar múltiples objetos centrados en una misma celda de la imagen. La idea del concepto anchor es agregar una dimensión más al vector de salida al definir un número de anchor predeterminado. De esta manera, es posible asignar un objeto a cada *anchor* y localizar múltiples objetos en una misma celda. Ahora para cada celda se definirán *N anchors* que actúan como detecciones individuales, lo que significa que en cada celda se podrán detectar hasta *N* objetos diferentes. Por ejemplo, en caso de elegir un valor de 2 *anchors* por cada celda, el vector de salida sería $Y = [P_C, B_X, B_Y, B_{w,}\, B_H,\ C_1, C_2,\ C_3, P_C,\ B_X, B_Y, B_{w,}\, B_H,\ C_1, C_2,\ C_3]^T$. Como se puede observar, ahora el vector permite dos detecciones para una misma celda. La intuición es que a la hora de tomar la decisión sobre qué objeto asocia con un *anchor*, se observa la forma de dicho objeto de manera que éste quedará asociado al *anchor* que mejor se ajuste a su forma. Otra razón para elegir una gran variedad de *anchors* es permitir que el modelo se especialice mejor y sea capaz de detectar objetos de diferentes formas. Como medida de similitud a la hora de asociar un *anchor* con un objeto, se utilizará *IoU*, la cual ya era utilizada por la arquitectura *SSD* para el mismo propósito.

Es muy habitual que un mismo objeto tenga asociado múltiples *bounding boxes*. En ese sentido, *YOLO* utiliza el algoritmo *Non Max Supression* (*NMS*) que descarta los *bounding boxes* ruidosos que se generan alrededor del *bounding box* principal. Este algoritmo ya era utilizado por *SSD* y actúa en tres fases. En la primera fase se descartan todas las *bounding boxes* cuyo valor de P_C es menor o igual a 0.6. En la segunda fase se selecciona el *bounding box* con el valor de P_C más alto. Por último, en la tercera fase, se descartan el resto de *bounding boxes* con *IoU* mayor o igual a 0.5. De esta manera se logra pasar de múltiples detecciones sobre un mismo objeto a una única detección sobre dicho objeto. Antes de aplicar *NMS* para limpiar las predicciones, *YOLO* utiliza los valores de confianza para cada una de las clases para filtrar los *bounding boxes*.

A continuación, se muestra un ejemplo gráfico completo del proceso de inferencia de *YOLO* aplicado a una imagen de ejemplo:

1. La red recibe una imagen pre-procesada sobre la cual se va a realizar el proceso de inferencia para obtener los objetos detectados en la imagen.

Ilustración 9 Imagen sobre la que se realizan las detecciones

2. Se calculan los vectores de salida para cada una de las celdas en las que se ha dividido la imagen de entrada. En este caso, la imagen se ha dividido en una cuadrícula de tamaño 3x3 con dos *anchors* para cada celda.

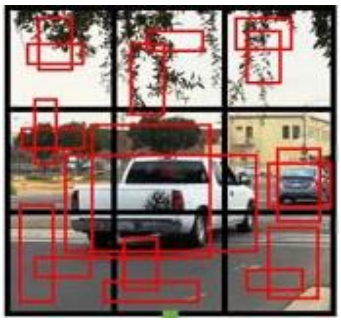

Ilustración 10 Vectores de salida sobre la imagen

3. Se aplica un filtro a partir de los valores de confianza para cada una de las clases, de manera que se descartan los *bounding boxes* con un valor de confianza bajo.

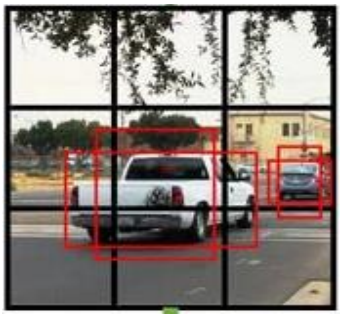

Ilustración 11 Filtrado en base al nivel de confianza

4. Se aplica *NMS* para seleccionar únicamente un *bounding box* por objeto detectado.

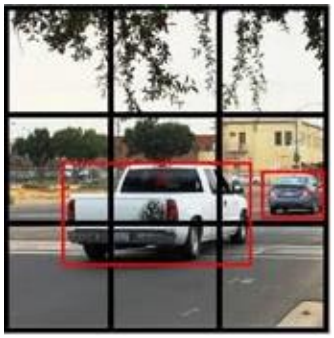

Ilustración 12 Resultado de aplicar algoritmo NMS

3.2 Solución Especifica SSD

La primera solución propuesta al problema de la detección de carteles publicitarios en imágenes estáticas utiliza una arquitectura de red *Single Shot Detector* (SSD) pre-entrenada con el dataset *Microsoft COCO*. El uso de una red neuronal ya pre-entrenada con un *dataset* de gran tamaño resulta beneficioso ya que, en lugar de comenzar el entrenamiento de la red inicializando los pesos aleatoriamente, estos pesos ya se encuentran inicializados para la tarea de detección de objetos, lo que ayuda a encontrar los pesos adecuados para la nueva tarea de detección de carteles publicitarios de una manera más rápida y eficiente.

Para realizar el entrenamiento de la red de una manera adecuada y precisa, se ha decidido utilizar la *API* para detección de objetos de *Tensorflow*. *Object Detection API* es un *framework* de código abierto basado en *Tensorflow* que permite el diseño, entrenamiento y despliegue de diferentes arquitecturas de *Deep Learning* orientadas a la tarea de detección

de objetos. Uno de los objetivos de este *framework* es permitir el diseño de prototipos de forma sencilla y rápida con el fin de pasar de una idea a un resultado en el menor tiempo posible.

A pesar de que la *API* permite el entrenamiento de arquitecturas mucho más precisas y potentes como *R-FCN* o *Faster R-CNN*, la elección de *SSD MobileNets* frente a los modelos anteriores radica en que los modelos de red *SSD* que utilizan *MobileNets* son mucho más ligeros y pueden desplegarse sobre dispositivos móviles para su ejecución en tiempo real, lo que permite su uso en aplicaciones de Realidad Virtual para dispositivos portátiles de pequeño tamaño. Los modelos como *R-FCN* o *Faster R-CNN* han sido descartados en este trabajo ya que requieren unos recursos computacionales mucho mayores que hacen inviable su ejecución en dispositivos móviles.

La arquitectura de red *SSD* implementa un método para detección de múltiples clases de objetos en imágenes mediante la generación de puntuaciones de confianza relacionadas con la presencia de objetos de cualquiera de las categorías en cada *bounding box*. Por otra parte, este tipo de red permite optimizar la asociación entre las bounding boxes y los objetos, ya que mediante la utilización de *anchors* se realiza la asignación de los *bounding boxes* en función de su ajuste a la forma del objeto. Su ejecución en tiempo real se debe a que realiza el proceso de inferencia en una única pasada, ya que no se vuelven a muestrear las características de la hipótesis de *bounding box*.

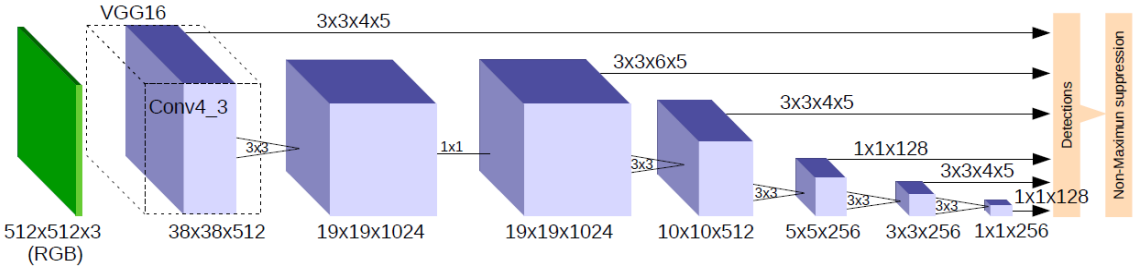

Ilustración 13 Arquitectura SSD Mobilenets

La arquitectura *SSD* se basa en una red convolucional *feed-forward* y su enfoque de detección de objetos actúa en dos fases. Durante la primera fase, *SSD* utiliza *VGG16* para extraer mapas de características y aplicar filtros de convolución para detectar los objetos utilizando la capa *Conv4_3*. Cada una de las predicciones realizadas por la red se compone de un *bounding box* junto con las puntuaciones de confianza para cada una de las clases (en este caso, 2), una para los carteles publicitarios y una clase adicional para cuando no se detecta ningún objeto; el objeto delimitado por el *bounding box* se asocia a la clase con la puntuación más alta. *Conv4_3* realiza un total de 38x38x4 predicciones, es decir que utiliza cuatro *anchors* predefinidos para obtener cuatro predicciones por cada una de las celdas independientemente de la profundidad de los mapas de características. Debido a que muchas de las predicciones no contienen ningún objeto, se utiliza la técnica *hard negative mining* para entrenar también con ejemplo negativos que irán asociados a la clase adicional indicada anteriormente. Un ejemplo de la arquitectura utilizada se puede observar en la ilustración 13.

En primer lugar, para entrenar este tipo de arquitectura con *Object Detection API* es necesario disponer de los datos de entrenamiento, es decir las imágenes junto con las

etiquetas, almacenados en un formato *TFRecord*, que es un formato de almacenamiento binario propio de *Tensorflow*. El uso de un formato binario como almacenamiento puede tener un impacto significativo en el tiempo de entrenamiento del modelo. *TFRecord* está optimizado para su uso con *Tensorflow* facilitando las funciones de importación y preprocesamiento de datos que proporciona la biblioteca, lo que resulta especialmente interesante para *datasets* que son demasiado grandes como para ser almacenados en la memoria ya que permite cargar desde disco solamente los datos que se requieren en el momento, es decir un *batch*, para luego procesarlos.

El *dataset* de carteles publicitarios creado en este trabajo contiene los datos etiquetados en formato *JSON*, ya que los datos han sido etiquetados con la herramienta *VGG Image Annotator*. Por esta razón, ha sido necesario transformar los datos antes de realizar el entrenamiento de la red *SSD* con *Object Detection API*. Para la transformación de los datos, se han diseñado dos funciones que permiten convertir los datos en formato *JSON* a formato *TFRecord*.

3.2.1 Transformación JSON-TFRecord

El proceso de transformación de los datos en formato *JSON* a formato *TFRecord* se lleva a cabo en dos fases. En la primera fase, se aplica una transformación a los datos en formato *JSON* de manera que se obtiene como resultado un fichero *CSV*, donde cada fila representa una detección de un cartel publicitario. En el fichero *JSON* cada una de las detecciones viene representada por cuatro coordenadas (x,y) pertenecientes a cada una de las esquinas del cartel mientras que para trabajar con *Object Detection API* es necesario utilizar únicamente dos coordenadas, la de la esquina superior izquierda y la de la esquina inferior derecha. En el fichero *CSV* cada una de las detecciones se representa mediante la siguiente información:

- ***filename*:** nombre de la imagen a la que pertenece la detección.
- ***width*:** ancho de la imagen.
- ***height*:** altura de la imagen.
- ***class*:** etiqueta de la clase a la que pertenece la detección, en este caso “addpanel”.
- ***xmin*:** coordenada X menor.
- ***ymin*:** coordenada Y menor.
- ***xmax*:** coordenada X mayor.
- ***ymax*:** coordenada Y mayor.

Para llevar a cabo dicha transformación, se ha diseñado una función llamada *json_to_csv.* Esta función recibe tres argumentos: la ruta del fichero *JSON*, la ruta donde se guardará el fichero *CSV* y el nombre de la clase a la que pertenecen las detecciones. Para cada una de las imágenes, se obtienen las detecciones etiquetadas en el fichero *JSON* y se escriben en el fichero *CSV* a través de la librería *CSV* de *Python*.

La segunda fase del proceso consiste en convertir el fichero *CSV* en un fichero *TFRecord* necesario para entrenar la red. El script *generate_tfrecord.py* junto con el fichero *dataset_util.py* proporcionado en el repositorio de *GitHub* de *Tensorflow* permiten convertir el fichero *CSV* con las detecciones en un fichero *TFRecord* listo para entrenar la red *SSD*. Antes de ejecutar dicho script, es necesario modificar la función *class_text_to_int* para añadir la etiqueta “addpanel”, es decir la clase que se quiere detectar.

El script *generate_tfrecord.py* es un script de línea de comandos que recibe dos argumentos como entrada. El primer argumento corresponde al *path* del fichero *CSV* que contiene la información sobre las detecciones mientras que el segundo argumento hace referencia al *path* donde se guardará el fichero *TFRecord* generado. El fichero *TFRecord* almacena la siguiente información para cada una de las detecciones:

- ***height*:** altura de la imagen
- ***width*:** ancho de la imagen
- ***filename*:** nombre de la imagen
- ***source_id*:** identificador de la imagen, se utiliza el *filename* como identificador.
- ***encoded*:** imagen en formato binario.
- ***format*:** extensión de la imagen, en este proyecto se ha utilizado la extensión *jpg*.
- ***xmin*:** coordenada X mínima normalizada.
- ***xmax*:** coordenada X máxima normalizada.
- ***ymin*:** coordenada Y mínima normalizada.
- ***ymax*:** coordenada Y máxima normalizada.
- ***text*:** etiqueta de clase en formato texto, en este proyecto "addpanel".
- ***label*:** etiqueta de clase en formato numérico, en este trabajo es 1.

Si se dispone de un fichero *CSV* con los datos etiquetados cuyo *path* es data/train_labels.csv y se quiere obtener el fichero *TFRecord*, se ejecutaría el script *generate_tfrecord.py* en una terminal mediante la siguiente instrucción:

python generate_tfrecord.py --csv_input=data/train_labels.csv --output_path=train.record

Una vez han sido generados los ficheros *TFRecord* para entrenamiento y validación, la siguiente etapa del proceso se corresponde a la configuración del entrenamiento. Antes de entrenar la red con el *dataset* de carteles publicitarios es necesario configurar una serie de parámetros de entrenamiento que resultarán determinantes en el aprendizaje de la solución óptima a nuestro problema.

Object Detection API utiliza una metodología de configuración a través de un fichero conocido como *pipeline*. En el *pipeline* se definen todos los parámetros de entrenamiento, el modelo de red pre-entrenado en caso de utilizar *Transfer Learning*, así como el *path* de los ficheros *TFRecord* de entrenamiento y validación. En este trabajo, se ha utilizado el *pipeline ssd_mobilenets_v1_pets.config* que se puede descargar desde el repo de *GitHub* de *Tensorflow*.

El *pipeline* de configuración contiene una serie de parámetros preconfigurados con un valor por defecto y otros que vienen sin configurar. Los parámetros sin configurar se indican como "*PATH_TO_BE_CONFIGURED*" y es el usuario el encargado de configurarlos antes de comenzar el entrenamiento. Estos parámetros son:

- ***fine_tune_checkpoint:*** modelo pre-entrenado utilizado como punto de partida para el entrenamiento de la red.
- ***train_input_reader:*** configuración de los datos de entrenamiento. *input_path* configura el fichero *TFRecord* con datos de entrenamiento mientras que *label_map_path* configura el fichero de traducción de etiquetas id-texto de la clase.

