Escola Tècnica Superior d'Enginyeria ETSE-UV

MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS

TRABAJO DE FIN DE MÁSTER

VISIÓN ARTIFICIAL APLICADA A LA DETECCIÓN DE PERSONAS

AUTOR:
HÉCTOR LÓPEZ PÉREZ

TUTORES:
VALERO LAPARRA PEREZ-MUELAS/ JORDI MUÑOZ MARI

09, 2022

Presidente del Tribunal Evaluador / *President del tribunal avaluador* via **CCA del Máster** / *CCA del Màster*

MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS

TRABAJO DE FIN DE MÁSTER

VISIÓN ARTIFICIAL APLICADA A LA DETECCIÓN DE PERSONAS

AUTOR:
HÉCTOR LÓPEZ PÉREZ

TUTORES:
VALERO LAPARRA PEREZ-MUELAS/ JORDI MUÑOZ MARI

TRIBUNAL:

PRESIDENTE/A: VOCAL 1:

Resumen

El objetivo de este proyecto es realizar una revisión teórica en el marco de la visión artificial con foco en la detección de personas, teniendo esta como fin último aplicar dicho conocimiento en el desarrollo de un modelo de detección e identificación de personas en secuencias de vídeo.

Para llevar a cabo esto se ha realizado un estudio tanto desde el punto de vista teórico como práctico de los modelos, herramientas y metodologías punteras, desde el ámbito de aplicación en imágenes hasta el de vídeo.

Por último, para el desarrollo del modelo decidimos basarnos en una herramienta puntera de detección en imágenes y, siguiendo metodologías vistas, implementar sobre esta la detección en vídeo, para analizar de primera mano las capacidades inherentes a las distintas aproximaciones. [1]

Palabras clave: Machine Learning, Visión artificial, detección, tracking, red recurrente, red convolucional.

[1] El código y los vídeos generados en este proyecto están disponibles clicando aquí: Código del proyecto

Abstract

The aim of this project is to perform a theorical review within the framework of Artificial Vision with a focus on people detection, with the ultimate goal of applying this knowledge to the development of a model for the detection and identification of people in video sequences.

In order to do this, we have carried out a theoretical and practical study of leading models, tools and metadologies, from the field of images to that of video.

Finally, for the development of the model, we decided to base it on a cutting-edge image detection tool and, following the methodologies we have seen, to implemente video detection on top of it, in order to analyse first-hand the capabilities inherent to the different approaches. [2]

Key Words: Machine Learning, Artificial vision, detection, tracking, recurrent network, convolutional network.

[2]The code and videos generated in this project are available by clicking here: Project code

Índice

1. Introducción

La visión artificial es un campo en desarrollo dentro de la inteligencia artificial, el cual se ocupa de dotar a los ordenadores de la capacidad de extraer información de imágenes o vídeos. La manera de proceder de los algoritmos de este campo busca asimilarse a la visión humana. Uno de los principales problemas de este campo actualmente, y que analizaremos en este trabajo se trata de la detección de objetos.

La detección de objetos consiste en la identificación de estos (personas, animales, vehículos, etc.) así como su posición dentro de una imagen o vídeo. Hay distintos métodos aplicables para conseguir este objetivo, siendo uno de ellos el entrenamiento previo de una red neuronal para detectar dichos objetos. Existen dos aproximaciones similares a este problema, la segmentación semántica y la segmentación de instancias. La primera asocia cada píxel de la imagen a una clase (objeto), mientras que la segunda identifica distintos elementos de la misma clase por separado.

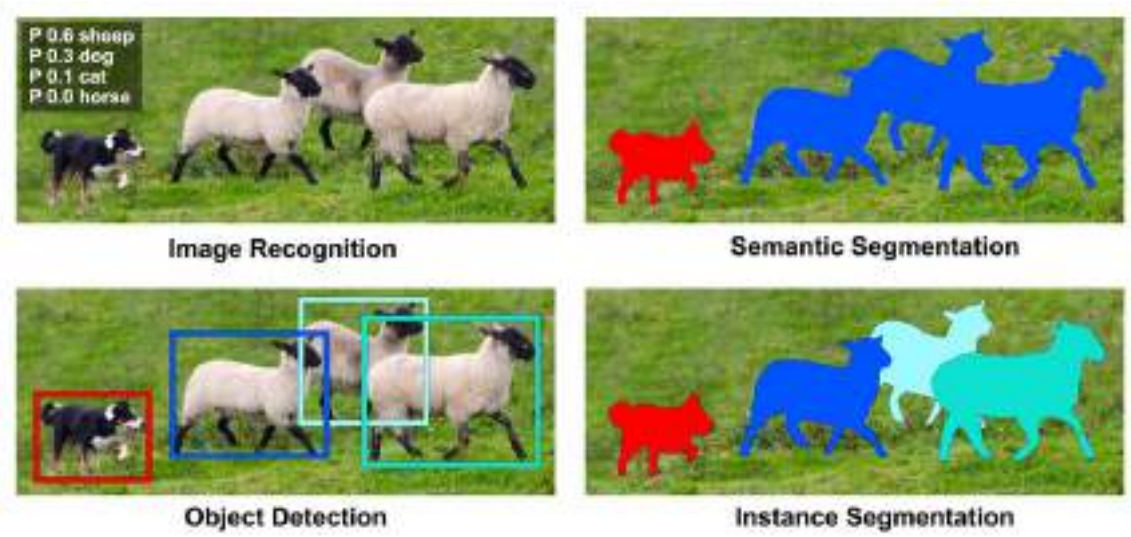


Figura 1: Ejemplo de segmentación semántica y de instancias para una misma imagen.

