Detección de Objetos con Asociación de Datos Embebida

Jaime Mallo Antelo
Tutores: Manuel Mucientes Molina y Víctor Brea Sánchez

Trabajo de Fin de Máster, Universidad de Santiago de Compostela
Master Universitario en Tecnologías de Análisis de Datos Masivos: Big Data

***Resumen*—El *tracking* visual de objetos está en pleno auge debido a sus numerosas aplicaciones, entre las que se encuentran los vehículos autónomos o la videovigilancia. En esta tarea se busca precisión a la par que velocidad siendo un requisito indispensable el funcionamiento en tiempo real. En este campo, una tendencia rompedora ha sido la introducción de detectores basados en aprendizaje profundo capaces de plantear hipótesis sobre la posición de los objetos de interés a la par que proposiciones de vectores de características, conocidos como *embeddings*, para los diferentes objetos identificados, orientados a la asociación de detecciones a lo largo del tiempo, soluciones que detectan y siguen objetos con una única red neuronal convolucional. Como respuesta a la necesidad de mejorar la capacidad discriminativa de estos vectores de características, entre objetos con identidades diferentes, en este TFM se plantea la integración de un *autoencoder* en una red de detección con *embeddings*. Con esta arquitectura es posible reducir la dimensionalidad y eliminar el ruido de los *embeddings*, potenciando la capacidad discriminativa de estos mediante la reformulación de las características de interés. El sistema propuesto ha sido evaluado en los conjuntos de datos de referencia para *tracking* mostrando mejora en rendimiento sin descuidar la velocidad de ejecución, permitiendo su funcionamiento en tiempo real.**

1. Introducción

El campo de visión por computador es una disciplina o área de la inteligencia artificial que busca dotar a un sistema de las capacidades necesarias para analizar y procesar imágenes y vídeo con el objetivo de extraer información de valor [25].

Este área dentro del campo de la inteligencia artificial, aunque bastante amplia, presenta cuatro líneas que, con la introducción del *Deep Learning* o aprendizaje profundo, han experimentado un avance significativo en los últimos años: la clasificación de imágenes, la detección de objetos, la segmentación y el *tracking*. Aunque en sus orígenes estas líneas se han desarrollado de forma independiente, actualmente comienza a presentarse un panorama donde la integración conjunta de algunas de ellas muestra determinadas ventajas.

Aquella en la que se centrará la atención es la detección de objetos en vídeos, buscando una extensión o aproximación a *tracking* de objetos. La detección de objetos consiste en determinar la posición de un determinado objeto en una imagen. Dado que estos suelen tener formas notablemente irregulares, el objeto es delimitado mediante un rectángulo, *bounding box*, de modo que se disponga de la posición y aspecto de este. Por su parte, el *tracking* consiste en asociar medidas a lo largo del tiempo a un objeto y estimar a partir de dicha asociación el estado del mismo, otorgándole una identidad única a lo largo del tiempo.

El encaje de detección y *tracking* resulta bastante natural dada la gran cantidad de aplicaciones que requieren de estas dos tareas de forma conjunta, tómese por caso los vehículos autónomos, dispositivos de vigilancia automática y muchos otros sistemas de análisis de vídeo. Así, uno de los componentes proporciona las características y posición de un determinado objeto, para un fotograma dado, y el otro componente asocia dichas características a lo largo del tiempo en sucesivos fotogramas.

Un requisito importante para estos sistemas suele ser la necesidad de funcionamiento en tiempo real. Este requisito se convierte en un reto puesto que los detectores de objetos con mayor precisión necesitan tiempos de cómputo superiores a los 100 ms para resoluciones de imagen HD720. En términos de visión por computador se entiende como tiempo real la capacidad de procesar 25-30 imágenes por segundo, tasa de reproducción más común en vídeos [7].

Hasta hace poco tiempo las arquitecturas del estado del arte abordaban la detección y el *tracking* de forma independiente con la finalidad de cumplir con el requisito del tiempo real, ejecutando para ello el detector un número limitado de veces. Sin embargo, en estos momentos existen determinadas arquitecturas estado del arte que permiten ejecutar ambas tareas de forma simultánea mediante la utilización de redes neuronales convolucionales (*Convolutional Neural Networks*, CNNs) profundas, aprovechando la extracción de características propia de la detección de objetos para establecer asociaciones entre identidades para fotogramas consecutivos.

La principal innovación en este tipo de arquitectura, conocida como JDE (*Jointly Detection and Embeddings*) [28], es la posibilidad de obtener, a partir de la red neuronal convolucional profunda, que constituye el detector, un vector de características conocido como *embedding*, que define la identidad de un objeto en base a sus características en términos de color, forma y contorno en un determinado momento de tiempo y que puede ser utilizado para reidentificar dicho objeto en

otro momento temporal.

Adicionalmente, dado que el aspecto de un objeto varía a lo largo del tiempo, en los sucesivos fotogramas que componen el vídeo, este planteamiento requiere de la actualización de dichos *embeddings* o vectores de características de modo que se enriquezca la identidad asociada a un determinado objeto en base a los diferentes *embeddings* emparejados en los sucesivos fotogramas.

Este trabajo parte de un detector de tipo JDE [28] y estudia la integración de un *autoencoder* para modificar los *embeddings*, de forma que mejore la capacidad discriminativa y asociativa de estos vectores de características.

La nueva arquitectura busca poner de manifiesto la mejora que puede alcanzar el sistema utilizando vectores de características de menor tamaño, donde cobren mayor importancia y protagonismo las características más representativas de cada objeto, evitando los problemas asociados al trabajo con espacio de alta dimensionalidad.

De forma más precisa, se propone como hipótesis de partida que la utilización de 512 características para representar cada uno de los objetos de interés, propuesta en la arquitectura JDE de partida, resulta una dimensionalidad demasiado elevada, dificultando la asociación de datos, existiendo ciertas características poco discriminativas.

A modo de resumen, las contribuciones de este trabajo son las siguientes:

- Creación de un conjunto de datos de *embeddings* para el entrenamiento de arquitecturas que requieran vectores de características representativos de personas.
- Implementación e integración de técnicas de reducción de la dimensionalidad, eliminando el ruido existente, y enriquecimiento de características mediante la aplicación de diferentes tipos de *autoencoders*.
- Presentación de resultados significativos que muestren los beneficios de utilizar vectores de características de menor tamaño a los propuestos en la arquitectura original JDE, conservando su capacidad de funcionamiento en tiempo real.

2. Trabajo Relacionado

Previo a la llegada de las redes neuronales convolucionales, gracias al aumento de las capacidades de cómputo, dos estrategias dominaban el estado del arte de detección: detectores de objetos específicos de una clase y detectores de objetos en movimiento.

Los detectores para objetos de clases específicas estaban basados en la identificación de estos mediante el registro de elementos, formas, colores o texturas permitiendo a posteriori buscar objetos con estas características en una imagen. Algunas representaciones de estos algoritmos son Viola Jones [26], [27] y detectores HOG [5], entre otros. Por su parte, los detectores de objetos móviles, como por ejemplo, los sustractores de fondo [1], [13], [24], se centran en buscar cambios en secuencias de imágenes. Básicamente, estas técnicas separan el fondo del frente buscando identificar los cambios que se producen en el frente de la escena. Estas técnicas, aunque innovadoras en su momento, presentan algunos problemas como la imposibilidad de trabajar con cámaras móviles o incluso abordar escenas diferentes proporcionadas por varias cámaras, la precisión estaba sujeta a la presencia de un entorno sin distracciones, como podrían ser los cambios de iluminación o la aparición de sombras, existiendo problemas también a la hora de delimitar objetos dispuestos muy juntos ente ellos como agrupaciones de personas muy cercanas.

Las redes neuronales convolucionales han permitido un gran avance en los problemas de detección, solventando en su mayoría los problemas asociados a las soluciones anteriores. Así, los detectores basados en aprendizaje profundo pueden dividirse en dos grupos o categorías, *two-stage detection* o detección en dos pasos y *one-stage detection* o detección en un único paso.

Los detectores en dos pasos se conocen con este nombre porque la detección sucede en dos etapas. En la primera de ellas el modelo propone una serie de regiones de interés, las cuales son candidatas a presentar un objeto del tipo buscado. En la segunda etapa, un clasificador se encarga de determinar, en estas regiones, la existencia o no de un objeto asociado a ciertas clases. Estos cobraron especial relevancia a partir de 2014 con la introducción de R-CNN [11], que utiliza búsqueda selectiva para la proposición de regiones de interés y CNN (*Convolutional Neural Network*) para la extracción de características de cada una de las regiones de interés, siendo posteriormente utilizadas para predecir las categorías y *bounding boxes*. Tomando esta arquitectura como punto de partida surgen un número importante de mejoras, como por ejemplo, Fast RCNN [10] capaz de incorporar un detector y un regresor para el ajuste de las *bounding boxes* bajo la misma red o Faster RCNN [23] que incorpora lo que se conoce como RPN (*Region Proposal Network*). En 2017, aparece una técnica denominada FPN (*Feature Pyramid Network*) [18], que se centra en la obtención de características con alto contenido semántico independientemente de la escala seleccionada. Si bien hoy en día existen multitud de técnicas de detección de dos etapas, muchas de ellas se inspiran en la red FPN, utilizando pirámides de características.

Aunque este tipo de arquitecturas supuso una gran mejora en cuanto a la precisión de este tipo de sistemas, continuaba existiendo un problema y ese es la imposibilidad de funcionar en tiempo real debido al coste computacional de su ejecución.

La otra tendencia en cuanto a detectores son los *one-stage detectors* o detectores en un único paso. La diferencia con los anteriores es que omiten la fase de propuesta de regiones y aplican directamente detección sobre un denso muestreo de posibles ubicaciones. Lo buscado con esta tendencia es reducir los tiempos de cómputo sacrificando en ciertos casos algo de precisión.

