Universidad Zaragoza
1542

Trabajo Fin de Grado

Seguimiento y Segmentación de Múltiples Objetos con Descriptores Aprendidos

Multi-Object Tracking and Segmentation with Learnt Descriptors

Autor

Daniel Cay Delgado

Directora

Berta Bescós Torcal

Codirector

José Neira Parra

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2020

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Daniel Cay Delgado,

con nº de DNI 73229695H en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Seguimiento y Segmentación de Múltiples Objetos con Descriptores Aprendidos,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 10 de noviembre de 2020

Fdo: Daniel Cay Delgado

Seguimiento y Segmentación de Múltiples Objetos con Descriptores Aprendidos

RESUMEN

En este TFG se ha estudiado uno de los problemas a los que se enfrenta la visión por computador en la actualidad, que es el del seguimiento y segmentación de múltiples objetos a lo largo de un vídeo. Dicho problema consiste en identificar de forma única a cada uno de los objetos que aparecen en un vídeo mediante una máscara que se adapta a su forma y contorno a nivel de píxel (*instance segmentation*).

En los últimos años, los sistemas de seguimiento de objetos han estado estancados debido a que la tecnología de *object detection* (identificar a cada objeto con una caja delimitadora o *bounding box* que lo rodea) se había explotado al máximo. En la actualidad, gracias a la aparición de la *instance segmentation*, la posibilidad de lograr grandes mejoras ha vuelto a aparecer. Es por ello que se ha desarrollado un sistema de seguimiento y segmentación de objetos que parte del uso de técnicas clásicas de visión por computador y de las últimas tecnologías en el ámbito, como es la red neuronal de segmentación semántica Mask R-CNN. Se busca utilizar nuevos métodos con el fin de diferenciar el sistema realizado del resto, aportando así nueva información acerca de este problema. La principal innovación del trabajo se basa en el uso de descriptores aprendidos, información sobre la apariencia de los objetos a seguir que se va a extraer de Mask R-CNN para utilizarla en beneficio de nuestro sistema, logrando así mejoras en su funcionamiento.

En cuanto al desarrollo del trabajo, se ha realizado una primera aproximación al seguimiento usando el solapamiento entre objetos de distintas imágenes. Después, se ha utilizado un algoritmo de predicción con el fin de solucionar algunos problemas que la primera aproximación tiene, y tras esto, se ha incorporado el uso de los descriptores aprendidos extraídos de Mask R-CNN para tener en cuenta la apariencia de los objetos, mejorando así los resultados del sistema. Por último, para evaluar el trabajo desarrollado y compararlo con el estado del arte se han usado las métricas de visión por computador MOTSA, MOTSP y sMOTSA.

Para el beneficio de la comunidad de la visión por computador, el sistema está disponible en `https://github.com/DanielCay/TFG` y un ejemplo del resultado final logrado se puede ver en `https://youtu.be/Xw1aob3RjWw`.

Multi-Object Tracking and Segmentation with Learnt Descriptors

ABSTRACT

This Bachelor Thesis has studied one of the problems facing computer vision today, the tracking and segmentation of multiple objects throughout a video. This problem consists of uniquely identifying each of the objects that appear in a video by means of a mask that adapts to its shape and its outline at pixel level (instance segmentation).

In recent years, object tracking systems have been stagnating because the technology of object detection (identify each object with a bounding box around it) had been exploited to the full. Today, thanks to the emergence of instance segmentation, the possibility of achieving great improvements has reappeared. For this reason, an object tracking and segmentation system has been developed, based on the classic computer vision techniques and the latest technologies in the field, such as the semantic segmentation neural network called Mask R-CNN. The aim is to use new techniques in order to differentiate the system carried out from the rest, thus providing new information about this problem. The main innovation of our work is based on the learnt descriptors, information about the appearance of the objects to be followed that will be extracted from Mask R-CNN to be used in the benefit of our system, thus achieving improvements in its operation.

As for the development of the work, a first approach to the tracking has been made with the overlapping between objects of different images. Then, a prediction algorithm has been used in order to solve some problems that the first approach has, and after that, the learnt descriptors extracted from Mask R-CNN have been incorporated to take into account the appearance of the objects, improving the results of the system. Finally, the computer vision metrics MOTSA, MOTSP and sMOTSA have been used to evaluate the work developed and compare it with the state of the art.

For the benefit of the computer vision community, the system is available at `https://github.com/DanielCay/TFG` and an example of the final result can be seen at `https://youtu.be/Xw1aob3RjWw`.

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a mis padres y a mi hermano todo el apoyo que me han dado siempre, y en especial estos últimos años durante la carrera que han sido tan duros física y psicológicamente.

Por otro lado, quiero dar las gracias a Berta Bescós, directora del TFG, y al codirector José Neira, porque a lo largo del trabajo me han ayudado siempre que lo he necesitado y me han permitido ampliar mis conocimientos en el campo que me gusta.

Por último quiero mencionar a mis amigos, que han sido la mejor forma de evadirme y desconectar cuando lo he necesitado.

Índice

Capítulo 1

Introducción

La visión por computador tiene por objetivo replicar el comportamiento del ojo humano analizando imágenes y vídeos de la misma manera que lo hacen las personas. Muchas aplicaciones dentro del campo de la visión por computador se han visto impulsadas en los últimos años gracias a los avances en inteligencia artificial y en aprendizaje profundo (*deep learning* [21]), que han permitido incluso sobrepasar las capacidades del ser humano realizando determinadas tareas. Ejemplos de estas aplicaciones son el reconocimiento y segmentación de objetos, los vehículos autónomos, *etc.*

Uno de los factores clave para lograr los avances en estas áreas es la cantidad de datos a la que se tiene acceso hoy en día. Otro es la disposición de la capacidad computacional requerida para procesar toda esa información. Lo anterior, junto con el desarrollo de nuevos algoritmos, ha permitido que ciertos sistemas sean más precisos que los humanos a la hora de reaccionar a estímulos visuales. Los primeros experimentos de visión por computador comenzaron en los años 50, siendo en los años 70 cuando se empezó a usar comercialmente para distinguir textos mecanografiados y escritos a mano, pero es ahora cuando el desarrollo de este campo está sufriendo un crecimiento exponencial.

En este contexto, uno de los problemas planteados dentro de la visión por computador es el de seguimiento de objetos a lo largo de una secuencia de vídeo identificando de forma única a cada uno de los objetos que aparecen. Los estudios recientes relacionados con este problema muestran que el seguimiento basado en detectar e identificar objetos con una caja que los rodea (*object detection*) está comenzando a saturarse, es decir, es muy difícil mejorar los resultados actuales con este método (Figura 1.1a). Por ello, las mejoras sólo se ven posibles si para realizar el seguimiento se identifica a los objetos de una manera más precisa, con una máscara que se adapte lo mejor posible a su contorno (*instance segmentation*). Este problema se conoce como seguimiento y segmentación de múltiples objetos (*multi-object tracking*

and segmentation). A diferencia del seguimiento de objetos basado en la *object detection*, este método aún está por explotar, y aún está por ver todo su potencial (Figura 1.1b).

(a) Ejemplo *object detection* [16]

(b) Ejemplo *instance segmentation* [26]

Figura 1.1: *Object detection* e *instance segmentation*

1.1. Motivación

Para 2022 se estima que la visión por computador y el mercado de hardware muevan la cantidad de 48.6 billones de dólares al año. Ese dato refleja el auge que experimenta el campo de la visión por computador, motivo por el cual se ha decidido realizar un sistema de detección y seguimiento de objetos. Se va a buscar utilizar los últimos avances en este campo, al mismo tiempo que se exploran nuevas técnicas que diferencien al sistema desarrollado de otros realizados hasta la fecha.

1.2. Objetivos y metodología

El objetivo principal del proyecto es el desarrollo, adaptación, mejora y evaluación de algoritmos para la detección, segmentación y seguimiento de objetos móviles en secuencias de vídeo obtenidas con una cámara también móvil. La solución a este problema tiene aplicaciones en muchos campos relacionados con la robótica móvil, por ejemplo, los coches autónomos. Para ello, se va a desarrollar un sistema que, en una secuencia de fotogramas, sea capaz de detectar los objetos potencialmente móviles que aparecen, delimitarlos en su región de movimiento (con una *bounding box*), identificarlos por su apariencia en el fotograma (máscara a nivel de píxel), y estimar el movimiento de cada objeto separadamente a lo largo de dicha secuencia. Para realizar este sistema, se va a utilizar como base la red neuronal Mask R-CNN [11], que dada una imagen es capaz de realizar la detección de objetos y su segmentación. Se va a modificar dicha red con el fin de incorporarle el seguimiento de objetos a lo largo de una secuencia de vídeo. Como se ha comentado en la introducción, esta técnica recibe el nombre de *multi-object tracking and segmentation*.

Para abordar el trabajo, se han definido una serie de objetivos parciales que se enumeran a continuación:

1. Realizar un estudio del estado del arte relacionado con la detección y seguimiento de objetos (*object tracking*).

2. Aprender a usar la red neuronal Mask R-CNN, la base del sistema desarrollado. Tras ello, se deberá estudiar en profundidad con el fin de saber cómo modificarla y adaptarla para las necesidades del sistema a implementar.

3. Desarrollar un sistema de seguimiento de objetos únicamente con información sobre la posición que estos ocupan.

4. Añadir un algoritmo de predicción para que el sistema sea más robusto.

5. Utilizar información asociada a la apariencia de los objetos.

6. Evaluar el comportamiento del sistema utilizando las métricas comúnmente usadas para evaluar sistemas de seguimiento (CLEAR MOTS [2]).

Se va a seguir el modelo de desarrollo en espiral, modelo iterativo en el que en cada una de sus iteraciones se debe: determinar objetivos, analizar riesgos, desarrollar el software y verificarlo. En la Figura 1.2 se muestra el diagrama de Gantt del proyecto.

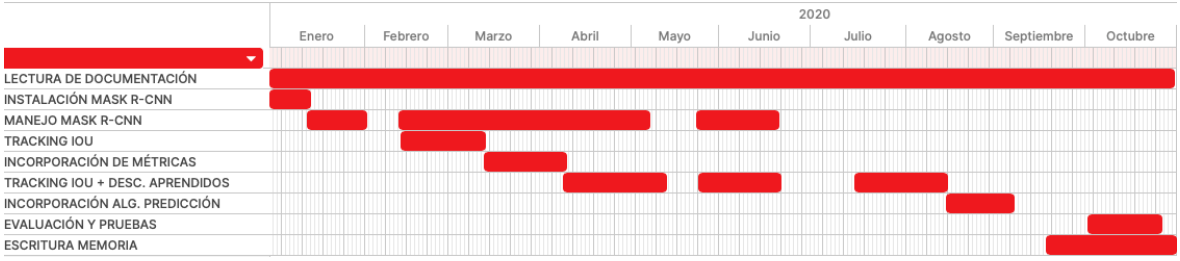


Figura 1.2: Diagrama de Gantt del TFG

1.3. Estructura de la memoria

Tras presentar la introducción, la motivación que ha llevado a realizar el proyecto, y los objetivos y metodologías que se han tenido en cuenta, se va a mostrar la estructura que sigue el documento.

En el Capítulo 2, aparece una visión general de algunos campos de la informática con los que está relacionado el proyecto realizado. Se va a dar una breve introducción a los campos de la visión por computador, aprendizaje automático [6] y *deep learning* (redes neuronales), y dentro de este último se hace hincapié en las redes neuronales

convolucionales (CNN). La lectura de este capítulo busca facilitar el entendimiento del resto del documento.

El Capítulo 3 se ocupa de explicar el estado del arte acerca del seguimiento y segmentación de múltiples objetos. Por un lado se hace referencia a la red neuronal Mask R-CNN, que se utiliza como punto de partida del proyecto desarrollado, y es lo más reciente en segmentación de múltiples objetos dada una imagen. Se va a realizar un repaso superficial de su estructura y de lo que es capaz de hacer. Por otro lado, se ahonda en la red neuronal Track R-CNN, la cual tiene el mismo objetivo que el sistema implementado, realizar el seguimiento y segmentación de múltiples objetos (usando diferentes técnicas a nuestro sistema). También se va a explicar su estructura a grandes rasgos y cómo se comporta.

En cuanto al Capítulo 4, este se centra en explicar el diseño e implementación del proyecto. En primer lugar, se muestra el punto de partida, las tecnologías usadas para la implementación y los *datasets* utilizados. Después, comienza la explicación de cómo se ha ido desarrollando el sistema, pasando por una primera etapa de familiarización con Mask R-CNN y selección de clases de objetos sobre los que se va a realizar el seguimiento, para después entrar en una primera aproximación al seguimiento y segmentación de objetos en la que se utiliza el solapamiento que hay entre las posiciones que ocupan los objetos. A la primera versión implementada se le van añadiendo partes que provocan una mejora en el funcionamiento del sistema, como son el algoritmo de predicción, y el uso de descriptores aprendidos para tener en cuenta la apariencia de los objetos, siendo esto último el punto más innovador del proyecto.

El Capítulo 5 se centra en la realización y análisis de diversas pruebas que van a determinar cómo de bien funciona el sistema y cómo afectan las distintas mejoras a su comportamiento. En primer lugar se explican las métricas utilizadas, a continuación se muestran pruebas para justificar la selección de determinados parámetros, y por último, se realiza una comparativa entre los resultados de nuestro sistema y los de Track R-CNN, la red neuronal que representa el estado del arte en lo que al problema enfrentado se refiere.

Para terminar, en el Capítulo 6 se exponen las conclusiones obtenidas tras el desarrollo y evaluación del proyecto, pudiendo ver así la aportación que el sistema realizado tiene para el problema enfrentado. Tras esto, se muestra un apartado de modos de fallo y líneas futuras en el que se analizan los puntos débiles del sistema, y se proponen mejoras y pruebas a realizar con el fin de perfeccionar su funcionamiento.

Capítulo 2

Fundamentos

En este apartado se dan unas bases sobre los temas relacionados con el sistema desarrollado para facilitar el entendimiento de los apartados posteriores. Se muestra una aproximación a la visión por computador, aprendizaje automático y redes neuronales. Dentro de estas últimas se hará referencia más concretamente a las CNNs.

2.1. Visión por computador

La visión por computador es una campo de la ciencia computacional que se centra en replicar partes de la visión humana con el fin de lograr identificar y procesar objetos en imágenes y vídeos de la misma forma que lo hacen los humanos.

Antes de la llegada del *deep learning*, la precisión de la visión por computador estaba limitada en algunas aplicaciones. Por ejemplo, si se quería usar para reconocimiento facial se debían seguir estos pasos:

1. Crear una base de datos con imágenes de la cara de todas las personas que se quiere ser capaz de reconocer.

2. Por cada una de las imágenes, extraer información clave que permita diferenciar la cara de una persona de todas las demás, como la distancia entre los ojos, la distancia entre el labio superior y la nariz, la anchura del tabique nasal, *etc.*

3. Por último, conseguir nuevas imágenes de esas personas y calcular de nuevo las medidas del paso anterior, teniendo en cuenta el ángulo en el que las nuevas imágenes fueron tomadas.

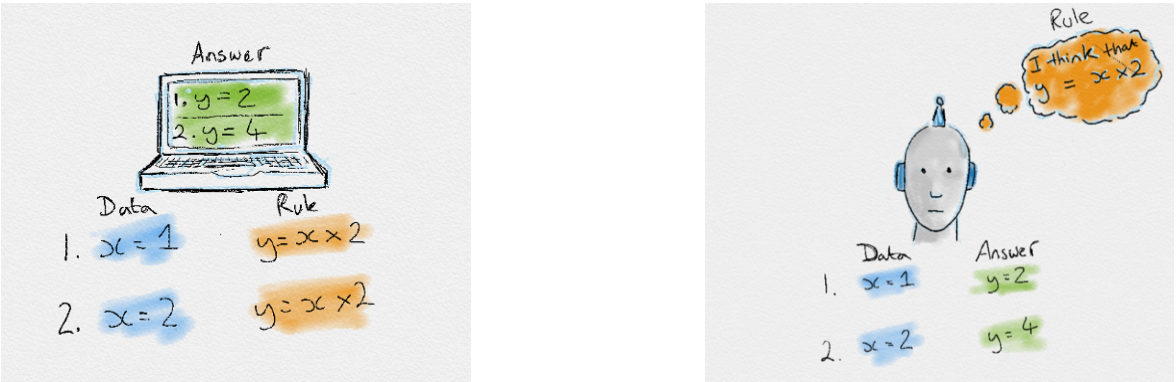
Una vez realizado este trabajo, el sistema sería capaz de decir a qué persona de la base de datos se corresponde la nueva imagen. Se puede observar que la mayoría del trabajo realizado es manual, la automatización es mínima, y aún así el error es alto.

Aquí es donde juega un papel muy importante el aprendizaje automático, que supone una aproximación distinta para la resolución de los problemas de la visión por computador. Con el aprendizaje automático, los desarrolladores ya no tienen que escribir cada una de las reglas que componen sus aplicaciones, sino que elaboran programas capaces de detectar por ellos mismos patrones en las imágenes o en otros tipos de información, consiguiendo así resolver muchos problemas que hasta ahora se consideraban muy complejos. En la actualidad, el subcampo del aprendizaje automático conocido como *deep learning*, que se basa en el uso de las redes neuronales, está permitiendo lograr grandes avances, es por eso que la visión por computador está en auge y crece exponencialmente.

La visión por computador ya está en muchos de los productos que se usan diariamente, como en los coches autónomos, en las aplicaciones de reconocimiento facial, en sistemas para el mundo sanitario, sistemas de realidad aumentada, *etc.*

2.2. Aprendizaje automático

En los últimos 50 años ha habido una explosión de información. Todos estos datos de los que se dispone hoy en día solo son útiles si se analizan y se buscan los patrones escondidos en ellos. El aprendizaje automático o *machine learning* [6] se usa para encontrar de forma automática estos patrones que de otra forma serían muy difíciles y costosos de obtener, para que una vez hallados, pueden usarse para predecir eventos futuros y tomar todo tipo de decisiones complejas. Tradicionalmente, los algoritmos combinan reglas creadas por los seres humanos con los datos de los que se dispone con el fin de dar respuestas a un problema (2.1a). En cambio, el aprendizaje automático usa los datos y las respuestas a un problema para descubrir y aprender las reglas que hay detrás de ese problema (2.1b).



(a) (b)

Figura 2.1: Comportamiento de los algoritmos tradicionales (2.1a) frente al comportamiento de los algoritmos de aprendizaje automático (2.1b) [6]

Los pasos a seguir para desarrollar un algoritmo de aprendizaje automático son:

1. Recolectar los datos para que el algoritmo aprenda.
2. Preprocesar los datos para que estén en el formato óptimo para que el sistema aprenda lo mejor posible.
3. Entrenar el sistema alimentándolo con esos datos.
4. Comprobar si el modelo es capaz de realizar buenas predicciones en base a las reglas que ha aprendido, utilizando para ello nuevos datos de entrada.