Vemos en la figura 1 un ejemplo de la diferencia entre segmentación semántica, la cual no distingue entre elementos de un mismo objeto, frente a la segmentación de instancias, la cual los identifica por separado.

Desde la conducción autónoma hasta procesos de control industriales pasando por tareas de vigilancia, esta 'nueva' tecnología muestra un papel creciente en la industria y la sociedad, pudiendo suponer una gran mejora en materia de seguridad, prevención de accidentes y automatización de la producción. Por todo esto está en constante cambio y su evolución ha sido notable en los últimos años. No obstante se trata de un campo con un horizonte muy amplio todavía a la vista, sobretodo en casos de visión artificial en vídeo y en tiempo real. Debido a su complejidad se trata de un campo en el que aún hay margen para grandes avances.

El objetivo de este trabajo es realizar un repaso teórico y práctico de los fundamentos de la visión artificial. Desde las bases teóricas y métodos que la sustentan, hasta los principales algoritmos utilizados en la actualidad, y las diferentes características y utilidades de estos, con el fin de conocer sus capacidades y limitaciones.

Para reducir el ámbito de estudio nos centraremos únicamente en la detección de personas dentro de este campo. Primero ponemos nuestro foco en la detección de personas en formato imagen, para después hacer una revisión de la detección de personas en vídeo y los distintos métodos existentes.

Por último, con el fin de aportar una visión práctica y clarificar mejor el estado del arte de la 'detección' y seguimiento de personas en vídeo implementamos un par de casos de uso sencillos basándonos en métodos y herramientas punteras.

Figura 2: Esquema del proyecto.

Para tener una visión más esquemática de las partes del proyecto y su evolución mostramos en la figura 2 las distintas etapas llevadas a cabo y que podemos encontrar en este trabajo.

2. Estado del arte de la detección en imágenes

Al ser este un campo tan amplio y la gama de problemas a afrontar tan variada, existen distintas aproximaciones que pueden reducir considerablemente la complejidad. Por ejemplo, en el caso de detección para una cámara fija se puede substraer el fondo conocido de la imagen para, basándose en esta diferencia, facilitar la detección. En nuestro caso vamos a abordar el problema de la forma más general posible, sin realizar ninguna suposición previa. Conviene puntualizar que los métodos mencionados aquí son de una complejidad considerable y no se busca hacer una revisión matemática sino conceptual.

2.1. Redes Convolucionales

Actualmente en el campo del aprendizaje profundo el método más utilizado para la extracción de características en imágenes es el conocido como *Convolutional Neuronal Network* (CNN). Este se basa en la aplicación de capas convolucionales, responsables de la extracción de características, junto con capas Pooling, encargadas de reducir la dimensionalidad y con ello el coste computacional. Por último, a la salida de las redes convoluciones se le aplica un aplanado (Flatten) para obtener un vector de una única dimensión, con el cual a partir de una capa totalmente Conexa (Fully-Connected) y aplicando la función Soft-max ($s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$) se extraen las probabilidades de pertenencia a las distintas clases de estudio.

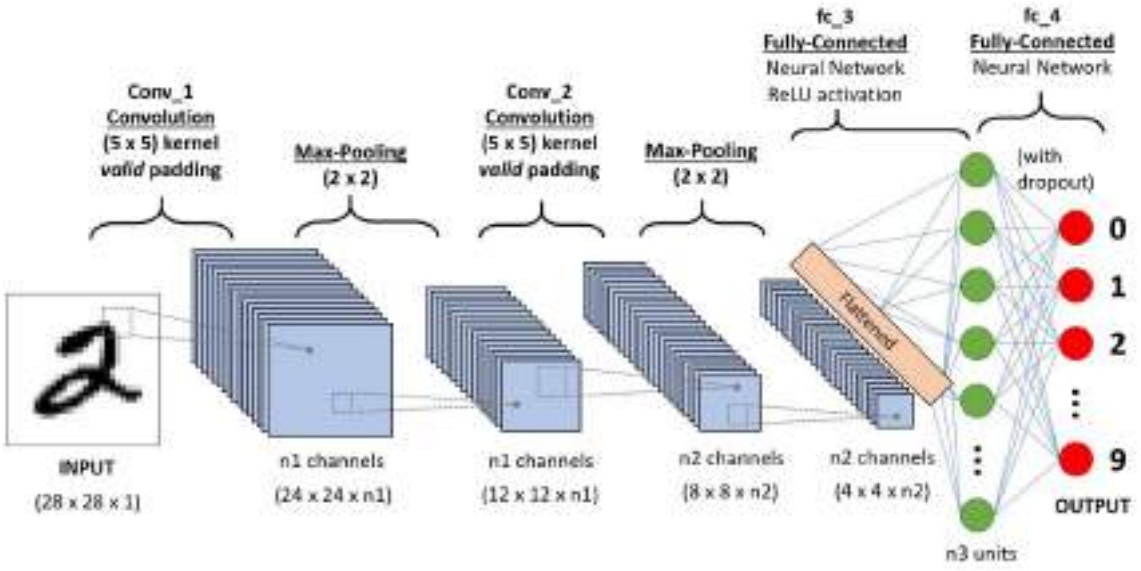


Figura 3: Esquema de las partes del modelo de red convolucional.

Vemos en la figura 3 un esquema simple de las distintas partes que intervienen en este modelo. A continuación vamos a explicar con más detalle cada una de las capas típicas (pueden darse implementaciones alternativas) de estas redes.

2.1.1. Capa convolucional

El proceso llevado a cabo en esta capa se basa en la aplicación de distintos filtros, como se muestra en la figura 4, para recalcar propiedades concretas de la imagen. Esto se hace aplicando una matriz kernel sobre cada píxel de la imagen original, pudiendo generar distintos tamaños de salida dependiendo del tipo de aplicación (valid, same o full).