El primer detector de este tipo fue YOLO (*You Only Look Once*) [21], que divide la imagen en regiones o celdas y predice las *bounding boxes* y la probabilidad de cada clase en cada una de las regiones. Especial mención

merece la versión 3 [22] de este detector, en el cual se añade la arquitectura FPN, lo que permite la detección de objetos en varias escalas, incluso objetos pequeños, un problema recurrente en las primeras versiones.

En relación con el *tracking* visual, una de las aproximaciones de mayor éxito son los *trackers* basados en filtros de correlación discriminativos [4]. Estos filtros modelan la apariencia del objeto de modo que una vez detectado e identificadas sus características en el primer fotograma, en los siguientes se utiliza la técnica de ventana deslizante para recorrer la imagen en busca de dicho objeto a partir del filtro con sus características, determinando la posición del objeto como aquella donde se obtiene la mayor correlación con el filtro definido para el objeto.

Asimismo, en *tracking* también han tenido un éxito notable los filtros basados en modelos de movimiento, como los filtros de Kalman [14], los cuales buscan determinar la posición de un objeto en base a su trayectoria, considerando la característica de continuidad del movimiento.

Sin embargo, el auge de las redes neuronales convolucionales no sólo tuvo impacto a nivel de detección sino también a nivel de *tracking* apareciendo dos tendencias dominadoras del estado del arte. Las primeras de ellas, explotaban los beneficios del aprendizaje profundo mediante la utilización de CNN para la correlación discriminativa [6], [3], [30]. La otra de las tendencias son las redes Siamesas [2], que proponen la utilización de una CNN precalculando las características del objeto de interés con anterioridad para posteriormente localizar dichas características en los siguientes fotogramas. El problema de todos estos *trackers* es su escalabilidad con el número de objetos y la necesidad de operar de forma conjunta con un detector.

Todo sistema completo de detección y *tracking* requiere de la asociación de datos. Este componente es el encargado de integrar la información aportada por el detector y el *tracker*. Los avances en este campo están representados por los *trackers* conocidos como MOT (*Multiple Object Trackers*), centrados en el paradigma *tracking-by-detection* [17], [19]. Este tipo de soluciones requieren de dos componentes de computo intensivo como son el detector y un modelo de *embeddings* encargado de extraer las características de los objetos para proceder a la asociación de detecciones y *trackers*, lo que se conoce como métodos SDE (Detección y *Embeddings* Separados). Estos sistemas tienen como principal limitación su tiempo de inferencia, al estar desacoplada la detección de la asociación de datos. Esto no sucede con los detectores de tipo JDE (Detección y *Embeddings* Conjuntos) en los que las características extraídas de bajo nivel son utilizadas tanto para detección como para la asociación de datos, con el ahorro en tiempo de cómputo que ello supone.

3. Arquitectura del Sistema

El sistema dispone de varios componentes, cada uno de los cuales juega un rol diferente en la identificación de objetos en tiempo real. Este se compone de una red neuronal convolucional profunda encargada de determinar la posición de cada uno de los objetos, así como la extracción de un vector de características que representa a cada uno de los objetos de interés. Como el objetivo es asociar detecciones a lo largo del tiempo, el sistema dispone de un *framework* encargado de esta tarea. Adicionalmente, con la finalidad de mejorar la asociación de datos, se incluye un *autoencoder*, donde radica la aportación principal de este trabajo, que permite eliminar el ruido y resumir las características de cada objeto, facilitando y mejorando la calidad de la asociación de datos. La arquitectura del sistema se ilustra en la figura 1. A continuación, se describen sus componentes y funcionalidades.

3.1. Modelo de Detección y *Embeddings* (JDE)

La arquitectura JDE se compone de una red neuronal convolucional que introduce un concepto novedoso en las áreas de *tracking* y detección como es la integración en una misma red de un detector y un modelo de *embeddings*. Esto permite no sólo determinar la posición de los objetos existentes en una determinada imagen, sino también la obtención de un resumen de características para cada objeto identificado. Esto resulta clave para garantizar el funcionamiento en tiempo real, puesto que se reutiliza la propia red de detección para la obtención de estos vectores de características sin añadir un coste computacional importante, como sí sucedería si se utilizasen dos arquitecturas independientes, cada una de ellas destinada a una tarea.

3.1.1. Extractor de Características (*Backbone*). La red utilizada emplea como extractor de características generales de la imagen la arquitectura DarkNet-53 [20]. Una de las características que hace popular a esta arquitectura, en comparación con otras, es su velocidad, sin perder precisión. La arquitectura de esta red que se puede ver en la figura 2 se compone de una serie de capas de convolución entre las que se intercalan capas de tipo *shortcut*, las cuales permiten fusionar características de la capa inmediatamente anterior con características extraídas en capas anteriores, enriqueciendo la representación.

3.1.2. FPN (Feature Pyramid Network). Debe tenerse en cuenta que los objetos de interés se pueden situar a una mayor o menor distancia del objetivo encargado de captar la escena con lo que su escala puede variar notablemente, incluso dentro de la misma escena. Esto pone de manifiesto la necesidad de trabajar con diferentes escalas. Este problema debe ser abordado de forma cuidadosa para evitar un incremento notable en el tiempo de computación.

Por ello, se utiliza una arquitectura FPN (*Feature Pyramid Network* o pirámide de características), que se integra en la red, permitiendo, ahora sí, realizar predicciones a diferentes escalas. La principal ventaja es que utiliza el valor semántico generado por el mapa de características de menor resolución, el obtenido en las últimas capas de la red, para aportar mayor valor a las características

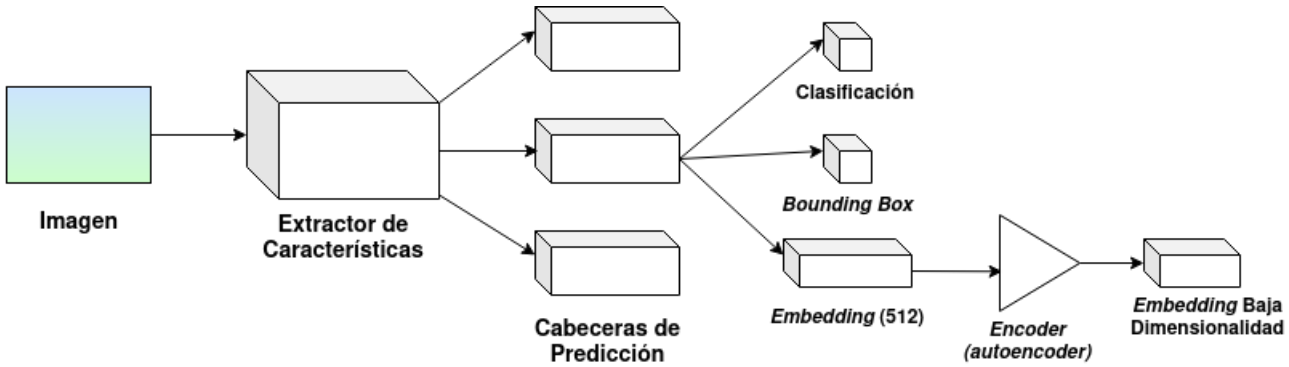

Figura 1: Arquitectura del sistema presentado en este trabajo.

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

Figura 2: Disposición de las capas que conforman la arquitectura Darknet-53 (Imagen extraída de Joseph Redmon et al. [22]).

propias de una escala de mayor tamaño o con una mayor resolución.

Así, a partir de las características con mayor valor semántico, obtenidas mediante un proceso de *downsampling*, se reconstruyen mapas de características de mayor resolución donde encontrar objetos de mayor escala, utilizando, para evitar distorsionar la posición de los objetos, las características aportadas por el *backbone* para la resolución correspondiente mediante un proceso de *upsampling*. De este modo, se realizaría una pasada hacia delante en la red y una reconstrucción hacia atrás, combinando las características con mayor valor semántico del nivel o escala inmediatamente anterior con las propias de cada escala proporcionadas por al red. Para el problema abordado se utilizan tres escalas de 1/32, 1/16 y 1/8 respectivamente, en relación al tamaño de entrada definido por la red. La figura 3 ilustra el funcionamiento de esta arquitectura.

3.1.3. Cabecera de Predicción (*Prediction Head*). Estos tres mapas de características obtenidos mediante la técnica anteriormente expuesta permiten detectar objetos

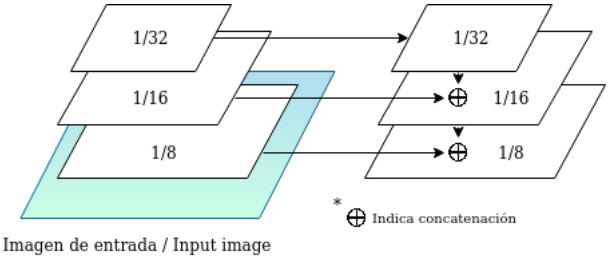

Figura 3: FPN (Feature Pyramid Network o Pirámide de Extracción de Características).

en diferentes escalas. Tras una parte común en la red encargada de extraer características, esta contará con tres componentes diferentes en cuanto a dimensiones pero idénticos en cuanto a configuración para detectar objetos en cada una de las escalas. Este nuevo componente se conoce como cabecera de predicción o *prediction head* y permite extraer la posición del objeto de interés, el vector de características, la clase y la confianza en la detección, medida que se puede traducir en la probabilidad de que realmente aquello que se ha identificado sea un objeto de interés y no el fondo de la imagen. Estas cabeceras de predicción o *prediction heads* se componen de una serie de capas de convolución adaptadas a las tres escalas de características empleadas por la red.