- ***eval_input_reader:*** configuración de los datos de validación. *input_path* configura el fichero *TFRecord* de validación mientras *label_map_path* configura el fichero de traducciones de las clases.
- ***num_classes:*** número de clases del *dataset*. En este trabajo únicamente se utiliza la clase "addpanel".

El resto de los parámetros de configuración del fichero tienen un valor preconfigurado por defecto. Sin embargo, resulta conveniente modificar dichos valores con el objetivo de encontrar los parámetros óptimos para nuestro problema concreto, ya que los valores por defecto puede que no sean adecuados para el problema que estamos intentando abordar. Los parámetros por defecto más interesantes son los siguientes:

- ***anchor_generator:*** configura los *aspect_ratio* de los *anchors* predefinidos utilizados por la red.
- ***box_predictor:*** configura diferentes hiperparámetros de la red de predicción, como el tamaño de los *kernels* de convolución, el uso de *dropout* o la función de activación entre otros.
- ***image_resizer:*** configura el re-escalado de las imágenes de entrada a un tamaño fijo.
- ***feature_extractor:*** configura el tipo de arquitectura encargada de la extracción de características, así como sus hiperparámetros.
- ***train_config:*** configura diferentes parámetros de entrenamiento como *batch_size*, *optimizer*, *learning rate*, *momentum* o *decay*.

En este trabajo, el *pipeline* de configuración ha sido modificado convenientemente con el objetivo de que la red aprenda durante el entrenamiento a detectar los carteles publicitarios con la mayor precisión posible.

Como modelo pre-entrenado se utilizado *ssd_mobilenet_v1_coco_11_06_2017,* con el dataset *Microsoft COCO*. El objetivo principal es la detección de carteles publicitarios en imágenes estáticas, por lo tanto, el parámetro *num_classes* ha sido configurado con el valor 1, ya que únicamente se detectan objetos de la clase "*addpanel*". Además, el fichero *label_map* contiene una única traducción, de clase "*addpanel*" a ID = 1, ya que ID=0 está reservado para cuando se produce ninguna detección. El parámetro *dropout* ha sido configurado de manera que solamente el 80% de las neuronas se tienen en cuenta durante la fase de entrenamiento, siendo el 20% restante ignoradas. Debido a la potencia limitada del ordenador utilizado para realizar el entrenamiento de la red, se ha configurado un *batch_size* entre 6 y 10 imágenes. Como algoritmo de optimización se ha utilizado la raíz cuadrada media *RMSProp*, un algoritmo de aprendizaje que mantiene el promedio de los gradientes cuadrados para cada peso y luego lo divide por la raíz cuadrada de la media del gradiente al cuadrado.

Una vez configurado el *pipeline* con los parámetros anteriores, se han realizado diferentes entrenamientos de la red con el *dataset* de paneles publicitarios durante 176.000 *epochs*. Para cada uno de los entrenamientos se han utilizado diferentes valores de *learning rate*, que varían desde 0.001 hasta 0.004, y los parámetros *momentum* y *decay* configurados con un valor de 0.9. El parámetro *momentum* es utilizado para evitar cambios bruscos en los pesos de un *epoch* a otro, de manera que un valor de *momentum* demasiado bajo haría

fluctuar los pesos en exceso. Por su parte, el parámetro *decay* es utilizado para evitar el sobreajuste de la red impidiendo que ésta memorice el conjunto de entrenamiento y se obtengan malos resultados durante la evaluación. Durante el entrenamiento, se han utilizado 5800 imágenes del *dataset* como conjunto de entrenamiento y únicamente 100 imágenes como conjunto de validación.

Para ejecutar el entrenamiento de la red, *Object Detection AP*I proporciona un script *CLI*, es decir un script que se ejecuta por línea de comandos que permite llevar a cabo el entrenamiento de la red. El script de *train* recibe dos parámetros, el *path train_dir* donde se guardarán los pesos de la red resultantes del proceso de entrenamiento y el *path* del *pipeline* de configuración *pipeline_config_path.* Un ejemplo de ejecución del proceso de entrenamiento sería:

python legacy/train.py --*train_dir*=training/ --*pipeline_config_path*= training/ssd_mobilenet.config

Además, una vez entrenada la red es necesario obtener grafo del modelo con los pesos congelados. Mediante el script *export_inference_graph* proporcionado por la propia *Object Detection API* es posible generar el grafo a través de la ejecución del script por medio de la siguente instrucción:

python export_inference_graph.py \
  --*input_type* image_tensor \
  --*pipeline_config_path* training/ssd_mobilenet.config \
  --*trained_checkpoint_prefix* training/model.ckpt-18000 \
  --*output_directory* publicity_graph

Como resultado de la ejecución, se genera el grafo en el directorio indicado con el parámetro *output_directory* y dicho modelo utilizará los pesos indicados en *trained_checkpoint_prefix.*

El rendimiento de la red se ha evaluado mediante un *Jupyter notebook* donde se han definido diferentes funciones de ayuda al proceso de inferencia sobre el conjunto de datos de test, además de funciones para la generación de informes *CSV* e informes gráficos. Durante el proceso de evaluación, toda la información acerca de las detecciones se almacena en una estructura llamada *predictedPrecisionMap*, de manera que una vez finalizado el proceso de inferencia sobre el conjunto de datos de test se pueden obtener diferentes métricas de rendimiento a partir de la información almacenada en la estructura.

Para cada una de las imágenes se guarda el número total de píxeles de la imagen, las proporciones de la imagen, el número de paneles etiquetados en el *groundtruth* para la imagen y la información de cada una de las predicciones de la red para la imagen. Cada predicción de la red se representa mediante cuatro valores: el valor de *IoU*, el tamaño original de cartel en porcentaje, el tamaño de cartel predicho por la red y las coordenadas de la predicción. En la ilustración 14 se muestra un ejemplo de la estructura:

```
{
  "imagename": {
    "imagesize": "nº píxeles totales de la imagen",
    "height_width": "(h,w) tupla con el tamaño de la imagen original",
    "panelsOnImages": "nº de paneles etiquetados en groundtruth",
    "boxes": {
      "0": {
        "IoU": "valor de IoU como precisión",
        "panelSize": "tamaño bounding box de grountruth",
        "predictPanelSize": "tamaño bounding box predicho",
        "predictedBox": "[xmin,ymin,xmax,ymax] coordenadas predichas"
      },
      "1":{
        ...
      }
    }
  },
  "imagename2":{
    ...
  }
}
```

Ilustración 14 Estructura PredictedPrecisionMap que guarda las detecciones

El proceso de evaluación consta de las siguientes etapas:

1. El primer paso del proceso de evaluación consiste en cargar el modelo ya entrenado en el *dataset* de paneles publicitarios junto a las respectivas etiquetas de clases, aunque en este trabajo solamente se utiliza la etiqueta "*addpanel*".

2. Una vez cargado el modelo, se genera una gráfica que muestra información acerca de los datos del conjunto de test. La gráfica muestra el número de carteles que hay para cada tamaño de cartel en el conjunto de datos de test. El tamaño ha sido calculado como *nº píxeles cartel / nº píxeles imagen*. A continuación, se muestra la gráfica generada donde se observa que la mayoría de los carteles ocupan menos del 10% de la imagen y el tamaño máximo de cartel es aproximadamente de 40-50% de la imagen.

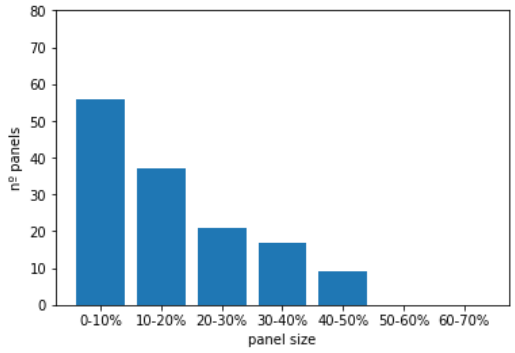

Ilustración 15 Gráfica imágenes de test distribuidas por tamaño

3. El proceso de inferencia sobre el conjunto de test se realiza de manera individual sobre cada una de las imágenes. En primer lugar, la imagen es cargada en memoria mediante la librería *PIL* y transformada a *numpy array* expandiendo sus dimensiones en el eje X.

4. Mediante la función *run_inference_for_single_image* la red realiza el proceso de inferencia sobre la imagen transformada y se obtiene como resultado un diccionario *output_dict* que almacena la información acerca de las predicciones en la imagen. La

información más importante es: número de detecciones en la imagen(*num_detections*), clases a la que pertenecen las detecciones(*detection_classes*), las coordenadas de los *bounding boxes*(*detection_boxes*) y la medida de confianza calculada por la red para cada detección(*detection_scores*). Este diccionario es utilizado tanto para visualizar las detecciones sobre la imagen con *visualize_boxes_and_labels_on_image_array* como para rellenar la estructura *predictedPrecisionMap*.

5. Una vez procesada la imagen, se calcula su tamaño (h, w) y se obtienen las detecciones de *groundtruth* para la imagen mediante la función *boundingBoxGroundTruth* que devuelve una lista con los *bounding boxes* etiquetados para la imagen. A partir de esta información se rellenan los campos *imagesize*, *height_width* y *panelsOnImage* de la estructura *predictedPrecisionMap.* Además, los *bounding boxes* de *groundtruth* serán utilizados posteriormente para calcular el *IoU* de las predicciones.

6. Después, a partir de *output_dict,* se rellena la información de cada detección en la estructura *predictedPrecisionMap*. Un aspecto importante es que solamente se tienen en cuenta aquellas detecciones con una confianza o *detection_score* superior a 0.5, siendo el resto de las detecciones descartadas. Para cada detección en *detection_boxes* con una confianza mayor a 0.5 se obtiene las coordenadas *Xmin*, *Ymin*, *Xmax* e *Ymax* que definen el *bounding box* predicho por la red. Dicho *bounding box* es asociado a su respectivo *bounding box* de *groundtruth* en base a la distancia de sus centros, de manera que esta distancia sea la menor posible. A partir de ambos *bounding boxes* se calcula el *IoU* mediante la función *bb_intersection_over_union* y los tamaños de cartel mediante la función *pannelPercentageOccupacy*. Sobre la imagen se mostrará el *bounding box* de *groundtruth* de manera que se pueda observar el grado de solapamiento entre la predicción y el cartel etiquetado. A partir de los datos calculados en esta fase, se rellena el campo boxes de la estructura *predictedPrecisionMap* que hace referencia a cada una de las detecciones. Para cada detección se guarda el *IoU* calculado, el tamaño del cartel etiquetado *PanelSize*, el tamaño del cartel predicho por la red *predictPanelSize* y las coordenadas del *bounding box* predicho.

7. Una vez realizado el proceso de inferencia sobre todas las imágenes de test y rellenado la información en la estructura *predictedPrecisionMap*, dicha estructura es utilizada para la generación de los informes *CSV* e informes *gráficos*. Como resultado de la evaluación se muestran 3 informes *gráficos*:
    - El primer informe muestra el número de carteles detectados en función de la precisión, es decir en función de los distintos valores de *IoU* posibles. Para ello se utiliza la función *predictionsHistogram* que a partir de *predictedPrecisionMap* genera la gráfica utilizando *Matplotlib*.
    - El segundo informe muestra el número de carteles detectados para cada tamaño de cartel diferente. Para ello se utiliza la función *percentagePanelSizeVsTotal.*
    - El tercer informe muestra la precisión media de las detecciones para cada uno de los tamaños de cartel diferentes. Para generar el informe también se utiliza la función *percentagePanelSizeVsTotal* que genera ambas gráficas a la vez.

Por cada uno de los informes gráficos generados también se genera un informe *CSV* con la misma información. Estos tres informes muestran información general en función de diversos criterios, sin embargo, no muestran ninguna información de manera individual para cada una de las imágenes. La función *createCsvSeparatedData* genera un informe *CSV* con información individual para cada una de las detecciones sobre las imágenes. Cada una de las filas del informe se corresponde con una detección sobre una de las imágenes. Cada detección se representa en el informe mediante las siguientes columnas: el nombre de la imagen *imagename*, el tamaño de la imagen *sizeImage*, el número de paneles en la imagen *panelsOnImage*, el id de la detección *box_id* siendo el id un valor entre 0 y *N*, el tamaño de la detección *sizeBox* y la precisión *IoU.*

3.2.2 Funciones

A continuación, se explica detalladamente en qué consiste cada una de las funciones utilizadas durante el proceso de evaluación del sistema propuesto.

load_image_into_numpy_array

La función *load_image_into_numpy_array* recibe como único argumento una imagen cargada en memoria con la librería *PIL* y la convierte a *Numpy array*. En primer lugar, se obtiene la altura *h* y la anchura *w* mediante la función *size*. Una vez obtenido el tamaño de la imagen, se transforma la imagen en un *array* de tamaño (h, w, 3) a través de la librería *Numpy*. El tamaño se representa como la terna (*Height*, *Width*, *Channels*).

run_inference_for_single_image

La función *run_inference_for_single_image*, proporcionada por la propia *API*, recibe dos argumentos: la imagen en formato *Numpy array* y un grafo que almacena el modelo de red entrenado previamente; y lleva a cabo el proceso de inferencia sobre una única imagen para obtener los *bounding boxes* de las detecciones, la clase a la que pertenece cada detección y la medida de confianza de la red.