- Valid: El filtro se aplica únicamente sobre píxeles de la imagen original, devolviendo una salida de dimensión menor a la original dependiendo del tamaño del filtro.

- Same: El filtro puede aplicarse sobre píxeles que pasan del margen de la imagen, los cuales se rellenan con 0's (padding) con el fin de devolver la misma dimensión que la imagen original
- Full: El filtro se aplica a cada pixel de la imagen original en todas las posiciones posibles, rellenando los márgenes exteriores con 0's. Aumenta así el tamaño de la salida dependiendo del tamaño del filtro.

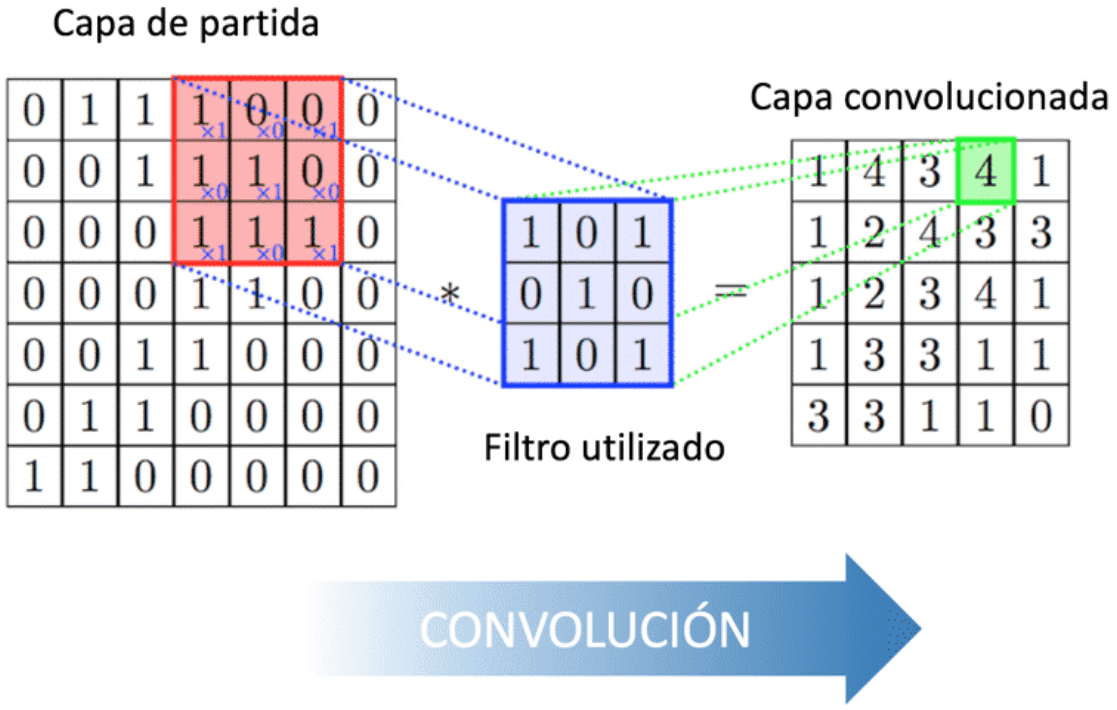


Figura 4: Ejemplo de aplicación de un filtro convolucional tipo valid.

Los pesos de estos filtros serán las variables que el modelo ajustará para minimizar el valor de una función de pérdida durante la etapa de entrenamiento.

2.1.2. Capa Pooling

Como hemos dicho antes, esta se encarga de reducir la dimensionalidad de la salida proporcionada por la capa convolucional. Para esto se divide la imagen en subregiones de píxeles en forma de cuadrado $(n \times n)$ y de cada una se extrae solamente un valor como se ve en la imagen 5. Hay distintos criterios o formas a aplicar para elegir dicho valor de submuestreo. Generalmente se usa Max-Pooling, que consiste en seleccionar el valor máximo, pero también existe el Average-Pooling en el que se toma la media o el Min-Pooling, tomando el mínimo.

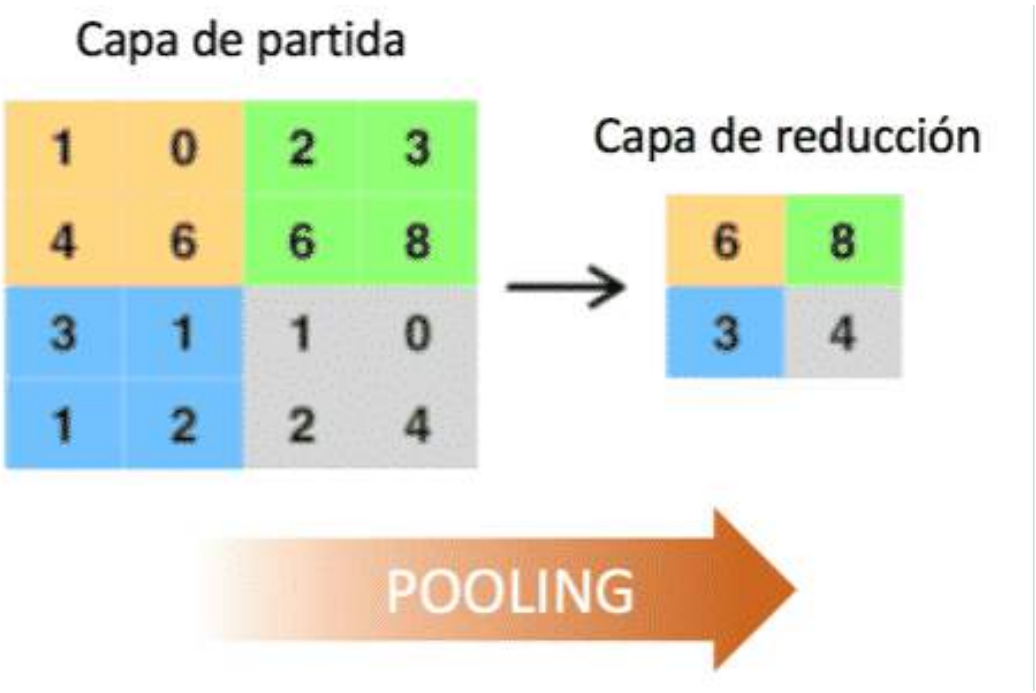


Figura 5: Ejemplo de aplicación de un filtro Max-Pooling.