Para que la red aprenda a delimitar los objetos mediante un rectángulo, habitualmente conocidos como *bounding boxes*, se utiliza una técnica basada en *anchors*. Los *anchors* son *bounding boxes* o rectángulos predefinidos de un cierto ancho y alto. El procedimiento se describe en la figura 4. Básicamente, lo que se busca con estas estructuras es que la red no tenga que aprender de cero a delimitar los objetos, en cuyo caso el entrenamiento sería mucho más lento y menos efectivo, sino que estos *anchors* están adaptados a las dimensiones y al aspecto o ratio de los objetos de interés.

3.1.4. Función de Coste. Un aspecto fundamental en el entrenamiento de cualquier red neuronal, más si cabe en este caso dada la complejidad de la arquitectura, es la función de coste utilizada. Como la red busca ajustar la probabilidad de existencia de un objeto en una posición dada, la *bounding box* o rectángulo que delimita el objeto y el vector de características que lo representa, también conocido como *embedding*, requiere

Figura 4: Utilización de *anchors* para permitir a la red aprender de forma sencilla a delimitar los objetos de interés en las diferentes celdas en que se divide la imagen.

de la combinación de tres funciones de coste, una por cada uno de estos aspectos a aprender.

En relación a la detección de objetos se utilizan dos funciones de coste. Por una parte, una función de clasificación para lo cual se utiliza la entropía cruzada, presentada en la ecuación 1,

$$L_\alpha = -\sum_{c}^{C} y_{oc} \cdot log(p_{oc}) \tag{1}$$

donde C representa el número de clases existentes, y_{oc} un valor binario que vale 1 para la clase correcta de la observación o y 0 en otro caso y p_{oc} la probabilidad predicha para la observación o de pertenecer a la clase c.

Por otra, la función encargada de lo que se conoce como *bounding box regression* es la función de coste smooth-L1, presentada en la siguiente ecuación,

$$L_\beta = \begin{cases} 0,5 \cdot x^2 & \text{if} \quad |x| < 1 \\ |x| - 0,5 & \text{otherwise} \end{cases} \tag{2}$$

donde x es la distancia L1 entre dos vectores compuestos por los valores que definen la *bounding box* que delimita el objeto (posición de la esquina superior izquierda y esquina inferior derecha). Esta función permite medir la distancia entre la *bounding box* predicha y la proporcionada por el *groundtruth* tomada como referencia [9].

La función de coste para los *embeddings* busca que la red sea capaz de aprender un espacio de 512 dimensiones donde instancias de la misma identidad se encuentren cercanas entre sí e instancias de diferentes identidades se encuentren bastante distanciadas.

$$L_\gamma = -log\frac{exp\left(f^T g^+\right)}{exp\left(f^T g^+\right) + \sum_i exp\left(f^T g_i^-\right)} \tag{3}$$

La función de coste utilizada, presentada en la ecuación 3, es una adaptación de la entropía cruzada, donde se considera para cada instancia, f^T, de un *mini-batch*, ciertos ejemplos como positivos y otros como negativos en función de si dos objetos tienen o no la misma identidad y con ello deben estar cerca o no en el espacio. Adicionalmente, al tratarse de la entropía cruzada, se incorpora la probabilidad de la clase positiva, g^+, a la que pertenece el *anchor* y la probabilidad de las clases negativas, g_i^-.

Naturalmente, al utilizar la retropropagación del error, para el ajuste de los pesos de la red neuronal convolucional profunda, se necesita combinar las tres funciones de coste existentes en una, permitiendo así el aprendizaje de la red.

$$L_{total} = \sum_{i}^{M} \sum_{j=\alpha,\beta,\gamma} \frac{1}{2}\left(\frac{1}{e^{s_j^i}} L_j^i + s_j^i\right) \tag{4}$$

La formulación de dicha función de coste global se muestra en la ecuación 4. Ésta se presenta como la suma lineal ponderada del coste para los diferentes componentes y diferentes escalas, donde L_α se corresponde con la función de coste que permite a la red aprender a clasificar el objeto, L_β la función de coste asociada al aprendizaje de las *bounding boxes* y L_γ la función de coste encargada del ajuste de los *embeddings* durante el entrenamiento.

De la ecuación anterior cabe puntualizar que M es el número de cabeceras de predicción o *prediction heads*, tres para esta arquitectura. Adicionalmente, se puede ver como existen una serie de coeficientes, s_j^i, que es necesario determinar para cada una de las funciones de coste. Estos podrían ser idénticos para las tres funciones de coste utilizadas o bien definidos diferentes pero de forma aleatoria mediante prueba y error. Sin embargo, aunque el funcionamiento de estas técnicas no tendría por que ser malo, alcanzando resultados aceptables tras varios intentos, en ningún caso sería óptimo. Este motivo lleva a utilizar una técnica que permite aprender de forma automática dichos pesos utilizando el concepto de *task independent uncertainty* [15], que consiste en aprender la incertidumbre homocedástica, dependiente de la tarea y no de los datos, para ponderar la función de coste de cada tarea en una función de coste global.

3.2. Asociación de Datos

Aunque la red de detección y generación de vectores de características representativos de los diferentes objetos utilizados es la parte donde radica, principalmente, lo novedoso de esta arquitectura, es necesario utilizar alguna estrategia para asociar detecciones a lo largo del tiempo para los sucesivos fotogramas. Dicho de otro modo, una vez se conoce la posición y dimensiones de un objeto se busca determinar si aparece en varias ocasiones en diferentes momentos de tiempo.

Para ello, se utilizarán diversas técnicas ampliamente extendidas en *tracking* que de forma conjunta ofrecen un gran rendimiento proporcionando robustez al sistema.

3.2.1. Filtrado de Detecciones. Junto con la posición del objeto y el vector de características devuelto por la red se obtiene una métrica que resume la confianza de la detección, esto es, la probabilidad de que el objeto identificado sea de la clase predicha. Para reducir el número de falsos positivos obtenidos, siendo esto la aceptación de una hipótesis en una posición donde no hay ningún

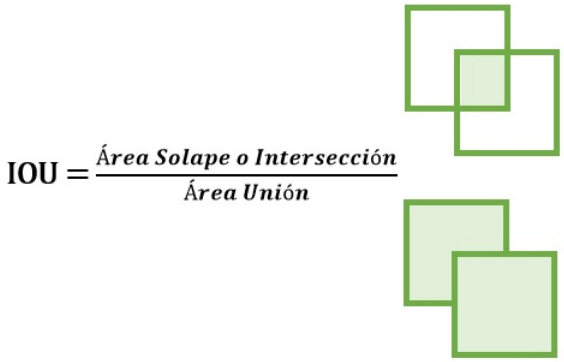

Figura 5: IOU (Intersection Over Union) o solape entre dos *bounding boxes*.

objeto de interés, debe establecerse un valor de corte o umbral a partir del cual se considera una detección. Así pues, la confianza es uno de los parámetros del modelo, y su valor depende de la aplicación.

3.2.2. IOU (Intersection Over Union o Intersección Sobre la Unión). El IOU (*Intersection Over Union*) mide el solape o superficie común entre dos *bounding boxes* o rectángulos que delimitan los objetos de interés, tal y como ilustra la figura 5.

El principio en que se sustenta esta técnica, tan sencilla como efectiva, es que los objetos en vídeos describen trayectorias continuas, de modo que entre un fotograma y el siguiente, suponiendo vídeos con una tasa de reproducción común (entre 24 y 30 FPS), la posición de los objetos variará pero no lo hará de forma abrupta. Téngase en cuenta que una tasa de 30 FPS, supone que entre un fotograma y el siguiente hay una salto temporal de 0,03 segundos, habitualmente imposibilitando un cambio de posición muy notable.

Un problema que se puede presentar, en este sentido, es la dificultad de asociar mediante esta técnica detecciones que se encuentren en posiciones muy cercanas o incluso parcialmente ocluidas, puesto que dicha cercanía puede conllevar cambios de identidad en el proceso de asociación indeseados. Es por ello, que esta técnica suele ser complementada por alguna otra que aporte robustez al sistema en este tipo de situaciones, utilizando no sólo la posición del objeto sino también sus características.

3.2.3. *Embeddings* o Vectores de Características. La red empleada para la detección de objetos es utilizada también para la generación de vectores de características representativos de cada objeto detectado. Estos describen de forma compacta la forma, el color y el aspecto de los objetos de interés.

Para poder asociar *embeddings* de dos fotogramas diferentes que representen la misma identidad debe utilizarse alguna medida de distancia. Para ello se utiliza la distancia del coseno, presentada en la ecuación siguiente,

$$similaridad(A, B) = \frac{A \cdot B}{||A|| \times ||B||} \qquad (5)$$

definida como la similitud entre dos vectores, A y B, en el espacio que determina el valor del coseno del ángulo comprendido entre los dos vectores. Esta métrica es interesante para este fin ya que toma valores en el intervalo $[-1, 1]$, intervalo acotado.

Un factor a tener en cuenta es que los vectores de características para un mismo objeto en diferentes fotogramas serán diferentes, aunque similares. Esto se debe a que la apariencia de los objetos cambia en el tiempo debido a múltiples factores como el movimiento, la orientación, la iluminación, etc. Por ello, es interesante utilizar toda la información existente para una identidad en relación con los *embeddings*. Esto puede aumentar el coste computacional de forma importante, al requerir del calculo de distancias con todos los vectores de características obtenidos en los sucesivos fotogramas para cada identidad. Para evitarlo se combinan las características de una identidad para los sucesivos fotogramas en los que aparece. De forma precisa, lo que se hace es fusionar cada nuevo vector de características con los anteriores, ponderando el nuevo vector de características en un 10 % y los anteriores, previamente fusionados, en un 90 %, minimizando el impacto de una asociación incorrecta. En cualquier caso, estos umbrales son también hiperparámetros del algoritmo y como tales dependientes de la aplicación bajo estudio.