A partir del grafo del modelo, se extraen los tensores de entrada y salida de la red, y se guardan en un diccionario. A través del diccionario se obtendrá cada una de las detecciones sobre la imagen. En primer lugar, se filtrarán las predicciones de manera que las detecciones con dimensión cero serán eliminadas del tensor "*detection_boxes*" ya que se consideran detecciones erróneas. Este filtrado se realiza por medio de la función *squeeze* de la librería *Tensorflow*. Posteriormente, con ayuda de la función *cast*, se obtendrá el número de detecciones reales en la imagen, de manera que serán eliminados del tensor "detection_boxes" aquellos *bounding boxes* que no se corresponden a detecciones reales. Mediante la sesión del modelo, se ejecuta la función *run* que realiza la inferencia y se obtiene como resultado un diccionario con toda la información acerca de las detecciones predichas por el modelo. La información almacenada es el número de detecciones totales "*num_detections*", la clase de cada una de las detecciones "*detection_classes*", la medida de confianza "*detection_scores*" y el *bounding box* de cada detección "*detection_boxes*".

bb_intersection_over_union

La función *bb_intersection_over_union* recibe como argumentos dos *bounding boxes* pertenecientes a la detección de la red y al *groundtruth* respectivamente y calcula la precisión de la detección mediante la métrica *IoU*. Cada uno de los *bounding boxes* se representa mediante las coordenadas *Xmin*, *Ymin*, *Xmax* e *Ymax*. En primer lugar, se calculan

las coordenadas *Xmin*, *Ymin*, *Xmax* e *Ymax* de la intersección a partir ambos *bounding boxes*. Los valores *Xmin* e *Ymin* de la intersección se calculan como el valor máximo de los valores mínimos, es decir $V_{min} = \max(Va_{min}, Vb_{min})$ mientras que los valores *Xmax* e *Ymax* se calculan como el valor mínimo de los valores máximos, es decir $V_{max} = \min(Va_{max}, Vb_{max})$. A partir de las coordenadas de la intersección se calcula el área de esta a la que se le denomina *interArea*. Después, se calcula el área de la unión como (*areaBox1* + *areaBox2* – *interArea*). Por último, una vez calculadas las áreas de la intersección y de la unión respectivamente, se calcula el *IoU* como (*interArea*) / (*areaBox1* + *areaBox2* - *interArea*).

pannelPercentageOccupacy

La función *pannelPercentageOccupacy* recibe dos argumentos, el número total de píxeles de la imagen y el número de píxeles del panel respectivamente, y calcula el porcentaje que ocupa el cartel respecto al tamaño total de la imagen. Como resultado la función devuelve un valor entre 0 y 1 que representa el porcentaje, por ejemplo, un valor de 0.25 significa que el cartel ocupa el 25% de la imagen. El porcentaje de ocupación es calculado como *nº pixeles cartel / nº pixeles totales*.

getCenterBox

La función *getCenterBox* recibe como entrada un *bounding box* con sus coordenadas *Xmin*, *Ymin*, *Xmax* e *Ymax*, y calcula el centro del *bounding box*. En primer lugar, se calcula la altura y anchura del *bounding box* como *h = (Xmax – Xmin)* y *w = (Ymax – Ymin)* respectivamente. A partir de la altura y la anchura calculadas en el paso anterior, se obtiene el centro del *bounding box* como la coordenada *(Xmin + w/2, Ymin + h/2).*

boundingBoxGroundTruth

La función *boundingBoxGroundTruth* recibe dos argumentos, un fichero *JSON* con los *bounding boxes* del *groundtruth* y el nombre de una imagen para la cual se quieren obtener los *bounding boxes*. Como resultado, se obtiene una lista de *bounding boxes* pertenecientes a la imagen indicada. En primer lugar, se carga el fichero *JSON* en memoria a partir de la librería *JSON* de *Python* y se accede a los datos etiquetados para la imagen solicitada. Para cada cartel etiquetado se tiene una lista de coordenadas *X* y una lista de coordenadas *Y* pertenecientes a las cuatro esquinas del cartel. A partir de estas listas se extraen las coordenadas *Xmin*, *Ymin*, *Xmax* e *Ymax* que representan el *bounding box* del cartel. Una vez procesados todos los carteles para la imagen, se devuelve una lista con todos los *bounding boxes* de la imagen.

groundTruthSizeGraph

La función *groundTruthSizeGraph* recibe como entrada dos argumentos, un fichero *JSON* con el *groundtruth* del conjunto de datos de test y un *path* donde se guardará un fichero *CSV*. La función utiliza la información almacenada en el fichero *JSON* para generar un informe *CSV* y un informe gráfico que mostrará el número de carteles etiquetados en función de los diferentes tamaños de cartel.

En primer lugar, se carga en memoria la información almacenada en el fichero *JSON*. Esta información se va recorriendo imagen a imagen, de manera que para cada una de las imágenes es necesario obtener el porcentaje de ocupación de cada una de las detecciones etiquetadas. Para obtener el porcentaje de ocupación de cada detección es necesario

obtener previamente el tamaño de la imagen en número total de píxeles, que se calcula como el producto entre *height* y *witdh*. A partir del tamaño de la imagen y el tamaño del *bounding box* etiquetado, se calcula el porcentaje de ocupación mediante la función *pannelPercentageOccupancy.* El número de *bounding boxes* de cada tamaño se va acumulando hasta que todas las imágenes y sus respectivos *bounding boxes* hayan sido procesados.

Una vez procesada toda la información, se genera el informe gráfico a través de la librería *Matplotlib*. Para ello, se crea una figura nueva mediante la función *subplots*, de manera que en el eje *X* se representan los diferentes tamaños de cartel mientras que en el eje *Y* se representa el número de carteles para un determinado tamaño. Haciendo uso de la función *set_xlabel* se asigna la etiqueta *panel size* al eje *X*; por su parte con la función *set_ylabel* se asigna la etiqueta *nº panels* al eje *Y*. Por último, por medio de la función *xticks* se establece el rango de los tamaños de cartel, que en este informe van desde 0-10% hasta el 60-70% de ocupación y mediante la función *bar* se crea la gráfica con los diferentes valores acumulados durante el procesado de los datos.

Debido a que los valores en el eje *Y* del informe gráfico muestran sus valores en decenas, es difícil apreciar el número exacto de carteles para cada tamaño, por ejemplo, es difícil saber con exactitud si hay 65 o 68 carteles para un tamaño determinado. Por este motivo, además de generar un informe gráfico, se genera un informe *CSV* que muestra la misma información, solo que en este informe si es posible apreciar el número exacto de carteles.

Para ello, se crea un fichero *CSV* en el *path* indicado como argumento de la función. El fichero se crea en modo escritura 'w' y se utiliza un objeto *writer* de la librería *CSV* de *Python* para escribir la información del informe. El informe muestra el número de carteles para cada uno de los diferentes tamaños, por lo que el informe tendrá dos columnas *box size* y nº *panels*, que hacen referencia al tamaño del cartel y al número de carteles respectivamente.

predictionsHistogram

La función *predictionsHistogram* recibe dos argumentos, la estructura con toda la información acerca del proceso de inferencia *predictedPrecisionMap* y un *path* donde se guardará el informe *CSV*. La función utiliza el mapa *predictedPrecisionMap* para generar un informe gráfico y un informe *CSV* que muestra el número de detecciones en función de cada valor de precisión, es decir en función del *IoU* de la predicción.

En primer lugar, se accede a la información almacenada en el mapa *predictedPrecisionMap* para cada una de las detecciones, es decir que, para cada imagen se recorre la información de los *boxes* y se acumulan los datos en función del valor del *IoU* de la detección. Mediante la librería *Matplotlib*, al igual que en la función anterior, se crea una figura de manera que el eje *X* representa los diferentes valores de *IoU* y el eje *Y* representa el número de detecciones para el valor de *IoU*.

Además, al igual que en la función anterior, se genera un informe *CSV* que muestra la información de manera más precisa. En este caso, el informe muestra el número de detecciones en función de la precisión obtenida para la predicción, por lo que en este informe las columnas que se mostrarán son *precision* y *nº boxes*, que hacen referencia a cada una de las precisiones y al número de detecciones con dicha precisión.

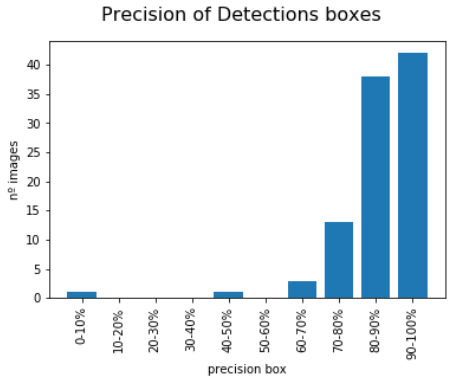

Ilustración 16 Número de detecciones en base a la precisión

percentagePanelSizeVsTotal

La función *percentagePanelSizeVsTotal* recibe dos argumentos, la estructura con la información del proceso de inferencia *predictedPrecisionMap* y un *path* donde se guardará el informe *CSV*. La función utiliza el mapa *predictedPrecisionMap* para generar dos informes gráficos y un único informe *CSV* que engloba la información de los dos informes gráficos. El primer informe gráfico muestra el número de detecciones para cada uno de los tamaños de cartel; por su parte, el segundo informe gráfico muestra la precisión *IoU* media para cada uno de los tamaños.

En primer lugar, se accede a la información almacenada en el mapa *predictedPrecisionMap* para cada una de las detecciones, es decir que, para cada imagen se recorre la información de los *boxes* y se acumulan los valores de *IoU* de cada predicción en función de los tamaños. Acumulando los datos de esta manera, es posible generar los dos informes a la vez, ya que es posible saber la cantidad de carteles de cada tamaño, así como su precisión media.

Al igual que en las dos últimas funciones explicadas, los informes gráficos han sido generados a través de la librería *Matplotlib*. En el primer informe, el eje *X* representa los diferentes tamaños de cartel mientras que el eje *Y* representa el número de detecciones para los diferentes tamaños. Para la generación de este informe es necesario conocer los diferentes tamaños y el número de carteles. Para ello, a partir de la información acumulada para los diferentes tamaños, es posible obtener el número de carteles de un tamaño comprobando la longitud de la lista con los valores *IoU* acumulados para cierto tamaño. Por su parte, en el segundo informe el eje *X* representa cada uno de los tamaños de cartel mientras que el eje *Y* representa la precisión media para cada tamaño. Para obtener el valor de la precisión media, se suman todos los valores de *IoU* acumulados para un cierto tamaño y se divide entre el número de valores acumulados.

En esta función, en lugar de generar un informe *CSV* para cada informe gráfico, se ha optado por unificar la información de ambos informes gráficos en un único informe *CSV*. En este informe *CSV* se utilizan tres columnas diferentes: *size, nº boxes* y *precisión mean.* Dichas columnas hacen referencia al tamaño de cartel, el número de detecciones y la precisión media de las detecciones respectivamente.

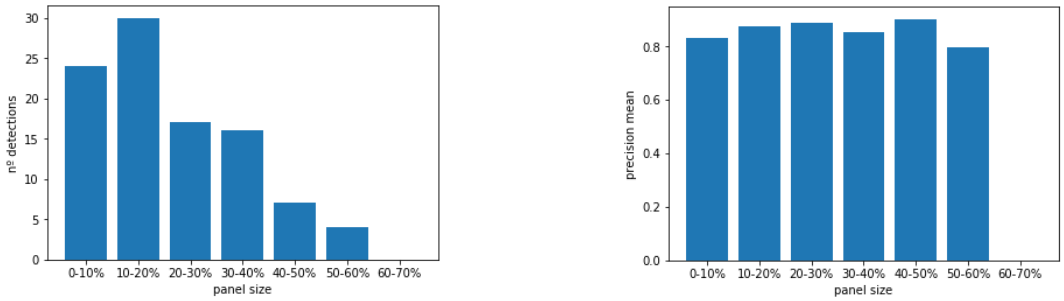

Ilustración 17 Gráficas nº detecciones vs tamaño y tamaño vs precisión media

createCsvSeparatedData

La función *createCsvSeparatedData* recibe dos argumentos, la estructura que almacena la información acerca del proceso de inferencia *predictedPrecisionMap* y el *path* donde se guardará el informe *CSV*. Esta función únicamente genera un informe *CSV* con la información separada para cada una de las predicciones de la red. Los informes anteriores mostraban información general en función de diferentes criterios mientras que en este informe se muestra información individual para cada una de las detecciones.

Para la generación de este informe *CSV*, se utilizan seis columnas que representan cada una de las detecciones. Las columnas utilizadas son las siguientes: *imagename*, *sizeImage*, *panelsOnImage*, *box_id*, *sizeBox* y *IoU*. Estas columnas hacen referencia al nombre de la imagen procesada, el tamaño de la imagen como tupla (h, w), el número de paneles de *groundtruth* de la imagen, el identificador de la detección (valor entre 0 y *PanelsOnImage* - 1), el tamaño de la detección como tupla (h, w) y la precisión de la detección respectivamente.

visualize_boxes_and_labels_on_image_array

La función *visualize_boxes_and_labels_on_image_array* recibe un total de 8 parámetros, 5 de ellos obligatorios y 3 de ellos opcionales. Los parámetros obligatorios hacen referencia a la imagen sobre la que se visualizaran las detecciones, los *bounding boxes*, las clases, las medidas de confianza y las traducciones de las clases. Por su parte, los parámetros opcionales hacen referencia al uso de máscaras si se trata de un problema de segmentación, al uso de coordenadas normalizadas y al grosor del *bounding box* mostrado. A partir de esta función es posible visualizar sobre la imagen los diferentes *bounding boxes* correspondiente a las detecciones. Además, por cada una de las detecciones se muestra la etiqueta de clase traducida y la medida de confianza proporcionada por la red.

3.3 Solución Especifica YOLO

La segunda solución propuesta al problema de la detección de carteles publicitarios en imágenes estáticas utiliza una arquitectura de red *You Only Look Once* (YOLO) pre-entrenada con el *dataset Imagenet*.

Con el objetivo de llevar a cabo el entrenamiento de la red de una manera adecuada, se ha utilizado la herramienta *Darknet*. *Darknet* es un *framework Open Source* escrito en C y CUDA para la detección de objetos en imágenes mediante una arquitectura de red *YOLO*. Las principales ventajas que proporciona el *framework* son la rapidez y facilidad de instalación, lo que permite un prototipado ágil posibilitando el cálculo tanto en *CPU* como *GPU*. A diferencia de otros tipos de arquitecturas de red que aplican el modelo a una imagen en

múltiples ubicaciones y escalas, *YOLO* divide la imagen en regiones y predice en una única pasada los *bounding boxes* y sus respectivos valores de confianza para cada región de la imagen.

La elección de este tipo de arquitectura viene motivada principalmente por la velocidad de inferencia de la red, que permite su uso para detección de objetos en tiempo real. Esta velocidad se debe a que realiza la predicción de un único *forward* o pasada, a diferencia de otros sistemas como *R-CNN*, que requieren más de una pasada para realizar la predicción sobre una sola imagen. De esta manera, *YOLO* es más de 1000 veces más rápido que *R-CNN* y 100 veces más rápido que *Fast R-CNN* [40].

YOLO implementa un método de detección de objetos multiclase en imágenes dividiendo la imagen en celdas independientes y realizando predicciones sobre cada una de las celdas. De esta manera, para cada una de las celdas, se calculan *N bounding boxes* junto con sus respectivos valores de confianza, además mediante uso de *anchors* permite un ajuste óptimo de los *bounding boxes* a la forma de cada objeto. Debido a que la mayor parte de una imagen no contiene ningún objeto, y por lo tanto la confianza en la mayor parte de las celdas será muy baja, se aplica un umbral de confianza de manera que los *bounding boxes* que no superan el umbral son descartados. Una vez descartados *los bounding boxes* que no contienen ningún objeto, a los restantes se les aplica el algoritmo de *Non Maxima Suppression* para evitar *bounding boxes* duplicados y superpuestos sobre un mismo objeto.

En este trabajo se ha utilizado *YOLOv3*, que predice un único *bounding box* para cada objeto y calcula una puntuación de confianza mediante regresión logística. *YOLOv3* utiliza la red *Darknet-53*, formada por 53 capas convolucionales entrenadas con ImageNet para la extracción de características además de 53 capas adicionales para la tarea de detección, es decir que *YOLOv3* contiene un total de 106 capas convolucionales. Esta red es más eficiente y rápida que *Darknet-19* (utilizada por *YOLOv2*) o las redes *ResNet-101* o *ResNet-152*. Además, utiliza una serie de bloques residuales y de *upsampling*. A diferencia de las redes neuronales tradicionales, donde la salida de una capa se alimenta como entrada a la siguiente capa, en las redes con bloques residuales una capa se alimenta a la siguiente capa y directamente a subsiguientes capas a varios saltos de distancia. De esta manera, los bloques residuales permiten el flujo de información desde las capas iniciales hasta las capas finales.