2.1.3. Capas Fully-Connected

Se tratan, como su nombre indica, de capas donde todas las neuronas están conectadas a todas las neuronas de la capa siguiente, como vemos en la figura 6. Por último, a la salida de estas capas se le aplica una función Soft-Max para reescalar

los valores en el rango [0,1], dándonos la probabilidad de pertenencia del objeto a detectar a distintas clases.

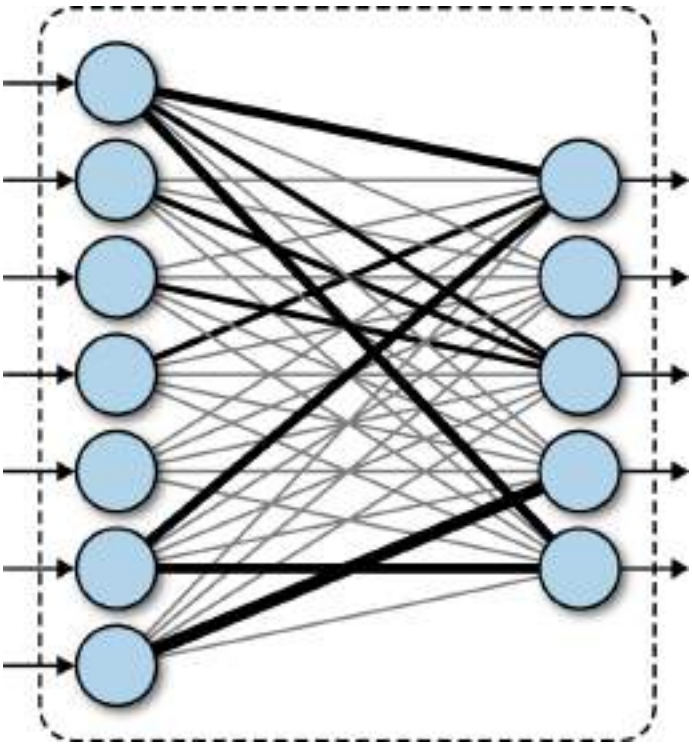


Figura 6: Ejemplo de dos capas de neuronas Fully-Connected.

Sin embargo, al realizar detección de objetos en imágenes es necesario tener información espacial de estos, por lo que el método de extracción de características necesitaba ser adaptado a este requisito. Además, una red convolucional estándar tiene una salida de tamaño fijado, mientras que para nuestro problema esta salida es variable, debido a que el número de apariciones del objeto en cuestión a detectar no está fijada. Una solución para este problema es buscar los objetos en determinadas regiones de la imagen aplicando en cada una la red CNN, pero esto podría ser computacionalmente muy costoso debido a la gran cantidad de estas. Por ello resulta necesario para la detección de objetos algún método para determinar en que subregiones conviene buscar dichos objetos (aplicar la red convolucional).

Esto lleva a la principal división de los algoritmos de detección, dando lugar a

dos grupos diferenciados según la manera de abordar dicho problema. Siguiendo este criterio, la primera aproximación que estudiaremos divide el proceso de detección de objetos en dos etapas, añadiendo a la capa básica de detección de objetos de distintas clases en las diferentes regiones una etapa previa donde se extraen las regiones potenciales de contener un objeto.

2.2. Detección en dos etapas (Two Stages): De R-CNN a Mask R-CNN

Vamos a explicar aquí los principales métodos existentes para solucionar este problema, así como la lógica que los sustenta y la evolución que han ido sufriendo conforme estos se refinaban.

2.2.1. R-CNN

Para solucionar el problema del gran número de regiones se busca dividir la imagen en zonas susceptibles de presentar objetos a detectar [1]. Para identificar dichas regiones existen distintas técnicas. Una de las más usadas es el método de búsqueda selectiva [2], el cual combina la búsqueda exhaustiva y la segmentación, seleccionando aproximadamente 2000 regiones con forma rectangular basándose en las propiedades similares de estas. Mostramos en la figura 7 a continuación un esquema básico de su funcionamiento.

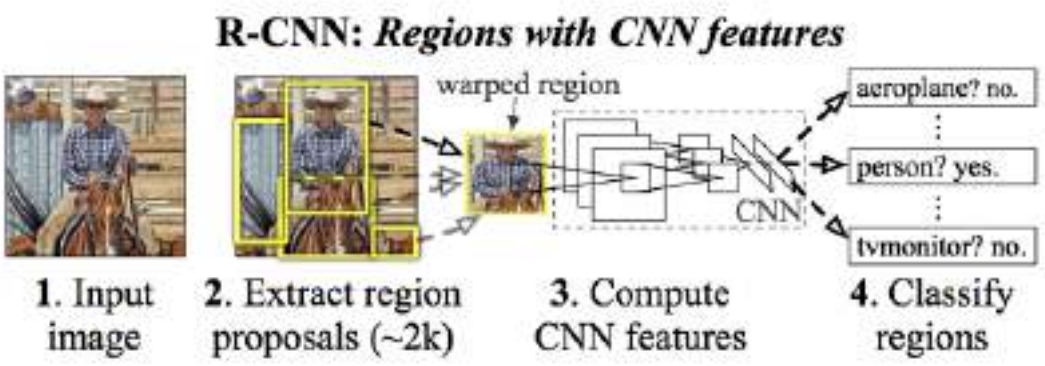


Figura 7: Proceso de segmentación y convolución en el algoritmo R-CNN [1].