La terminología principal asociada a este campo es:

- *Dataset*: Conjunto de datos con características (*features*) relevantes para resolver el problema en cuestión. Se compone de:
  - Datos de entrenamiento: Entrenar el sistema.
  - Datos de validación: Comprobar si el sistema está aprendiendo bien.
  - Datos de test: Comprobar si el sistema ha aprendido correctamente.
- Características (*features*): Rasgos que caracterizan los datos y que alimentan al algoritmo de aprendizaje automático para ayudar a que este aprenda.
- Modelo: Es el algoritmo entrenado listo para utilizarse.

Existe una gran variedad de algoritmos dentro del aprendizaje automático que se pueden clasificar entre los algoritmos de aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semi-supervisado y aprendizaje por refuerzo.

Por último, existe un subcampo del aprendizaje automático conocido como *deep learning* que está basado en las redes neuronales. A continuación, se va a proceder a explicar sus bases para entender el por qué del gran auge de las redes neuronales en la actualidad y el qué las hace tan especiales.

2.3. Redes neuronales

Las redes neuronales [21] son en torno a lo que gira el *deep learning*. Al fin y al cabo, tienen el mismo objetivo que cualquier otro algoritmo de aprendizaje automático, hacer buenas predicciones.

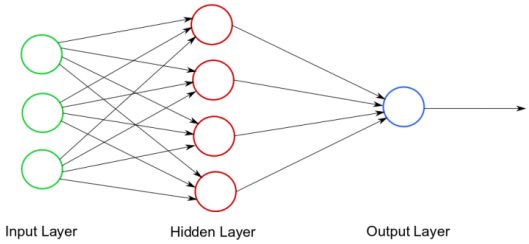


Figura 2.2: Red neuronal básica

Si se observa la Figura 2.2, una red neuronal se compone normalmente de: 1. la capa de entrada (*input layer*) de color verde, 2. un número variable de capas ocultas (*hidden layers*) de color rojo, y 3. la capa de salida (*output layer*) de color azul.

Los pasos a seguir con una red neuronal son los mismos que los comentados para cualquier otro algoritmo de aprendizaje automático, la diferencia radica en cómo aprenden, ya que buscan imitar el comportamiento del cerebro humano. Su entrenamiento consiste en introducir un dato de entrada a través de la *input layer* y conocer el valor objetivo (*label* o etiqueta) que debería devolver la *output layer* para esa entrada. Por cada dato de entrada, la red predice una salida, y se compara dicha salida con la etiqueta. El entrenamiento de la red tiene por objetivo ir reduciendo lo máximo posible el error entre la etiqueta y la salida dada por la red para cada dato de entrada. La red debe recibir muchos datos de entrada distintos, de manera que las neuronas que la componen, conforme van viendo nuevos datos, vayan ajustando una serie de pesos con el fin de ir reduciendo el error entre los valores esperados para cada entrada (sus etiquetas), y los valores que predice la red para esas entradas.

Por ejemplo, si se quiere crear una red capaz de diferenciar entre imágenes de perros y gatos será necesario entrenar dicha red con muchas imágenes de ambos animales. Véase en la Figura 2.3 que al introducir la imagen de un gato, la red predice que hay un 75 % de probabilidades de que sea un gato y un 25 % de que sea un perro. La salida objetivo (la etiqueta para esa entrada) sería que dijera con un 100 % de probabilidad que esa entrada es un gato. Como ha habido un error entre la salida esperada y la dada por la red, la red neuronal va a realizar ajustes en sus neuronas para disminuir dicho error para futuras entradas.

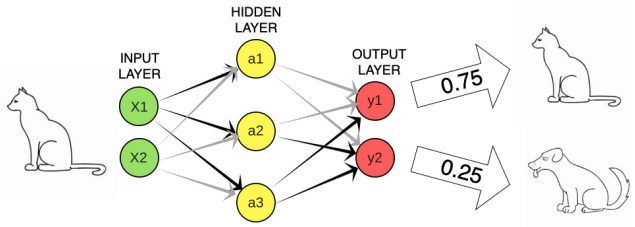


Figura 2.3: Ejemplo de red que devuelve una salida en base a la entrada

A continuación se definen alguno términos asociados a las redes neuronales:

- Función de pérdida (*loss*): Calcula el error entre la salida dada por la red y la salida objetivo para cada dato de entrada. El objetivo es que conforme aprenda la red su valor se vaya reduciendo.

- Función de activación: Define la salida de una neurona para cada entrada que le llega. Hay muchas distintas: sigmoid, relu, softmax...

- Época: Constituye una pasada de todos los datos de entrenamiento por la red. Las redes se suelen entrenar durante varias épocas, por lo que los datos de entrenamiento se pasan varias veces por la red.

- Ratio de aprendizaje: Determina cómo de rápido intenta aprender la red. Cuanto más rápido, más probable que no aprenda tan bien.

- Sobreentrenamiento: Se da cuando la red aprende demasiado bien los datos de entrenamiento, de manera que cuando se usa con datos que no ha visto durante el entrenamiento funciona mal al no generalizar correctamente.

2.3.1. CNNs - Redes neuronales convolucionales

Originalmente, los modelos de clasificación de imágenes utilizaban los píxeles directamente para clasificar las imágenes. Por ejemplo, se pueden clasificar distintas razas de gatos usando características como su color con el histograma de colores de los píxeles de la imagen, y la forma de sus orejas usando detección de bordes. Este método da resultados aceptables para casos sencillos. Sin embargo, obtener características de mayor complejidad permite enfrentarse a situaciones más complejas.

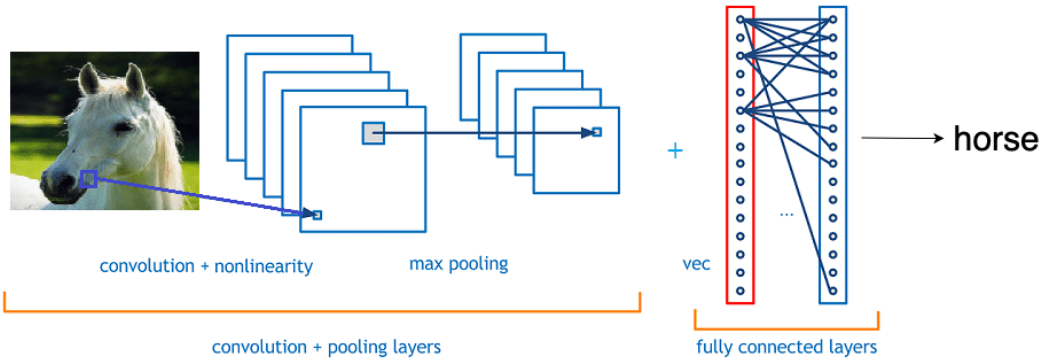


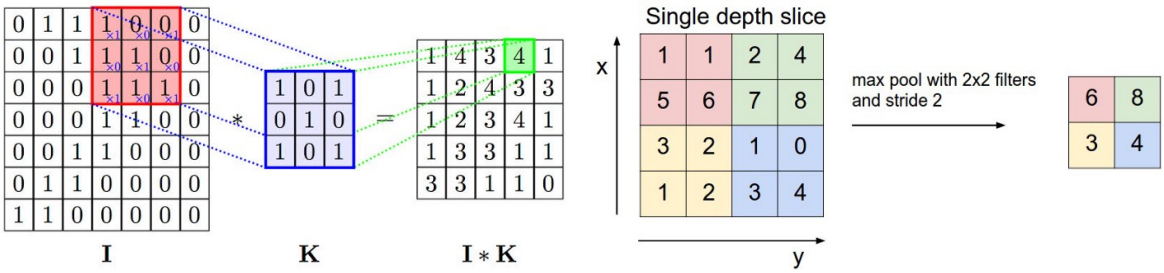
Figura 2.4: Red neuronal convolucional (CNN) [7]

Una CNN [14, 17] es un tipo de red neuronal que a diferencia de la clasificación de imágenes clásica donde se definen las características de la imagen, gestiona los píxeles

de las imágenes que se le pasan, entrena el modelo, y reconoce patrones de forma automática con independencia de dónde se encuentren en las imágenes. En la Figura 2.4 se muestra un ejemplo de una red neuronal de este tipo.

A continuación se definen algunos términos asociados a las CNNs:

- Convolución: Extrae las características más relevantes de la imagen I multiplicando sus píxeles por un filtro K y guardando el resultado (Figura 2.5a).
- *Pooling*: Permite reducir el tamaño de los datos y el tiempo de procesamiento recorriendo los píxeles de la imagen con un filtro. Está caracterizado por el *size*, que especifica el tamaño del filtro, y el *stride* o zancada, que indica de cuantos en cuantos píxeles se desplaza el filtro (Figura 2.5b).



(a) Aplicación de una convolución a una imagen [22]

(b) La imagen de la izquierda se ha reducido a la de la derecha gracias a la aplicación de *pooling* [14]

Figura 2.5: Ejemplos de convolución y *pooling*

Capítulo 3

Estado del arte

En este trabajo de fin de grado se va a desarrollar un sistema de seguimiento de múltiples objetos haciendo uso de la *instance segmentation*. Es decir, mediante la detección e identificación de los objetos con una máscara que se adapta a su contorno a nivel de píxel. A continuación, se procede a explicar las principales tecnologías tenidas en cuenta para el desarrollo del sistema.

3.1. Mask R-CNN

Mask R-CNN [11] es una red neuronal cuyo objetivo es identificar los objetos de una imagen haciendo uso de la *instance segmentation*. Para entender mejor qué hace, es necesario comprender una serie de conceptos que están presentes en la Figura 3.1:

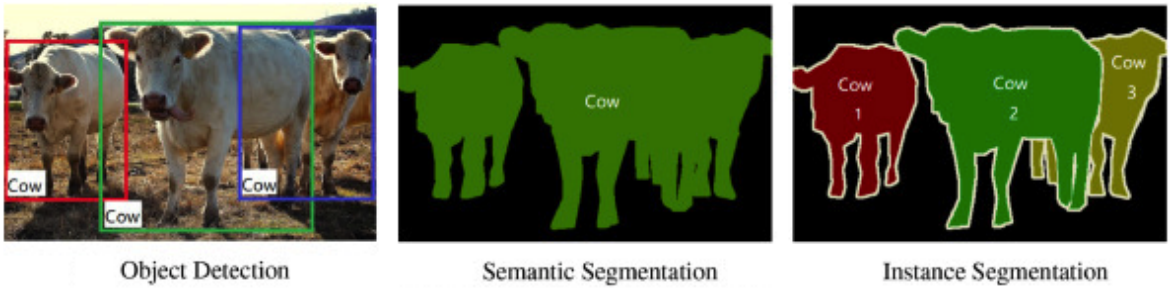


Figura 3.1: Diferencias entre *object detection*, *semantic segmentation* e *instance segmentation* [27]

- *Object detection*: Consiste en que para una imagen dada se conozca la clase a la que pertenece cada objeto que aparece, así como las coordenadas de una caja delimitadora o *bounding box* que contiene a cada objeto.

- *Image segmentation*: Se basa en generar una máscara a nivel de píxel que contiene la forma exacta de cada objeto que aparece en la imagen (poseer la máscara de un objeto implica conocer su *bounding box*). Se pueden diferenciar 2 tipos:

1. *semantic segmentation*: si hay más de una instancia de una misma clase no las diferencia, y 2. *instance segmentation*: cada objeto detectado es una instancia distinta, da igual que 2 o más objetos sean de la misma clase, se interpretan como distintos.

Mask R-CNN es básicamente una extensión de Faster R-CNN [24], ya que este último es un algoritmo que realiza la *object detection* sobre una imagen, y Mask R-CNN lo ha usado como base y le ha incorporado la máscara a nivel de píxel, logrando así la *instance segmentation*. Por último, su arquitectura presente en la Figura 3.2 consta de las siguientes partes:

1. *Backbone Model* (*Feature extraction*): Extrae características de una imagen dada.
2. *Region Proposal Network* (RPN): Con la información del paso 1, genera regiones de interés (ROI) en la imagen en las que el modelo predice que hay algún objeto.
3. Por cada ROI, la red se divide en 3 ramas: una encargada de generar una *bounding box*, otra de decir a qué clase pertenece el objeto que aparece y con qué probabilidad, y otra que genera una máscara a nivel de píxel.

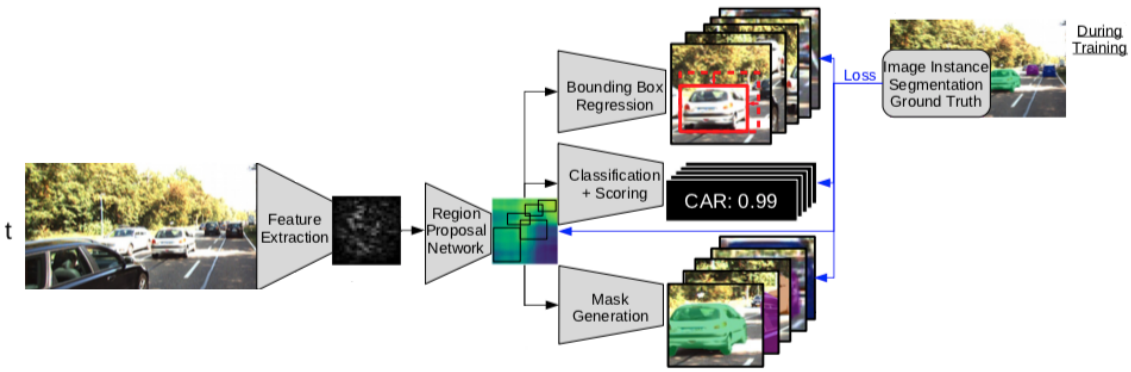


Figura 3.2: Estructura de Mask R-CNN [11]

3.2. Track R-CNN

Ya se ha visto que Mask R-CNN es una red neuronal capaz de realizar la *instance segmentation* sobre un *frame*. Track R-CNN [26] extiende a Mask R-CNN añadiéndole convoluciones 3D con el fin de conseguir información temporal, de manera que se pueda hacer seguimiento de objetos a lo largo de una secuencia de varios *frames* (un vídeo).

En la Figura 3.3 se puede observar de color amarillo lo que se ha añadido a la arquitectura de Mask R-CNN para lograr el seguimiento de objetos.

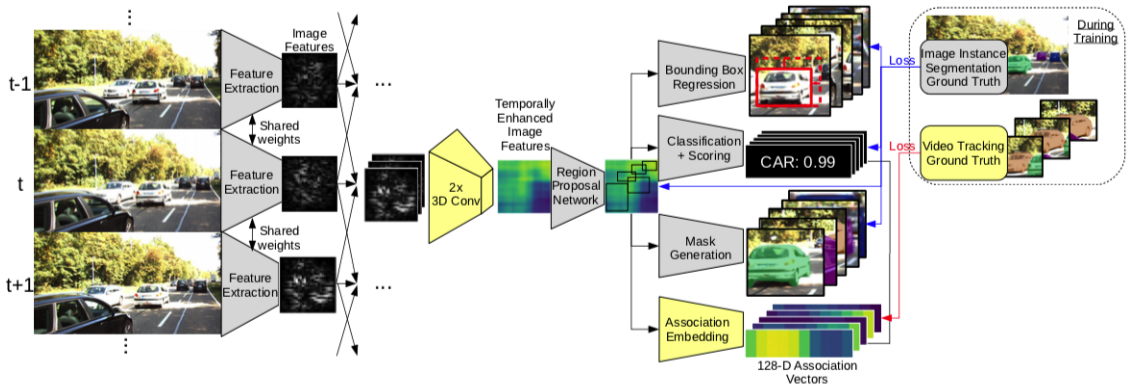


Figura 3.3: Estructura de Track R-CNN [26]

Su arquitectura se constituye por:

1. *Backbone Model* (*Feature extraction*): Ahora se extraen características de la imagen actual, pero también de la anterior y la siguiente con el fin de obtener información temporal.

2. Capas convolucionales 3D: Obtienen la información relevante procedente de la *Feature extraction* para poder relacionar objetos entre distintos *frames*.

3. RPN: Genera ROIs en la imagen. Regiones en las que el modelo predice que hay algún objeto.

4. La red se divide en 4 ramas: Las 3 comentadas para Mask R-CNN, y la que va a establecer la asociación final entre objetos de distintos *frames* para realizar su seguimiento, que está constituida por un vector de asociación de objetos (*association embedding* [20]). Se considera el mismo objeto aquello que está más cerca en el vector de asociación. En la Figura 3.4 se puede ver que los coches que se encuentran más cerca en dicho vector se parecen mucho.

Figura 3.4: Ejemplo de *association embedding* [26]

Capítulo 4

Diseño e implementación

Una vez visto el contexto en el que se sitúa el sistema desarrollado y las tecnologías en las que se basa, se procede a explicar todos los pasos que lo han dado lugar.

El objetivo del sistema es ser capaz de identificar de forma única con un id a cada instancia de un objeto que aparezca a lo largo de un vídeo. Para ello, en primer lugar es necesario realizar la *instance segmentation* en cada *frame*, así que se ha utilizado como base Mask R-CNN. Track R-CNN también toma esta red neuronal como base, la diferencia con nuestro sistema radica en qué se ha añadido a Mask R-CNN para lograr el seguimiento de los objetos a lo largo de varios *frames*. Nuestro sistema va a utilizar la posición de los objetos, su apariencia y un algoritmo de predicción [19], conceptos que se van a explicar a continuación en profundidad.

4.1. Tecnologías usadas para la implementación

Puesto que el lenguaje de programación en el que Mask R-CNN está desarrollado es **Pyhton** y va a ser necesario modificar y añadir elementos a su código, el sistema se ha implementado en este mismo lenguaje. Para programar en Python se ha utilizado **Anaconda**, que es una distribución libre y abierta que instala lo necesario para programar en este lenguaje, además de que permite crear distintos entornos, de manera que en cada uno de ellos se puedan tener unos paquetes diferentes instalados, aquellos que interesen según lo que se esté desarrollando en cada entorno. La edición de código se ha realizado a través de **Jupyter notebook**.

4.2. *Datasets* utilizados

Durante el desarrollo del sistema se han usado dos *datasets* cuya estructura se comenta a continuación:

– *Common Objects In Context* (COCO) [18]: *Dataset* de imágenes muy grande

diseñado para tratar la *object detection*, *segmentation* y demás problemas relacionados con la visión por computador. Este nombre también es usado para hacer referencia al formato en el que se guarda dicho *dataset*, ya que muchos programas están estructurados expresamente para trabajar con el formato que este sigue. Contiene 123.287 imágenes en las que aparecen 886.284 instancias de 80 clases de objetos distintos. Mask R-CNN está entrenada con COCO, siendo la gran dimensión del *dataset* lo que permite que esta red neuronal sea capaz de detectar 80 clases de objetos.