La extracción de características se realiza en 3 escalas diferentes utilizando un método similar a *Feature Pyramids Network* (*FPN)*. Por su parte, las detecciones de la red se realizan aplicando kernels de tamaño 1x1xN a los mapas de características obtenidos en cada una de las 3 escalas. De esta manera, el mapa de características resultante tiene una altura y anchura idénticas a las del mapa de características anterior, sin embargo, su profundidad se ha reducido a N características únicamente.

En la ilustración 18 se puede observar la arquitectura de la red *YOLOv3*. Las detección de los *bounding boxes* en la primera escala se producen en la capa 82 de la red. Durante las primeras 81 capas, la red está formada por capas de convolucionales junto a bloques residuales que se encargan de extraer una serie de mapas de características. Una vez realizadas las detecciones en la primera escala, el mapa de características de la capa 79 es procesado por una serie de capas convolucionales antes de ser muestreado por una capa

de *upsampling* x2 para aumentar sus dimensiones. Posteriormente, el mapa de características muestreado se combina en profundidad junto con el mapa de características de la capa 61 para obtener información semántica más significativa de las primeras capas de la red. El mapa de características resultante es procesado por una serie de capas convolucionales 1 x 1 que se encargan de fusionar las características. En la capa 94, la red realiza la detección de los *bounding boxes* sobre la segunda escala. Al igual que para la capa 79, el mapa de características de la capa 91 es procesado por un conjunto de capas convolucionales antes de ser muestreado por una capa de *upsampling* x2. Este mapa muestreado se combina en profundidad con el mapa de características de la capa 36 antes de ser procesado por una serie de capas convolucionales 1x1 para fusionar la información. Finalmente, en la capa 106 la red realiza la detección de *bounding boxes* en la última escala. Las predicciones en esta última escala se benefician de todos los cálculos anteriores, ya que la concatenación de las capas muestreadas con las primeras capas de la red ayuda a preservar las características de grano fino obtenidas en las primeras capas de convolución. En la primera escala se detectan los objetos más grandes, en la segunda se detectan los objetos medianos mientras que en la última escala se detectan los objetos más pequeños.

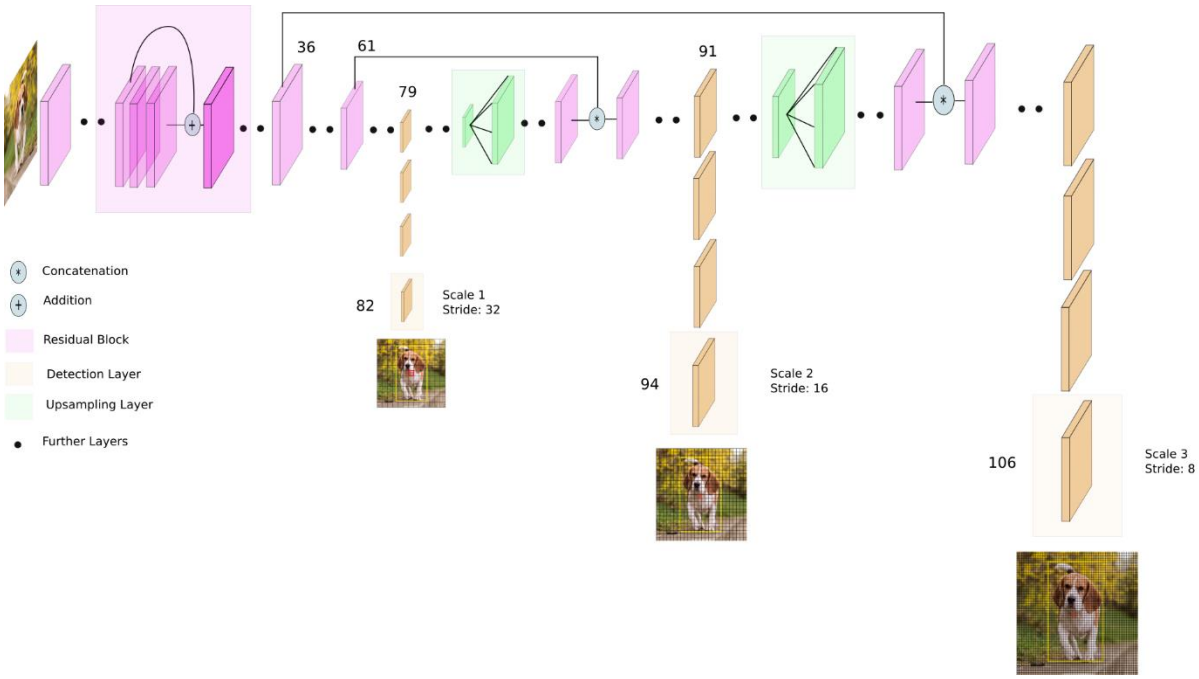

Ilustración 18 Arquitectura de red YOLOv3

Para las detecciones, *YOLOv3* calcula 3 *anchors boxes* para cada una de las escalas, de manera que en total se predicen 9 bounding boxes para cada una de las celdas de la imagen. En la ilustración 19 se muestra un ejemplo de los *anchors boxes* calculados para una celda de la imagen. La mayoría de los bounding boxes predichos no contienen ningún objeto, por lo tanto la confianza en la mayor parte de las celdas será muy baja. Para filtrar los *bounding boxes* que no contienen ningún objeto se aplica un umbral de confianza de manera que los *bounding boxes* que no superan el umbral son descartados. A los *bounding boxes* restantes se les aplica el algoritmo de *Non Maxima Suppression* para evitar *bounding boxes* duplicados y superpuestos sobre un mismo objeto.

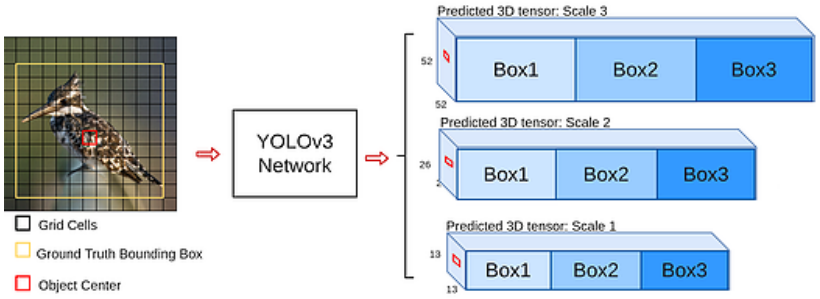

Ilustración 19 Anchors de YOLO para una celda de la imagen

Para la tarea de clasificación, *YOLOv3* utiliza clasificadores de regresión logísticos independientes para cada una de las clases en lugar de utilizar una capa *softmax* tradicional. De esta manera un objeto puede ser clasificado con múltiples etiquetas. Por ejemplo, suponiendo que se tiene un modelo entrenado para las clases "persona" y "mujer" y una imagen en la que aparece una mujer; *softmax* producirá unas probabilidades divididas para estas 2 clases, por ejemplo, 0,4 y 0,45 para cada una de las clases respectivamente mientras que los clasificadores independientes generan una probabilidad para cada clase, por ejemplo 0,8 para la clase "mujer" y 0.9 para la clase "persona" lo que permite etiquetar el objeto como persona y mujer.

La versión *YOLOv3* mejora la precisión promedio para objetos pequeños y a medida que aumenta la precisión promedio *Mean Average Precission* (*mAP)* los errores de localización disminuyen. Esto se debe principalmente a los bloques residuales que propagan las información de las primeras capas de la red hasta las capas finales, lo que favorece a preservar las características de grano fino de las capas tempranas.

El primer paso para entrenar la red *YOLOv3* con el *framework Darknet* es transformar los datos etiquetados del *dataset* de paneles publicitarios al formato aceptado por la red. El *dataset* de paneles guarda la información de las etiquetas en un fichero *JSON*, de modo que cada elemento etiquetado se representa mediante una lista de cuatro coordenadas *(x, y)* pertenecientes a cada una de las esquinas del cartel. Por su parte, *YOLOv3* representa cada *bounding box* mediante la siguiente información: *<object-class-id><center-x><center-y><width><height>*. La columna *object-class-id* representa la clase del objeto, se trata de un número entero en un rango desde 0 hasta (número de clases - 1). En este trabajo, ya que únicamente se detectan carteles publicitarios, la clase "*addpanel"* se establece con el valor 0. Las columnas *center-x* y *center-y* representan las coordenadas *X* e *Y* normalizadas del centro del *bounding box* respectivamente. Las columnas *width* y *height*, también normalizadas, representan el ancho y el alto del *bounding box* respectivamente.

Como resultado del proceso de transformación de los datos al formato aceptado por *YOLOv3*, para cada una de las imágenes que conforman el *dataset* se genera un fichero individual que almacena cada una de las detecciones asociadas a la imagen. De esta manera si una imagen solamente tiene un *bounding box* etiquetado como *groundtruth*, el fichero generado solamente tendrá una fila con los valores *object-class-id, center-x, center-y, width* y *height* mientras que si la imagen contiene varios *bounding boxes* etiquetados como *groundtruth* el fichero generado tendrá una fila por cada *bounding box* con los valores mencionados anteriormente. Una vez finalizado el proceso de conversión de los datos, se tienen aproximadamente 6000 ficheros con las etiquetas adaptadas al formato aceptado

por *YOLOv3*, es decir un fichero por cada una de las imágenes del *dataset* de carteles publicitarios. Para todos estos datos, el valor de *object-class-id* será 0 ya que en este trabajo únicamente se trabaja con la clase "*addpanel"*, por lo que la primera columna de todos los ficheros generados tendrá el valor 0.

Una vez que todos los datos han sido transformados al formato aceptado por *YOLOv3*, es necesario configurar el archivo *darknet.data*, proporcionado por el propio *framework*. A partir de este fichero se le transfiere al *framework* información necesaria acerca de las especificaciones del detector, así como algunos *paths* relevantes para el entrenamiento del sistema. Los parámetros configurados en este fichero son:

- ***classes:*** número de clases del *dataset* utilizado, en este caso será 1 ya que únicamente se detectan objetos de la clase *addpanel*.
- ***train:*** *path* absoluto del fichero que almacena la lista imágenes utilizadas como datos de entrenamiento de la red.
- ***valid:*** *path* absoluto del fichero que almacena la lista de imágenes utilizadas como datos de validación de la red.
- ***names:*** *path* absoluto del fichero que contiene los nombres de cada una de las clases del *dataset*. En este trabajo el fichero únicamente contiene la etiqueta *addpanel*.
- ***backup:*** *path* de un directorio existente que almacenará los pesos intermedios de la red generados durante el proceso de entrenamiento.

Además del archivo darknet.data, *YOLOv3* también requiere de un archivo de configuración llamado *darknet-yolov3.cfg* que almacena diferentes parámetros necesarios para el entrenamiento de la red. Los parámetros de entrenamiento más importantes los siguientes:

batch y subdivisions

El parámetro *batch* especifica el tamaño de lote utilizado durante el proceso de entrenamiento, es decir el número de imágenes que se utilizan para actualizar los pesos mientras que el parámetro *subdivisions* se utiliza para dividir el tamaño de lote cuando se producen errores de memoria insuficiente al procesar un lote.

Durante el proceso de entrenamiento se actualizan iterativamente los pesos de la red en función de los errores cometidos en el *dataset* utilizado. El *dataset* utilizado para entrenar la red contiene varios miles de imágenes, por lo que no es práctico utilizar todas las imágenes a la vez para actualizar los pesos, por ello se utiliza un subconjunto de los datos que se denomina *batch*. Aunque es posible especificar un *batch* muy grande, puede que no se disponga de suficiente memoria para procesar tal cantidad de datos. Por ello, *Darknet* utiliza el parámetro *subdivisions* que permite procesar una fracción del *batch*. En caso de error, se procesarán *batch/subdivisions* datos, pero la iteración o actualización de los pesos no se completará hasta que se hayan procesado todas las imágenes.

En este trabajo, debido a las limitaciones de entrenar en CPU, se ha utilizado un *batch* de entre 4 y 8 imágenes y valor para el parámetro *subdivisions* de 2.

Width, Height y Channels

Los parámetros *width*, *height* y *channels* especifican el tamaño de las imágenes de entrada a la red, así como el número de canales de la imagen. Las imágenes de entrada son

redimensionadas al tamaño especificado en estos parámetros antes de ser utilizadas para entrenar la red.

El *framework* de *Darknet* utiliza unos valores por defecto de 416×416 sin embargo los resultados pueden mejorar utilizando un tamaño mayor, por ejemplo, de 608×608, pero también requeriría más tiempo entrenar. Cuando se utilizan imágenes a color, el parámetro *channels* deberá tener el valor 3, ya se están procesando imágenes con 3 canales, es decir en *RGB*.

En este trabajo, se ha utilizado los valores configurados por defecto, por lo que las imágenes del *dataset* de paneles utilizadas para entrenar la red *YOLOv3* serán imágenes *RGB* con un tamaño de 416x416 píxeles.

Momentum y Decay

El archivo de configuración contiene algunos parámetros que controlan cómo se actualizan los pesos de la red. Estos parámetros son *momentum* y *decay*. El parámetro *momentum* se utiliza para penalizar los grandes cambios de peso entre cada una de las iteraciones mientras que el parámetro *decay* se utiliza como término de penalización para evitar el sobreajuste de la red.

Debido a que los pesos de la red se actualizan en base a un pequeño lote de imágenes y no con el *dataset* completo, los valores de los pesos fluctúan bastante durante las actualizaciones de una iteración a otra, por lo que es necesario controlar estas variaciones mediante el parámetro *momentum*. Además, debido a la cantidad de pesos que poseen estas redes, se pueden adaptar fácilmente a cualquier información de entrenamiento o incluso sobreajustar los datos de entrenamiento, lo que significa que la red memoriza los datos de entrenamiento, pero no es capaz aprender el concepto subyacente para discriminar, lo que se traduce en unos malos resultados de test. Para mitigar este problema se utiliza el parámetro *decay*.

En este trabajo se han realizado diferentes entrenamientos de la red de manera que en cada uno de los entrenamientos se han utilizado diferentes configuraciones para cada uno de los parámetros mencionados. Para el parámetro *momentum* se han utilizado diferentes valores en el rango 0.8-0.9 mientras que para el parámetro *decay* se han utilizado valores en el rango 0.0003-0.0005.

Learning Rate, Steps, Scales, Burn In

El parámetro *learning rate* hace referencia a la tasa de aprendizaje, que controla la velocidad con la que la red aprende en base al lote actual de datos. Típicamente se utiliza un valor entre 0.01 y 0.0001, de manera que cuanto mayor es la tasa más rápido se encuentra una solución, aunque podría no ser la solución óptima.