La capa CNN extrae las características de las distintas regiones rectangulares (Bounding Box) y se utilizan a continuación estas como entrada a un algoritmo Support Vector Machine [3], **SVM**, para la clasificación, el cual devuelve una probabilidad de que el objeto se encuentre en dicha región. Además, para ajustar más la región en cuestión al objeto y paliar posibles errores de segmentación iniciales, se optimizan 4 parámetros de desplazamiento que indican si el ajuste al objeto mejora o no en las 4 direcciones de la caja. Sin embargo, este método requería de un tiempo de computación alto al tener que realizar la convolución de las cerca de 2000 regiones propuestas.

2.2.2. Fast R-CNN

Para solucionar este problema, su mismo desarrollador propuso un método [4] en el que toda la imagen era transformada mediante una red convolucional. Después, del mapa de atributos generado se extraen las regiones de interés (RoI) correspondientes a las regiones propuestas por el algoritmo de búsqueda selectiva. A continuación, mediante una capa RoI-Pooling se les da la misma dimensión a las distintas regiones para por último realizar la clasificación de estas y predecir así la clase y ajustar los parámetros de desplazamiento al igual que antes. Vemos estas etapas ilustradas en la figura 8

[3]Algoritmo de aprendizaje supervisado para la clasificación basado en encontrar un hiperplano donde se maximice la separación entre las clases a predecir [3]

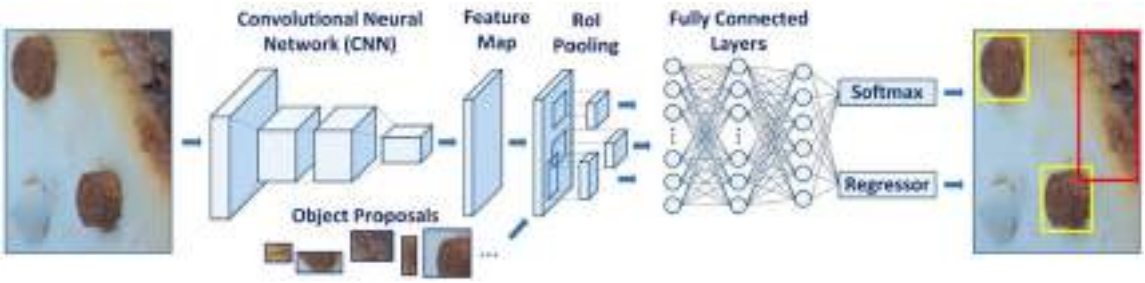


Figura 8: Proceso de segmentación y convolución en el algoritmo Fast R-CNN [4].

2.2.3. Faster R-CNN

En siguiente paso en la evolución de estos modelos fue el usar, en vez de búsqueda selectiva, una red separada para predecir las regiones propuestas directamente sobre el mapa de características dado por la capa convolucional. Después estas regiones son transformadas por una capa RoI pooling, cuya salida se usa a continuación para clasificar la imagen análogamente a los demás casos.

2.2.4. Mask R-CNN

Este modelo extiende el algoritmo de Faster R-CNN al incluir en paralelo un método para predecir la máscara del objeto, es decir, su perfil concreto dentro de la región marcada por la caja (bounding Box). [5] Esto se consigue añadiendo un término a la función de perdida que codifica K (número de clases) máscaras binarias. Vemos un ejemplo de esto en la figura 9, donde primero se localiza el objeto y aplicando la máscara binaria píxel a píxel se determinan todos los que pertenecen a este y, por tanto, su perfil.

El hecho de ser capaz de determinar el perfil exacto del objeto a detectar es clave para problemas tales como la clasificación de acciones.

Figura 9: Ejemplo del funcionamiento y de la salida de la detección del modelo Mask R-CNN [6].

2.3. Algoritmos de detección en Una Etapa (One Stage)

Los algoritmos de detección basados en dos etapas implican la necesidad de entrenar y optimizar dos componentes independientes, lo cual conlleva un mayor tiempo de procesado y dificulta el proceso de optimización, aunque por regla general los resultados muestran una precisión mayor. Una alternativa potencialmente más rápida es realizar directamente la detección de objetos sin la etapa previa de proposición de regiones (region proposal). Procedemos en este apartado a describir los principales ejemplos para este enfoque.

2.3.1. RetinaNet

Este modelo introduce principalmente las propiedades de Feature Piramidal Network (FPN), el cual explicamos con más detalle en la sección 4.1.1, permitiendo así la detección con precisión a distintas escalas, y Focal Loss [4], que permite lidiar

[4]Se trata en un algoritmo que introduce un factor modulador a la función de pérdida de entropía para focalizar el entrenamiento en los casos mal clasificados, asignando dinámicamente mayor im-

con el gran desbalance entre las clases en primer plano y en el fondo durante el entrenamiento, respecto a otros modelos similares.