– Kitti [9] modificado por Track R-CNN: Contiene un total de 51 vídeos divididos en 29 de test y 21 de *training*, que consisten en un conjunto de *frames* en los que se observa lo que va capturando una cámara colocada sobre un vehículo que circula por una gran variedad de carreteras. Esta variante del *dataset* proporcionada por Track R-CNN incluye la información necesaria sobre los objetos para enfrentarse al problema del seguimiento y segmentación de múltiples objetos. Se utiliza tanto para entrenar la red neuronal convolucional que se explicará en profundidad más adelante, como para evaluar lo bien que funciona el sistema implementado.

4.3. Desarrollo del sistema

4.3.1. Familiarización con Mask R-CNN

Puesto que esta red neuronal se usa como punto de partida, el primer paso es descargarla desde `https://github.com/matterport/Mask_RCNN`. A continuación, se ha realizado un estudio a fondo de los ficheros que la componen ya que va a ser necesario modificarlos y añadir código donde corresponda.

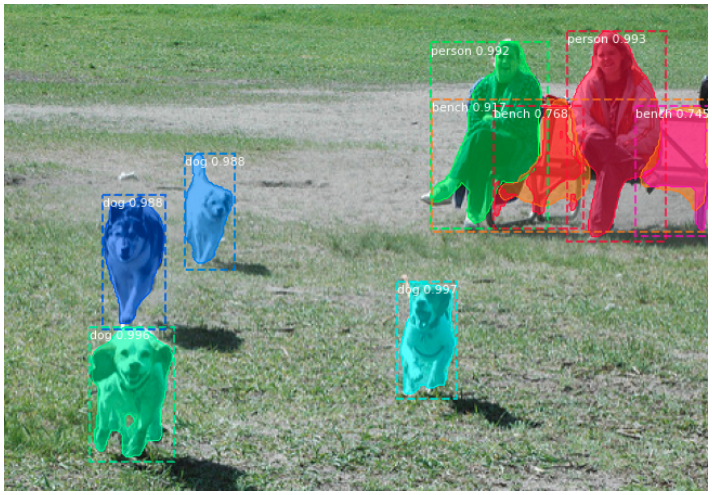


Figura 4.1: Aplicación de Mask R-CNN sobre una imagen

Para comprobar que funciona correctamente se han probado algunas imágenes como la de la Figura 4.1. Como se puede observar, para una imagen dada, Mask R-CNN devuelve por cada objeto que aparece su *bounding box*, la clase a la que pertenece y con qué probabilidad, y la máscara a nivel de píxel.

4.3.2. Selección de las clases a detectar

Por defecto, Mask R-CNN es capaz de detectar 80 clases distintas de objetos, pero para el sistema desarrollado se ha decidido centrarse en el seguimiento de coches únicamente, ya que los vídeos utilizados para probar el sistema son grabados desde un vehículo y la mayoría de los objetos que aparecen son coches. Además, por las herramientas de las que se dispone, solo se puede evaluar lo bien que funciona el sistema con coches y personas. Sería posible realizar la detección de otras clases pero para ello sería necesario disponer de datos etiquetados de dichas clases para poder así entrenar la red neuronal que incluye el sistema, la cual se explicará más adelante.

4.3.3. Seguimiento usando superposición de objetos

Atributos de un objeto

En primer lugar, a la información que aporta Mask R-CNN por defecto por cada objeto se ha añadido nueva, siendo entonces los atributos para cada objeto detectado:

- **ID**: Identificador único para cada objeto que permite realizar su seguimiento en una secuencia de vídeo.
- **COLOR**: Color expresado en RGB que se asocia a cada id con el fin de representar cada objeto de un color distinto, facilitando así la diferenciación a nivel visual entre unos objetos y otros.
- **ID_CLASE**: Identificador que indica a qué clase pertenece el objeto (es un coche, un semáforo, una moto...). Sobre la imagen procesada, en vez de poner el id_clase, se coloca el nombre de la clase que está asociada a ese identificador para facilitar el entendimiento de los resultados.
- **PUNTUACIÓN Ó *SCORE***: Expresa la probabilidad de que el objeto pertenezca a la clase indicada por el id_clase.
- ***BBOX***: Se refiere a la *bounding box*, una caja delimitadora que encierra al objeto detectado y que se representa con cuatro valores (x1,x2,y1,y2). La esquina superior izquierda es igual a (x1,y1) y la esquina inferior derecha es igual a (x2,y2).

– **MÁSCARA**: Matriz de píxeles de la dimensión del *frame* que se ajusta al contorno del objeto detectado, y que contiene todo 0s excepto en las posiciones donde va colocada la máscara, que hay 1s.

En la Figura 4.2 se observa cómo se muestran los atributos sobre un objeto.

Figura 4.2: Atributos proporcionados por Mask R-CNN acerca de un objeto

En las figuras que se van a mostrar con fines explicativos, así como en el sistema final, pueden no mostrarse algunos de estos atributos con el fin de mejorar el entendimiento de los resultados.

Intersection over union (IoU)

A continuación, se procede a realizar la primera aproximación al *tracking* o seguimiento de objetos usando el solapamiento de objetos entre distintos *frames*. Por ejemplo, como se puede observar en la Figura 4.3, los coches que en el *frame* 0 aparecen de color azul y rojo vuelven a salir en el *frame* 1, puesto que los coches de este nuevo *frame* aparecen del mismo color que los del 0. Se ha detectado que vuelven a aparecer debido al solapamiento entre las máscaras de los objetos de ambos *frames*.

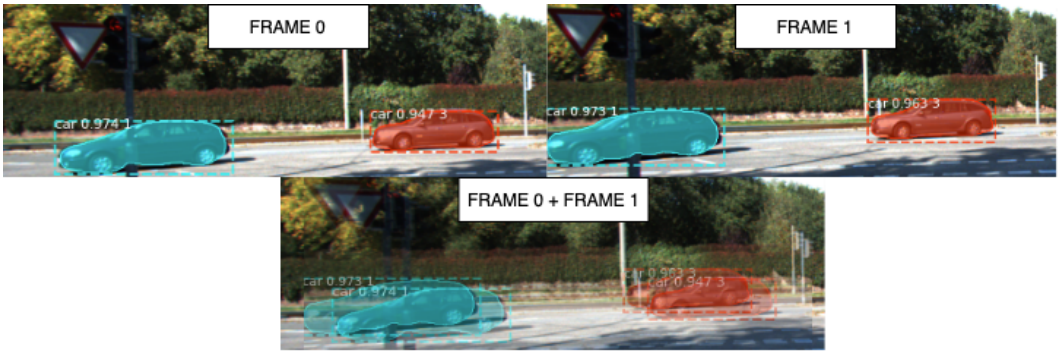


Figura 4.3: Ejemplo de seguimiento de objetos por solapamiento

Para saber cómo de solapados están dos objetos se ha utilizado la métrica IoU, también conocida como índice de Jaccard [23]. Como de cada objeto se posee su

bounding box y su máscara, se va a realizar la IoU para los 2 atributos entre cada par de objetos. Esta métrica, como su propio nombre indica, consiste en calcular la intersección entre las *bounding boxes* o máscaras de 2 objetos, y dividirla para la unión de estas (Figura 4.4a). Cuanto mayor sea el solapamiento entre 2 objetos, más cercano a 1 será el resultado devuelto por la IoU, mientras que si no están nada solapados, la IoU devolverá 0 (Figura 4.4b)

(a) Representación de la IoU (b) Ejemplo de cómo varía la IoU

Figura 4.4: Definición visual (a) y ejemplos (b) de la métrica IoU.

Algoritmo húngaro

Es posible que varios objetos del *frame* actual tengan solapamiento con más de uno de los objetos ya vistos en *frames* anteriores. Para solucionar este problema se usa el algoritmo húngaro [15], que se encarga de asociar cada objeto del *frame* actual con uno solo de los ya vistos de manera que se maximice el solapamiento total. Para ello, se elabora una matriz que contiene en las columnas los objetos vistos en el *frame* actual y en las filas los objetos vistos en *frames* anteriores. El valor de cada posición se corresponde a la IoU entre la pareja de objetos correspondiente.

En el ejemplo de la Figura 4.5, los dos objetos del *frame* actual tienen solapamiento con los dos objetos ya vistos, por lo que el algoritmo húngaro busca la combinación más óptima, que en este caso será asociar *frameActual_1* con *frameAnterior_1* y *frameActual_2* con *frameAnterior_2*. Es lo más óptimo ya que como se ha comentado se busca maximizar el solapamiento total, y con la combinación elegida el solapamiento total es $0{,}7 + 0{,}6 = 1{,}13$. Si se hubiera realizado la asociación contraria, el resultado hubiera sido $0{,}3 + 0{,}2 = 0{,}5$.

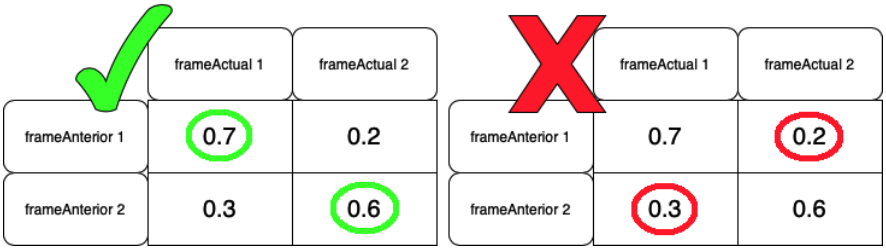


Figura 4.5: Asociación establecida entre los objetos del *frame* actual y los vistos en *frames* anteriores en función del algoritmo húngaro

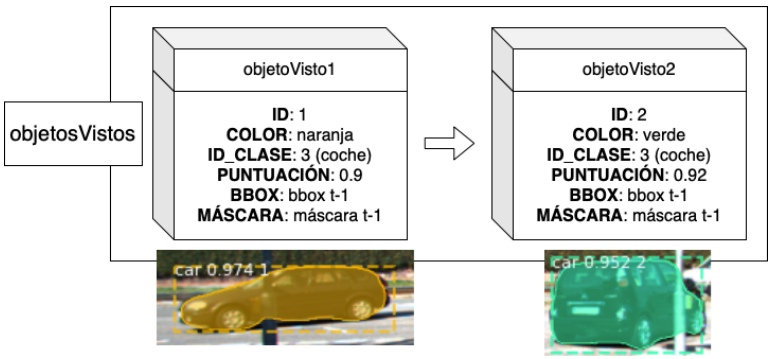
Para el algoritmo húngaro se ha usado el módulo *Munkres* de Python.

Ejemplo de comportamiento del sistema

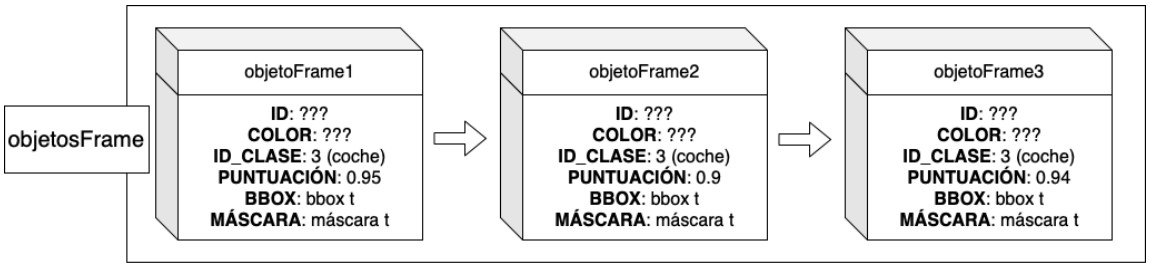
Una vez conocidos estos conceptos, dado un *frame t*, los pasos que sigue el algoritmo para realizar el seguimiento de los objetos basándose en el solapamiento es el siguiente:

1. Se guarda una lista *objetosVistos* con los objetos vistos hasta el *frame t*-1 con sus atributos actualizados hasta el último *frame* en el que han sido localizados (como mucho el *frame t*-1). El contenido se muestra en la Figura 4.6a.

2. Se usa Mask R-CNN para obtener la *bounding box*, clase y probabilidad de que pertenezca a esa clase, y máscara a nivel de píxel de cada objeto detectado en el *frame t*. Esta información se guarda en la lista *objetosFrame*, utilizada para mostrar las detecciones sobre el *frame actual*. Como se observa en la Figura 4.6b, en este punto los objetos de esta lista no tienen ni id ni color porque aún no se sabe si se van a asociar con un objeto ya visto, o si van a ser objetos nuevos no vistos anteriormente.



(a) Contenido de la lista *objetosVistos*



(b) Contenido de la lista *objetosFrame*

Figura 4.6: Contenido *objetosVistos* y *objetosFrame*

3. Se crea una matriz que tiene en las columnas los objetos vistos en el *frame* actual *t* (*objetosFrame)* y en las filas los objetos vistos desde el primer *frame* del vídeo hasta el *frame t*-1 (*objetosVistos*). En cada posición se coloca el resultado de realizar la IoU entre las máscaras del par de objetos que corresponda.

4. Se aplica el algoritmo húngaro sobre esta matriz (Figura 4.7).

Figura 4.7: Resultado del algoritmo húngaro entre los objetos del *frame* actual y los vistos en *frames* anteriores

En la asociación de objetos se pueden dar 3 situaciones:

a) Si el número de objetos en *objetosVistos* es el mismo que en *objetosFrame*, el algoritmo húngaro emparejará todos los objetos sin dejar ninguno libre.

b) Si el número de objetos en *objetosVistos* es mayor al de *objetosFrame*, los n objetos de *objetosFrame* se emparejarán con n objetos de *objetosVistos* (con los más óptimos según el algoritmo húngaro).

c) Si el número de objetos de *objetosVistos* es menor al de *objetosFrame*, quedarán objetos del *frame* actual sin emparejar. Serán considerados como nuevos objetos no vistos en *frames* anteriores.

Para poder emparejar 2 objetos su solapamiento debe ser siempre mayor que 0.

5. Se actualiza *objetosVistos* y *objetosFrame* en función de las asociaciones establecidas en el paso anterior.

a) Cuando un objeto de *objetosVistos* se ha asociado con uno de *objetosFrame*, los atributos de *bounding box* y máscara del segundo se asignan a los del primero, ya que *objetosVistos* debe contener la información más actualizada de los objetos que se han visto a lo largo de los *frames* (Figura 4.8 flechas rojas). Cuando se da esta asociación, los atributos id y color del objeto de *objetosVistos* se asignan al de *objetosFrame*, de esta forma se mostrará el objeto en el *frame* actual del mismo color y con el mismo id que en *frames* anteriores en los que haya aparecido (Figura 4.8 flechas azules).

b) Se puede ver en la Figura 4.9 que si un objeto del *frame* actual no se asocia con ningún objeto de *objetosVistos* es considerado un objeto nuevo que no se ha visto antes, por lo que se le asigna un nuevo id y color no usados por ningún otro objeto, y se añade a la lista de *objetosVistos*.

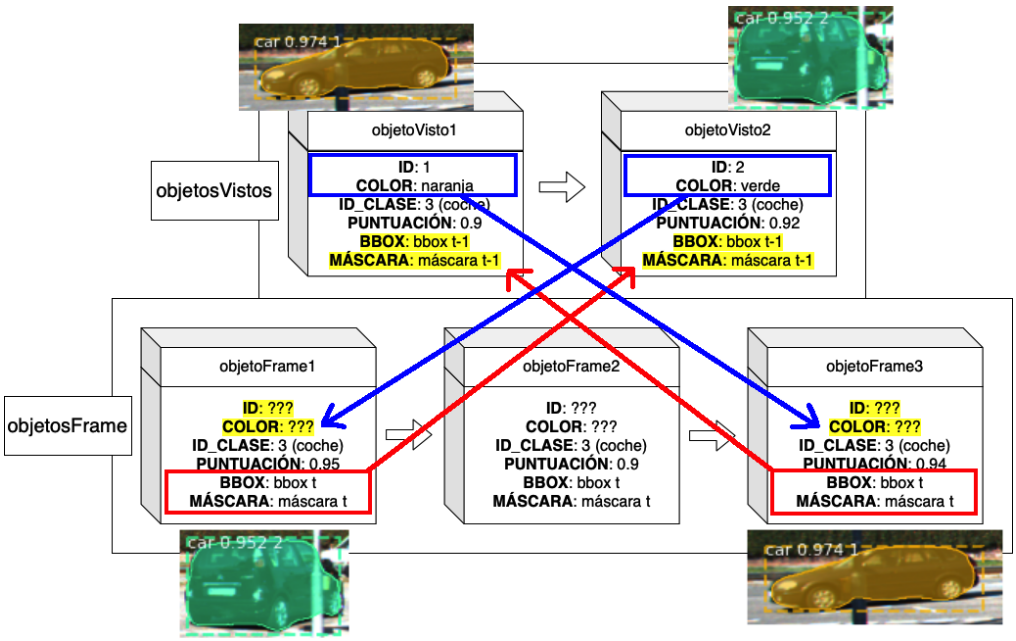


Figura 4.8: Actualización de *bounding box* y máscara de los objetos de *objetosVistos* y de id y color de los objetos de *objetosFrame*

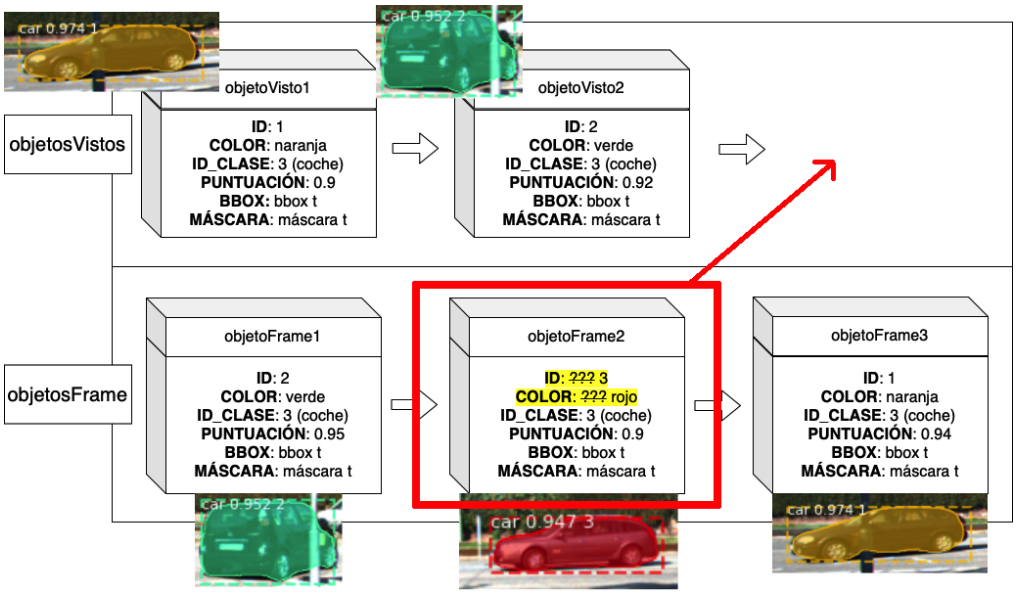


Figura 4.9: Asignación de un nuevo id y color a *objetoFrame2* y añadido de este objeto a la lista de *objetosVistos*

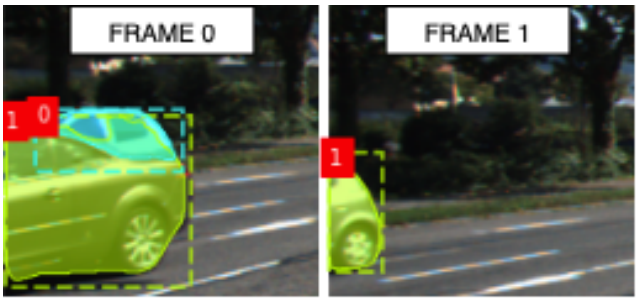
6. En la Figura 4.10 se muestra el *frame* actual t recorriendo la lista *objetosFrame*.