Al principio del proceso de entrenamiento, se comienza con cero información por lo que la tasa de aprendizaje debe ser alta, pero a medida que la red neuronal va procesando los datos los pesos deben cambiar de forma menos agresiva, por lo que la tasa de aprendizaje debe reducirse con el tiempo.

Darknet proporciona los parámetros *steps* y *scales* para controlar la disminución de la tasa de aprendizaje a lo largo del entrenamiento. En ese sentido, la tasa de aprendizaje permanecerá con un valor fijo durante el número de *steps* especificado, pero una llegado a ese punto, la tasa de aprendizaje disminuirá su valor, de manera que se calculan las nuevas

tasas de aprendizaje multiplicado la tasa de aprendizaje anterior por el valor de *scales*, de manera que a medida que avanza el entrenamiento, el valor de la tasa de aprendizaje irá decreciendo.

Si bien la tasa de aprendizaje debe ser alta al principio y luego ir disminuyendo, en los últimos tiempos se ha demostrado empíricamente que la velocidad del entrenamiento tiende a aumentar si tenemos una tasa de aprendizaje más baja durante un corto período de tiempo, por ello se utiliza el parámetro *burn_in*. Este parámetro incrementa lentamente la tasa de aprendizaje durante *burn_in batches*, es decir que se aumenta un poco la tasa de aprendizaje para reducir el tiempo de entrenamiento de la red.

En este trabajo, se han utilizado diferentes configuraciones para cada uno de los parámetros que se indican a continuación (excepto para *steps* y *scales*, para los cuales se han mantenido los valores por defecto proporcionados por el propio *framework* de *Darknet)*. Para el parámetro *learning rate* se ha utilizado diferentes valores en el rango 0.01-0.001 mientras que para el parámetro *burn_in* se han utilizado diferentes valores en el rango 400-1000. Los parámetros *steps* y *scales* utilizan unos valores por defecto de 1800 y 0.1 respectivamente. Además, es posible indicar el parámetro *max_batches* que indica el número de iteración que durará el proceso de entrenamiento. Para este parámetro se ha decidido utilizar un valor de 5000 iteraciones.

Data Augmentation

Una red neuronal requiere de una gran cantidad de datos de entrenamiento, ya que se necesitan miles de imágenes para lograr un buenos resultados. La creación de un *dataset* propio con miles de imágenes conlleva mucho tiempo y esfuerzo. Para este trabajo se han recopilado aproximadamente 6000 imágenes donde cada uno de los *bounding boxes* pertenecientes a carteles publicitarios ha sido etiquetado a mano. Para ello, se utilizó la herramienta *VGG Image Annotator*.

Para aprovechar al máximo estos datos, *Darknet* permite aplicar la técnica de *Data Augmentation*, que consiste en aplicar diferentes operaciones sobre la imagen para transformar esta de manera que la red aprenda diferentes variabilidades a las contenidas en el *dataset*. *Darknet* permite aplicar diferentes transformaciones sobre la imagen; por ejemplo, la operación de rotación o transformar los colores de la imagen a partir de unos valores de saturación, exposición y tono. Los parámetros proporcionados por *Darknet* para realizar estas transformaciones son *angle*, *saturation*, *exposure* y *hue*.

En este trabajo, se han utilizado los valores por defecto proporcionados por el propio *framework* de *Darknet*. Para el parámetro *angle* se utiliza el valor 0, de manera que se aplican rotaciones con un ángulo de rotación aleatorio. Para los parámetros *saturation* y *exposure* se ha utilizado un valor de 1.5 mientras que para el parámetro *hue* se ha utilizado un valor de 0.1.

Una vez configurados todos los parámetros, ya es posible comenzar el entrenamiento de la red. Cuando se entrena un detector de objetos con un *dataset* propio, es buena idea utilizar como punto de partida un modelo ya pre-entrenado sobre conjunto de datos muy grande como *ImageNet* o *Microsoft COCO* aunque dicho *dataset* no contenga imágenes del objeto que se intenta detectar. Este proceso se conoce como *transfer learning*. Esta técnica resulta interesante ya que en lugar de aprender los pesos desde cero, el modelo pre-entrenado ya

contiene los pesos convolucionales entrenados para un *dataset* de gran tamaño, por lo que usando estos pesos como pesos iniciales la red es capaz de aprender más rápido. En este trabajo se ha empleado la técnica de *transfer learning* utilizando una red *Darknet-53* pre-entrenada en el *dataset ImageNet*.

Para llevar a cabo el proceso de entrenamiento de la red, el propio *framework* de *Darknet* proporciona un archivo ejecutable que posibilita la ejecución de diferentes comandos. Actualmente soporta dos comandos diferentes, el comando *detect* que permite efectuar el proceso de inferencia a partir de un detector ya entrenado y el comando *detector* que permite ejecutar tanto el entrenamiento de la red como el proceso de inferencia. El modo de uso del ejecutable es el siguiente:

$ **./darknet** <*comando*> <*parámetros*>

Dependiendo del comando a ejecutar, es necesario proporcionar unos parámetros u otros. En este trabajo únicamente se ha utilizado el archivo ejecutable para llevar a cabo el proceso de entrenamiento de la red, de manera que solamente se ha utilizado el comando *detector*. Este comando recibe una serie de parámetros necesarios para su ejecución:

$ **./darknet** *detector* [train/test/valid] [cfg] [weights (optional)]

El primer parámetro que hay que indicar con el comando *detector* es el modo de ejecución, es decir, si se ejecuta en modo entrenamiento *train*, en modo evaluación *test* (equivalente al comando *detect*) o si se ejecuta en modo validación *valid*. En este trabajo únicamente se ha utilizado el modo *train* ya que el archivo ejecutable solo se ha utilizado para entrenar la red. En modo *train*, es necesario proporcionar una serie de parámetros adicionales:

$ **./darknet** detector train <file.data> <file.cfg> <file.weights>

Los parámetros proporcionados son los siguientes:

- ***file.data:*** *path* del fichero de configuración .data que se utiliza para indicar el número de clases del *dataset*, el directorio donde se guardan los pesos así como diferentes rutas relativas a los datos de entrenamiento.
- ***file.cfg:*** *path* del fichero de configuración .cfg que almacena toda la información acerca de la configuración del entrenamiento de la red.
- ***file.weights:*** *path* del fichero con los pesos iniciales entrenados con *ImageNet*

Un ejemplo de la ejecución del entrenamiento con los datos reales sería:

$ **./darknet** *detector train* darknet.data darknet-yolov3.cfg darknet53.conv.74

Durante toda la ejecución del proceso de entrenamiento, se muestra diferente información acerca de los resultados del entrenamiento en las diferentes escalas. Como *YOLO* realiza detecciones en tres escalas, para cada *batch* de entrenamiento se muestra información en tres escalas conocidas como región 82, región 94 y región 106 que representan la capa de predicción grande, intermedia y pequeña respectivamente. Por cada escala, la información que se muestra es la siguiente:

- ***Region:*** escala de las detecciones.
- ***Avg IoU:*** valor promedio del *IoU* de cada imagen en la subdivisión actual.

- ***Class:*** tasa de clasificación (un valor cercano a 1 representa una detección precisa.
- ***Obj:*** debe ser in valor cercano a 1).
- ***No Obj:*** debe ser un valor cercano a 0.
- ***5R/75R:*** ratio de muestra positiva detectadas sobre la muestra positiva real en todas las imágenes de subdivisión. Si todas las muestras positivas son detectadas el valor será 1. Se calcula como *recall*/*count*.
- ***Count:*** número de objetos detectados en la escala.

Una vez procesada la información en cada una de las 3 escalas, se muestra la información relativa al *batch* actual. La información mostrada es la siguiente:

- ***Nº iteration:*** número de iteración actual.
- ***Loss:*** representa el valor de perdida general.
- ***Avg:*** valor de perdida medio. Cuanto menor sea el valor mejor.
- ***Rate*****:** tasa de aprendizaje actual.
- ***Seconds*****:** tiempo empleado en procesar el *batch* actual.
- ***Images*****:** número total de imágenes procesadas.

A continuación, se muestra un ejemplo de la información de entrenamiento para la iteración 4678 del entrenamiento de la red *YOLO* en el *dataset* de carteles publicitarios:

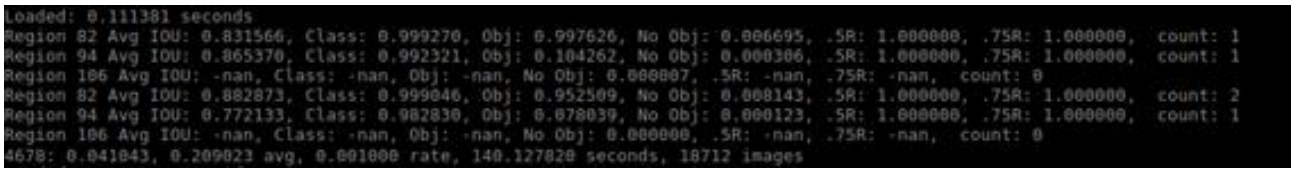

Ilustración 20 Información de entrenamiento con Darknet

A medida que avanza el entrenamiento, se irán almacenando los pesos entrenados para el *dataset* de carteles publicitarios en el directorio indicado en el fichero *darknet.data*, de modo que a la hora de realizar el proceso de inferencia se utilizan los pesos "congelados" resultantes que han sido almacenados en dicho directorio.

Una vez acabado el entrenamiento de la red y guardado los pesos aprendidos, es necesario evaluar el rendimiento de la red sobre el conjunto de datos de test. Para ello, al igual que en la primera solución propuesta, se ha creado un *Jupyter notebook* que realiza el proceso de inferencia sobre cada una de las imágenes y genera una serie de informes para mostrar información acerca de los resultados obtenidos. Al igual que para el sistema *SSD* propuesto, toda la información acerca de las detecciones de la red *YOLO* es almacenada en una estructura llamada *predictedPrecisionMap* que es utilizada para obtener diferentes métricas de rendimiento acerca del proceso de evaluación. Para cada una de las imágenes, *predictedPrecisionMap* guarda el número total de píxeles de la imagen, las proporciones de la imagen, el número de paneles etiquetados en el *groundtruth* para la imagen y la información de cada una de las predicciones de la red obtenidas a partir de la inferencia realizada sobre la imagen. Cada predicción de la red se representa mediante cuatro valores: el valor de *IoU*, el tamaño original de cartel en porcentaje, el tamaño de cartel predicho por la red y las coordenadas de la predicción.

Al principio del *Jupyter notebook* se definen una serie de variables que hacen referencia al *path* de los directorios y ficheros que van a ser utilizados durante la evaluación del sistema propuesto. Los directorios especificados son: ***test_images*** que se refiere al directorio que contiene las imágenes de test sobre las que se realiza el proceso de inferencia, ***test_results*** que se refiere al directorio donde se guardan las imágenes resultantes del proceso de inferencia y ***csvReportsPath*** que hace referencia al directorio donde se guardan los informes *CSV* generados durante la evaluación del sistema. Por su parte, los ficheros definidos son ***jsonPath*** que se refiere al fichero *JSON* que contiene la información acerca del *groundtruth* de los *bounding boxes* del conjunto de datos de test, ***labelsPath*** que indica el fichero que contiene los nombres de las clases a detectar, ***weightsPath*** que referencia al fichero de pesos entrenados en el *dataset* de paneles publicitarios y ***configPath*** que se refiere al fichero de configuración *.cfg* de la red *YOLO*.

A diferencia de la primera solución propuesta que utilizaba *Tensorflow* para realizar las detecciones con la red *SSD*, esta segunda solución utiliza el módulo *Deep Neural Networks* de *OpenCV* para realizar el proceso de inferencia sobre cada una de las imágenes con la red *YOLO*. El proceso de evaluación del sistema propuesto consta de las siguientes etapas:

1. En primer lugar, se genera una gráfica que muestra información acerca de las imágenes del conjunto de datos de test. La gráfica muestra el número de carteles que hay para cada tamaño de cartel en el conjunto de datos de test, siendo el tamaño calculado como *nº píxeles cartel / nº píxeles imagen*. Para generar dicha gráfica se ha utilizado la función *groundTruthSizeGraph*, que es exactamente la misma que se utilizaba en la primera solución propuesta.

2. Después, se carga el modelo de red en memoria. Para ello se utiliza la función *readNetFromDarknet* proporcionada por el módulo *dnn* de OpenCV. La función readNetFromDarknet recibe como parámetros el fichero de configuración *.cfg* y los pesos entrenados en el *dataset* de carteles publicitarios y devuelve el modelo de red *YOLO* listo para el proceso de inferencia. Además, se cargan las etiquetas de clases mediante la función *load_labels*, aunque en este trabajo la única clase utilizada es "addpanel".

3. El proceso de inferencia sobre el conjunto de test se realiza de manera individual sobre cada una de las imágenes. En primer lugar, la imagen es cargada en memoria mediante la función *imread* de OpenCV y se obtiene su tamaño mediante su atributo *shape*, que devuelve una tupla con *(height, width, channels)* que representa el tamaño de la imagen.

4. Una vez cargada la imagen, se rellena la estructura *predictedPrecisionMap* con la información acerca de la imagen. La información que se rellena es el tamaño de la imagen en píxeles *imagesize* calculado como height x width, el tamaño de la imagen en formato tupla *height_width* y el número de carteles etiquetados de *groundtruth* en la imagen *panelsOnImage.* Para obtener esta última información se ha utilizado la función *boundingBoxGroundTruth*, que ya se utilizada en la primera solución propuesta, y que permite obtener los *bounding boxes* de *groundtruth* para una imagen determinada.

5. Posteriormente, se realiza el proceso de inferencia sobre la imagen para obtener los *bounding boxes* detectados. Para ello se utiliza la función *predict_on_image*, que recibe como argumentos el modelo de red cargado en memoria y la imagen sobre la que se realiza la inferencia. Como resultado, se obtienen las coordenadas de las detecciones *boxes*, los valores de confianza de cada *bounding box confidences* y los identificadores de las clases a las que pertenecen las detecciones *classIDs*. Para evitar *bounding boxes* superpuestos, se aplica el algoritmo *non-máxima supression* que permite obtener un único *bounding box* por cada objeto detectado. Para ello, se utiliza la función *NMSBoxes* del módulo *dnn* de OpenCV que recibe como parámetros los *bounding boxes* y sus valores de confianza así como dos umbrales, un umbral de confianza y un umbral NMS. Todos los *bounding boxes* con una confianza menor al umbral de confianza son descartados. Además, de los *bounding boxes* restantes, todos aquellos cuyo *IoU* respecto al *bounding box* con mayor confianza sea superior al umbral NMS también son descartados, ya que se consideran dos detecciones diferentes sobre el mismo objeto. De esta manera se obtiene un único *bounding box* para cada una de las detecciones. En este trabajo se ha establecido un umbral de confianza de 0.5 y un umbral NMS de 0.3.