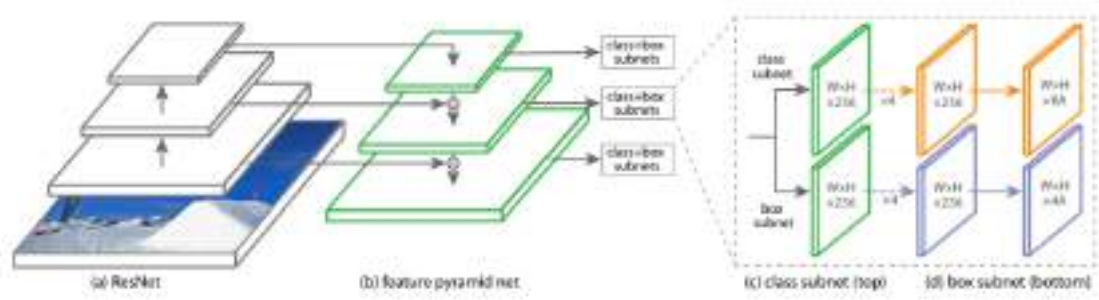


Figura 10: Esquema del modelo de una etaoa RetinaNet. [7]

Como vemos en la figura 10 para las zonas donde evaluar la presencia de objetos se parte de una serie de Anclajes (Anchors), que son básicamente cajas con distintas áreas, cubriendo las escalas de la FPN, y de distintos ratios (1:2,1:1,2:1), que mapean toda la imagen a procesar. Para realizar las predicciones se tiene una capa principal, que se encarga de obtener el mapa de características para una imagen de entrada, y dos 'subnets':

Classification Subnet: Encargada de predecir la probabilidad de encontrar un objeto en cada posición espacial para todos los Anclajes y todas las clases a considerar.

Box Regression Subnet: Esta capa se ocupa de ajustar las cajas de los Anclajes de cada localización espacial a un hipotético objeto en dicha zona. Para ello consta de 4 parámetros por Anclaje que buscan ajustar este a los valores reales de las 'cajas' existentes.

2.3.2. YOLO (You Only Look Once)

Este es otro de los modelos de *One Stage* con mejor desempeño. Una de las principales diferencias es que se usan características de toda la imagen para realizar

portancia a los casos difíciles [8]

la detección. Para ello se divide la imagen en una maya de celdas de $S \times S$ en las que se predicen B 'cajas' con una puntuación de confianza. La arquitectura de la red para realizar estas predicciones sigue un esquema análogo al del resto de modelos, usando capas convolucionales anidadas y MaxPooling para la extracción de características, y capas fully connected para las probabilidades de pertenencia a las clases y las coordenadas de estas como vemos en la figura 11 a continuación.

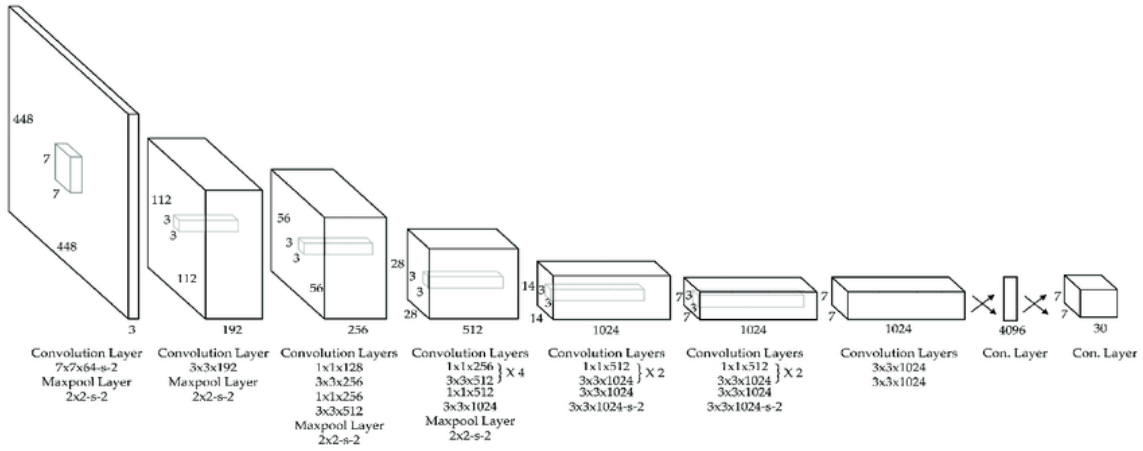


Figura 11: Esquema de la arquitectura del modelo YOLO. [9]

También hay una versión todavía más rápida de este algoritmo (Fast YOLO) donde se utilizan un menor número de capas convolucionales y menos filtros en ellas.

Por lo general tenemos que los algoritmos con dos etapas obtienen una mayor precisión en la detección pero si queremos una detección puramente en tiempo real con un alto número de imágenes por segundo los algoritmos con una única etapa pueden mostrar una mayor velocidad de procesado, por lo que no hay un tipo mejor para todos los casos.

3. Estado del arte de la visión artificial en vídeos

Continuamos nuestro recorrido a través de este campo de la inteligencia artificial centrándonos ahora en la aplicación en vídeos. Cabe destacar que, como es obvio, esta se apoya en gran medida en la detección en imágenes, y, por tanto, complementa los métodos vistos anteriormente.

3.1. Detección en vídeo

Todos los algoritmos introducidos en el apartado 2 nos permiten realizar detección de objetos con alta precisión en imágenes, pero al tratar con vídeos necesitamos un modelo que nos permita aprender la secuencia temporal inherente entre las distintas imágenes de este. Por otro lado, la detección en vídeo también posibilita la tarea de la identificación de objetos a lo largo de las secuencias.

El principal problema a abordar en el marco de la detección en vídeo viene cuando los objetos a detectar son parcial o totalmente ocluidos. En este caso, la detección en dicha imagen fallaría y deberíamos valernos de las imágenes anteriores para extrapolar dicha información. Para dotar al modelo de la capacidad de coherencia temporal actualmente la aproximación más utilizada se basa en el uso de redes recurrentes, en concreto LSTM, en conjunción con las detecciones en cada imagen del vídeo. Añadiendo estas como una capa adicional, la cual recibirá como entrada información relativa a las detecciones.