Figura 4.10: *Frame* con los objetos detectados en él

Punto fuerte y punto débil

Realizar el seguimiento y segmentación de objetos usando únicamente el solapamiento en situaciones en las que hay objetos bien separados funciona bien (Figura 4.11a). Sin embargo, en el momento en el que los objetos se empiezan a superponer es donde comienzan los problemas. En la Figura 4.11b, en el *frame* 0 se detectan dos coches con id igual a 0 y 1. En el *frame* 1 se observa que al objeto de los dos que aún permanece dentro de la imagen se le ha asignado el id 1 cuando realmente se le debería haber asignado el id 0, ya que se trata del otro coche. Le asigna el id equivocado ya que si se observa la máscara del objeto del *frame* 1, tiene más solapamiento en el *frame* 0 con la máscara del objeto de id 1 que con el de id 0.

(a) Ejemplo seguimiento con IoU funciona bien



(b) Ejemplo seguimiento con IoU funciona mal

Figura 4.11: Ejemplo de punto fuerte (a) y punto débil (b).

En base a este ejemplo se ha podido observar cuál es el límite del seguimiento de objetos utilizando únicamente el solapamiento entre la posición de los objetos. Es por ello que utilizando otra clase de información como el color, la forma o la textura de los objetos se podría obtener un sistema más robusto.

4.3.4. Incorporación del algoritmo de predicción CSRT

En nuestro sistema, de la detección de objetos usando la *instance segmentation* se encarga Mask R-CNN, pero para aquellos casos en los que falle su detección, se va a recurrir a la *object detection* realizada por un algoritmo de predicción. Destacar que las predicciones de dicho algoritmo no se muestran sobre los *frames* ya que este no realiza la segmentación de objetos. Su uso permite que los objetos detectados por Mask R-CNN en el *frame* actual se asocien con mayor probabilidad con el objeto correcto.

Además, cabe mencionar que su utilización va a ser clave para el caso de los objetos en oclusión ya que Mask R-CNN no los va a detectar.

Para su implementación se ha utilizado la librería **OpenCV** [5], la cual proporciona muchas herramientas para el campo de la visión por computador, incluyendo una gran variedad de *trackers*. El algoritmo de predicción que se ha seleccionado tras probar con distintos (véase el subapartado 5.2.1) es el *Channel and Spatial Reliability Tracker* (CSRT) [19]. En el uso de este algoritmo va a entrar en juego la *bounding box* de los objetos (*object detection*), puesto que los *trackers* disponibles en OpenCV no trabajan con máscaras. Dichos *trackers* son capaces de estimar la posición de la *bounding box* de un objeto en el *frame* actual a partir de su posición en el *frame* anterior.

El algoritmo CSRT utiliza los histogramas de gradiente orientado (HoG) y los colores como características del *frame* que se está analizando. En base a estas, los filtros de correlación con los que trabaja el algoritmo son capaces de determinar la posición en dicho *frame* del objeto que se está siguiendo.

Por qué usar el algoritmo de predicción

En la Figura 4.12a se observa que en el *frame* 0 aparece un objeto. En el *frame* 12 (Figura 4.12b), el objeto que aparece es realmente el mismo que el del *frame* 0, pero como se ha estado desde el *frame* 1 hasta el 11 sin actualizar su *bounding box* y máscara porque Mask R-CNN no lo ha detectado a lo largo de estos *frames*, la última información que se tiene de ese objeto es su *bounding box* y máscara en el *frame* 0. Aunque lo correcto sería decir que el objeto del *frame* 0 y el 12 es el mismo esto no va a ocurrir, puesto que la superposición de las máscaras entre ambos *frames* es nula (Figura 4.12c). Utilizando el algoritmo de predicción sí que va a ser posible decir que son el mismo objeto, puesto que el algoritmo CSRT va a estimar dónde está la *bounding box* del objeto del *frame* 0 en el *frame* 12, pudiendo ver en la Figura 4.12d que hay solapamiento entre la predicción de CSRT y la *bounding box* del objeto en el *frame* 12. Para determinar cuándo entra en juego el algoritmo CSRT se ha añadido el atributo edad, que especifica cuántos *frames* lleva Mask R-CNN sin detectar un objeto de *objetosVistos*.

En definitiva, cuando en un *frame* no se actualice la posición de la máscara de un objeto de *objetosVistos* porque Mask R-CNN no lo ha detectado, se recurrirá al algoritmo de predicción para estimar dónde debería estar la *bounding box* del objeto en ese *frame*. De esta forma, en todos los *frames* se actualizará como mínimo la *bounding box* de todos los objetos presentes en *objetosVistos*, ya sea porque el algoritmo húngaro los ha asociado con un objeto del *frame* actual (se actualizará también la máscara), o porque el algoritmo de predicción ha estimado su posición.

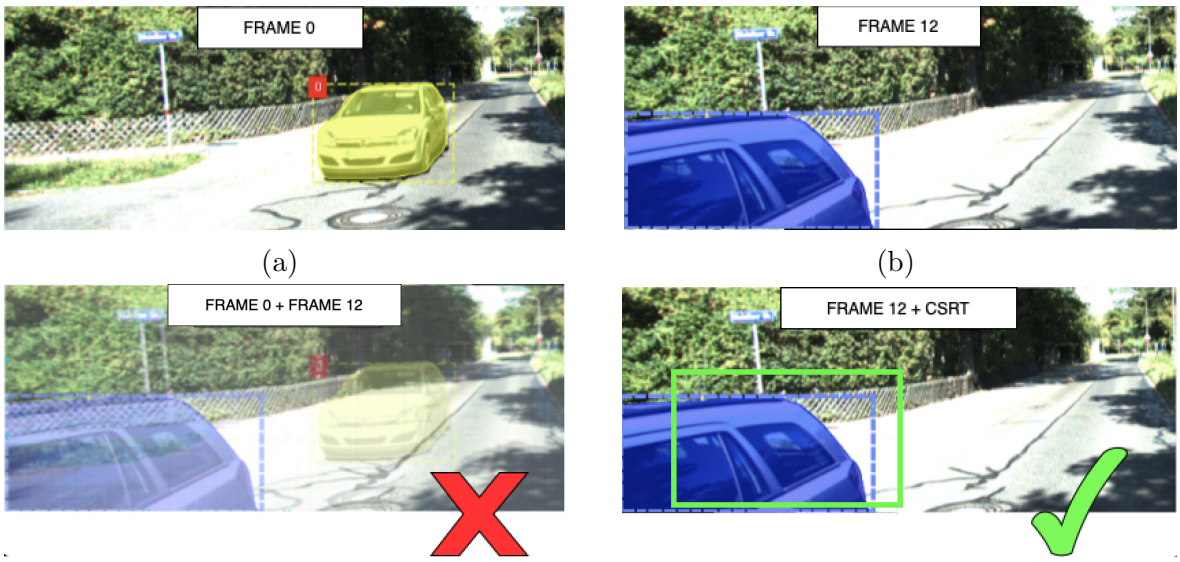


Figura 4.12: Ejemplo de aplicación algoritmo CSRT

Ejemplo de comportamiento del sistema con IoU + CSRT

FRAME t

1. En la Figura 4.13 aparecen las listas *objetosVistos* y *objetosFrame*.

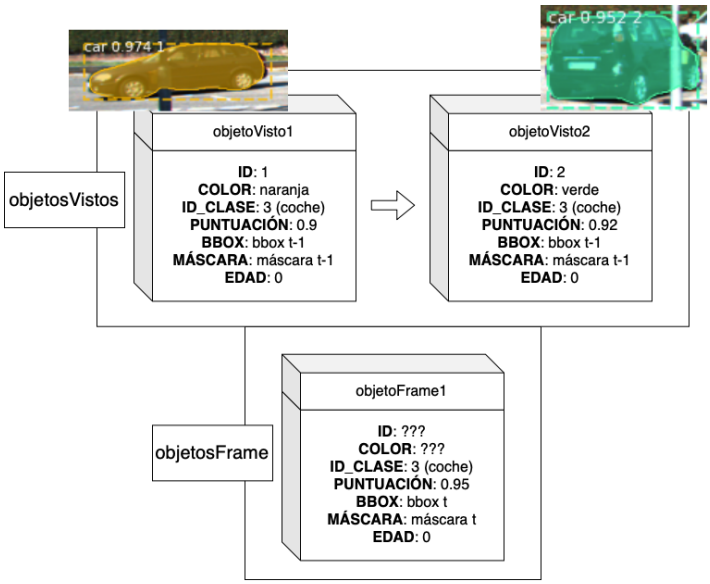


Figura 4.13: Contenido *objetosVistos* y *objetosFrame*

Los objetos de *objetosFrame* siempre tienen edad igual a 0 porque se están detectando en el *frame* actual. El que *objetoVisto1* y *objetoVisto2* tengan edad igual a 0 significa que han sido vistos en el *frame* *t*-1 y que por lo tanto su posición a nivel de máscara y *bounding box* está lo más actualizada posible.

2. Se calcula la matriz con las IoU y se aplica el algoritmo húngaro (Figura 4.14).

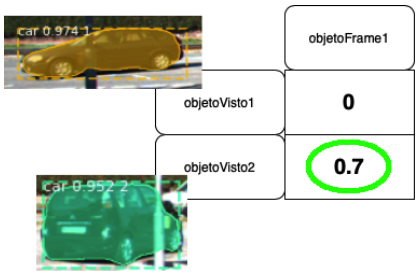


Figura 4.14: Matriz con la IoU entre los objetos del *frame* actual y los objetos vistos en *frames* anteriores + Aplicación del algoritmo húngaro sobre la matriz

3. Se asocia *objetoVisto2* con *objetoFrame1* (Figura 4.15).

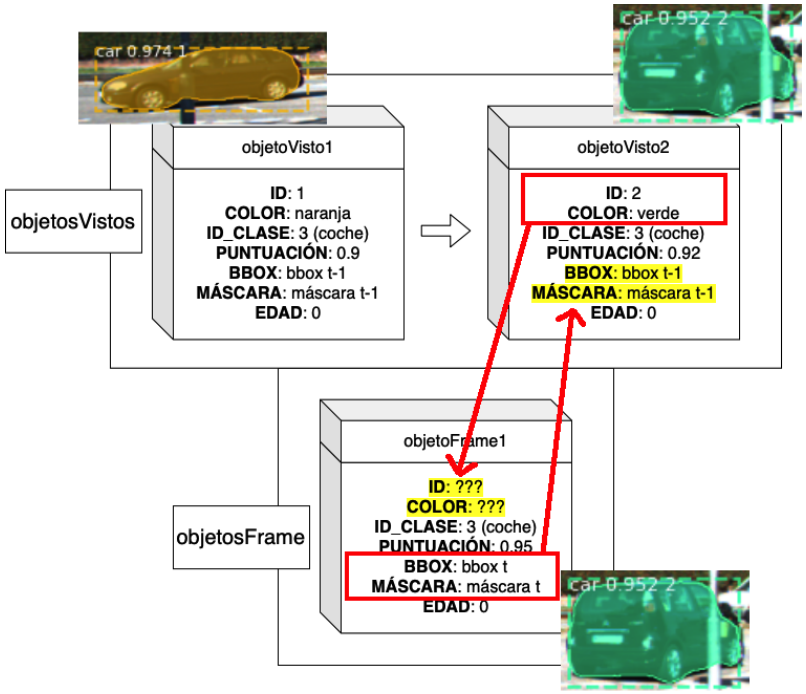


Figura 4.15: Actualización atributos de *objetoVisto2* y *objetoFrame1*

4. *objetoVisto1* queda sin asociar con ningún objeto del *frame* actual por lo que como se ve en la Figura 4.16 se incrementa en 1 su edad.

Figura 4.16: Incremento en una unidad del atributo edad de *objetoVisto1*

5. Como se observa en la Figura 4.17, el algoritmo de predicción CSRT se usa para estimar la posición de *objetoVisto1* en el *frame* actual al ver que su edad es mayor que 0. De esta manera, aunque no se haya asociado con ningún objeto del *frame t*, y por lo tanto no se haya podido actualizar su máscara a su posición en el *frame* actual, se ha actualizado la posición de su *bounding box*.

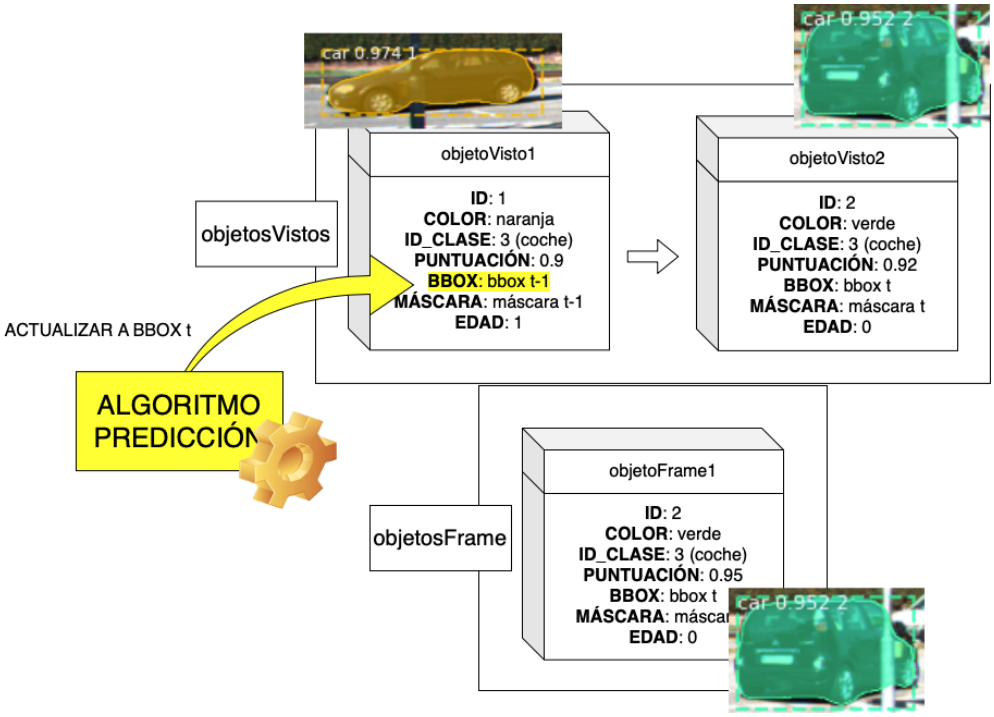


Figura 4.17: Aplicación del algoritmo de predicción sobre *objetoVisto1* actualizando su *bounding box* a la posición en el *frame t*

6. Se muestra el *frame t* con cada objeto detectado y su información asociada recorriendo la lista *objetosFrame*. Obsérvese la Figura 4.18.

Figura 4.18: *Frame* con las detecciones del seguimiento por solapamiento

FRAME t+1

1. Las listas *objetosVistos* y *objetosFrame* tienen la forma de la Figura 4.19. Se puede observar cómo la *bounding box* de *objetoVisto1* está actualizada al *frame t* gracias al algoritmo de predicción, pero en cambio, la máscara más reciente que se tiene de este coche es del *frame t*-1.

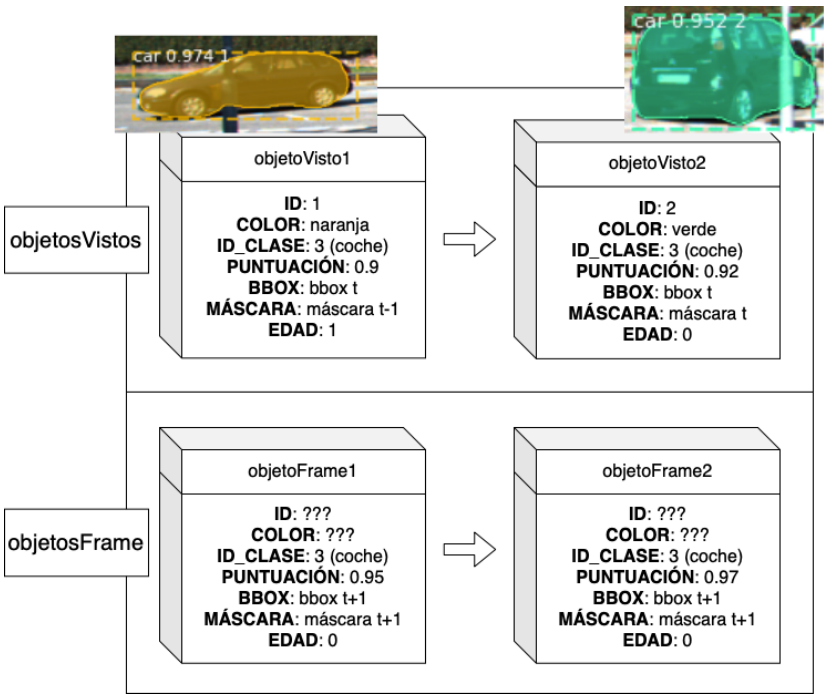


Figura 4.19: Contenido de *objetosVistos* y *objetosFrame*

2. Se calcula la matriz con las IoU para cada par de objetos y se aplica el algoritmo húngaro (véase la Figura 4.20). La diferencia con respecto al procesamiento del *frame t* es que al tener *objetoVisto1* la edad mayor a 0, su posición más reciente se ha estimado usando el algoritmo de predicción, y como este algoritmo solo funciona con *bounding boxes*, la IoU entre *objetoVisto1* y un objeto del *frame* actual debe hacerse usando la *bounding box* en vez de la máscara.