6. Después de filtrar las detecciones de la red con el algoritmo NMS para obtener una única detección por cada objeto en la imagen, se recorre cada una de las detecciones con el objetivo de guardar la información en la estructura *predictedPrecisionMap* y mostrar el resultado gráficamente. Por cada detección filtrada, se obtienen las coordenadas en formato *YOLO*, es decir *Xmin*, *Ymin*, *W* y *H*, y a partir de estas se calculan las coordenadas *Xmin*, *Ymin*, *Xmax* e *Ymax* que definen el *bounding box*. Una vez conocidas las coordenadas del *bounding box*, se dibuja la *ROI*, la etiqueta de clase y el valor de confianza sobre la imagen mediante la función *rectangle* y *putText* de OpenCV. Posteriormente, se obtiene el *bounding box* del *groundtruth* asociado a la detección en base a la distancia de sus centros al igual que se hace en la primera solución propuesta. Ahora que se tiene el *bounding box* de la detección y el *bounding box* del *groundtruth*, es posible calcular la precisión de la detección. para ello se utiliza la función *intersection_over_union (IoU)* que recibe como argumentos ambos *bounding boxes* y devuelve el valor de *IoU* obtenido. Además, a partir de las coordenadas *Xmin*, *Ymin*, *Xmax* e *Ymax* del *bounding box* del *groundtruth*, se dibuja la *ROI* en la imagen mediante la función *rectangle* de OpenCV. Por último, se rellena la información *boxes* de la estructura *predictedPrecisionMap* con los datos acerca de la detección. Esta información es la precisión de la detección *IoU*, el tamaño real del panel *panelSize* y el tamaño de la detección *predictPanelSize* calculados mediante la función *pannelPercentageOccupacy*, y las coordenadas *Xmin*, *Ymin*, *Xmax* e *Ymax* de la detección.

7. Una vez realizado el proceso de inferencia sobre todas las imágenes de test y rellenado la información en la estructura *predictedPrecisionMap*, dicha estructura es utilizada para la generación de los informes *CSV* e informes gráficos. Como resultado de la evaluación se muestran 3 informes gráficos:
    - El primer informe muestra el número de carteles detectados en función de la precisión, es decir en función de los distintos valores de *IoU* posibles. Para ello se utiliza la función *predictionsHistogram* que a partir de *predictedPrecisionMap* genera la gráfica utilizando *Matplotlib*.

- El segundo informe muestra el número de carteles detectados para cada tamaño de cartel diferente. Para ello se utiliza la función *percentagePanelSizeVsTotal.*
- El tercer informe muestra la precisión media de las detecciones para cada uno de los tamaños de cartel diferentes. Para generar el informe también se utiliza la función *percentagePanelSizeVsTotal* que genera ambas gráficas a la vez.

Por cada uno de los informes gráficos generados también se genera un informe *CSV* con la misma información.

Estos tres informes muestran información general en función de diversos criterios, sin embargo, no muestran ninguna información de acerca de las predicciones para cada una de las imágenes individualmente. La función *createCsvSeparatedData* genera un informe *CSV* con información individual para cada una de las detecciones sobre las imágenes. Cada una de las filas del informe se corresponde con una detección sobre una de las imágenes. Cada detección se representa en el informe mediante las siguientes columnas: el nombre de la imagen *imagename*, el tamaño de la imagen *sizeImage*, el número de paneles en la imagen *panelsOnImage*, el id de la detección *box_id* siendo el id un valor entre 0 y N, el tamaño de la detección *sizeBox* y la precisión *IoU.*

3.3.1 Funciones

A continuación, se muestra detalladamente en qué consisten cada una de las funciones propuestas para la implementación de la solución mediante la red *YOLO*. Algunas de las funciones utilizadas son las mismas que las implementadas y explicadas en la primera solución, por lo tanto, éstas no van a ser explicadas de nuevo. Las nuevas funciones implementadas son las siguientes:

load_labels

La función *load_labels* recibe como único argumento el *path* del fichero que contiene los nombres de las clases a las que pueden pertenecer las detecciones. Como resultado se devuelve una lista con los nombres de cada una de las clases almacenadas en el fichero. Esta función es utilizada para obtener el nombre de la clase, en este caso "*addpanel*", para mostrar la etiqueta de clase y el valor de confianza de cada una de las detecciones sobre una imagen.

En primer lugar, se leen las líneas del fichero. Para ello, se abre el fichero con la función *open* de *Python*, y después se leen todas las líneas a través de la función *read*. Después, es necesario eliminar todos los espacios en blanco del principio y final de la cadena leída, para ello se utiliza la función *strip* de *Python*.

Por último, la cadena es separada en diferentes líneas, una para cada posible clase. Para ello se utiliza la función *split* de la librería *string* de *Python*.

La función es utilizada para obtener el nombre de la clase, en este trabajo *addpanel*, que se muestra junto a la precisión para cada una de las detecciones sobre una imagen.

get_net_prediction

La función *get_net_prediction* recibe como argumentos la red cargada en memoria y la imagen sobre la cual se realiza la inferencia. Como resultado, se obtienen las capas de salida de la red junto con las detecciones resultantes del proceso de inferencia.

En primer lugar, para llevar a cabo la inferencia es necesario diferenciar entre las capas de salida y el resto de capas de la red. Mediante la función *getLayerNames*, del módulo *dnn* de *OpenCV*, se obtienen los nombres de cada una de las capas de la red. Una vez conocidas todas las capas, se utiliza la función *getUnconnectedOutLayers* para obtener únicamente las capas pertenecientes a la salida.

En *OpenCV*, las imágenes de entrada a la red deben estar en un formato determinado conocido como *blob*. Para convertir la imagen en *blob* se utiliza la función *blobFromImage*. Esta función escala los valores de los píxeles de la imagen a un rango 0 a 1 utilizando un factor de escala 1/255. Además, reescala la imagen a un tamaño de 416x416 píxeles.

El *blob* obtenido se le pasa a la red neuronal mediante la función *setInput* y se lleva a cabo la inferencia mediante la función *forward* que devuelve la lista de *bounding boxes* predichos como salida de la red.

predict_on_image

La función *predict_on_image* recibe dos argumentos, la red neuronal cargada en memoria y la imagen sobre la que se realiza la predicción. Como resultado, se obtiene la lista de *bounding boxes* pertenecientes a las detecciones, sus valores de confianza y las clases a las que pertenecen las detecciones.

En primer lugar, se calcula el tamaño de la imagen de entrada a través del atributo *shape* de la imagen, que devuelve una tupla con los valores *height* y *width*. Después, se lleva a cabo la inferencia sobre la imagen de entrada mediante la función *get_net_prediction* de manera que se obtienen las capas de salida de la red que almacenan los resultados de la inferencia.

Para cada una de las capas de salida, se obtienen sus respectivas detecciones sobre la imagen de manera que para cada una de las detecciones se calcula la clase a la que pertenece la detección y su valor de confianza. Este valor de confianza es utilizado para filtrar las detecciones de modo que si el valor de confianza es menor que 0.5 se descarta la detección.

Para cada una de las detecciones válidas con una confianza superior a 0.5 se calculan los valores de la capa de salida. Estos valores son *centerX* y *centerY*, que hacen referencia al centro del *bounding box*, además de los valores *height* y *width* de la detección. Estos valores están normalizados, por lo que para obtener los valores en coordenadas de la imagen es necesario multiplicar los valores normalizados por el height y width de la imagen calculados anteriormente.

A partir de los valores en coordenadas de la imagen, se calculan las coordenadas del *bounding box* en el formato utilizado por *YOLO*, es decir *Xmin*, *Ymin*, *width* y *height*. Una vez que ya se han calculado las coordenadas del *bounding box*, su valor de confianza y la clase a la que pertenece la detección, la información es acumulada.

Una vez procesadas todas y cada una de las capas junto a sus detecciones, se devuelven la información de todas las detecciones a través de tres listas con los datos acerca de los *bounding boxes*, sus valores de confianza y las clases.

4. Resultados

En esta sección, se analizan los resultados de detección y localización de carteles usando los dos tipos de redes profundas consideradas en este trabajo: *SSD* y *YOLO*.

4.1 Single Shot Detector (SSD)

A continuación, se muestran los resultados cualitativos y cuantitativos con respecto a la detección de paneles en las imágenes del conjunto de datos de test. Para obtener estas métricas de rendimiento se han utilizado los informes generados durante el proceso de evaluación del sistema, así como la estructura *predictedPrecisionMap* que almacena la información del proceso de inferencia sobre cada una de las imágenes de test. Los mejores resultados se han obtenido tras 176.000 iteraciones con un *batch_size* de 6-10 imágenes, un *learning rate* de 0.001-0.004 y valor de 0.9 para los parámetros *momentum* y *decay*. La ilustración 21 muestra varias detecciones de paneles producidas por el sistema propuesto bajo diferentes condiciones como iluminación nocturna, perspectiva, oclusiones parciales o un tamaño reducido de cartel.

Ilustración 21 Detecciones de la red SSD

Con respecto a los resultados cuantitativos, se ha utilizado la métrica de *Intersection over Union* (*IoU*) como medida de precisión en las detecciones. Con esta métrica, la precisión de la detección se calcula al dividir la intersección de los *bounding boxes* de la detección y del *groundtruth* por la unión correspondiente de ambos *bounding boxes*. Los respectivos

valores de *IoU* calculados para los paneles detectados en las imágenes de la ilustración 21 fueron de 0.93, 0.94, 0.88 y 0.93 respectivamente.

Dada una imagen de entrada, la red *SSD* devuelve para cada punto de la imagen de entrada una medida de confianza que representa la posibilidad de encontrar un panel centrado en dicho punto de la imagen. En el contexto de nuestro modelo, se ha buscado la ausencia de falsos positivos, ya que es preferible, desde el punto de vista de las aplicaciones relacionadas con los anuncios, dejar algún panel publicitario sin detectar que producir una detección donde no existe ningún cartel publicitario. Por lo tanto, utilizando la muestra de aprendizaje, se ha establecido un umbral de confianza de 0.5 para considerar una detección como correcta. Por lo tanto, si la neurona en la capa de salida tiene un valor superior a 0.5, la red considera que hay un panel publicitario en ese punto.

Usando este valor de umbral, se obtiene un valor de *IoU* mayor que cero para todos los carteles en las imágenes de test excepto en un caso, de modo que tal caso podría ser considerado como falso positivo. La tabla 3 muestra el número de carteles detectados con este umbral y la tasa de *IoU* obtenida en cada caso. El uso de este umbral deja una tasa de falso rechazo del 30%, es decir que 44 de los 140 paneles de las imágenes de test no son detectados. Muchos de los carteles no detectados se corresponden a carteles de pequeño tamaño (es decir, incluso mucho más pequeños que el 10% del tamaño de la imagen).

Valor IoU	Nº Paneles Detectados
[0.5, 0.6)	0
[0.6,0.7)	3
[0.7, 0.8)	13
[0.8, 0.9)	38
[0.9, 1.0]	42

Tabla 3 Número de detecciones de SSD para cada valor de IoU

4.1.1 Clasificación por tamaños

La tabla 4 muestra la de distribución de los carteles publicitarios de acuerdo a los tamaños para las imágenes de test. A la hora de evaluar los resultados hay que tener en cuenta que la mayoría de los paneles que aparecen en las imágenes son pequeños y tienen un tamaño inferior o igual al 10% (aproximadamente un 40% de total); un 81,5% de los paneles presentan un tamaño inferior o igual al 30%; y todos los paneles son más pequeños o iguales que la mitad de la imagen. En este estudio, no se han considerado paneles muy grandes (es decir, más del 50% del tamaño de la imagen), ya que a medida que aumenta el tamaño del cartel su detección se vuelve más fácil incluso con un fondo de imagen complejo.

Tamaño de Panel	Nº Paneles GroundTruth
0-10%	56
11-20%	37
21-30%	21
31-40%	17
41-50%	9

Tabla 4 Número de carteles GroundTruth de acuerdo a los tamaños

Con respecto a cómo influye el tamaño del panel en la precisión de la detección, la tabla 5 muestra que el porcentaje de detección aumenta a medida que aumenta el tamaño del panel en la imagen. Este valor crece desde alrededor del 43,5% en paneles que cubren menos del 10% de la imagen hasta el 94% cuando el tamaño del panel ocupa aproximadamente el 40% de la imagen. Además, un hecho interesante es que cuando se detectan paneles en las imágenes, su precisión correspondiente al valor del *IoU* es, en la mayoría de los casos, superior al 85%, independientemente del tamaño del panel. Otro detalle a tener en cuenta es que a medida que aumenta el tamaño de cartel también aumenta la precisión de las detecciones, por ejemplo, la precisión media de los carteles que ocupan menos del 10% de la imagen es de un 83,1% mientras que la precisión de los carteles que ocupan aproximadamente el 50% de la imagen asciende hasta 90,3% de precisión.

Tamaño Panel	Nº Real Paneles	Nº Detecciones	% detecciones	Precisión Media
0- 10%	56	24	43%	0.831
11-20%	37	30	81%	0.876
21-30%	21	17	81%	0.887
31-40%	17	16	94%	0.855
41-50%	9	7	78%	0.903

Tabla 5 Precisión de las detecciones de SSD por tamaño de cartel

4.1.2 Detecciones por precisión

La tabla 6 muestra como la mayoría de predicciones propuestas por la red alcanzan unos valores de precisión bastante altos, aproximadamente el 82% de las detecciones alcanza una precisión superior al 80% mientras que solamente el 5% de las detecciones tiene una precisión inferior al 70%. Además, es posible apreciar que tan solo se produce un falso positivo, ya que únicamente la detección con un *IoU* del 10% puede considerarse como falso positivo.

Precisión	Nº Detecciones
0-10%	1
11-20%	0
21-30%	0
31-40%	0
41-50%	1
51-60%	0
61-70%	3
71-80%	13
81-90%	38
91-100%	42

Tabla 6 Número de detecciones de SSD para cada valor de IoU

4.1.3 Detecciones por otros aspectos

Para observar cómo afectan las diferentes condiciones de variabilidad de las imágenes a la precisión de las detecciones, se ha evaluado el rendimiento del sistema en función de tres condiciones diferentes bajo las que aparecen los carteles en las imágenes. Los tres aspectos que se han tenido en cuenta han sido:

- ***Nocturnidad:*** Imágenes nocturnas vs Imágenes diurnas.
- ***Frontalidad:*** Imágenes frontales vs Imágenes no frontales.
- ***Oclusiones:*** Imágenes con oclusiones vs Imágenes sin oclusiones.

Para la extracción de estas métricas de rendimiento se han utilizado tres ficheros *JSON* que contienen cada una de las imágenes de test separadas según los criterios de clasificación especificados. Los ficheros *JSON* utilizados son *front_classification*, *nightly_classification* y *occluded_classification*. A partir de estos, se han generado tres informes *CSV*, uno para cada criterio de clasificación, de manera que cada uno de los informes muestra la siguiente información:

- ***Type:*** tipo de imagen, por ejemplo, si es nocturna o diurna.
- ***Iou_Min:*** valor mínimo de *IoU* para un tipo determinado.
- ***IoU_Max:*** valor máximo de *IoU* para un tipo de determinado.
- ***IoU_Mean:*** valor medio de *IoU* para un tipo.
- ***Total_images:*** número total de imágenes de un tipo determinado.
- ***No_detected:*** número de imágenes sin detección.
- **False_positive:** número de falsos positivos en las detecciones.