3.2.4. Algoritmo de Optimización de las Asociaciones. Hasta el momento se han presentado técnicas para determinar la distancia o similitud entre dos objetos, una basada en la posición del objeto, IOU, y otra basada en las características de este, distancia del coseno entre *embeddings*. Para un fotograma dado, las distancias deben calcularse entre todos los objetos detectados para dicho fotograma y los anteriores, considerando como mínimo la información del fotograma anterior. El resultado son dos matrices de distancias entre las hipótesis planteadas para el fotograma actual y las aceptadas para los fotogramas anteriores donde cada celda define la distancia entre dos detecciones en base a IOU o distancia del coseno entre *embeddings* en sendas matrices.

Para un buen funcionamiento del sistema, esta asociación de información debe realizarse de forma que se optimice el global de los emparejamientos, no sólo buscando asociaciones factibles de forma individual sino de forma global. Con tal fin se utiliza lo que se conoce como método Húngaro, un algoritmo que dada una matriz de coste, con los elementos representados mediante las filas y las columnas, busca la combinación de filas y columnas que minimice o maximice, según el caso, el coste global de las múltiples asignaciones.

3.2.5. Filtros de Kalman. Aunque la asociación de objetos mediante *embeddings* proporciona múltiples ventajas presenta un problema importante y es que estos no consideran información espacial sobre la posición del objeto. Esto puede provocar que se asocien objetos muy distantes en dos fotogramas consecutivos, siendo altamente improbable un cambio de posición tan abrupto.

Precisamente a modo de revisión de las asociaciones, en base a los *embeddings* o vectores de características, se utilizan filtros de Kalman [14]. Lo que se busca utilizando esta técnica es determinar si la posición en la que se encontraría el objeto en el siguiente fotograma

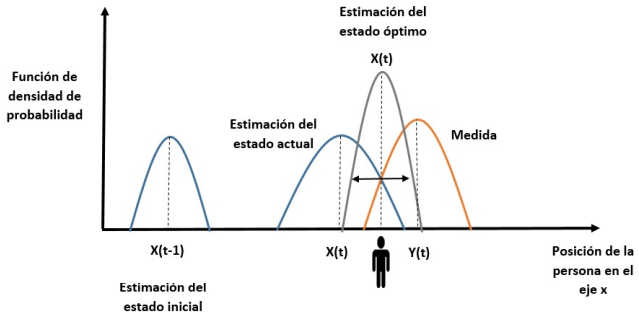

Figura 6: Utilización del filtro de Kalman para estimar la posición de una persona para el eje X.

de acuerdo a la asociación es factible o no.

Los filtros de Kalman son algoritmos que se encargan de estimar o predecir los parámetros de interés en base a la localización, velocidad y dirección ante la presencia de medidas con ruido. Busca estimar variables de interés cuando no pueden ser medidas de forma directa. Esto permite que se combinen medidas con diferentes grados de ruido.

Para este problema, lo que se busca es conocer la posición de un objeto en el momento $t+1$ conocida la posición en el momento t y $t-1$ o incluso momentos anteriores a estos, considerando la continuidad característica del movimiento. De este modo se puede determinar si la posición en la que se encuentra el objeto, de acuerdo a la asociación de información realizada, es factible en base a su trayectoria.

Esta técnica, ilustrada en la figura 6, describe un proceso compuesto por una etapa de predicción y otra de actualización, presentado en las ecuaciones 6 y 7 respectivamente.

$$Ecuación\ de\ predicción: X(t) = F*X(t-1)+V_q(t-1) \quad (6)$$

$$Ecuación\ de\ actualización: Y(t) = H*X(t)+V_p(t) \quad (7)$$

En las ecuaciones anteriores $X(t)$ y $Y(t)$ son la variable de estado a estimar y la variable medida respectivamente, correspondiéndose con la posición del objeto en el momento t. Por su parte, F es la matriz de transición de estados y H la matriz de medidas. $V_q(t)$ y $V_p(t)$ representan el ruido del sistema y las medidas, en ese orden.

De este modo, en las sucesivas iteraciones, coincidiendo con los fotogramas que componen el vídeo, para cada identidad se realiza la predicción de la posición en base a información de fotogramas anteriores, y tras la asociación se lleva a cabo la etapa de actualización del estado con la nueva información sobre dicha identidad.

3.2.6. *Framework* o Marco de Asociación de Datos.

Hasta el momento se han presentado los diferentes componentes utilizados por el *framework* o marco de asociación de datos en sucesivos fotogramas. Para un correcto funcionamiento, es necesario poner en conjunto todos ellos, de modo que se utilicen las hipótesis planteadas por el detector para determinar las identidades de los objetos presentes en el vídeo. El flujo del sistema en el procesado de un vídeo de entrada se presenta en el algoritmo 1.

Algoritmo 1: Asociación de detecciones en vídeo.

Input: Vídeo o fotogramas.
Resultado: Posición identidades identificadas en el vídeo.

Inicialización;
tracks_activos = [ ]
tracks_no_confirmados = [ ]
tracks_perdidos = [ ]
tracks_eliminados = [ ]
mientras *No final vídeo o hay fotograma* **hacer**
  detecciones = **Detector**(fotograma);
  detecciones = **EvaluarConfianzaDetecciones** (detecciones, umbral_confianza);
  grupo_tracks = tracks_activos + tracks_perdidos;
  distancias = **DistanciaCoseno** (grupo_tracks, detecciones);
  distancias = **Kalman** (distancias, grupo_tracks, detecciones);
  emparejamientos, u_grupo_tracks, u_detecciones = **MétodoHúngaro** (distancias);
  distancias = **IOU** (u_grupo_tracks, u_detecciones);
  emparejamientos, u_grupo_tracks, u_detecciones = **MétodoHúngaro** (distancias);
  distancias = **IOU** (tracks_no_confirmados, u_detecciones);
  emparejamientos, u_no_confirmados, u_detecciones = **MétodoHúngaro** (distancias);
  ActualizarTracksActivos (emparejamientos, tracks_activos);
  EvaluarTracksPerdidos (u_grupo_tracks, tracks_eliminados, tracks_perdidos, max_tiempo_vida);
  DefinirNuevasIdentidades (u_detecciones, tracks_no_confirmado);
  EliminarFalsosPositivos (u_no_confirmados);
fin

En la asociación de datos tiene un papel importante el estado de los *tracks*, entendiendo por estos un conjunto de detecciones de diferentes fotogramas pertenecientes a la misma identidad. Así, existen 4 clases, los activos, aquellos a los que se ha asociado una nueva detección en el último fotograma, los no confirmados, entendiendo por estos detecciones que no han sido asociadas a ningún *track* en el último fotograma y para los que se desconoce si describen un nuevo objeto que ha entrado en la escena

o bien si se trata de un fallo del detector, los perdidos, aquellos *tracks* a los que no se ha asociado ninguna detección en el último fotograma, y los eliminados, considerados aquellos que han abandonado la escena.

El sistema procesa los fotogramas del vídeo de entrada uno a uno. Cada vez que se recibe un nuevo fotograma este pasa por el detector (línea 7 del algoritmo) dando como resultado una serie de hipótesis que deben ser evaluadas. Estas hipótesis plantean la existencia de un objeto de interés en una determinada posición y proporcionan una descripción de las características de este mediante un *embedding*. Adicionalmente, cada hipótesis viene acompañada de una confianza, que explica la probabilidad de que la hipótesis represente un objeto de interés. Este valor es utilizado para eliminar aquellas hipótesis por debajo de un umbral de confianza (línea 8 del algoritmo), consideradas como poco fiables y que por ello contribuyen a la aparición de falsos positivos.

Tanto la posición como los *embeddings* serán utilizados para asociar las nuevas hipótesis a hipótesis de los fotogramas anteriores para definir las identidades presentes a lo largo del vídeo. Al comienzo se intentarán asociar las nuevas hipótesis a los *tracks* activos y aquellos que se encuentran perdidos temporalmente.

Así, la primera de las técnicas utilizadas para asociar las detecciones del nuevo fotograma con las de anteriores es la distancia del coseno entre *embeddings* (línea 10 a 12 del algoritmo). Como se comentaba con anterioridad, se utiliza el filtro de Kalman para determinar si una asociación dada es factible. Tras este proceso no todas las detecciones tienen por qué haber sido asociadas a un *track*. Por ello, las hipótesis restantes se intentan asociar en base a su posición utilizando como tecnica para determinar la distancia entre las nuevas hipótesis y los *tracks* existentes IOU (líneas 13 y 14 del algoritmo). En ambos casos, tanto utilizando los *embeddings* como la posición, se optimizan los emparejamientos mediante el método Húngaro, buscando el menor coste global en términos de distancia del coseno e IOU respectivamente.

Una vez se han emparejado las nuevas hipótesis con los *tracks* activos y perdidos temporalmente, se procede a la asociación de las hipótesis restantes con los *tracks* no confirmados, asociados a objetos aparecidos únicamente en el fotograma inmediatamente anterior (líneas 15 y 16 del algoritmo). Para este emparejamiento se utiliza la posición mediante IOU como se hizo anteriormente. La utilización de la posición en lugar de los *embeddings* responde a la incertidumbre existente en torno a una hipótesis para el que se desconoce si se trata de un falso positivo, de modo que la posición resulta un mejor indicador.

Una vez se da por finalizado el proceso de emparejamiento, se actualizan los *tracks* activos (línea 17 del algoritmo), incluyendo en este grupo los ya activos en el fotograma anterior, los confirmados, aquellos *tracks* presentes en más de un fotograma, y aquellos que se encontraban perdidos y a los que se les ha asociado una hipótesis en el presente fotograma. Además, se eliminan aquellos *tracks* perdidos que han superado el tiempo máximo de vida (línea 18 del algoritmo), número de fotogramas durante el cual se considera que pueden volver a aparecer. Por último, se define un *track* no confirmado para las detecciones no emparejadas (línea 19 del algoritmo) y se eliminan los no confirmados del fotograma anterior que no han sido actualizados para el nuevo fotograma (línea 20 del algoritmo).