3.1.1. LSTM

Este tipo de redes, llamadas Long-Short Time Memory ([10]) permiten a la red neuronal considerar las entradas de momentos temporales anteriores, tanto mediante un término de memoria a largo plazo como uno de memoria a corto plazo. Esto posibilita el uso de esta 'memoria' para realizar las predicciones y así dotar al modelo de la coherencia temporal buscada.

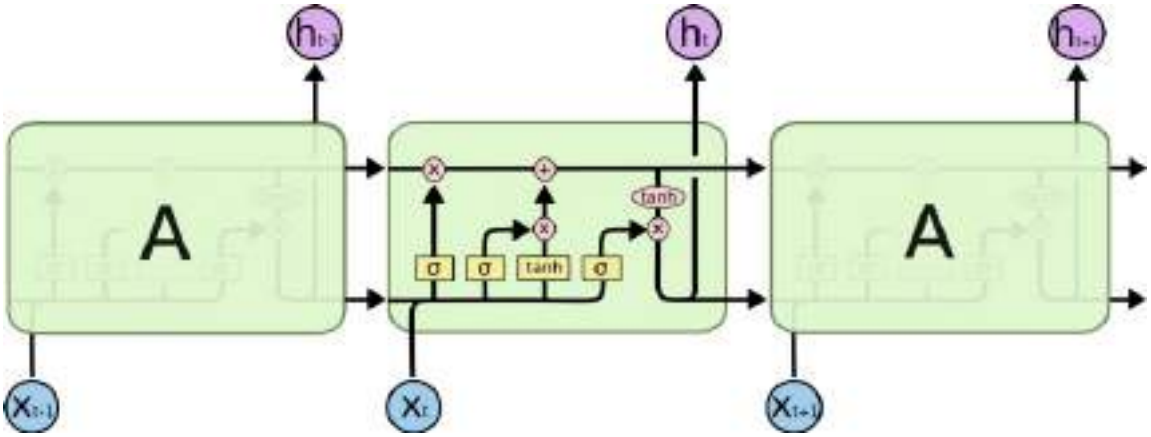


Figura 12: Esquema de una neurona de una red LSTM.

En la figura 12 vemos un esquema general y sencillo de una neurona en distintas etapas temporales. La clave de estas redes es que existe un término de memoria a largo plazo, como podemos apreciar en la parte superior de la imagen, que no es sometido a ninguna función de activación, lo cual disminuiría su relevancia rápidamente. Por otro lado, observamos un término de memoria a corto plazo proveniente de la misma neurona en el estadio temporal anterior. Y por último, la entrada de la red en el momento actual. Todo esto se considera para generar una salida en este tipo de neuronas. Los parámetros que determinan cada una de estas componentes se ajustan por retropropagación (backpropagation).

Vamos a definir otros conceptos relacionados con los parámetros y funcionamiento de las neuronas LSTM en redes neuronales y que conviene conocer para la definición y desarrollo de las redes basadas en estas.

Función de activación

Se entiende como función de activación a la función que se le aplica a las entradas de las neuronas para generar una salida. Existen distintos tipos de estas como vemos en la figura 13, y la comprensión de las distintas propiedades de cada una de ellas es un punto clave en el desarrollo de cualquier problema de inteligencia artificial, ya que estas, sobre todo en la capa de salida, limitan los resultados que se obtienen.

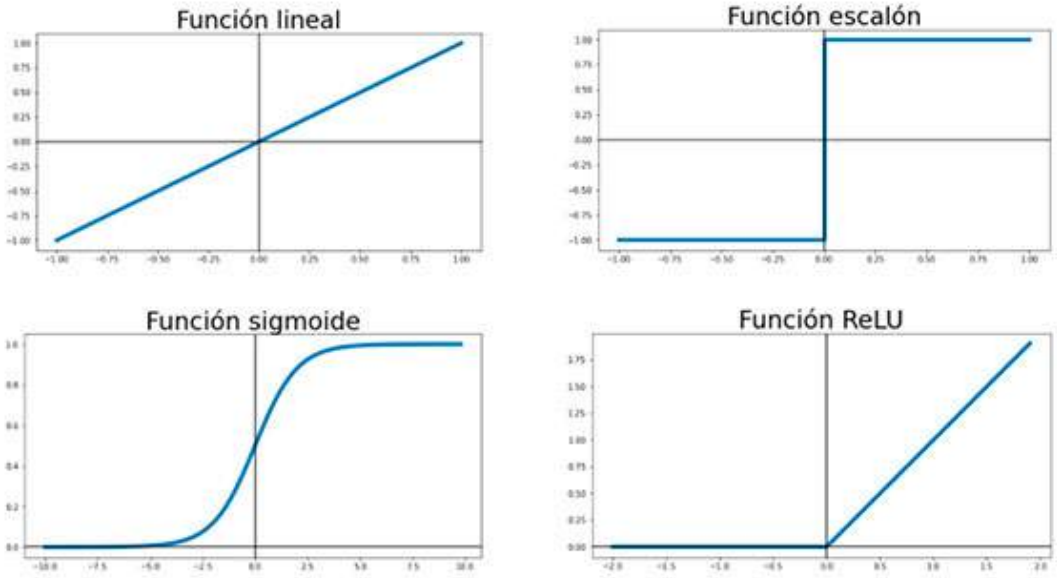


Figura 13: Representación de las principales tipos de funciones de activación.

'Capa' Bidireccional

Conviene explicar esta propiedad, que adquiere especial sentido cuando las distintas posiciones del vector de entrada simbolizan distintos momentos temporales. Esta se basa sencillamente en alimentar a la neurona con las entradas en dirección directa e inversa, como vemos en la figura 14, por lo que sería como tener dos capas de neuronas, cuyas salidas después vienen concatenadas y se les aplica la función de activación de manera conjunta. Con esto nos aseguramos de que la importancia que se le da a la tendencia temporal real no prevalezca sobre la inversa, y, por tanto, que el peso que reciben durante el entrenamiento las detecciones más recientes y más lejanas sea la misma para nuestra red.