En la tabla 7, se puede observar que, a diferencia de lo que pueda parecer a priori, las condiciones de nocturnidad de las imágenes no afectan a la precisión de las detecciones, ya que la precisión media de las detecciones sobre las imágenes nocturnas es de aproximadamente el 96% y el *IoU* mínimo es del 85%. Otro aspecto importante a tener en cuenta es que en las imágenes nocturnas no se ha producido ningún falso positivo, si bien en una de las imágenes no se ha detectado ningún panel publicitario. Por su parte, en las imágenes diurnas, la precisión máxima alcanzada ha sido mayor, casi un 98%. Sin embargo, se puede observar que la precisión media es algo menor, ya que ésta es de aproximadamente el 87% si bien es verdad que la mayoría de las imágenes de test se corresponden a imágenes diurnas, por lo que es normal que la precisión media sea algo menor.

Tipo	IoU Min	IoU Max	IoU Medio	Imágenes Totales	Sin Detecciones	Falsos Positivos
Nocturnas	0.852	0.955	0.911	5	1	0
Diurnas	0.497	0.978	0.870	128	36	1

Tabla 7 Comparativa Imágenes Diurnas vs Nocturnas con SSD

En la tabla 8 se observa la comparativa entre los carteles publicitarios frontales y los carteles en perspectiva. A pesar de que la precisión de los carteles en perspectiva o no frontales alcanza unos valores aceptables, ya que la precisión media es del 86% y la precisión mínima es aproximadamente del 74%, el sistema propuesto sí que se ve afectado en ese aspecto a la hora de realizar las detecciones, ya que se puede observar que tan solo en el 50% de las imágenes que contenían carteles en perspectiva se ha producido alguna detección. Por lo tanto, se puede concluir que, si bien la perspectiva de los carteles publicitarios no afecta a

la precisión de sus detecciones, sí que afecta a la tasa de detecciones, ya que solo en la mitad de las imágenes se ha logrado al menos una detección correcta.

Tipo	IoU Min	IoU Max	IoU Medio	Imágenes Totales	Sin Detecciones	Falsos Positivos
Frontal	0.497	0.978	0.873	113	27	1
No Frontal	0.737	0.935	0.860	20	10	0

Tabla 8 Comparativa Imágenes Frontales vs No Frontales con SSD

La tabla 9 muestra la comparativa entre las detecciones de carteles publicitarios ocluidos y los carteles sin oclusiones. Se puede observar que la precisión media de las detecciones cuando los carteles publicitarios presentan oclusiones decrece considerablemente en comparación a cuando no se producen oclusiones, ya que la precisión media con oclusiones es del 79,4% mientras que sin oclusiones alcanza hasta el 87,4%. Además, se observa que el número de imágenes sin detecciones también ha aumentado considerablemente, ya que aproximadamente en el 66% de las imágenes no se ha producido ninguna detección. Por lo tanto, se puede concluir que las oclusiones producidas sobre los carteles publicitarios tienen un efecto negativo tanto en la precisión de las detecciones como a la hora de localizar el cartel dentro de la imagen.

Tipo	IoU Min	IoU Max	IoU Medio	Imágenes Totales	Sin Detecciones	Falsos Positivos
Ocluidas	0.694	0.913	0.794	12	8	1
No Ocluidas	0.497	0.978	0.874	121	29	0

Tabla 9 Comparativa Imágenes Ocluidas vs No Ocluidas con SSD

4.2 You Only Look Once (YOLO)

A continuación, se muestran los resultados cualitativos y cuantitativos con respecto a la detección de paneles en las imágenes del conjunto de datos de test utilizando la arquitectura de red *YOLO*. Para obtener estas métricas de rendimiento se han utilizado los informes generados durante el proceso de evaluación del sistema, así como la estructura *predictedPrecisionMap* que almacena la información del proceso de inferencia sobre cada una de las imágenes de test. Los mejores resultados se han obtenido después de entrenar el sistema durante 4800 iteraciones, con un *batch_size* de 4-8 imágenes, un *learning rate* de 0.0001-0.01 y unos valores de 0.0003-0.0005 para el parámetro *decay* y de 0.8-0.9 para el parámetro *momentum*. La ilustración 22 muestra varias detecciones de paneles producidas por el sistema *YOLO* bajo diferentes condiciones como iluminación nocturna, perspectiva, oclusiones parciales o un tamaño reducido de cartel.

Ilustración 22 Detecciones de la red YOLO

Con respecto a los resultados cuantitativos, se ha utilizado la misma métrica que en la primera solución propuesta, es decir *Intersection Over Union* (*IoU*). La precisión de la detección se calcula al dividir la intersección de los *bounding boxes* de la detección y del *groundtruth* por la unión correspondiente de ambos *bounding boxes*. Los respectivos valores de *IoU* calculados para los paneles detectados en las imágenes de la ilustración 22 fueron de 0.70, 0.95, 0.38 y 0.70 respectivamente.

En este sistema propuesto se ha intentado buscar una solución que produzca la menor cantidad de falsos positivos, ya que es preferible, desde el punto de vista de una aplicación, dejar algún panel publicitario sin detectar que producir una detección donde no existe ningún cartel publicitario. Por lo tanto, se ha establecido un umbral de confianza de 0.5 para considerar una detección como correcta.

Usando este valor de umbral, se obtiene un valor de *IoU* mayor que cero para la mayoría de los carteles en las imágenes de test excepto en 9 detecciones que tienen un *IoU* entre 0-10%, de modo que tales casos se consideran como falsos positivo. La tabla 10 muestra el número de carteles detectados con este umbral y la tasa de *IoU* obtenida en cada caso. El uso de este umbral deja una tasa de falso rechazo del 10%, es decir que 14 de los 140 paneles de las imágenes de test no son detectados. Muchos de los carteles no detectados se corresponden a carteles de pequeño tamaño.

Valor IoU	Nº Paneles Detectados
[0.5, 0.6)	14
[0.6,0.7)	18
[0.7, 0.8)	35
[0.8, 0.9)	46
[0.9, 1.0]	13

Tabla 10 Número de detecciones de YOLO para cada valor de IoU

4.2.1 Clasificación por tamaños

La Tabla 11 muestra la de distribución de los carteles publicitarios de acuerdo a los tamaños para las imágenes de tests. A la hora de evaluar los resultados hay que tener en cuenta que la mayoría de los paneles que aparecen en las imágenes son pequeños y tienen un tamaño inferior o igual al 10% (aproximadamente un 40% de total); un 81,5% de los paneles presentan un tamaño inferior o igual al 30%; y todos los paneles son más pequeños o iguales que la mitad de la imagen. En este estudio, no se han considerado paneles muy grandes (es decir, más del 50% del tamaño de la imagen), ya que a medida que aumenta el tamaño del cartel su detección se vuelve más fácil incluso con un fondo de imagen complejo.

Tamaño de Panel	Nº Paneles GroundTruth
0-10%	56
11-20%	37
21-30%	21
31-40%	17
41-50%	9

Tabla 11 . Número de carteles GroundTruth de acuerdo a los tamaños

Con respecto a cómo influye el tamaño del panel en la precisión de la detección, la tabla 12 muestra que el porcentaje de detección aumenta a medida que aumenta el tamaño del panel en la imagen. Este valor crece desde alrededor del 82% en paneles que cubren menos del 10% de la imagen hasta el 100% cuando el tamaño del panel ocupa aproximadamente el 50% de la imagen. Además, un hecho interesante de esta red *YOLO* es que detecta una mayor cantidad de paneles en las imágenes que la red *SSD* sin embargo su precisión correspondiente al valor del *IoU* es inferior a la obtenida por el sistema *SSD*. Otro detalle a tener en cuenta es que a medida que aumenta el tamaño de cartel también aumenta la precisión de las detecciones, por ejemplo, la precisión media de los carteles que ocupan menos del 10% de la imagen es de un 63,1% mientras que la precisión de los carteles que ocupan aproximadamente el 50% de la imagen asciende hasta 86,7% de precisión.

Tamaño Panel	Nº Real Paneles	Nº Detecciones	% detecciones	Precisión Media
0- 10%	56	46	82%	0.631
11-20%	37	35	95%	0.712
21-30%	21	21	100%	0.815
31-40%	17	15	88%	0.869
41-50%	9	9	100%	0.867

Tabla 12 Precisión de las detecciones de YOLO por tamaño de cartel

4.2.2 Detecciones por precisión

La tabla 13 muestra como la mayoría de predicciones propuestas por la red alcanzan unos valores de precisión bastante aceptables, aunque menores que los obtenidos con la red *SSD*. Aproximadamente el 43,70% de las detecciones alcanza una precisión superior al 80%, además el 70% de las detecciones tiene una precisión superior al 70%. También hay que mencionar que se producen 9 falsos positivos, lo que se corresponde con un 6,7% de las detecciones.

Precisión	Nº Detecciones
0-10%	9
11-20%	0
21-30%	0
31-40%	0
41-50%	0
51-60%	14
61-70%	18
71-80%	35
81-90%	46
91-100%	13

Tabla 13 Número de detecciones de YOLO para cada valor de IoU

4.2.3 Detecciones por otros aspectos

Para observar cómo afectan las diferentes condiciones de variabilidad de las imágenes a la precisión de las detecciones, se ha evaluado el rendimiento del sistema en función de tres condiciones diferentes bajo las que aparecen los carteles en las imágenes. Los tres aspectos que se han tenido en cuenta han sido:

- ***Nocturnidad:*** Imágenes nocturnas vs Imágenes diurnas.
- ***Frontalidad:*** Imágenes frontales vs Imágenes no frontales.
- ***Oclusiones:*** Imágenes con oclusiones vs Imágenes sin oclusiones.

Para la extracción de estas métricas de rendimiento se han utilizado tres ficheros *JSON* que contienen cada una de las imágenes de test separadas según los criterios de clasificación especificados. Los ficheros *JSON* utilizados son *front_classification*, *nightly_classification* y *occluded_classification*. A partir de estos, se han generado tres informes *CSV*, uno para cada criterio de clasificación, de manera que cada uno de los informes muestra la siguiente información:

- ***Type:*** tipo de imagen, por ejemplo, si es nocturna o diurna.
- ***Iou_Min:*** valor mínimo de *IoU* para un tipo determinado.
- ***IoU_Max:*** valor máximo de *IoU* para un tipo de determinado.
- ***IoU_Mean:*** valor medio de *IoU* para un tipo.
- ***Total_images:*** número total de imágenes de un tipo determinado.
- ***No_detected:*** número de imágenes sin detección.
- **False_positive:** número de falsos positivos en las detecciones.

En la tabla 14, se puede observar que, a diferencia de lo que pueda parecer a priori, las condiciones de nocturnidad de las imágenes no afectan a la precisión de las detecciones, ya

que la precisión media de las detecciones sobre las imágenes nocturnas es de aproximadamente el 84% y el IoU mínimo es del 71%. Otro aspecto importante a tener en cuenta es que en las imágenes nocturnas no se ha producido ningún falso positivo. Por su parte, en las imágenes diurnas, la precisión máxima alcanzada ha sido ligeramente mayor, ya que ha alcanzado un 95%, sin embargo, se puede observar que la precisión media es algo menor, ya que esta es de aproximadamente el 76% si bien es verdad que la mayoría de las imágenes de test se corresponden a imágenes diurnas, por lo que es normal que la precisión media sea algo menor. Además, en estas imágenes diurnas se han producido un total de 9 falsos positivos.

Tipo	IoU Min	IoU Max	IoU Medio	Imágenes Totales	Sin Detecciones	Falsos Positivos
Nocturnas	0.713	0.943	0.841	5	0	0
Diurnas	0.358	0.950	0.763	128	7	9

Tabla 14 Comparativa Imágenes Diurnas vs Nocturnas con YOLO

En la tabla 15 se observa la comparativa entre los carteles publicitarios frontales y los carteles en perspectiva. A pesar de que la precisión media de los carteles en perspectiva o no frontales alcanza unos valores aceptables, ya que la precisión media es del 71%, la precisión mínima bastante baja, aproximadamente del 38,4%. Hay que destacar que a diferencia de la red *SSD*, donde la perspectiva sí que afectaba considerablemente a las detecciones, para la red *YOLO* los resultados obtenidos para las imágenes en perspectiva son muy similares a los obtenidos para los carteles frontales. Por lo tanto, a pesar de que la precisión mínima es bastante baja en ambos casos, se puede considerar que la frontalidad del cartel no afecta ni a la precisión de las detecciones ni a la propia detección del cartel, ya que en ambos casos se producen detecciones en más del 90% de los carteles.

Tipo	IoU Min	IoU Max	IoU Medio	Imágenes Totales	Sin Detecciones	Falsos Positivos
Frontal	0.358	0.950	0.775	113	5	8
No Frontal	0.384	0.924	0.717	20	2	1

Tabla 15 Comparativa Imágenes Frontales vs No Frontales con YOLO

La tabla 16 muestra la comparativa entre las detecciones de carteles publicitarios ocluidos y los carteles sin oclusiones. Se puede observar que la precisión media de las detecciones cuando los carteles publicitarios presentan oclusiones decrece considerablemente en comparación a cuando no se producen oclusiones, ya que la precisión media con oclusiones es del 65,1% mientras que sin oclusiones alcanza hasta el 77,3%. Además, se observa que el número de imágenes sin detecciones también ha aumentado considerablemente, ya que en el 25% de las imágenes con paneles ocluidos no se ha producido ninguna detección mientras que en las imágenes con sin paneles ocluidos solamente en aproximadamente el 3% de las imágenes no se ha producido ninguna detección. Por lo tanto, se puede concluir que las oclusiones producidas sobre los carteles publicitarios tienen un efecto negativo tanto en la precisión de las detecciones como a la hora de localizar el cartel dentro de la imagen.

Tipo	IoU Min	IoU Max	IoU Medio	Imágenes Totales	Sin Detecciones	Falsos Positivos
Ocluidas	0.383	0.825	0.651	12	3	3
No Ocluidas	0.358	0.950	0.773	121	4	6

Tabla 16 Comparativa Imágenes Ocluidas vs No Ocluidas con YOLO

4.3 Comparativa SSD vs YOLO

En este trabajo se ha realizado un estudio acerca de la detección de objetos en imágenes estáticas, en concreto detección de paneles publicitarios en espacios públicos. Para ello, se han propuesto dos soluciones al problema utilizando diferentes arquitecturas convolucionales de *Deep Learning* con capacidad para operar en tiempo real. La primera solución utiliza una arquitectura *Single Shot Detector* (*SSD*) mientras que la segunda solución propuesta utiliza una arquitectura *You Only Look Once* (*YOLO*). Estos sistemas propuestos reciben como entrada una imagen y generan como salida un conjunto de coordenadas que representan las detecciones, sus valores de confianza y la clase a la que pertenece cada detección. Ambas soluciones han sido entrenadas y evaluadas en las mismas condiciones, es decir con el mismo *dataset* y con la misma distribución de imágenes para entrenamiento, validación y test.