3.3. Optimización de los *Embeddings* o Vectores de Características

El planteamiento inicial propone utilizar *embeddings* con 512 características. La hipótesis de partida es que trabajar con vectores de características de alta dimensionalidad, como es el caso, es contraproducente, añadiendo complejidad que directamente afecta de forma negativa al resultado.

Cuando se utilizan representaciones muy complejas, aunque bien es cierto que se suelen capturar todas las características de los objetos, se puede capturar una cantidad de ruido importante que haga que los resultados no sean óptimos o puedan ser mejorados.

De forma más técnica, cuando se aumenta la dimensionalidad el volumen del espacio aumenta exponencialmente haciendo que los datos disponibles se vuelvan más dispersos. Al ser utilizadas distancias en dicho espacio para asociar detecciones, aunque es interesante distanciar instancias con identidades diferentes, puede darse el caso de que todos los objetos parezcan distantes y diferentes en multitud de aspectos, lo que impide que las estrategias de asociación de características sean eficaces.

En este contexto, se plantea la utilización de *autoencoders* [12]. Aunque existen multitud de variantes de este modelo, todos ellos presentan una serie de rasgos comunes, buscan en primer lugar proyectar el espacio actual en un espacio latente, de menor dimensionalidad, y posteriormente, a partir del espacio latente, intentan reconstruir la entrada, proporcionando como salida la entrada reconstruida. Están conformados por un *encoder* o codificador, que se encarga de transformar el espacio inicial en el espacio latente, y por un *decoder* o decodificador que busca reconstruir el espacio original a partir del nuevo espacio generado.

Esta técnica resulta muy interesante porque permite, por una parte, reducir la dimensionalidad de los vectores de características, que es lo buscado en última instancia, y, por otra, minimizar el ruido propio de estas representaciones, eliminando aquellas características comunes a la clase o tipo de objeto que no son de interés en la asociación de datos, siendo estas asumidas por el modelo. Al intentar reconstruir la entrada a partir de un espacio de menor dimensionalidad, espacio latente, deben encontrarse las características diferenciadoras, ya que las comunes pueden ser aportadas por la red a través de los parámetros sin necesidad de incluirlas en este espacio.

De forma general, los *autoencoder* son muy utilizados en visión por computador y otras áreas para reconstruir imágenes con baja resolución, como PCA (Análisis de Componentes Principales) o como técnica

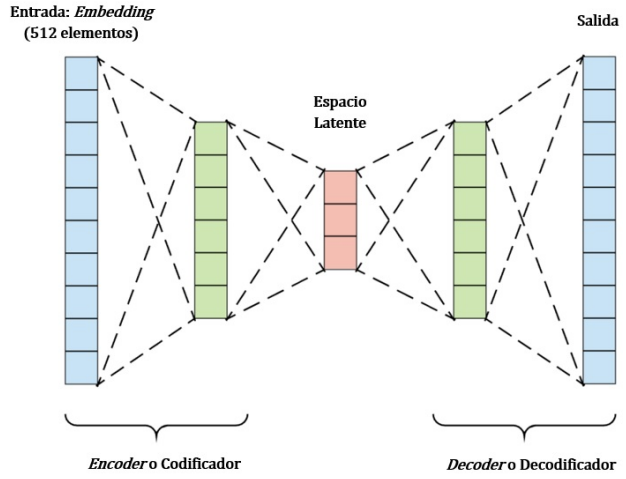

Figura 7: *Autoencoder* estándar.

para la detección de anomalías, entre otras.

Una de las peculiaridades de este modelo es que mientras para el entrenamiento se utiliza la totalidad de los componentes, en producción no es así, utilizándose únicamente el codificador o el decodificador en función de la tarea que se busca resolver con esta técnica. En este problema resulta bastante claro que la parte de interés es el codificador puesto que es el componente que permite reducir la dimensionalidad de los vectores de características enriqueciendo la representación.

Aunque existen múltiples variantes de esta técnica, el foco se pondrá sobre los *autoencoders* estándar y los *autoencoders* variacionales.

3.3.1. *Autoencoder* Estándar. Este tipo de *autoencoder* es el más conocido, representando el concepto puro de *autoencoder*. Este se presenta de forma esquemática en la figura 7. Para la construcción de este modelo se crearán dos módulos simétricos, el codificador, que permite transformar cada una de las entradas de 512 elementos a un espacio de menor dimensión, y el decodificador, idéntico al anterior pero con las capas en orden inverso. De este modo, la salida del primero constituye la entrada del segundo.

Al partir de un vector de 512 características, un tamaño de entrada relativamente pequeño, se utilizarán capas de tipo totalmente conectadas o *fully connected*. El número de capas y neuronas de estas depende de la reducción que se desee realizar, considerándose una buena aproximación reducciones múltiplo de 2, empleando una capa por cada una de estas reducciones.

El aprendizaje del modelo se realiza buscando que la entrada original sea lo más similar posible a la salida, es decir, se desea que a partir de las características del espacio latente el decodificador sea capaz de reconstruir la entrada. Para ello, la función de coste empleada es el error cuadrático medio, utilizando la suma como reducción.

$$MSE = \frac{1}{K}\sum_{j=1}^{K}\sum_{i=1}^{N}(Salida_{ji} - Entrada_{ji})^2 \quad (8)$$

Esta se expresa en la ecuación 8 donde N representa el número de elementos de cada vector de características, 512, y K el número total de ejemplos utilizados.

Evidentemente, como durante la etapa de predicción se desea reducir la dimensionalidad de los *embeddings* se utilizará únicamente el codificador, aunque para el entrenamiento se requiere de ambas partes.

3.3.2. *Autoencoder* Variacional. Los *autoencoder* variacionales [16] son una alternativa a los estándar, disponiendo de algunas características que hacen a este modelo interesante para el problema. Este es un modelo generativo, modelos con gran auge reciente gracias a las posibilidades que ofrecen. Es por ello que en la literatura hay quien entiende que no deben considerarse *autoencoders* como tal ya que, aunque la estructura es similar, el funcionamiento es diferente.

Uno de los problemas que puede aparecer con los *autoencoders* estándar es que en ocasiones el espacio latente puede no ser continuo, de modo que ciertos ejemplos no utilizados en entrenamiento podrían ser distribuidos en el espacio sin necesidad de estar cercanos a otros que sí lo estaban en el espacio original. Este aspecto se soluciona con los variacionales los cuales buscan no sólo reconstruir la entrada a partir del espacio latente, sino también que el espacio latente siga una determinada distribución. Al forzar al espacio latente a seguir una determinada distribución este será continuo y acotado, incluso para ejemplos no aprendidos por el modelo.

Para conseguir que el espacio latente siga la distribución deseada, el codificador se formula como un modelo probabilístico encargado de mapear la entrada a los componentes de una distribución, la media y la desviación típica, tal y como se ilustra en la figura 8. Así, el codificador se define como $q(z|x)$, donde x representa la observación o entrada y z un ejemplo del espacio latente, que toma como entrada una observación y genera como salida un conjunto de parámetros que especifican la distribución del espacio latente. Aunque la distribución se describe a través de la media y la varianza, normalmente, se suele utilizar la log-varianza ya que proporciona una mayor estabilidad numérica. El decodificador se define como $p(x|z)$ que toma un ejemplo del espacio latente, z, como entrada y busca regenerar la entrada original x.

Uno de los problemas que presenta este modelo, basado también en redes neuronales, es que el espacio latente se proporciona en términos de una distribución a través de la media y la varianza, pero no en forma de vectores en el nuevo espacio construido. Esto resulta en un cuello de botella porque la retropropagación del error, técnica utilizada para el entrenamiento y aprendizaje del modelo, no puede fluir a través de la red. Para

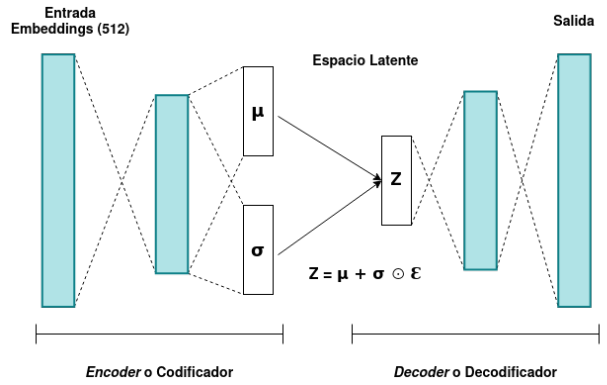

Figura 8: *Autoencoder* variacional.

su utilización se requiere de un único punto de partida desde el cual se reparte el error entre los componentes anteriores, situación no dada en este caso, al disponer de dos elementos de salida en el decodificador.

Esto lleva a utilizar lo que se conoce como *reparameterization trick* o truco de reparametrización, presentado en la siguiente ecuación.

$$z = \mu + \sigma \odot \epsilon \tag{9}$$

Esta técnica busca aproximar z mediante los parámetros conocidos donde μ y σ representan la media y la desviación estándar de la distribución. El término ϵ puede ser pensado como un ruido aleatorio usado para mantener la estocasticidad de z. Se generará ϵ a partir de una distribución normal estándar. Ahora si, se puede utilizar la retropropagación del error a través de μ y σ en el codificador.

En cuanto a la arquitectura de la red, se utilizarán capas totalmente conectadas buscando con cada una de ellas la reducción a la mitad de la dimensionalidad. La principal diferencia con respecto a la arquitectura estándar es que la última capa del codificador está dividida en dos, una de ellas para recoger la media y otra la log-varianza de la distribución.