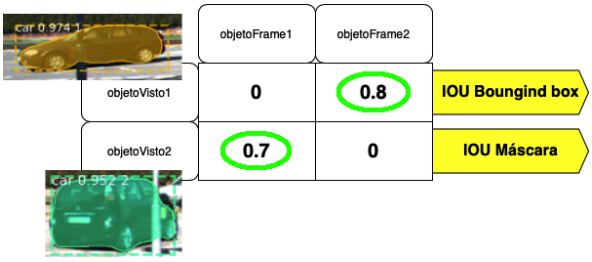


Figura 4.20: Matriz que contiene la IoU entre los objetos del *frame actual* y los objetos vistos en *frames anteriores* + Aplicación del algoritmo húngaro

Por lo tanto, si la edad de un objeto de *objetosVistos* es igual a 0, la IoU se calcula entre máscaras, y si es mayor a 0, la IoU se calcula entre *bounding boxes*.

3. Se asocia *objetoVisto2* con *objetoFrame1*, y por lo tanto se realiza el intercambio de atributos oportuno como se puede ver en la Figura 4.21.

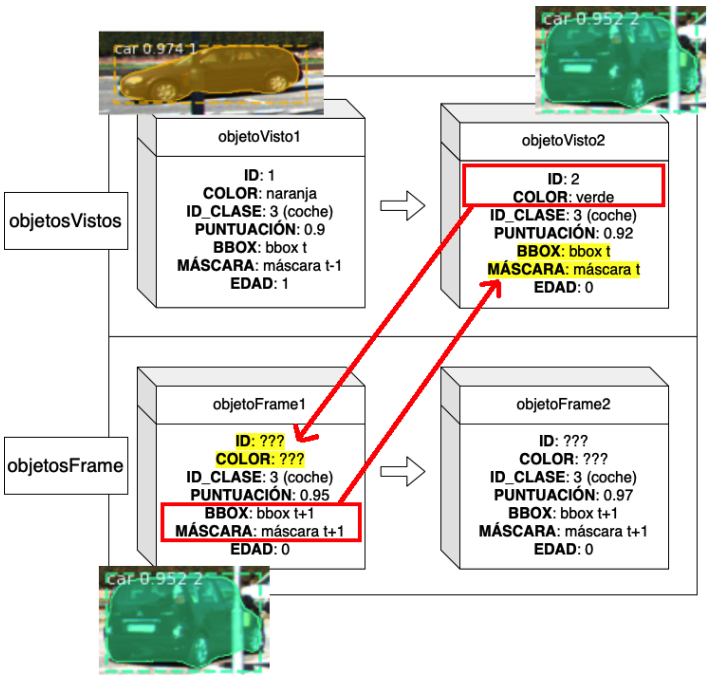


Figura 4.21: Actualización de *bounding box* y máscara de *objetoVisto2* y de id y color de *objetoFrame1* al asociarlos, ya que se trata del mismo objeto

4. Se asocia *objetoVisto1* con *objetoFrame2*. Como *objetoVisto1* se acaba de asociar con un objeto del *frame* actual, se pone de nuevo su edad a 0 y se actualiza su máscara y *bounding box* al *frame* t+1 (Figura 4.22).

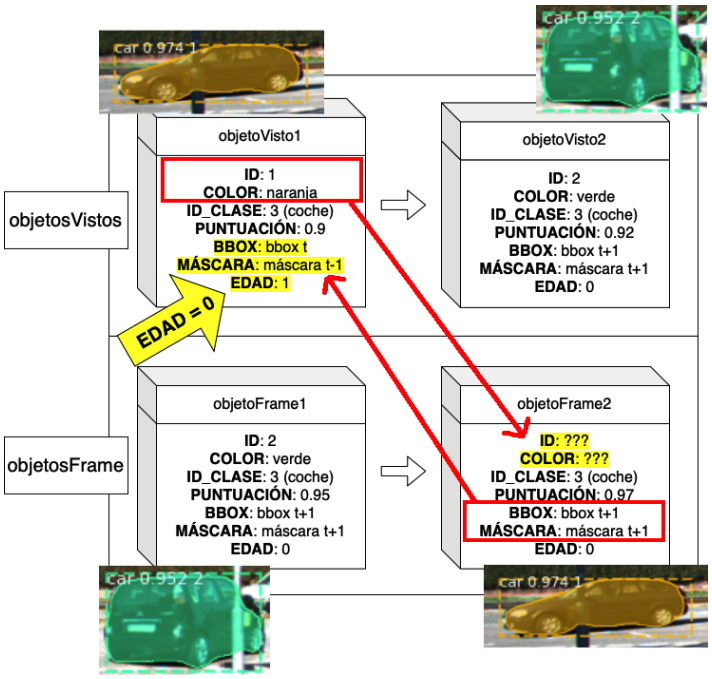


Figura 4.22: Actualización de *bounding box*, máscara y edad de *objetoVisto1*, y de id y color de *objetoFrame2* al asociarlos, ya que se trata del mismo objeto

5. Como se puede ver en la Figura 4.23, se muestra el *frame* t+1 con cada objeto

detectado y su información asociada recorriendo la lista *objetosFrame*.

Figura 4.23: *Frame* con las detecciones del seguimiento por solapamiento

4.3.5. Incorporación de la CNN con Descriptores Aprendidos

Con esta técnica se va buscar que el seguimiento de los objetos no dependa exclusivamente de la superposición de sus máscaras (o *bounding boxes* si se está usando el algoritmo de predicción), sino también de su color, forma y muchos más rasgos que la red neuronal es capaz de identificar. En definitiva, se busca extraer de Mask R-CNN un descriptor de cada objeto que proporcione información adicional acerca de él, haciendo así más robusto el sistema de *tracking* de objetos. Este es el punto más innovador y distintivo del TFG, con el que se ha querido comprobar si realmente es posible mejorar el sistema utilizando información de los objetos que está "escondida" por las capas de Mask R-CNN, ya que esta red neuronal devuelve de cada objeto la información propia de la *instance segmentation*, descartando por ello parte de la información que posee de cada uno, y la cual es posible que se pueda aprovechar para mejorar el sistema.

Selección de la mejor capa de la que extraer el descriptor

Recordando la estructura de Mask R-CNN, en la Figura 4.24 se muestran sus salidas.

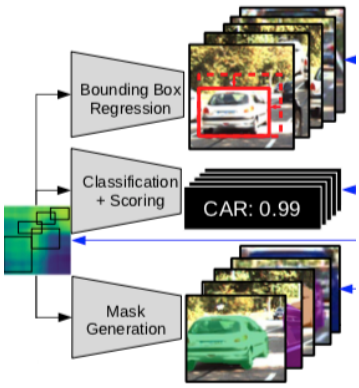


Figura 4.24: Salidas de Mask R-CNN

Para lograrlas hay un punto en el que la red se separa en 3 ramas, pero inmediatamente tras esta separación no están las salidas. Antes, hay una serie de capas intermedias que van a ir depurando la información que les llega con el fin de generar la

bounding box, clasificación por clase y probabilidad de que el objeto pertenezca a esa clase, y máscara. Como lo que se busca ahora es obtener información de los objetos detectados distinta de la que nos dan estas 3 salidas, lo que se ha hecho es acceder a la capas anteriores a la salida de la rama de la máscara (Figura 4.25).

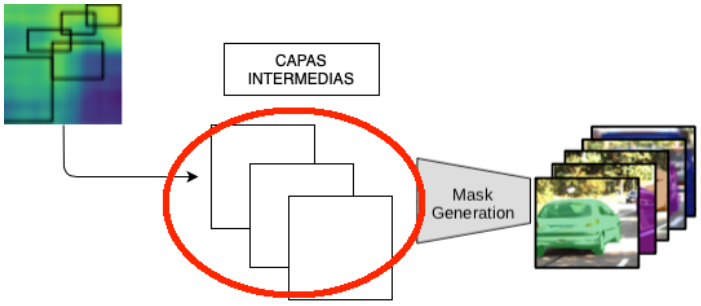


Figura 4.25: Capas intermedias de Mask R-CNN anteriores a la salida de la máscara

El que se hayan usado las capas de esta rama se debe a que la máscara es al fin y al cabo la salida que aporta información más precisa en cuanto a la apariencia real del objeto, que es lo que se está buscando. De esta forma, se va a disponer de la información que dé una de esas capas acerca del objeto (descriptor aprendido por esa capa), además de su *bounding box* y su máscara.

A continuación se debe elegir la mejor capa, entendiendo como mejor aquella que permita identificar mejor de forma única a un objeto y poder así diferenciarlo de los demás. Las capas que se han considerado como candidatas son *roi_align_mask*, *activation_71*, *activation_72* y *activation_73*. Se han ordenado de más lejos a más cerca de la salida de la rama de la máscara (Figura 4.26).

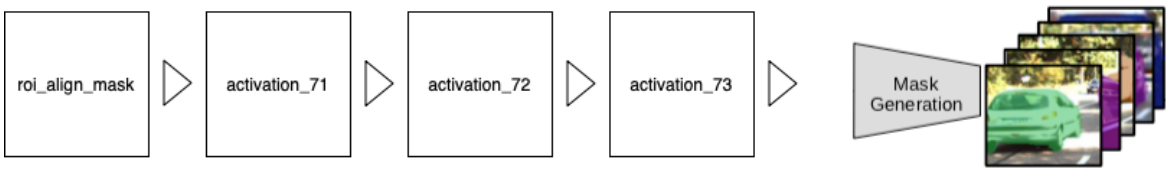


Figura 4.26: Capas intermedias candidatas y su posición con respecto a la salida de Mask R-CNN que devuelve la máscara

Cada una de estas capas está formada por 256 filtros de tamaño 14×14, lo que constituye un descriptor con información del objeto (Figura 4.27). Cada uno de estos filtros es una especie de imagen reducida que contiene información acerca del objeto distinta de la que aportan el resto de filtros (un filtro detecta bordes, otro colores, otro texturas, *etc*). Estos $14 \times 14 \times 256$ valores almacenados en una capa van a cambiar según el objeto del que se trate, por lo que la capa seleccionada va a ser aquella cuyo

descriptor devuelto para cada objeto se diferencie lo mejor posible del descriptor del resto de objetos.

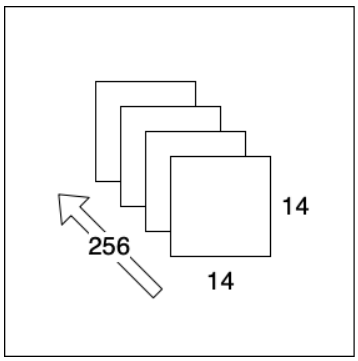


Figura 4.27: Dimensión del descriptor devuelto por una capa

Analizando las pruebas que se muestran en el subapartado 5.2.2 (Selección de la capa de la que extraer el descriptor) dentro del apartado de Análisis de resultados, las diferencias entre los objetos usando los descriptores de la capa *roi_align_mask* son mayores. De aquí se deduce que la capa *roi_align_mask* es más sensible a cambios puesto que 2 objetos distintos están a mayor distancia, siendo por ello la capa seleccionada.

De esta manera, el seguimiento de los objetos va a realizarse teniendo en cuenta el solapamiento entre objetos y su apariencia, para esto último haciendo uso del descriptor asociado a cada objeto que aporte la salida de la capa *roi_align_mask*. Esto da lugar a un nuevo atributo llamado RAM (iniciales de *roi_align_mask*), cuya incorporación a la lista de atributos de un objeto se presenta en la Figura 4.28.:

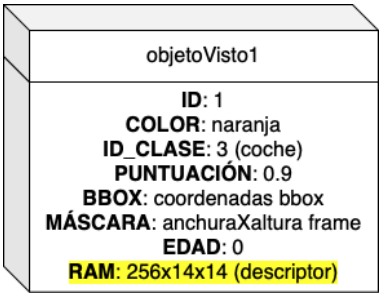


Figura 4.28: Incorporación del atributo RAM a los objetos

Implementación de una CNN para la utilización del descriptor aprendido

La información que se tiene de los objetos en forma de máscara y *bounding box* se ha utilizado a través del cálculo de la IoU (para medir la superposición de los objetos), la pregunta es cómo usar el descriptor de un objeto para mejorar el seguimiento de objetos. La respuesta es implementar una red neuronal que reciba como entrada los descriptores de 2 objetos y que en base a ellos indique en la salida la probabilidad de que se trate del mismo objeto en distintos *frames*.

Si la red neuronal va a tener como entrada los descriptores de 2 objetos, es decir, dos matrices de tamaño $14 \times 14 \times 256$, lo que supone un total de 100352 valores, van a hacer falta demasiados datos de entrenamiento para poder entrenar la red, puesto que se van a necesitar muchas neuronas para poder procesar toda la información procedente de los dos descriptores. Para solucionar este problema, se va a usar una CNN para así unificar en la red neuronal la reducción de las dimensiones de los descriptores y el cálculo de la probabilidad de que los 2 descriptores de entrada de la red estén vinculados al mismo objeto en distintos *frames*.

La CNN implementada presenta la siguiente estructura:

1. Reducción de las dimensiones de los 2 descriptores de entrada (Figura 4.29).

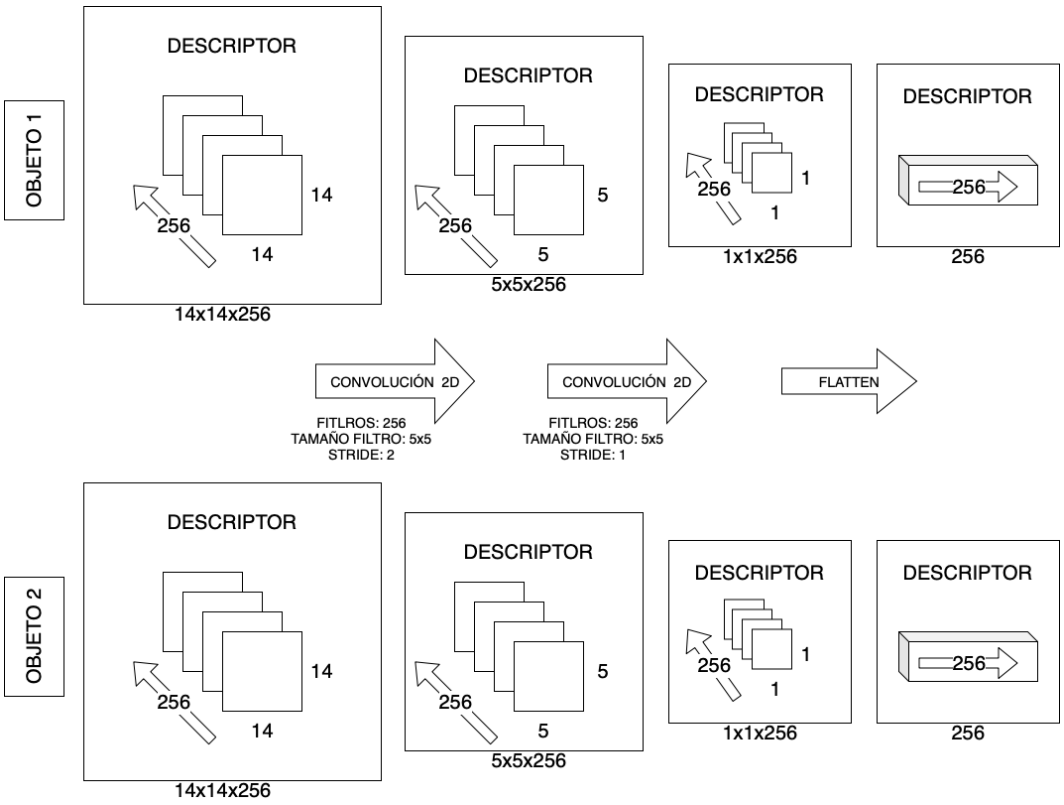


Figura 4.29: Primera parte de la CNN.

El descriptor extraído de la capa *roi_align_mask* atraviesa 2 capas convolucionales que lo comprimen, logrando pasar de un descriptor de tamaño $14 \times 14 \times 256$ a uno de $1 \times 1 \times 256$. Después de cada capa convolucional hay una capa *BatchNormalization* que estandariza los datos que pasan por ella, y otra *Activation* con función de activación *reLu*. Tras esto, se usa una capa *Flatten* para dejar una única dimensión, pasando de 1x1x256 a un *array* de 256 valores.

2. Comprobación de si los 2 descriptores pertenecen al mismo objeto en distintos *frames* (Figura 4.30).

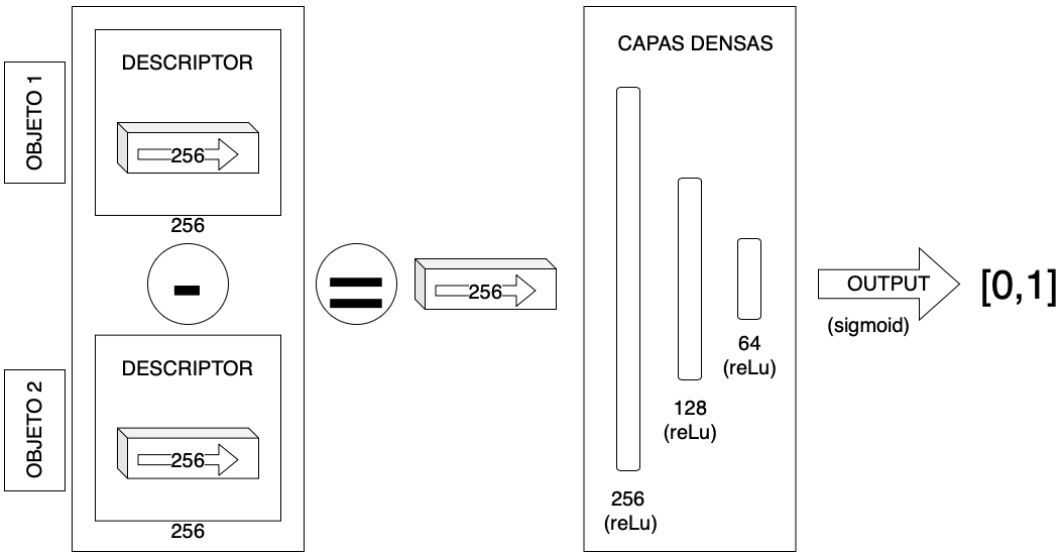


Figura 4.30: Segunda parte de la CNN

En esta parte de la red, se resta cada posición del descriptor reducido de un objeto con la misma posición del descriptor del otro objeto, generando un vector de 256 valores. Como cada uno de los 256 valores que conforman un descriptor representa un rasgo del objeto, realizando la resta se obtiene cómo de diferentes son los 2 objetos para cada rasgo. A continuación, el vector resultante atraviesa 4 capas densas (todas las neuronas de una capa están conectadas con todas las de la siguiente). Las 3 primeras capas tienen 256, 128 y 64 neuronas respectivamente y utilizan la función de activación *reLu*. La capa de salida tiene una única neurona y devuelve un único valor comprendido entre 0 y 1 gracias a la función de activación *sigmoid*. Cuanto más se acerque a 1 la salida, significará que la red encuentra más parecidos los dos descriptores de entrada, por lo que habrá más posibilidades de que sean descriptores del mismo objeto en *frames* distintos. Si la red devuelve 0.8, estará considerando que hay un 80 % de probabilidades de que los dos descriptores de entrada hagan referencia al mismo objeto.