Debido a la ausencia de *datasets* acerca de carteles publicitarios, ha sido necesario crear un *dataset* propio con imágenes que contengan al menos un cartel publicitario. Ya que se trata de carteles publicitarios en espacios públicos de exterior, es necesario dotar al *dataset* de la variabilidad suficiente a través de escenas reales que recojan las diferentes condiciones puedan afectar al rendimiento del sistema. Para ello, se han recopilado imágenes en las que los carteles aparecen en diferentes condiciones climatológicas como lluvia o nieve, en diferentes momentos del día como por la noche o al atardecer y a diferentes distancias para obtener distintos tamaños de cartel. En total el *dataset* creado contiene aproximadamente 6000 imágenes de carteles publicitarios de manera que el 92% de las imágenes se utilizan para entrenamiento, el 5% se utilizan para validación y el 3% se utilizan como imágenes de test.

Durante el proceso de entrenamiento la red extrae automáticamente las características más importantes que distinguen a los carteles publicitarios. Esta extracción se lleva a cabo en las capas convolucionales de la red que extraen mapas con las características más relevantes. Con su potencia, este tipo de capas son capaces de inferir características mucho más discriminantes que a simple vista no se pueden apreciar y que nos permiten realizar una detección más precisa.

En cuanto a los resultados obtenidos para cada una de las soluciones propuestas en las pruebas de evaluación realizadas, se puede concluir que la arquitectura de red *SSD* es capaz de realizar unas detecciones más precisas que *YOLO* y con una tasa de falsos positivos mucho menor, sin embargo *YOLO* obtiene una tasa de falso rechazo considerablemente menor, un 10% frente al 30% obtenido por la red *SSD*, lo que significa que es capaz de detectar una cantidad superior de carteles. Además, las métricas de evaluación en función de los criterios de nocturnidad, frontalidad y oclusiones han revelado que las condiciones

de nocturnidad no afectan ni a *SSD* ni a *YOLO*. Sin embargo, las condiciones de frontalidad en las imágenes afectan a la precisión de la red *SSD* pero no a *YOLO*, por lo que las imágenes no frontales son más complejas de detectar para la red *SSD*. Por último, los resultados muestran que las oclusiones producidas sobre las imágenes sí que afectan a ambas redes, ya que en ambos casos la precisión ha sido notablemente inferior cuando los carteles aparecían ocluidos en la imagen respecto a cuando no se producía ninguna oclusión.

5. Conclusiones

En esta sección, se resumen las conclusiones de este trabajo de fin de máster así como el trabajo futuro que no se ha podido desarrollar durante el presente trabajo.

5.1 Conclusiones Finales

En este trabajo aborda el problema de la detección y localización de carteles publicitarios en escenas exteriores. Este problema resulta interesante para diferentes sectores de la industria, como la publicidad o el mantenimiento. El sector publicitario puede aprovechar este tipo de soluciones para reemplazar el contenido de un cartel publicitario por otro nuevo de mayor interés. También se puede emplear este tipo de soluciones para la comprobación automática del mobiliario urbano facilitando su mantenimiento de manera que se minimiza la intervención humana.

Las grandes dificultades a las que nos enfrentamos a la hora de tratar con un problema como éste son principalmente la variabilidad en la forma y tamaño de los carteles publicitarios, las condiciones climatológicas adversas y de iluminación debido a su localización en el exterior, las oclusiones de los carteles por parte de elementos externos y la propia complejidad del entorno.

Se trata de un trabajo novedoso, ya que hasta la fecha no existe ninguna publicación dedicada a la detección de carteles publicitarios en imágenes estáticas, ya que los trabajos más similares a éste abordan la detección de señales de tráfico [1], detección de matrículas en automóviles [21] o la detección de carteles de paneles de tráfico en autopistas [13]. Como consecuencia de ello, tampoco existen bases de datos públicas acerca de carteles publicitarios, por lo que ha sido necesario crear nuestra propia base de datos a partir de imágenes de carteles publicitarios recopiladas principalmente de *Google Street View* y *Google Images*. La base de datos creada está orientada a los problemas de detección de objetos utilizando una metodología de aprendizaje supervisado, ya que cada uno de los carteles de las imágenes ha sido etiquetado mediante la herramienta *VGG Image Annotator,* que permite seleccionar las coordenadas que delimitan la región de la imagen perteneciente al cartel publicitario. En total, la base de datos está formada por 6000 imágenes aproximadamente que dotan a ésta con la variabilidad suficiente como para actuar en un entorno real de manera eficiente.

Las soluciones propuestas para el problema localización y detección de carteles publicitarios están basadas en el paradigma de *Deep Learning*, más concretamente en las redes neuronales de convolución. Se han utilizado dos arquitecturas de red capaces de trabajar en tiempo real minimizando el número de falsos positivos, como son *Single Shot Detector* (*SSD*) y *You Only Look Once* (*YOLO*). Ambas arquitecturas funcionan en dos fases: la primera consiste en la extracción de características mediante las capas de convolución mientras que la segunda fase consiste en la obtención de los *bounding boxes* mediante regresión. Como resultado, ambos redes devuelven las coordenadas de los *bounding boxes* pertenecientes a las detecciones en las imágenes junto con etiqueta de clase y su medida de confianza respectivamente.

Ambas soluciones han sido evaluadas en igualdad de condiciones utilizando el *dataset* de carteles publicitarios creado en este trabajo. Los resultados de las pruebas de evaluación realizadas han demostrado que ambas redes neuronales funcionan de manera adecuada

sobre el conjunto de datos de test, ya que ambas superan el 70% de precisión media. La red *YOLO* es capaz de detectar una mayor cantidad de carteles publicitarios contenidos en las imágenes si bien, su precisión media resulta ligeramente inferior respecto a la red *SSD*. Además, se han obtenido resultados en base a diversos criterios como la nocturnidad, la frontalidad y las oclusiones. Los resultados han mostrado que la nocturnidad no afecta a la precisión de las detecciones de *YOLO* y *SSD*. Sin embargo, otros factores como la frontalidad del cartel o las oclusiones sí que afectan a la precisión.

5.2 Trabajo Futuro

A pesar de obtenerse unos resultados aceptables en la detección y localización de los carteles publicitarios, uno de los objetivos de futuro es evaluar el funcionamiento de ambos sistemas haciendo uso de otra base de datos específica para este tipo de problema. En abril de 2019 se ha publicado un trabajo que ha creado una base de datos llamada *ALOS* dedicada a la detección carteles publicitarios, si bien esta base de datos de momento no se encuentra disponible públicamente ya que el trabajo todavía se encuentra un una fase preliminar. Por lo tanto, no se ha podido realizar ninguna prueba de evaluación con dicho dataset.

Además, otra de las futuras investigaciones sería abordar el problema haciendo uso de otro tipo de redes neuronales que permiten realizar la segmentación a nivel de píxeles como por ejemplo *Mask R-CNN*. Estas pruebas, utilizando el mismo *dataset* que se ha creado en este trabajo, permitirían comparar la precisión de los sistemas de detección de objetos propuestos junto con el problema de segmentación a nivel de píxel, ya que obtener una máscara binaria a partir de los *bounding boxes* detectados resulta una tarea sencilla. De esta manera sería posible valorar si el esfuerzo que supone entrenar una red de segmentación es compensado con una precisión mayor en el sistema de segmentación semántica a nivel de píxel en problemas de este tipo.

Por último, resultaría interesante, que una vez detectados los carteles publicitarios en la imagen, desarrollar un sistema para sustituir en dichos carteles la publicidad real por una publicidad virtual basada en la experiencia previa del usuario o en base a unos intereses específicos orientados a los intereses de los propios usuario.

Como resultado del presente trabajo, se ha realizado una publicación [41] en el libro *Highlights of Practical Applications of Survivable Agents and Multi-Agent Systems. The PAAMS Collection*, pp.246-256.

Bibliografía

[1] Garcia, M., Sotelo, M., Martin, E.: Track sign detection in static images using matlab. In: IEEE Conference on Emerging Technologies and Factory Automation (ETFA '03) (2003)

[2] Aldershoff, Frank & Gevers, T. (2004). Visual tracking and localization of billboards in streamed soccer matches. 408-416. 10.1117/12.526871.

[3] Vazquez-Reina, Amelio & J. López Sastre, R & Lafuente Arroyo, S & Gil-Jiménez, Pedro. (2006). Adaptive traffic road sign panels text extraction. 295-300.

[4] R. Smith, "An overview of the tesseract ocr engine," in Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02, ser. ICDAR '07, 2007, pp. 629–633.

[5] Murty, R., Mainland, G., Rose, I., Chowdhury, A., Gosain, A., Bers, J., Welsh, M.: Citysense: An urban-scale wireless sensor network and testbed. In: 2008 IEEE International Conference on Technologies for Homeland Security (2008)

[6] Watve, A., Sural, S.: Soccer video processing for the detection of advertisement billboards. Pattern Recognition Letters 29(7), 994{1006 (2008).

[7] Ballan, Lamberto & Bertini, Marco & Jain, Arjun. (2008). A system for automatic detection and recognition of advertising trademarks in sports videos. MM'08 Proceedings of the 2008 ACM International Conference on Multimedia, with co-located Symposium and Workshops.

[8] Moutarde, Fabien & Bargeton, Alexandre & Herbin, Anne & Chanussot, Lowik. (2009). Modular Traffic Sign Recognition applied to on-vehicle real-time visual detection of American and European speed limit signs.

[9] Kardkovács, Zsolt & Paróczi, Zsombor & Varga, E & Siegler, Adam & Lucz, P. (2011). Real-time traffic sign recognition system. 1-5.

[10] Huang, Y., Hao, Q., Yu, H.: Virtual ads insertion in street building views for augmented reality. 18th IEEE International Conference on Image Processing (ICIP '11) pp. 1117{1120 (2011).

[11] Á. Gonzalez et al., "Automatic Traffic Signs and Panels Inspection System Using Computer Vision," in IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 2, pp. 485-499, June 2011.

[12] Á. González, L. M. Bergasa, J. J. Yebes and J. Almazán, "Text recognition on traffic panels from street-level imagery," 2012 IEEE Intelligent Vehicles Symposium, Alcala de Henares, 2012, pp. 340-345.

[13] Á. González, L. M. Bergasa and J. J. Yebes, "Text Detection and Recognition on Traffic Panels From Street-Level Imagery Using Visual Appearance," in IEEE Transactions on Intelligent Transportation Systems, vol. 15, no. 1, pp. 228-238, Feb. 2014.

[14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, 2014, pp. 740–755.

[15] Wong, D., Deguchi, D., Ide, I., Murase, H.: Vision-based vehicle localization using a visual street map with embedded surf scale. In: European Conference on Computer Vision (ECCV '14). pp. 167{179. Springer (2015)

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in NIPS, 2015, pp. 91–99.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," CoRR, vol. abs/1512.03385, 2015.

[18] R. Girshick, "Fast r-cnn," arXiv preprint arXiv:1504.08083, 2015.

[19] Bojarski, Mariusz & Del Testa, Davide & Dworakowski, Daniel & Firner, Bernhard & Flepp, Beat & Goyal, Prasoon & D. Jackel, Lawrence & Monfort, Mathew & Muller, Urs & Zhang, Jiakai & Zhang, Xin & Zhao, Jake & Zieba, Karol. (2016). End to End Learning for Self-Driving Cars.

[20] Jung, Seokwoo & Lee, Unghui & Jung, Jiwon & Hyunchul Shim, David. (2016). Real time Traffic Sign Recognition system with deep convolutional neural network. 31-34. 10.1109/URAI.2016.7734014.

[21] Panchal, T., Patel, H., Panchal, A.: License plate detection using harris corner and character segmentation by integrated approach from an image. Procedia Computer Science 79, 419{425 (2016).

[22] Zhu, Zhe & Liang, Dun & Zhang, Songhai & Huang, Xiaolei & Li, Baoli & Hu, Shimin. (2016). Traffic-Sign Detection and Classification in the Wild. 2110-2118. 10.1109/CVPR.2016.232

[23] Jung, Seokwoo & Lee, Unghui & Jung, Jiwon & Hyunchul Shim, David. (2016). Real-time Traffic Sign Recognition system with deep convolutional neural network. 31-34. 10.1109/URAI.2016.7734014.

[24] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in ECCV. Springer, 2016, pp. 21–37.

[25] Anthopoulos, L.: Understanding Smart Cities: A Tool for Smart Government or an Industrial Trick? Springer (2017)

[26] Chi, Lu & Mu, Yadong. (2017). Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues.

[27] Y. Wei, N. Song, L. Ke, M. Chang and S. Lyu, "Street object detection / tracking for AI city traffic analysis," 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld / SCALCOM / UIC / ATC / CBDCom / IOP / SCI), San Francisco, CA, 2017, pp. 1-5.

[28] Chi, Lu & Mu, Yadong. (2017). Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues.

[29] “Google TensorFlow Object Detection API,” https://research.googleblog.com/2017/06/supercharge-your-computer-vision models.html.

[30] Intasuwan, T., Kaewthong, J., Vittayakorn, S.: Text and object detection on billboards. 10th International Conference on Information Technology and Electrical Engineering (ICITEE '18) pp. 6{11 (2018)

[31] He, Kaiming & Gkioxari, Georgia & Dollar, Piotr & Girshick, Ross. (2018). Mask R-CNN. IEEE Transactions on Pattern Analysis and Machine Intelligence. PP. 1-1. 10.1109/TPAMI.2018.2844175.

[32] Fernández Alcantarilla, Pablo & Stent, Simon & Ros, Germán & Arroyo, Roberto & Gherardi, Riccardo. (2018). Street-view change detection with deconvolutional networks. Autonomous Robots. 42. 10.1007/s10514-018-9734-5.

[33] Hossari, Murhaf & Dev, Soumyabrata & Nicholson, Matthew & McCabe, Killian & Nautiyal, Atul & Conran, Clare & Tang, Jian & Xu, Wei & Pitié, François. (2018). ADNet: A Deep Network for Detecting Adverts.

[34] “Cloud vision api,” accessed 2018-04-05. [Online]. https://cloud.google.com/vision/

[35] Mapillary Vistas Dataset: https://www.mapillary.com/dataset/vistas

[36] ALOS Dataset: https://arxiv.org/pdf/1904.07776.pdf

[37] Cartociudad: http://www.cartociudad.es/

[38] Dutta, A., Gupta, A., Zissermann, A.: VGG Image Annotator (VIA) - Version: 1.0.6 (2016 (Accessed: 19 02 05)), http://www.robots.ox.ac.uk/ vgg/software/via

[39] Ho-Phuoc, Tien. (2018). CIFAR10 to Compare Visual Recognition Performance between Deep Neural Networks and Humans.

[40] https://pjreddie.com/darknet/yolo/

[41] Morera, Angel & Sanchez, Angel & D. Sappa, Ángel & Vélez, Jose. (2019). Robust Detection of Outdoor Urban Advertising Panels in Static Images. 10.1007/978-3-030-24299-2_21.