Por último, es necesario revisar la función de coste a utilizar en el aprendizaje del modelo. Por una parte, se desea que la salida de la red sea lo más parecida a la entrada, al igual que sucedía con el *autoencoder* estándar. Por otra, se busca que el espacio latente siga una distribución Gaussiana, con cierto grado de libertad, propia del espacio de características original donde *embeddings* de la misma identidad se encontraban cercanos en el espacio y aquellos con identidades diferentes separados entre sí.

Así, la función de coste es una función relativamente compleja, donde el primero de los componentes es el error cuadrático medio (MSE), funcionando exactamente igual que para la variante estándar del modelo, y el segundo componente es la divergencia de Kullback-Leibler (DKL). Como la primera de las componentes es ya conocida, se expondrá la segunda de ellas.

La divergencia de Kullback-Leibler permite determinar la diferencia existente entre dos distribuciones de probabilidad. Existe cierta controversia en cuanto a si se puede considerar o no una medida, ya que no es simétrica.

Es evidente que una de las distribuciones es la propia del espacio latente, la que ofrece la red, pero la otra debiera ser la distribución ideal para obtener las salidas deseadas. Como esta distribución se desconoce, se aproxima mediante una distribución centrada en el origen y de desviación 1, de ahí que se busque que el espacio latente describa una distribución Gaussiana.

$$DKL = \sum_{i=1}^{N} e^{\sigma_i} + \mu_i^2 - 1 - \sigma_i \tag{10}$$

La KL-divergencia se formula de acuerdo a la ecuación 10 donde μ y σ representan la media y la log-varianza devueltas por el codificador y N el número total de características de cada vector, que es igual al número de características deseado para el espacio latente.

La función de coste global, formulada en la ecuación 11,

$$Loss = \alpha_{MSE} \cdot MSE + \alpha_{DKL} \cdot DKL \tag{11}$$

se compone de ambos términos, los cuales pueden ser ponderados de la forma más adecuada favoreciendo el aprendizaje, normalmente buscando que ambas funciones de coste trabajen en escalas similares.

3.3.3. Integración del *Autoencoder* en la Arquitectura. Cualquiera de las dos soluciones propuestas es compatible con la arquitectura. El proceso de integración consiste en, para cada una de las detecciones que supera el umbral de confianza, reducir la dimensionalidad de los *embeddings* al tamaño deseado haciendo pasar estos por el codificador de alguno de los dos modelos propuestos.

Aunque los *autoencoders* han sido pensados para procesar los *embeddings* de forma individual, ya que el número de propuestas de detección aceptadas es variable y no conocido de antemano, se puede realizar procesamiento por lotes favoreciendo así la escalabilidad y reduciendo el coste computacional de dicho componente.

Una vez se dispone de los *embeddings* en un espacio latente de menor dimensionalidad, se procedería a la asociación de datos, siguiendo el curso normal propuesto en el *framework* o marco de asociación.

4. Experimentación

4.1. Detalles de implementación

La arquitectura presentada dispone de múltiples hiperparámetros los cuales deben ser ajustados para optimizar el rendimiento de la red de detección con *embeddings*.

4.1.1. Diseño *Anchors*. El detector aprende a delimitar los objetos de interés a partir de los *anchors*, es por ello, que el aspecto de estos, relación entre ancho y alto resulta crucial. Este viene determinado por la clase del objeto de interés. Normalmente, existen dos opciones o bien el entrenamiento de un sistema orientado a la detección de objetos de múltiples clases o bien centrado en alguna específica, buscando en este último caso mayor precisión y calidad en la detección de un tipo concreto.

Este trabajo ha considerado personas como obejto de interés, por lo que los *anchors* se han definido de

acuerdo al aspecto de estas. Así, por cada escala se utilizan 4 *anchors*, resultando en un total de 12, con un ratio de aspecto 1:3, el que mejor define el aspecto de las personas.

4.1.2. Umbral de Confianza en Detección. Junto con cada detección la red proporciona un valor de confianza, en el intervalo $[0, 1]$, que determina la probabilidad de que se trate de un objeto de interés. Con el fin de evitar falsos positivos, se ha establecido un umbral a partir del cual una detección es descartada y no utilizada en la asociación de datos. Este umbral se ha establecido en 0,5, por mantener un equilibrio entre falsos positivos y falsos negativos.

4.1.3. Tiempo de Vida Máximo *Tracks*. Este hiperparámetro suele estar ligado a la tasa de reproducción, de modo que se define como el tiempo durante el cual se considera que un objeto puede volver a aparecer. En la experimentación realizada parece apropiado un valor de un segundo, ya que, por ejemplo, para un vídeo a 30 FPS (Fotogramas por Segundo) si no se consigue asociar ninguna nueva detección a un *track* durante 30 fotogramas, muy posiblemente el objeto haya abandonado la escena. Dicho hiperparámetro depende en gran medida del problema que se desea abordar y como de importante es que, en caso de que un objeto reaparezca tiempo después se considere o no como una nueva identidad.

4.1.4. Configuración *Autoencoders*. Aunque para la construcción de los *autoencoders* se tomó la decisión de diseño de utilizar capas totalmente conectadas o *fully connected*, existen ciertos parámetros como son las funciones de activación con las que se ha experimentado para ver cuál se ajusta mejor a las necesidades, lo que en términos de rendimiento significa buscar la función de activación que favorece el aprendizaje de la red minimizando el resultado de la función de coste. Las propuestas iniciales han sido seleccionadas teniendo en cuenta el rango de valores tomados por los *embeddings*, [-1,1], lo que ha permitido descartar ciertas funciones de activación sin necesidad de experimentación.

Así, las configuraciones probadas han sido las siguientes:

- **C1**. Utilización de *LeakyRelu* como función de activación a excepción de la última de las capas del decodificador, donde se utiliza la función *Tahn*.
- **C2**. Utilización de *LeakyRelu* como función de activación a excepción de la última de las capas del decodificador, donde se utiliza la función identidad.
- **C3**. Utilización de *LeakyRelu* como función de activación en todas las capas de la red.
- **C4**. Utilización de *Tahn* como función de activación en todas las capas de la red.
- **C5**. Configuración sólo utilizada en el *autoencoder* variacional. Utilización de *Tanh* como función de activación en todas las capas de la red, salvo la capa de salida del codificador que utiliza una función identidad.
- **C6**. Configuración sólo utilizada en el *autoencoder* variacional, consiste en la utilización de *Tanh* como activación en todas las capas, salvo en la última del codificador donde para la media se utiliza la identidad como función de activación y para la varianza la función *Softplus*.

De las 4 configuraciones disponibles para el *autoencoder* estándar la que mejor rendimiento ofreció fue la C4. En lo que al variacional respecta, tanto la configuración C4 como la C6 ofrecieron un rendimiento destacado sobre el resto de configuraciones, utilizándose en última instancia la C4 por su rendimiento levemente superior.

4.1.5. Entrenamiento *Autoencoders*. Para el entrenamiento de los *autoencoder*, tanto estándar como variacional, se han utilizado alrededor de 300 épocas. Inicialmente, se ha definido como tasa de aprendizaje 10^{-4} para una reducción a *embeddings* de entre 2 y 16 características y 10^{-3} para una reducción a *embeddings* de entre 32 y 256 características. En ambos casos, se ha decrementado a la mitad esta tasa de aprendizaje en las épocas 30 y 150 y una décima parte en la época 70.

4.2. *Dataset Embeddings*

Para poder emplear los *autoencoders* como parte del sistema previamente deben entrenarse y validarse dichos modelos. Para tal fin se requiere de datos adecuados y adaptados para ello. Dado que no existen bases de datos de *embeddings*, ya que se trata de una información bastante específica, ha sido necesaria su generación.

Se han utilizado tres conjuntos de datos centrados en la detección de personas, los *datasets* Caltech[8] y PRW [29] para entrenamiento y el *dataset* MOT-16 para validar los modelos. Los diferentes fotogramas han sido procesados mediante el sistema, utilizando el *framework* propuesto sin la utilización del *autoencoder*, lo que ha permitido obtener los *embeddings* asociados a los objetos identificados. Como la calidad de los datos obtenidos depende de la calidad del sistema, fue necesario realizar un filtrado mediante la utilización de los respectivos *groundtruths*. De forma más precisa, se utilizó el método Húngaro para maximizar el solape entre las *bounding boxes* obtenidas como hipótesis y las propias del *groundtruth* de modo que se eliminen aquellas hipótesis que no se correspondan con objetos de interés y se consideren errores del sistema.

El resultado de este proceso son *embeddings* fiables, contrastados sobre el *groundtruth*, que pueden ser utilizados para entrenar y probar o validar los modelos.

4.3. Métricas

Con la aparición del MOT (Multiple Object Tracking)[19] se establecieron unas métricas de referencia que permiten conocer con exactitud la calidad de los resultados.

Existen dos métricas que sirven a modo de referencia de la calidad de un sistema de *tracking*, lo que se conoce como MOTA y MOTP. Estas son muy útiles ya que de forma rápida permiten determinar donde se encuentra el sistema frente a otros, así como determinar la calidad del sistema, gracias al empleo de otras métricas más sencillas.