El siguiente paso es obtener los datos de entrenamiento, validación y test necesarios para entrenar la red y posteriormente comprobar cómo de bien funciona. Como se puede observar en la Figura 4.31, para la obtención de los datos se han usado 205 objetos (coches). De cada uno de ellos, se han utilizado 10 descriptores, es decir, su aparición en 10 *frames* distintos, formando un total de 2050 muestras. Todas estas muestras se han extraído de los vídeos de *test* del *dataset* de Kitti modificado por Track R-CNN, ya que de estos vídeos no se proporciona *groundtruth*, y puesto que el proceso de obtención de datos se ha realizado de forma muy manual no ha sido necesario. Como la red debe recibir 2 descriptores de entrada para decidir si pertenecen o no al mismo objeto, hay

que formar parejas con estas muestras, formando un total de 1025 parejas. De ellas, la mitad serán parejas entre descriptores pertenecientes al mismo objeto (su etiqueta vale 1 ya que lo ideal sería que la red diga que hay un 100 % de probabilidades de que los descriptores pertenezcan al mismo objeto) y la otra mitad parejas entre descriptores de distintos objetos (su etiqueta vale 0 ya que lo ideal sería que la red diga que hay un 0 % de probabilidades de que los descriptores pertenezcan al mismo objeto).

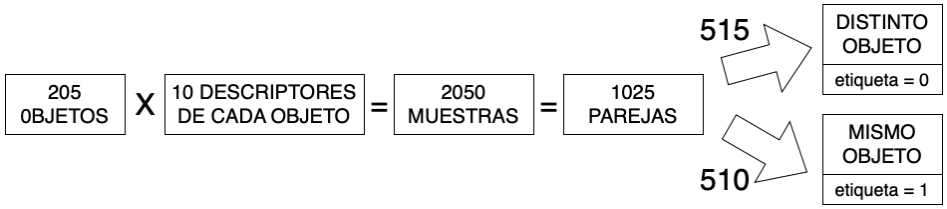


Figura 4.31: Obtención de las muestras

Una vez se tienen las 1025 parejas, hay que repartirlas entre los datos de entrenamiento, usados para que la red aprenda a diferenciar unos objetos de otros por sus descriptores; datos de validación, utilizados para asegurar que la red está aprendiendo correctamente (evitar sobreentrenamiento) y seleccionar qué época del entrenamiento interesa más; y datos de test, cuyo objetivo es comprobar cómo de bien funciona la red una vez se ha entrenado probándola con datos nuevos que no ha visto durante el entrenamiento. Véanse los porcentajes de reparto en la Figura 4.32.

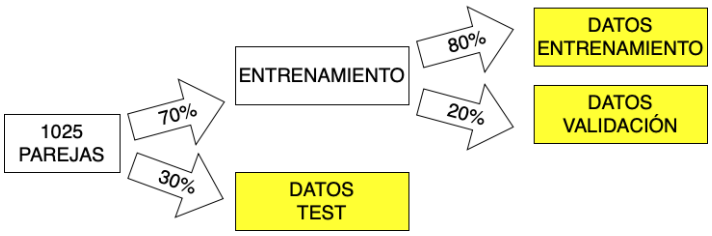


Figura 4.32: Reparto de las parejas entre datos de entrenamiento, validación y test

Tras repartir los datos entre los distintos grupos, el siguiente paso sería llevar a cabo un preprocesado sobre ellos con el fin de lograr que la red neuronal aprenda mejor y más rápido, pero este preprocesamiento no se ha realizado debido a que al extraer los descriptores aprendidos de una capa de Mask R-CNN, los valores de estos descriptores ya han sido normalizados gracias a una capa *BatchNormalization* presente en la red neuronal. Esta capa se encarga de estandarizar los datos centrándolos y posteriormente escalándolos, para que los distintos rasgos que representan los valores que conforman un descriptor estén en la misma escala.

Una vez se ha seleccionado la red y los distintos grupos de datos, el siguiente paso es entrenarla. Se ha utilizado la función de pérdida *binary_crossentropy* ya que se usa mucho para clasificadores binarios (se busca que la salida sea 0 ó 1), y un ratio de aprendizaje de 0.0001. Se ha entrenado durante 200 épocas y finalmente se ha utilizado el modelo entrenado hasta la época 136. Para tomar esta decisión se han observado las gráficas de *accuracy* y *loss* obtenidas con los datos de validación, presentes en la Figura 4.33.

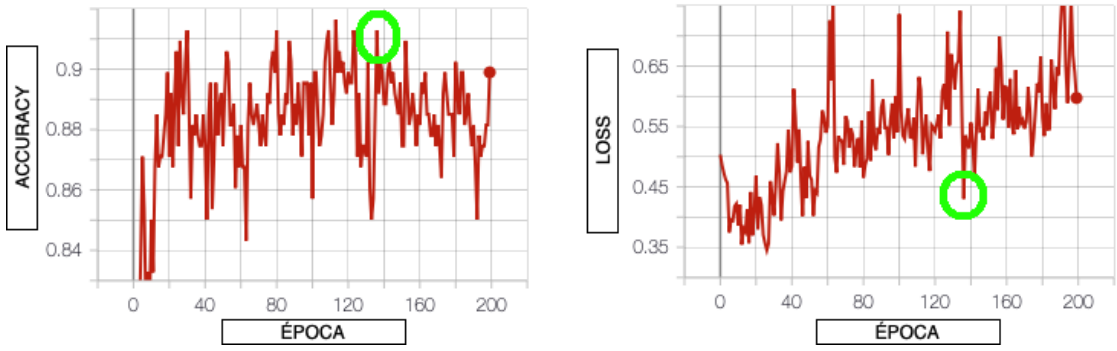


Figura 4.33: Gráficas de *accuracy* y *loss* de la CNN entrenada. La época 136 aparece rodeada porque es la época seleccionada

La gráfica de *accuracy* contiene para cada época la media de la distancia entre las salidas de la red para los datos de entrenamiento y las etiquetas de esos datos de entrenamiento (cuanto más se acerque a 1 mejor). La gráfica *loss* (función de pérdida) especifica para cada época la media del valor de la función de pérdida de entre todos los datos de entrenamiento (cuanto más se acerque a 0 mejor). Se ha buscado una época que represente un término medio entre una *accuracy* alta y una *loss* baja, y por ello, tras realizar pruebas con distintas épocas candidatas (véanse las pruebas en el subapartado 5.2.3) se ha seleccionado la 136.

Una vez entrenada la red, para evaluar cómo de bien funciona se han usado los datos de test, con los cuales se ha obtenido un *accuracy* del 93 %.

Integración de *Tracking* con Superposición de objetos + Algoritmo de predicción CSRT + Apariencia de objetos usando Descriptores Aprendidos

La incorporación de la CNN con los descriptores aprendidos al *tracking* por superposición de objetos más el algoritmo de predicción CSRT, solo afecta al paso en el que se calcula la matriz que contiene en cada posición cuánto se parece cada objeto del *frame* actual a cada objeto de los vistos en *frames* anteriores. Ahora, en vez de calcular únicamente la IoU entre los objetos, también se usan los descriptores de dichos objetos con la CNN implementada, con el fin de que esta indique con qué probabilidad estima que se trata o no de descriptores pertenecientes al mismo objeto en

distintos *frames*. Por lo tanto, se va a disponer de una matriz que contiene el resultado de la IoU y de otra que posee el resultado de la CNN, para cada par de objetos. Como se puede ver en la Figura 4.34, el paso siguiente es fusionar estas 2 matrices, dando un 80 % de peso a la información procedente de los resultados de la IoU, y un 20 % a la obtenida de la CNN. Esta combinación de pesos es la que genera unos mejores resultados del sistema (véase el subapartado 5.2.4).

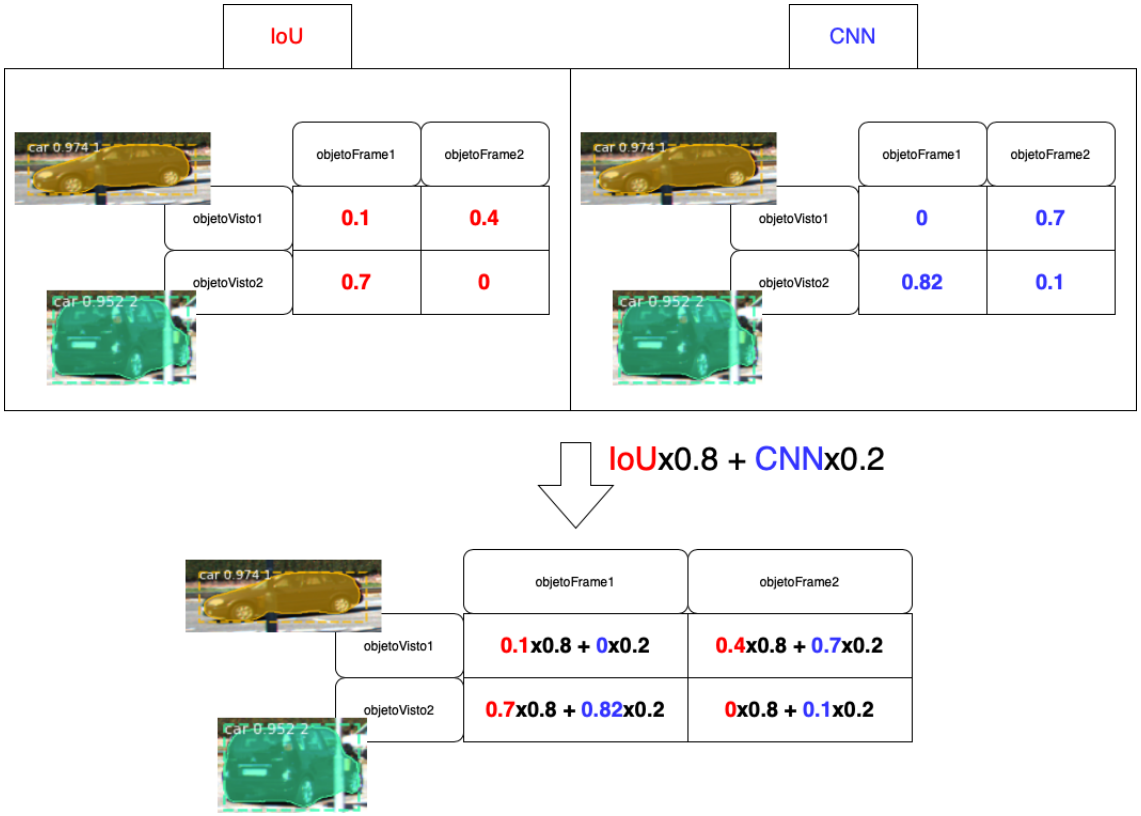


Figura 4.34: Cálculo de la matriz que contiene cuánto se parece cada objeto del *frame* actual a los objetos vistos en *frames* anteriores. Se da un 80 % peso a la información aportada por la IoU entre objetos y un 20 % a la aportada por la CNN

Tras esto se aplica el algoritmo húngaro, siendo el resto del procesamiento exactamente igual que para la versión del sistema sin CNN ni descriptores aprendidos.

4.3.6. Reentrenemiento de Mask R-CNN

El uso del algoritmo CSRT y de la CNN con los descriptores aprendidos va a ayudar a disminuir los fallos producidos al asignar a un coche el id de otro (errores en el seguimiento), ya que dificultan el hecho de confundir unos objetos con otros. Para rebajar los errores provocados por una débil detección de objetos es necesario reentrenar Mask R-CNN, ya que esta red neuronal es en definitiva en lo que se apoya el sistema desarrollado para realizar la *instance segmentation*. Es evidente que si no se detectan

correctamente los objetos que aparecen en un *frame* (no se detecta algún objeto que aparece o se dice que es un objeto algo que no lo es), va a ser muy complicado poder seguirlos.

Como se ha comentado al describir el *dataset* de COCO, Mask R-CNN está entrenada con dicho *dataset*, y la amplitud de este es lo que permite que la red neuronal en cuestión sea capaz de detectar 80 clases distintas de objetos. Para el reentrenamiento realizado, lo que se ha hecho es entrenar la red únicamente con la parte del *dataset* de COCO referida a la clase coche. La idea detrás de esto ha sido intentar que Mask R-CNN se especialice en la detección de los coches, que es al final el tipo de objeto en el que se centra nuestro sistema, sin embargo, los resultados obtenidos tras el reentrenamiento realizado son peores que sin él. Aunque se está seguro de que este punto lograría grandes mejoras en el comportamiento del sistema, hubiera sido necesario invertir mucho más tiempo, debido a que reentrenar la red es un proceso muy costoso para los recursos de los que se dispone (varios días para reentrenar una parte de la red), además de porque para dar con un reentrenamiento que sea útil para nuestro sistema sería necesario profundizar aún más en el entendimiento de Mask R-CNN.

Capítulo 5

Análisis de resultados

En este capítulo se explican en primer lugar las métricas utilizadas, y a continuación se exponen una serie de pruebas realizadas con el fin de seleccionar determinados parámetros del sistema (disponible en `https://github.com/DanielCay/TFG`), logrando así su funcionamiento más óptimo. Tras esto, se lleva a cabo un estudio de los resultados obtenidos por nuestro sistema y su comparación con los resultados de Track R-CNN.

5.1. Métricas para evaluar el sistema

Una vez está desarrollado el sistema el siguiente paso es poder evaluarlo, es decir, comprobar cómo de bien funciona. Como el estado del arte en seguimiento de múltiples objetos con *instance segmentation* es la red neuronal Track R-CNN, se ha decidido usar sus mismas métricas con el fin de poder comparar resultados. Para evaluar el sistema se necesita un *groundtruth*, es decir, es necesario tener la información de lo que idealmente debería detectar nuestro sistema en cada *frame*. De esta manera, se puede comprobar lo que detecta el sistema desarrollado en relación a lo que debería detectar, pudiendo así obtener métricas fiables.

5.1.1. Obtención del *groundtruth*

Se han utilizado como *groundtruth* los vídeos de *training* del *dataset* Kitti modificado por Track R-CNN. Proporcionan sobre cada detección de un objeto: el id de la clase a la que pertenece (id_clase), el identificador único del objeto en cuestión (id), el *frame* en el que está apareciendo el objeto y máscara del objeto en dicho *frame* (Figura 5.1). Además, se ha decidido asociar un color a cada objeto (a cada id) del *groundtruth* para facilitar su seguimiento a nivel visual. Cabe mencionar que solo se puede evaluar el buen funcionamiento del sistema para coches y personas puesto que no hay *groundtruth* para el resto de clases de objetos.

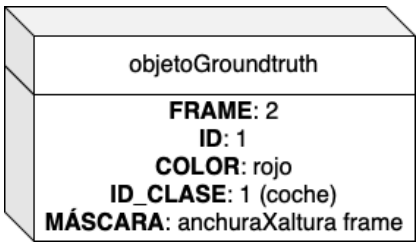


Figura 5.1: Atributos proporcionados por el *groundtruth* acerca de la aparición de un objeto en un *frame*

El *groundtruth* está guardado en un archivo *.txt* en el formato de COCO, por lo que el primer paso es leerlo del fichero para poder utilizarlo. Para ello se ha usado la API de *pycocotools*, la cual ayuda a cargar, analizar y visualizar anotaciones realizadas en el formato de COCO. Una vez se han leído los atributos, falta transformar la máscara de formato *run-length encoding* (RLE), formato de compresión de datos [25], a formato binario con el fin de que sea igual que las máscaras de los objetos que devuelve Mask R-CNN. Como se puede observar en la Figura 5.2, para realizar este paso se ha utilizado de nuevo la API de *pycocotools*.

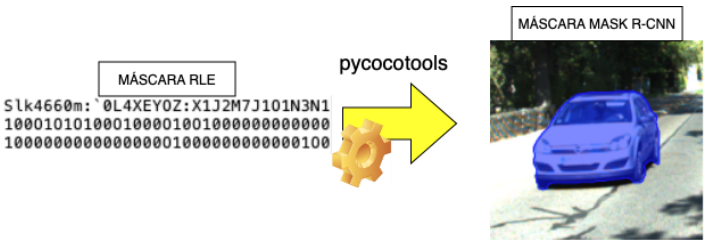


Figura 5.2: Decodificación de la máscara RLE de un objeto con *pycocotools*

El *groundtruth* proporcionado presenta los llamados puntos ciegos, que son zonas a ignorar en los *frames* puesto que los desarrolladores del *groundtruth* consideran que no se deben tener en cuenta para evaluar el correcto funcionamiento del sistema (coches muy lejanos difíciles de detectar, clases de objetos muy difíciles de diferenciar de un coche en algunos casos como las furgonetas, *etc*).

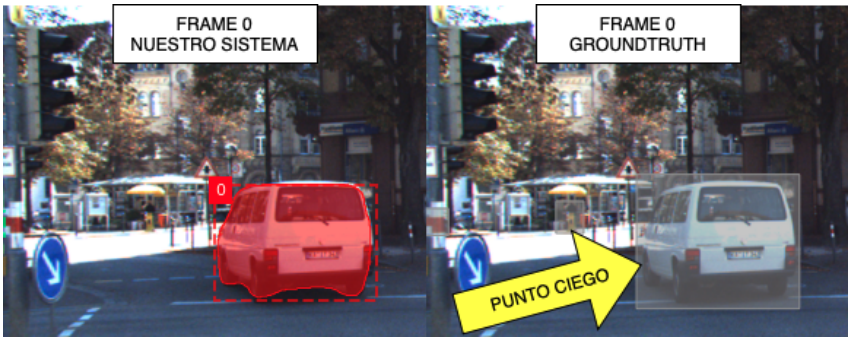


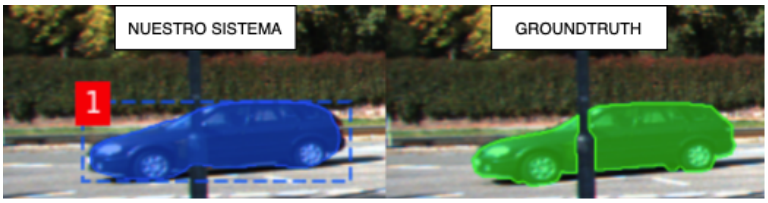
Figura 5.3: El objeto con id igual a 0 detectado por nuestro sistema se ignora para el cálculo de las métricas puesto que se encuentra en un punto ciego del *groundtruth*

Cualquier objeto que se detecte en un punto ciego se ignorará para el cálculo de las métricas (en la Figura 5.3 los puntos ciegos aparecen de color gris). Se considera que un objeto está dentro de un punto ciego cuando la IoU que tiene con el punto ciego con el que más se solapa es mayor a la IoU que tiene con el objeto del *groundtruth* con el que más se solapa.