La primera de ellas, en la ecuación 12, *Multiple Object Tracking Accuracy* permite determinar la calidad

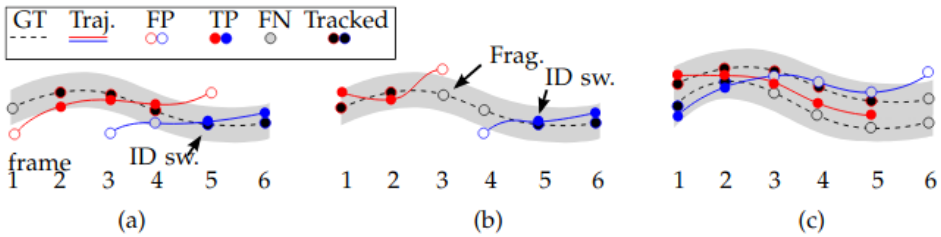

Figura 9: Situaciones de asignación de *trackers* a objetivos. (a) Un Cambio de ID (IDS) se produce cuando existe un intercambio entre el *track* rojo, previamente asignado, por el azul. (b) Lo que se conoce como *Track Fragmentation* (FM) se produce en el fotograma 3, porque el objeto es identificado en los fotogramas 1 y 2, posteriormente, la identificación se interrumpe, hasta que es reidentificado con una nueva identidad, marcada en azul, lo que genera a su vez un Cambio de Identidad (IDS). (c) Se puede ver como inicialmente, existen dos *tracks* correctamente definidos por las líneas roja y azul, hasta que en el fotograma 5, ambas pasan a describir la trayectoria del mismo objeto, lo que ocasione a su vez un Falso Positivo (FP), ya que debe descartarse una de las dos hipótesis, aquella con menor solape con el *track*, y un Falso Negativo (FN) en el fotograma 4 ya que la trayectoria descrita por el objeto inferior en el gráfico deja de tener asociada alguna hipótesis (Imagen extraída de A. Milan et al. [19]).

del sistema en términos de detección y definición de las identidades de los objetos.

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDS_t)}{\sum_t GT_t} \tag{12}$$

Se compone de 3 métricas más específicas. FN o Falsos Negativos, hace referencia a objetivos no reconocidos por ninguna hipótesis. Por su parte, FP o Falsos Positivos, se refiere a la situación contraria, la presentación de una hipótesis en una determinada región en la cual no existe objeto alguno. La métrica IDS o Intercambio de Identificador (*IDentifier Switch*) se encarga de recoger aquellas situaciones en las que a una misma identidad, de acuerdo al *groundtruth*, se le asignan en diferentes momentos de tiempo hipótesis con identidades diferentes. El término GT o *Grountruth*, hace referencia al número real de objetos existente. Como es de esperar, el símbolo t se refiere a cada uno de los fotogramas que componen el vídeo sobre el que se desea evaluar el sistema. Esta métrica toma valores entre $(-\infty, 100]$, siendo 100 la puntuación máxima.

La segunda de las métricas, en la ecuación 13, *Multiple Object Tracking Precision*, permite conocer la precisión de la localización de los objetos, por lo que realmente aporta más información sobre la calidad de las detecciones que sobre *tracking*, de ahí que ambas sean totalmente representativas y complementarias.

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t C_t} \tag{13}$$

En ella, $d_{t,i}$ representa el solape o área de la intersección del objeto i en el fotograma t con respecto al *groundtruth*. Cabe recordar que en *tracking*, los objetos serán delimitados mediante rectángulos, buscando contener todos los píxeles pertenecientes al objeto con el mayor ajuste posible. En el caso de C_t, esto representa el número de asociaciones para el fotograma t entre las hipótesis y el *groundtruth*. El dominio de esta métrica es $[0, 100]$, donde 100 representaría un solape perfecto entre las hipótesis y el *groundtruth*.

Tabla 1: *Autoencoder* Estándar y *framework* de partida para una resolución de 576×320 píxeles.

Tamaño *Embeddings*	FP	FN	IDS	MOTA	MOTP
2	7007	32544	2392	62,01 %	**21,91** %
4	7071	31902	1555	63,30 %	21,87 %
8	6920	32048	1418	63,42 %	21,89 %
16	6947	31958	1329	63,56 %	21,90 %
32	6924	**31870**	**1201**	**63,77 %**	21,85 %
64	6919	31945	1222	63,70 %	21,84 %
128	6917	32015	1287	63,57 %	21,89 %
256	6879	32137	1398	63.40 %	21,85 %
512	**6846**	32177	1330	63,45 %	21,86 %

Tabla 2: *Autoencoder* Variacional y *framework* de partida para una resolución de 576×320 píxeles.

Tamaño *Embeddings*	FP	FN	IDS	MOTA	MOTP
2	6251	34863	5223	58,03 %	21,81 %
4	6158	35119	4926	58,15 %	21,85 %
8	5896	35445	3881	59,04 %	21,59 %
16	5801	35367	3042	59,95 %	21,62 %
32	**5741**	35225	2823	60,34 %	21,61 %
64	5750	35315	2790	60,28 %	21,60 %
128	5750	35315	2790	60,28 %	21,60 %
256	5750	35315	2790	60,28 %	21,61 %
512	6846	**32177**	**1330**	**63,45 %**	**21,86 %**

4.4. Resultados

En esta sección se presentan los resultados obtenidos por el sistema, para las métricas de referencia, tanto para el *autoencoder* estándar como para el variacional, focalizando el interés en aquellos que ponen en relieve la ganancia obtenida. Para la validación del sistema se ha utilizado el *dataset* MOT-16, conjunto de vídeos de referencia en *tracking* de personas, requisito de validación indispensable para cualquier sistema que quiera presentar resultados relevantes en este campo.

El primero paso ha sido determinar si realmente la utilización de *embeddings* de menor tamaño era favorable en términos de MOTA y MOTP, las dos métricas más representativas en *tracking*. Los resultados obtenidos se muestran en las tablas 1 y 2 para la reducción mediante *autoencoder* estándar y variacional respectivamente.

Antes de analizar los resultados debe tenerse

en cuenta que, aunque el sistema admite vídeos o fotogramas con cualquier resolución, internamente, la red de extracción de características trabaja con 3 resoluciones de 576×320, 864×480 y 1088×608 píxeles. Los resultados de la tabla anterior reflejan los obtenidos para la resolución de 576×320 píxeles. Dado que en este caso particular se desea determinar si la utilización de *embeddings* con menos características supone una mejora, basta con visualizar los resultados para una de las tres resoluciones puesto que se busca una comparativa relativa con la utilización de *embeddings* de 512 características. En cualquier caso, los resultados son extensibles a las resoluciones restantes.

Así, estos arrojan resultados muy positivos para la reducción efectuada mediante el *autoencoder* estándar y menos en relación al variacional. Para el primero, tabla 1, se puede ver como el MOTA es superior al de referencia, *embeddings* con un tamaño de 512, para tamaños de 128, 64, 32 y 16. Esto pone de manifiesto que la reducción de la dimensionalidad mediante este modelo favorece la asociación de detecciones, por lo menos hasta 16 características. Aunque la ganancia en MOTA es del 0,3 % únicamente, se reducen un 9,7 % el número de Cambios de Identidad (IDS) y un 0,95 % el número de Falsos Negativos (FN), lo que es muy satisfactorio especialmente en cuanto a IDS, más teniendo en cuenta que el aumento de Falsos Positivos (FP) apenas alcanza el 1 %.

El resultado no es igual de satisfactorio para la reducción realizada mediante *autoencoders* variacionales, tabla 2. En este caso, la reducción de la dimensionalidad supone una pérdida de rendimiento en términos de MOTA, lo que denota, teniendo en cuenta el rendimiento del *autoencoder* estándar, que la complejidad adicional introducida por el modelo buscando no sólo reconstruir la entrada sino limitando el espacio latente mediante una distribución genera un efecto negativo en los resultados.

En cuanto al MOTP, aclarar que este no resulta tan interesante, puesto que realmente la mejora es en la asociación de datos no a nivel de detección, con lo que los cambios en este son mínimos teniendo en cuenta que el detector incorporado en el sistema no ha sido objeto de mejora.

Las métricas anteriores representan el rendimiento del sistema utilizando el *framework* presentado en la sección de arquitectura. Sin embargo, en el sistema presentado no sólo se utilizan los *embeddings* para asociar detecciones en los sucesivos fotogramas sino que también se utiliza IOU (*Intersection Over Union*), por lo que realmente es difícil conocer el impacto real de los nuevos *embeddings* propuestos.

Por este motivo, se ha reformulado el *framework* o marco de asociación de datos eliminando IOU y utilizando únicamente los *embeddings* para realizar la asociación de datos. De este modo, la asociación depende únicamente de aquello cuyo rendimiento se desea medir.

Así, los resultados con esta nueva forma de proceder se muestran en las tablas 3, 4 y 5. En ellas, se presentan los resultados para las 3 resoluciones con las que se puede trabajar para el *autoencoder* estándar, aquel que ofreció mejor rendimiento, tal y como se vio anteriormente.

Tabla 3: *Autoencoder* Estándar y *framework* basado en únicamente *embeddings* para una resolución de 576×320 píxeles.

Tamaño *Embeddings*	FP	FN	IDS	MOTA	MOTP
2	6945	33999	2945	60,25 %	**22,00 %**
4	7071	**31912**	1559	63,28 %	21,86 %
8	6889	32567	1693	62,73 %	21,86 %
16	6913	32362	1520	63,05 %	21,87 %
32	6903	31995	**1282**	**63,60 %**	21,84 %
64	6883	32094	1315	63,50 %	21,83 %
128	6863	32609	1600	62,80 %	21,85 %
256	6767	33511	1911	61,79 %	21,85 %
512	**6704**	33765	1879	61,64 %	21,85 %

Tabla 4: *Autoencoder* Estándar y *framework* basado en únicamente *embeddings* para una resolución de 864×480 píxeles.