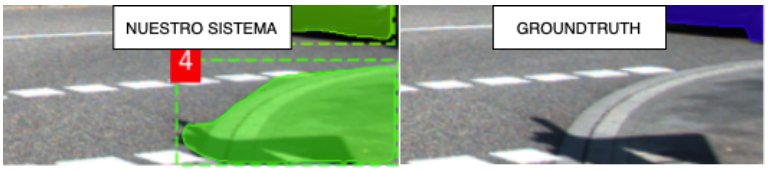
5.1.2. Métricas utilizadas

Una vez se ha obtenido el *groundtruth*, el siguiente paso es elegir las métricas a usar para evaluar el sistema. Como se ha comentado, se ha decidido utilizar las mismas que Track R-CNN. En primer lugar se van a mencionar los distintos factores que componen las métricas que son necesarios para calcularlas:

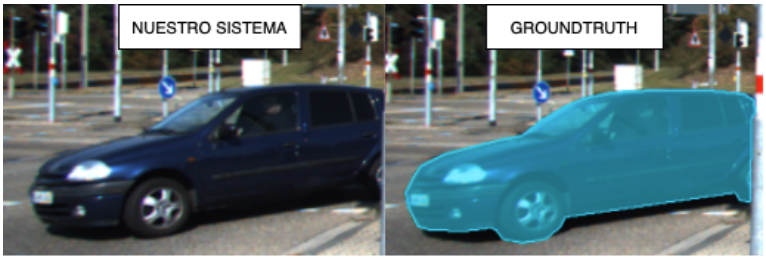
- *True positive* (TP): Nuestro sistema dice que hay un objeto donde el *groundtruth* dice que lo hay (Figura 5.4a).
- *False positive* (FP): Nuestro sistema dice que hay un objeto donde el *groundtruth* dice que no lo hay (Figura 5.4b).
- *False negative* (FN): Nuestro sistema no detecta un objeto donde el *groundtruth* dice que lo hay (Figura 5.4c).



(a) Ejemplo de *true positive*



(b) Ejemplo de *false positive*



(c) Ejemplo de *false negative*

Figura 5.4: Parámetros necesarios para definición de las métricas CLEAR MOTS [2].

– *True positive* solapamiento (TPS): Resultado de la IoU dado un TP (cuánto se solapan dos objetos considerados como *true positive*).

– Máscaras (M): Número de objetos detectados por el *groundtruth* (número de máscaras que no son puntos ciegos). En la Figura 5.5 se devolvería M igual a 4.

Figura 5.5: Ejemplo de número de máscaras detectadas en un frame

– Id *switch* (IDS): Nuestro sistema dice que un objeto no es el que el *groundtruth* dice que es (error en el seguimiento de un objeto a lo largo de los *frames*). En la Figura 5.6 se observa un error por IDS ya que en el *frame* t de nuestro sistema aparece un coche con id igual a 0, mientras que en el *frame* t+1 aparece el mismo coche con id igual a 14. Se sabe que es el mismo coche porque en el *frame* t y t+1 del *groundtruth* ese vehículo se mantiene del mismo color. El objeto con id igual a 4 no provoca un IDS ya que en los dos *frames* de nuestro sistema se mantiene con el mismo id (y por lo tanto mismo color), y en el *groundtruth* aparece también del mismo color en los dos *frames*, indicando así que se trata del mismo coche.

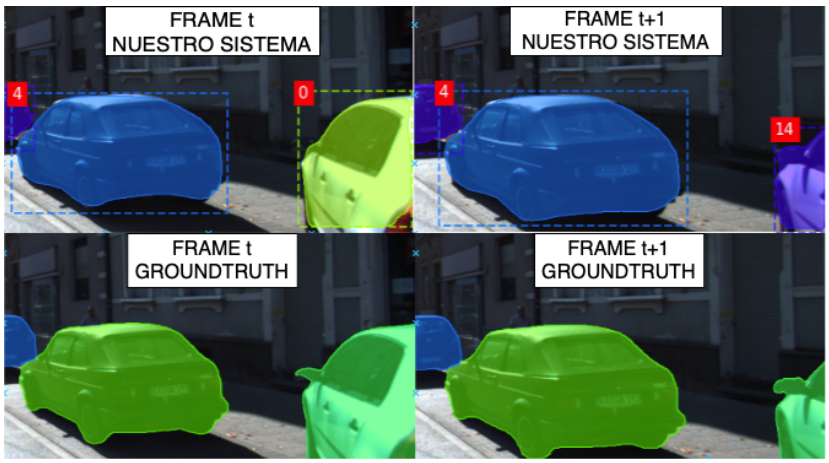


Figura 5.6: Ejemplo error por IDS

Los 3 primeros factores comentados tienen en cuenta si nuestro sistema detecta un objeto en el mismo sitio que lo hace el *groundtruth*, es decir, si se solapan los objetos que el *groundtruth* especifica que hay en el *frame* actual con los objetos que nuestro sistema ha detectado para ese mismo *frame*. Para comprobar este solapamiento se aplica nuevamente la IoU y el algoritmo húngaro. Para que un objeto del *groundtruth*

se asocie con uno detectado por el sistema, a parte de que el algoritmo húngaro los asocie, la IoU entre ambos debe ser como mínimo 0.5 [8].

Una vez conocidos estos conceptos, las métricas usadas para evaluar el sistema son una adaptación de las métricas *Multi-object tracking accuracy* (MOTA) y *Multi-object tracking precision* (MOTP) [2], ya que las utilizadas tienen en cuenta la segmentación de los objetos (su máscara). Estas métricas son:

- *Multi-object tracking and segmentation accuracy* (MOTSA): Representa cómo de bien se realiza el seguimiento de los objetos (Eqn. 5.1).

$$MOTSA = \frac{|TP| - |FP| - |IDS|}{|M|} \tag{5.1}$$

- *Multi-object tracking and segmentation precision* (MOTSP): Expresa la habilidad del sistema para estimar con precisión la posición de los objetos a nivel de máscara (Eqn. 5.2).

$$MOTSP = \frac{TPS}{|TP|} \tag{5.2}$$

- *Soft multi-object tracking and segmentation accuracy (sMOTSA)*: Es una combinación de las dos anteriores ya que tiene en cuenta lo bien que funciona el seguimiento de objetos, pero a la vez valora si se detecta correctamente la posición de estos (Eqn. 5.3).

$$sMOTSA = \frac{TPS - |FP| - |IDS|}{|M|} \tag{5.3}$$

Para el cálculo de las métricas, conforme se procesa cada *frame* se van actualizando los distintos factores: TP, FP, FN, TPS, M e IDS. Una vez que se han procesado todos, los factores contienen los valores acumulados de todos los *frames* procesados, es entonces cuando se procede a calcular MOTSA, MOTSP y sMOTSA.

En las pruebas realizadas en los siguientes apartados la métrica a tener en cuenta es MOTSA, puesto que las mejoras introducidas en el sistema (algoritmo CSRT y CNN con descriptores aprendidos) han buscado reducir los IDS, es decir, disminuir los errores producidos al asignar a un objeto el id de otro, siendo MOTSA la métrica en la que mejor se refleja la variación de este factor. Además, puesto que MOTSP no tiene en cuenta los IDS que es el factor que va a variar, el valor de esta métrica no va a cambiar.

5.2. Selección de parámetros del sistema

5.2.1. Selección del algoritmo de predicción

La librería de OpenCV dispone de muchos algoritmos de predicción. Con el fin de elegir el más adecuado para nuestro sistema, en la Tabla 5.1 se observan los resultados obtenidos al usar cada uno dentro de la versión del sistema que utiliza el solapamiento entre objetos junto con el algoritmo de predicción. Las pruebas se han realizado sobre los vídeos 1 y 20 del conjunto de *training* del *dataset* de Kitti, ya que se han considerado vídeos representativos del *dataset* completo por su complejidad y longitud.

	MOTSP	**MOTSA**	**sMOTSA**
CSRT	81.1	59.1	43.0
KCF	81.1	56.9	40.6
BOOSTING	81.1	59.0	42.9
MIL	81.1	57.2	41.1
TLD	81.1	58.2	42.1
MOSSE	81.1	52.5	36.4

Tabla 5.1: Comparativa entre distintos algoritmos de predicción: CSRT [19], KCF [12], BOOSTING [10], MIL [1], TLD [13] y MOSSE [3]

El algoritmo CSRT es con el que se ha obtenido unos mejor resultados para MOTSA, siendo por lo tanto el algoritmo de predicción seleccionado. Tanto en este como en los siguientes apartados, el elemento seleccionado para ser utilizado en el sistema se muestra de color amarillo.

5.2.2. Selección de la capa de la que extraer el descriptor

Como se ha explicado en el apartado de Diseño e Implementación, las capas candidatas de Mask R-CNN de las que extraer el descriptor con información de cada objeto son: *activation_71* (a_71), *activation_72* (a_72), *activation_73* (a_73) y *roi_align_mask* (ram), y se encuentran en la rama cuya salida da lugar a la máscara de los objetos. Recuérdese que el descriptor devuelto para un objeto por cada una de estas capas es de dimensión $14 \times 14 \times 256$.

El objetivo es seleccionar aquella capa cuyo descriptor dado para cada objeto se diferencie más del descriptor del resto de objetos. Para comprobar cómo de distintos son 2 objetos para una capa, como se puede comprobar en la Figura 5.7, se resta 1 a 1 cada posición del descriptor de un objeto dado por esa capa, con la misma posición del descriptor del otro objeto dado por esa misma capa, dando lugar así a una matriz $14 \times 14 \times 256$. Por último, se suman todos los valores que contiene la matriz. Cuanto mayor sea el resultado, más distintos considera esa capa a los dos objetos comparados.

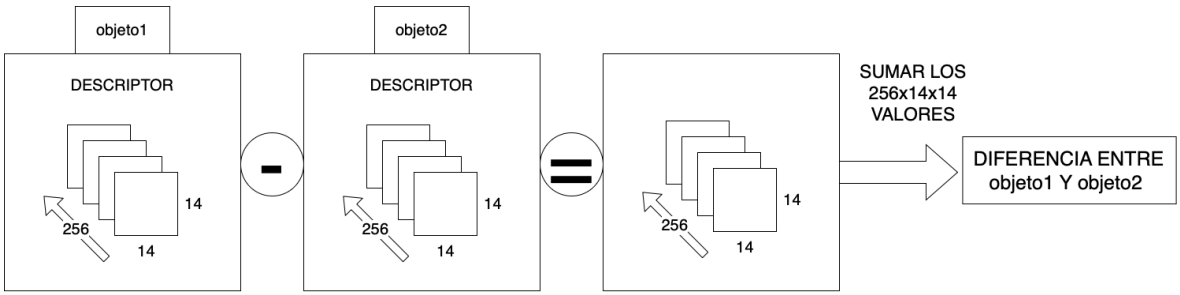


Figura 5.7: Operación aplicada sobre los descriptores de dos objetos para ver lo diferentes que son

La prueba llevada a cabo ha consistido en realizar todas las combinaciones dos a dos de los 4 coches que aparecen en el *frame* presente en la Figura 5.8.

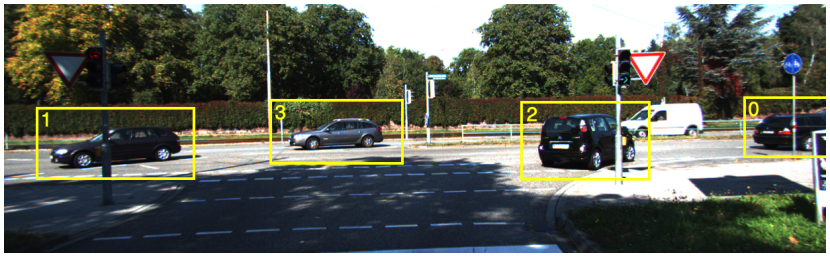


Figura 5.8: *Frame* 0 de la secuencia 4 del *dataset* de *training* de Kitti

En la Tabla 5.2 se puede ver que la mayor distancia entre cualquier par de objetos se da utilizando la capa *roi_align_mask*. Esto significa que los descriptores devueltos por esta capa son más sensibles a las diferencias entre los objetos, y que por lo tanto son capaces de identificar mejor únicamente a cada objeto. Es por eso que la capa seleccionada ha sido la *roi_align_mask*.

	0-1	**0-2**	**0-3**	**1-2**	**1-3**	**2-3**
a71	29775	27711	26460	26651	23499	28869
a72	29885	27712	25750	30327	22017	27988
a73	38368	32699	34149	41182	26509	36674
ram	139209	132285	117142	99159	126937	142326

Tabla 5.2: Comparativa de la capacidad de diferenciación de cada capa

Se esperaba que esta fuera la mejor capa a utilizar puesto que es la que se encuentra más lejos de la salida de la rama, y como dicha salida devuelve la máscara de un objeto, conforme más cerca de la salida esté una capa, más información del objeto se habrá descartado para conseguir únicamente su máscara, ignorando así toda la información adicional que se pudiera tener de él.

5.2.3. Selección de la época de la CNN

La CNN seleccionada se ha entrenado durante 200 épocas. Para elegir en qué época se comporta la red de forma más óptima, se han comparado los resultados obtenidos utilizando 4 épocas distintas del entrenamiento que cumplían los requisitos de *accuracy* alta y *loss* baja, usando una versión del sistema que realiza el seguimiento de objetos únicamente con la CNN y los descriptores aprendidos. Las pruebas se han realizado sobre los víideos 1 y 20 del conjunto de *training* del *dataset* de Kitti.

	MOTSP	**MOTSA**	**sMOTSA**
ÉPOCA 26	81.1	43.5	27.4
ÉPOCA 85	81.1	52.2	36.1
ÉPOCA 113	81.1	48.4	32.3
ÉPOCA 136	81.1	53.6	37.5

Tabla 5.3: Comparativa entre distintas épocas de la CNN

En la Tabla 5.3 se observa que la mejor época es la 136, puesto que el valor de MOTSA es mayor.

5.2.4. Selección de los pesos de IoU y CNN

Como se ha explicado en el apartado de Diseño e Implementación, para comparar dos objetos con el fin de comprobar si son el mismo usando el sistema final, se utiliza la información procedente del cálculo de la IoU y de la CNN con los descriptores aprendidos. Para ver cuánto peso se da a cada una de las 2 informaciones se han procesado los vídeos 1, 3, 4, 5, 9, 11, 12, 15 y 20 del conjunto de *training* del *dataset* de Kitti, realizando todas las combinaciones de pesos posibles.

En la Figura 5.9 se muestra el valor de MOTSA de color cian, el de sMOTSA de color verde y el del número de errores por IDS de color rojo para cada combinación de pesos, yendo desde 0 % peso de IoU y 100 % de CNN, a lo opuesto. Se puede observar que tanto centrándose en MOTSA como en los IDS, la mejor combinación es dar un 20 % de peso a la información procedente de la CNN y el uso de los descriptores aprendidos y un 80 % a la del cálculo de la IoU, puesto que con esta combinación se alcanza el valor mínimo de errores por IDS y el máximo de MOTSA.

Por último, al depender sMOTSA tanto de MOTSA como de MOTSP, y al mantener esta última un valor constante de 80.6 puesto que no se ve afectada por la variación de IDS, sMOTSA sigue el mismo patrón que MOTSA.

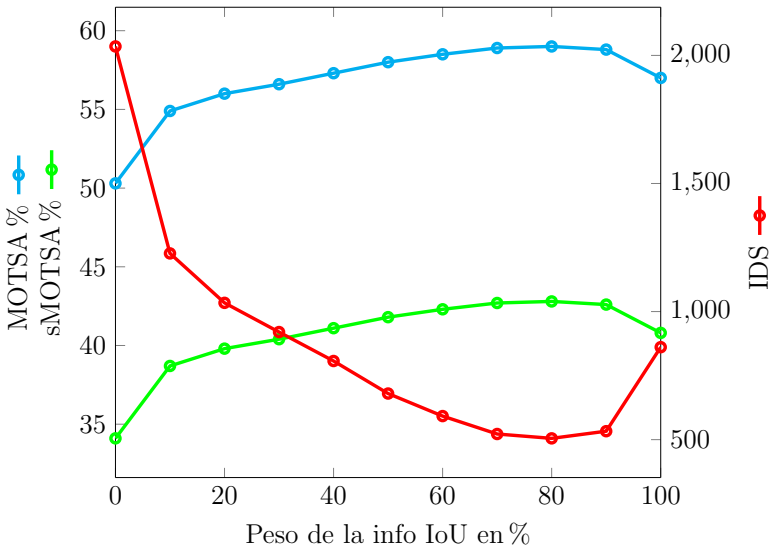


Figura 5.9: Valor de MOTSA, sMOTSA e IDS para cada combinación de pesos

5.3. Análisis del sistema

Para evaluar las diferentes versiones del sistema, así como para compararlo con el estado del arte, se utilizan los vídeos 2, 6, 7, 8, 10, 13, 14, 16 y 18 del conjunto de *training* del *dataset* de Kitti, los mismos que utiliza Track R-CNN [26].

5.3.1. Comparativa entre las distintas versiones del sistema

A continuación se va a analizar qué resultados se obtienen para las 3 versiones del sistema desarrollado, la primera versión en la que se realiza el seguimiento únicamente con el cálculo de la IoU, la que incorpora a la primera aproximación el uso del algoritmo de predicción CSRT, y la versión final, que añade el uso de la CNN y los descriptores aprendidos. En la Tabla 5.4 aparece tanto el resultado de las métricas para cada versión del sistema, como el valor para cada unos de los factores que componen dichas métricas (IDS, TP, TPS, FP, FN y M).