Tamaño *Embeddings*	FP	FN	IDS	MOTA	MOTP
2	6625	25031	1779	69,72 %	**19,93 %**
4	6648	**24900**	1591	69,98 %	19,91 %
8	6652	24905	1445	70,11 %	19,90 %
16	6607	24924	1374	70,20 %	**19,93 %**
32	6679	24903	1237	**70,27 %**	19,89 %
64	6678	24933	**1218**	**70,27 %**	19,88 %
128	6461	25293	1398	69,97 %	19,83 %
256	6384	25965	1670	69,19 %	19,85 %
512	**6347**	26259	1814	68,82 %	19,84 %

De este modo, evaluando puramente los *embeddings* se puede ver como la mejora es muy notable, de un 2 % de MOTA, una mejora considerable teniendo en cuenta que se trata de un sistema de *tracking* estado del arte. Salvo para 2 características, donde se produce un descenso en el rendimiento en algunas resoluciones, en los restantes, la reformulación y reducción del tamaño de los *embeddings* es muy favorable. Con esta nueva codificación se puede ver como aunque aumentan levemente los Falsos Positivos (FP) en un 4 % de media para las tres resoluciones, la reducción experimentada en cuanto a Falsos Negativos (FN) y Cambios de Identificador (IDS) es importante ascendiendo a un 5,8 % y a un 32 % respectivamente, con lo que el rendimiento general del sistema aumenta de forma muy satisfactoria, como refleja el MOTA, especialmente manteniendo la identidad de los objetos identificados a los largo del vídeo.

Aunque esta reformulación de los *embeddings* se muestra muy positiva en casi todas las reducciones, ya que no sólo se reduce la dimensionalidad sino que se elimina el ruido existente y se enriquecen las características diferenciadoras, alcanza su mayor rendimiento para una reducción a 64 y 32 características. Esto se evidencia de forma gráfica en la figura 10.

Tabla 5: *Autoencoder* Estándar y *framework* basado únicamente en *embeddings* para una resolución de 1088×608 píxeles.

Tamaño *Embeddings*	FP	FN	IDS	MOTA	MOTP
2	6721	22244	2252	71,72 %	**18,50 %**
4	6855	**21459**	1451	73,04 %	18,41 %
8	6754	21739	1508	72,83 %	18,34 %
16	6698	21778	1434	72,91 %	18,32 %
32	6781	21497	**1183**	**73,32 %**	18,32 %
64	6733	21573	1189	73,29 %	18,33 %
128	6647	21829	1340	72,99 %	18,29 %
256	6513	22397	1566	72,40 %	18,25 %
512	**6493**	23098	1749	71,61 %	18,25 %

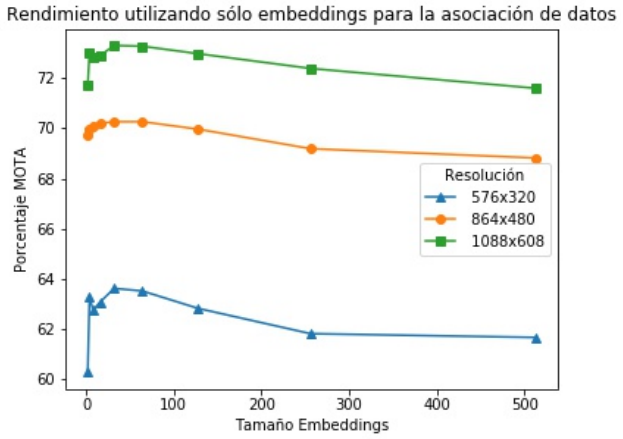

Figura 10: Representación de la métrica MOTA con diferentes tamaños de *embeddings* y resoluciones.

Tabla 6: Tasa de procesamiento media alcanzada por el sistema en Fotogramas por Segundo para las diferentes resoluciones.

Tipo de configuración	576 × 320	864 × 480	1088 × 608
JDE y autoencoder (embeddings de 2 características)	43,33 FPS	33,82 FPS	27,17 FPS
JDE y autoencoder (embeddings de 4 características)	45,17 FPS	33,51 FPS	27,54 FPS
JDE y autoencoder (embeddings de 8 características)	44.65 FPS	33.34 FPS	27.48 FPS
JDE y autoencoder (embeddings de 16 características)	43.70 FPS	33,48 FPS	27,57 FPS
JDE y autoencoder (embeddings de 32 características)	44,30 FPS	33,64 FPS	27,60 FPS
JDE y autoencoder (embeddings de 64 características)	44,62 FPS	33,65 FPS	27,30 FPS
JDE y autoencoder (embeddings de 128 características)	44,54 FPS	33,02 FPS	27,28 FPS
JDE y autoencoder (embeddings de 256 características)	44,44 FPS	33,20 FPS	27,34 FPS
JDE (embeddings de 512 características)	44,31 FPS	33,88 FPS	27,25 FPS

Los mejores resultados en todos los casos se consiguen para la mayor resolución de imagen, puesto que se dispone de un mayor número de características y con ello el valor semántico que se puede extraer de estas también es mayor. El problema está en la velocidad del sistema y es que al trabajar con mayor resolución el coste computacional es mayor lo que repercute directamente en el tiempo de procesamiento del sistema. Así, en función de las capacidades de cómputo de la máquina en que se despliega el sistema en muchas ocasiones se sacrifica rendimiento por velocidad para funcionar en tiempo real, siempre en un balance razonable.

Esto pone de manifiesto la necesidad de mostrar que el sistema no sólo mejora en cuanto a rendimiento puramente de *tracking*, sino que la reformulación de los *embeddings* no tiene un impacto negativo significativo en cuanto a tiempo.

Por ello, se han medido los tiempos para las diferentes resoluciones y tamaños de *embeddings* tomando como referencia los tiempos obtenidos para 512, configuración del sistema que no utiliza *autoencoders*.

En la tabla 6 se da muestra de ello. En ella se presenta la velocidad media de ejecución o procesamiento medida sobre el MOT-16 *train* utilizando una GPU Nvidia Tesla V100S 32GB.

Los resultados evidencian que el tiempo imputable a la incorporación de los *autoencoders* como parte del sistema es despreciable, poniendo en valor el 2 % ganado en MOTA en lo que a la asociación de datos mediante *embeddings* se refiere. Gran parte del éxito en cuanto a tiempo de procesamiento se debe a la configuración adoptada para la integración del *autoenoder* como parte del sistema que hace que sea totalmente escalable, al permitir el procesamiento conjunto de todos los *embeddings* de cada fotograma, lo que evita el impacto negativo que podría tener la presencia de un número elevado de hipótesis sobre el tiempo de ejecución.

5. Conclusiones

En este documento se propone una arquitectura en la que se integra un *autoencoder* junto con un detector JDE, con la finalidad de reducir la dimensionalidad de los *embeddings* utilizados en la asociación de datos. Esta reformulación de los *embeddings*, utilizando un menor número de características, favorece la asociación de datos lo que permite aumentar el MOTA obtenido por el sistema en un 2 %, en términos de asociación mediante *embeddings* y un 0,3 % a nivel general, destacando sobremanera en la reducción de los cambios de identidad. Todo ello se consigue sin perder velocidad de procesamiento, permitiendo funcionar al sistema en tiempo real. Como trabajo futuro se plantea el estudio de la adaptación de la arquitectura para sistemas embebidos en los que los recursos computacionales son mucho más limitados y donde cobra más importancia si cabe las mejoras en rendimiento dada las limitaciones establecidas para el funcionamiento en tiempo real.

Referencias

[1] Olivier Barnich and Marc Van Droogenbroeck. Vibe: A universal background subtraction algorithm for video sequences. *IEEE Transactions on Image processing*, 20(6):1709–1724, 2010.

[2] Luca Bertinetto et al. Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pages 850–865. Springer, 2016.

[3] Goutam Bhat et al. Learning discriminative model prediction for tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6182–6191, 2019.

[4] David S Bolme et al. Visual object tracking using adaptive correlation filters. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2544–2550. IEEE, 2010.

[5] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[6] Martin Danelljan et al. Atom: Accurate tracking by overlap maximization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4660–4669, 2019.

[7] P. Dendorfer et al. MOT20: A benchmark for multi object tracking in crowded scenes. *arXiv:2003.09003[cs]*, March 2020. arXiv: 2003.09003.

[8] P. Dollar et al. Pedestrian detection: A benchmark. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 304–311, 2009.

[9] Jerome Friedman et al. The elements of statistical learning. volume 1. Springer, 2009.

[10] Ross Girshick. Fast R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[11] Ross Girshick et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[12] Ian Goodfellow et al. Deep learning. pages 502–525. MIT Press, 2016. http://www.deeplearningbook.org.

[13] Martin Hofmann et al. Background segmentation with feedback: The pixel-based adaptive segmenter. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 38–43. IEEE, 2012.

[14] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[15] Alex Kendall et al. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *arXiv:1705.07115*, 2018.

[16] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114v10*, 2019.

[17] Leal-Taixé L. et al. MOTChallenge 2015: Towards a benchmark for multi-target tracking. *arXiv:1504.01942 [cs]*, April 2015. arXiv: 1504.01942.

[18] Tsung-Yi Lin et al. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[19] A. Milan et al. MOT16: A benchmark for multi-object tracking. *arXiv:1603.00831 [cs]*, March 2016. arXiv: 1603.00831.

[20] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[21] Joseph Redmon et al. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[22] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[23] Shaoqing Ren et al. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[24] Pierre-Luc St-Charles et al. Subsense: A universal change detection method with local adaptive sensitivity. *IEEE Transactions on Image Processing*, 24(1):359–373, 2014.

[25] Richard Szeliski. Computer vision: Algorithms and applications. Springer, 2010.

[26] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I. IEEE, 2001.

[27] Paul Viola and Michael J Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[28] Zhongdao Wang et al. Towards real-time multi-object tracking. *arXiv preprint arXiv:1909.12605*, 2019.

[29] Liang Zheng et al. Person re-identification in the wild. *arXiv:1604.02531*, 2017.

[30] Linyu Zheng et al. Learning features with differentiable closed-form solver for tracking. *arXiv preprint arXiv:1906.10414*, 2019.