	IOU	**IOU+CSRT**	**IOU+CSRT+CNN**
IDS	494	335	172
TP	6984	6984	6984
TPS	5704	5704	5704
FP	1350	1350	1350
FN	1045	1045	1045
M	8029	8029	8029
MOTSP	81.7	81.7	81.7
MOTSA	64.0	66.0	68.0
sMOTSA	48.1	50.1	52.1

Tabla 5.4: Comparativa entre las distintas versiones del sistema

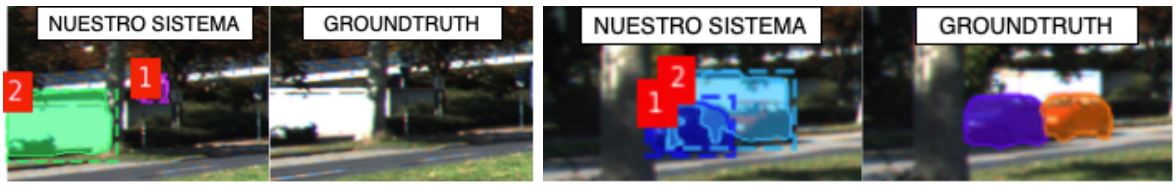
En primer lugar, de acuerdo a lo comentado al inicio del apartado de Análisis de resultados, las mejoras introducidas solo afectan a los errores por IDS, es por ello que el resto de factores de las métricas se mantiene igual. El segundo aspecto a valorar es que entre la primera versión del sistema y la segunda, MOTSA ha mejorado en un 2 % (de 64 % a 66 %), y entre la segunda y la tercera otro 2 % (de 66 % a 68 %). Se ha logrado pasar de 494 IDS en la primera versión a 335 en la segunda, y a 172 IDS en la versión final, habiendo por lo tanto una mejora sustancial en los errores de este tipo. Cabe destacar que el uso de la CNN con los descriptores aprendidos ha permitido reducir casi a la mitad los errores por IDS con respecto a la versión anterior, pero al tener MOTSA en cuenta los FP (que no se han podido reducir), esta alta disminución de los errores por IDS debida a la introducción de la CNN con descriptores aprendidos se plasma únicamente en una mejora de un 2 %. Por último, es importante mencionar que el uso de la CNN con descriptores aprendidos no empeora los resultados del sistema para ningún vídeo con respecto a la versión que no la usa.

Un ejemplo del resultado final logrado se puede ver en `https://youtu.be/Xw1aob3RjWw`, donde se observa el funcionamiento del sistema tanto para vídeos del *dataset* de Kitti como para vídeos tomados por nosotros en la ciudad de Zaragoza. De esta manera se ve la capacidad del sistema para generalizar a escenas urbanas desconocidas.

5.3.2. Variado del umbral de asociación con el *groundtruth*

Para asociar un objeto detectado por nuestro sistema con uno del *groundtruth* se ha estipulado que debe haber como mínimo un 50 % de solapamiento entre las máscaras de ambos. Como Track R-CNN usa este mismo umbral para sus resultados va a ser posible compararse con dicha red neuronal.

Nuestro sistema final devuelve un 68 % en MOTSA. El problema es que esta métrica se ve afectada por los FP, factor que no se ha podido mejorar ya que de detectar adecuadamente los objetos en un *frame* se encarga Mask R-CNN, y cuando dicha red dice que hay un coche donde no lo hay (Figura 5.10a), o si lo hay pero la máscara que le ha puesto no es suficientemente restrictiva como para tener un 50 % de solapamiento con un objeto del *groundtruth* (Figura 5.10b), se produce un FP. Por este motivo, el seguimiento de objetos como tal va mejor de lo que indica MOTSA (errores por IDS muy bajos), pero al verse afectada dicha métrica por lo bien o mal que coloque Mask R-CNN las máscaras sobre los objetos, el resultado que se obtiene empeora.



(a) Mask R-CNN detecta objetos donde no los hay

(b) Objeto con id=2 no se asocia con el naranja del *groundtruth* porque no se solapan un 50 %

Figura 5.10: Errores por FP

Por ello, en la Tabla 5.5 se muestra el resultado de MOTSA relajando el umbral de asociación con objetos del *groundtruth* del 50 % al 40, 30, 20 y 10 % respectivamente.

	50 %	**40 %**	**30 %**	**20 %**	**10 %**
MOTSA	68.0	72.7	75.7	76.9	77.3

Tabla 5.5: Resultados MOTSA con distinto umbral de asociación con *groundtruth*

5.3.3. Comparativa de nuestro sistema con Track R-CNN

La Tabla 5.6 muestra los resultados de Track R-CNN y de nuestro sistema. Para una comparación justa, ambos sistemas se han evaluado utilizando los mismos vídeos y siguiendo únicamente a coches. Además, el umbral establecido para asociar un objeto detectado con uno del *groundtruth* es 0.5 (50 % de solapamiento de las máscaras).

	MOTSP	**MOTSA**	**sMOTSA**
IOU+CSRT+CNN	81.7	68.0	52.1
TRACK R-CNN	87.2	87.8	76.2

Tabla 5.6: Comparativa entre los resultados de nuestro sistema y Track R-CNN

El que con Track R-CNN se obtengan mejores resultados se debe principalmente a que dicha red neuronal sí que modifica la capacidad de detección de Mask R-CNN, y a que no es *online* puesto que utiliza información de *frames* posteriores, a diferencia de nuestro sistema que procesa el *frame* actual utilizando únicamente información de los *frames* anteriores.

5.3.4. Análisis temporal

En este apartado se muestra el coste temporal de los puntos importantes del sistema:

– *Instance segmentation* de Mask R-CNN: Como indica en su artículo Mask R-CNN [11], 195 ms por imagen en una Nvidia Tesla M40 GPU.

Para los siguientes valores se ha calculado la media entre 700 parejas de objetos. Pruebas realizadas sobre un procesador 2,4 GHz Intel Core i5 en MacBook Pro.

– Algoritmo húngaro: Siendo n el número de objetos del *frame* actual y m el número de objetos vistos en *frames* anteriores, este algoritmo tiene un coste del orden $O(n^3)$ si $n > m$ y $O(m^3)$ si $m > n$. En la práctica se traduce en una media de 0,03 ms para que el algoritmo húngaro asocie un objeto con otro.

– Cálculo de la IoU entre 2 objetos utilizando sus *bounding boxes*: 2,1 ms

– Cálculo de la IoU entre 2 objetos usando sus máscaras: 0,009 ms

– Cálculo de la salida de la CNN para los descriptores de 2 objetos: 2,7 ms

El *dataset* de Kitti tiene una frecuencia de 10Hz (100 ms por *frame*), y por ello para que nuestro sistema se ejecute en tiempo real no se debe superar ese tiempo para procesar cada *frame*. En base al análisis temporal, nuestro sistema no podría funcionar a tiempo real, puesto que la realización de la *instance segmentation* por parte de Mask R-CNN ya tarda 195 ms (duplica los 100 ms por *frame* de Kitti). Para reducir el tiempo de esta parte del sistema se podría utilizar una tarjeta gráfica más potente, así como otra red neuronal que realice la *instance segmentation* con una precisión muy similar a Mask R-CNN pero en menor tiempo, como es Yolact [4]. Esta red neuronal es capaz de llevarla a cabo a 33,5 fps, es decir, procesa cada *frame* en 30 ms en una tarjeta gráfica Titan Xp.

Como para que el sistema se ejecute en tiempo real cada *frame* se tiene que procesar en 100 ms y se han utilizado 30 ms en realizar la *instance segmentation* con Yolact, quedan 70 ms dedicados al resto del sistema. El procesamiento de cada objeto de un *frame*, sumando el cálculo del algoritmo húngaro (0,03ms), más la IoU usando *bounding boxes* (2,1 ms), más la IoU usando máscaras (0,009 ms), más el cálculo de la salida de la CNN (2,7 ms), hacen un total de 4,8 ms. Por lo tanto, para que el sistema funcione a tiempo real se pueden llegar a procesar 14 objetos en cada *frame*, ya que la suma del tiempo de procesar este número de objetos es inferior a los 70 ms que quedaban por "gastar" de los 100.

Capítulo 6

Conclusiones, modos de fallo y líneas futuras

En este capítulo se van a presentar las conclusiones extraídas del sistema desarrollado teniendo en cuentas los resultados obtenidos. Además, se incluye un apartado en el que se exponen los puntos débiles del sistema y unas líneas futuras que buscan solucionar estos problemas, y en definitiva mejorar el sistema de Seguimiento y Segmentación de Múltiples Objetos con Descriptores Aprendidos implementado.

6.1. Conclusiones

Se ha comprobado que el uso de los descriptores aprendidos extraídos de Mask R-CNN mejora el *tracking* de objetos, puesto que se han conseguido reducir los errores por IDS un 50 % al utilizarlos. Además, su combinación con el solapamiento de objetos usando la IoU y el algoritmo de predicción CSRT han permitido crear un sistema más robusto que si se hubiera utilizado únicamente o el solapamiento de objetos o los descriptores aprendidos.

En este caso, los descriptores aprendidos se han usado a través de una red neuronal convolucional (CNN). Aunque la gran dimensión de cada descriptor ($14 \times 14 \times 256$) no permite identificar los rasgos específicos que este contiene sobre cada objeto, el uso de la CNN posibilita que la información que aporta en conjunto sobre cada uno mejore el seguimiento de los objetos.

Gracias al algoritmo de predicción CSRT y al uso de la CNN con descriptores aprendidos ha sido posible reducir enormemente la cantidad de errores por IDS (de 494 a 172), en cambio, los FP y FN no se han podido disminuir ya que dependen exclusivamente de la capacidad de Mask R-CNN de realizar la *instance segmentation*. Debido a esto, el usar varias clases de objetos distintos provoca que se den FP y FN en relación a cada clase, lo que dependiendo de la facilidad de Mask R-CNN para detectar correctamente cada una de dichas clases provocaría un empeoramiento mayor o menor

del funcionamiento del sistema.

El claro mejor funcionamiento de Track R-CNN sobre el sistema desarrollado se debe principalmente a que esta red neuronal sí que modifica la capacidad de detectar objetos de Mask R-CNN, y a que utiliza la información del *frame* anterior y posterior para realizar predicciones sobre el actual (nuestro sistema usa solo la del anterior, por lo que puede ser *online*).

Se puede concluir que se ha logrado implementar un sistema que es capaz de seguir objetos a lo largo de un vídeo en gran cantidad de situaciones, pero que como plasma el alto número de FP y FN, en ocasiones las máscaras no se colocan suficientemente bien sobre los objetos.

6.2. Modos de fallo y líneas futuras

En primer lugar, un punto interesante sería entrenar la CNN que usa descriptores aprendidos con otras clases que no fueran coches, con el fin de realizar el seguimiento y segmentación de otro tipo de objetos aprovechando la información de la CNN.

Los errores por IDS se han conseguido disminuir mucho, pero se puede lograr una reducción aún mayor. Es por ello que uno de los puntos a mejorar sería buscar una manera de utilizar los descriptores aprendidos que permitiera explotar aún más sus propiedades. Otra opción sería intentar exprimir más la CNN implementada con el fin de que sin cambiar la estructura del sistema, los descriptores aprendidos tras pasar por la CNN aportaran información más relevante de la que dan para diferenciarse aún mejor del resto de objetos.

El punto débil del sistema son los FP y FN. La única manera de disminuir ambos tipos de errores, provocando así un aumento de los TP, sería logrando una mejora en la *instance segmentation* (es imposible seguir correctamente los objetos si no se detectan de forma adecuada). Una posible opción sería la que se ha intentado, reentrenar Mask R-CNN, pero profundizando en ella en mayor medida. El problema de Mask R-CNN es que al detectar 80 clases de objetos distintos no es capaz de realizar la *instance segmentaion* con la precisión deseada. Por ello, este reentrenamiento debería buscar mejorar la capacidad de Mask R-CNN para realizar la *instance segmentation* sobre aquellas clases que interese detectar de acuerdo al ámbito en el que se utilice el sistema.

Otro punto a mejorar sería reducir el coste temporal del sistema de procesar un *frame*. La prioridad ha sido la funcionalidad del sistema y no su coste en tiempo.

Por último, permitir al usuario que utiliza el sistema configurar con mayor facilidad determinados parámetros de este: clases de objetos a detectar, la forma en la que se muestran los resultados, *etc*, daría lugar a una mejor experiencia de uso.

Bibliografía

[1] Babenko, Boris ; Yang, Ming-Hsuan ; Belongie, Serge: Visual tracking with online multiple instance learning. In: *2009 IEEE Conference on computer vision and Pattern Recognition* IEEE, 2009, S. 983–990

[2] Bernardin, Keni ; Elbs, Alexander ; Stiefelhagen, Rainer: Multiple object tracking performance metrics and evaluation in a smart room environment. In: *Sixth IEEE International Workshop on Visual Surveillance, in conjunction with ECCV* Bd. 90 Citeseer, 2006, S. 91

[3] Bolme, David S. ; Beveridge, J R. ; Draper, Bruce A. ; Lui, Yui M.: Visual object tracking using adaptive correlation filters. In: *2010 IEEE computer society conference on computer vision and pattern recognition* IEEE, 2010, S. 2544–2550

[4] Bolya, Daniel ; Zhou, Chong ; Xiao, Fanyi ; Lee, Yong J.: Yolact: Real-time instance segmentation. In: *Proceedings of the IEEE international conference on computer vision*, 2019, S. 9157–9166

[5] Bradski, Gary ; Kaehler, Adrian: *Learning OpenCV: Computer vision with the OpenCV library*. .°'Reilly Media, Inc.", 2008

[6] Chollet, Francois: Deep Learning with Python. 978-1617294433. In: *Shelter Island: Manning Publications* (2017), S. 384

[7] Coşkun, Musab ; Uçar, Ayşegül ; Yildirim, Özal ; Demir, Yakup: Face recognition based on convolutional neural network. In: *2017 International Conference on Modern Electrical and Energy Systems (MEES)* IEEE, 2017, S. 376–379

[8] Everingham, Mark ; Van Gool, Luc ; Williams, Christopher K. ; Winn, John ; Zisserman, Andrew: The pascal visual object classes (voc) challenge. In: *International journal of computer vision* 88 (2010), Nr. 2, S. 303–338

[9] Geiger, Andreas ; Lenz, Philip ; Stiller, Christoph ; Urtasun, Raquel: Vision meets robotics: The kitti dataset. In: *The International Journal of Robotics Research* 32 (2013), Nr. 11, S. 1231–1237

[10] Grabner, Helmut ; Grabner, Michael ; Bischof, Horst: Real-time tracking via on-line boosting. In: *Bmvc* Bd. 1, 2006, S. 6

[11] He, Kaiming ; Gkioxari, Georgia ; Dollár, Piotr ; Girshick, Ross: Mask r-cnn. In: *Proceedings of the IEEE international conference on computer vision*, 2017, S. 2961–2969

[12] Henriques, Joã. ; Caseiro, Rui ; Martins, Pedro ; Batista, Jorge: High-speed tracking with kernelized correlation filters. In: *IEEE transactions on pattern analysis and machine intelligence* 37 (2014), Nr. 3, S. 583–596

[13] Kalal, Zdenek ; Mikolajczyk, Krystian ; Matas, Jiri: Tracking-learning-detection. In: *IEEE transactions on pattern analysis and machine intelligence* 34 (2011), Nr. 7, S. 1409–1422

[14] Karpathy, Andrej u. a.: Cs231n convolutional neural networks for visual recognition. In: *Neural networks* 1 (2016), Nr. 1

[15] Kuhn, Harold W.: The Hungarian method for the assignment problem. In: *Naval research logistics quarterly* 2 (1955), Nr. 1-2, S. 83–97

[16] Leal-Taixé, Laura ; Milan, Anton ; Reid, Ian ; Roth, Stefan ; Schindler, Konrad: Motchallenge 2015: Towards a benchmark for multi-target tracking. In: *arXiv preprint arXiv:1504.01942* (2015)

[17] LeCun, Yann ; Haffner, Patrick ; Bottou, Léon ; Bengio, Yoshua: Object recognition with gradient-based learning. In: *Shape, contour and grouping in computer vision*. Springer, 1999, S. 319–345

[18] Lin, Tsung-Yi ; Maire, Michael ; Belongie, Serge ; Hays, James ; Perona, Pietro ; Ramanan, Deva ; Dollár, Piotr ; Zitnick, C L.: Microsoft coco: Common objects in context. In: *European conference on computer vision* Springer, 2014, S. 740–755

[19] LuNežič, A ; Vojíř, Tomáš ; Čehovin Zajc, L ; Matas, Jiří ; Kristan, Matej: Discriminative Correlation Filter TracNer with Channel and Spatial Reliability. In: *International Journal of Computer Vision* 126 (2018), Nr. 7, S. 671–688

[20] Newell, Alejandro ; Huang, Zhiao ; Deng, Jia: Associative embedding: End-to-end learning for joint detection and grouping. In: *Advances in neural information processing systems*, 2017, S. 2277–2287

[21] Nielsen, Michael A.: *Neural networks and deep learning*. Bd. 2018. Determination press San Francisco, CA, 2015

[22] Ostrovskaya, AA ; Semenov, NE ; Rubtsov, AO: Determination of the Requirements for the Neural Network for Recognition Algorithm UHRSI. In: *IOP Conference Series: Materials Science and Engineering*, 2019

[23] Real, Raimundo ; Vargas, Juan M.: The probabilistic basis of Jaccard's index of similarity. In: *Systematic biology* 45 (1996), Nr. 3, S. 380–385

[24] Ren, Shaoqing ; He, Kaiming ; Girshick, Ross ; Sun, Jian: Faster r-cnn: Towards real-time object detection with region proposal networks. In: *Advances in neural information processing systems*, 2015, S. 91–99

[25] Robinson, AH ; Cherry, Colin: Results of a prototype television bandwidth compression scheme. In: *Proceedings of the IEEE* 55 (1967), Nr. 3, S. 356–364

[26] Voigtlaender, Paul ; Krause, Michael ; Osep, Aljosa ; Luiten, Jonathon ; Sekar, Berin Balachandar G. ; Geiger, Andreas ; Leibe, Bastian: MOTS: Multi-object tracking and segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, S. 7942–7951

[27] Wu, Xiongwei ; Sahoo, Doyen ; Hoi, Steven C.: Recent advances in deep learning for object detection. In: *Neurocomputing* (2020)

Lista de Figuras

Lista de Tablas

Lista de Acrónimos

CNN Convolutional Neural Network

COCO Common Objects In Context

CSRT Channel and Spatial Reliability Tracker

FN False negative

FP False positive

IDS ID switch

IoU Intersection over Union

MOTA Multi-object tracking accuracy

MOTP Multi-object tracking precision

MOTSA Multi-object tracking and segmentation accuracy

MOTSP Multi-object tracking and segmentation precision

RLE Run-length encoding

ROI Region of Interest

RPN Region Proposal Network

sMOTSA Soft multi-object tracking and segmentation accuracy

TP True positive

TPS Solapamiento de True positive