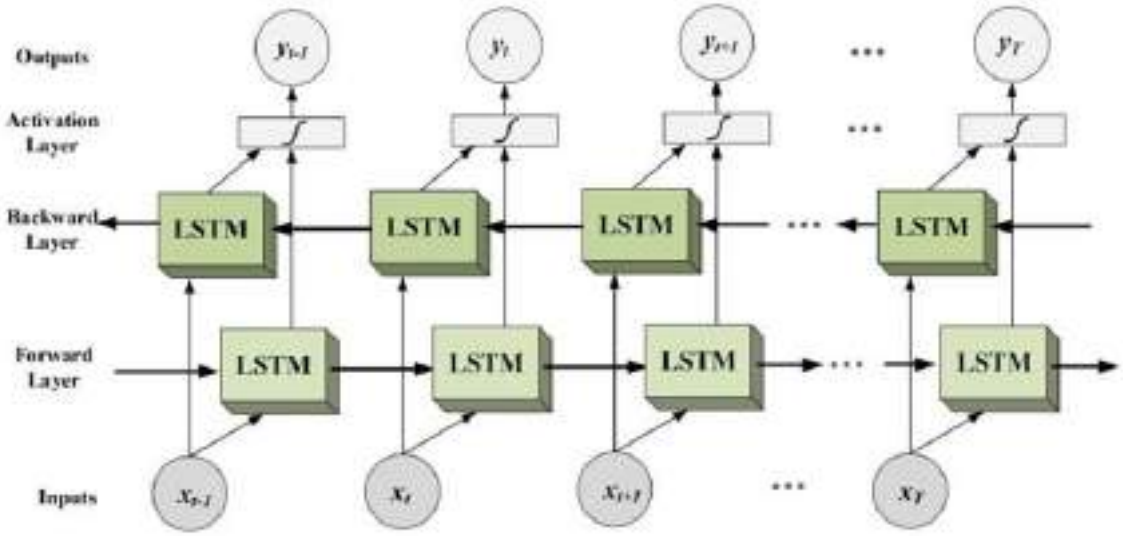


Figura 14: Esquema de una capa bidireccional

Actualmente, en los modelos de detección en vídeo, la integración de capas recurrentes se aplica a la salida de las capas convolucionales, tomando en el caso más profundizado, además de la posición, el vector de características proporcionado por estas de la imagen como entrada. Aunque este procedimiento está obteniendo una importancia creciente por sus posibilidades, es costoso de entrenar y requiere un procesamiento elevado que lo hace difícilmente compatible en la actualidad con la ejecución en tiempo real.

Esto nos lleva a pensar si no existe una manera más sencilla y computacionalmente menos costosa de establecer una relación entre las detecciones de distintas imágenes para mantener un seguimiento de los objetos, sin necesidad de ejecutar la detección en todas las imágenes. Para abordar dicho problema surgió el seguimiento (Tracking). A partir de ahora utilizaremos el término inglés Tracking para estar en concordancia con el nombre de este tipo de modelos.

3.2. Object tracking

Hasta ahora hemos explicado métodos de detección de objetos en imágenes mediante algoritmos de aprendizaje profundo. Debido al uso de estas funciones, este método tiene un tiempo de computación que para aplicar a vídeo de alta resolución en tiempo real puede ser inviable. Por otro lado, estas necesitan de capas recurrentes para relacionar la información entre imágenes. Por tanto, como nuestro objetivo final es aplicar la detección e identificación de personas a lo largo de una secuencia de vídeo, conviene estudiar aproximaciones alternativas.

Al contrario que los modelos de detección que buscan el objeto dentro de la imagen individual, los modelos de tracking tienen como entrada la posición en formato caja (bounding box) de este y se basan en seguirlo a través del resto de imágenes mediante distintas técnicas. Esto nos permitirá mantener identificado un objeto que hemos detectado en una de las imágenes previas. Además, el hecho de realizar tracking sobre un objeto en vez de la detección nuevamente en cada imagen nos permite sobre el papel aumentar significativamente nuestra velocidad de procesado. Vemos a continuación las principales técnicas utilizadas en este campo.

3.2.1. Correlation tracking

Como su propio nombre indica este modelo se basa en la correlación, la cual puede ser evaluada tanto en el dominio espacial como en el frecuencial. Dependiendo del filtro de correlación utilizado el resultado variará ligeramente. Dicha correlación se evalúa entre la posición conocida del objeto y la nueva imagen se obtiene siguiendo la siguiente expresión [11]:

$$C(i,j) = \frac{(S_{i,j} \cdot R)}{|S_{i,j}||R|} \tag{1}$$

Donde $S_{i,j}$ representa cada posición i,j de la imagen actual y R el vector que representa la anterior posición del objeto a seguir. Se normaliza entre los módulos para evitar que zonas con una intensidad elevada en la imagen tengan un valor de correlación alto. En el marco de este modelo de tracking tenemos distintas funciones en las librerías de Dlib y OpenCV, los principales modelos que encontramos implementados

en esta última son:

- KCF Tracker: Kernelized Correlation Filters, muestra problemas al lidiar con oclusión completa.
- CSRT Tracker: Discriminative Correlation Filter, más preciso que KFC pero ligeramente más lento.
- MOSSE Tracker: Minimum Output Sum of Squared Error, no se ve afectado por el cambio de apariencia del objeto. Se trata del modelo más rápido, lo cual obviamente afecta a su desempeño.

Independientemente del modelo que se us,e surge el problema de que es necesario inicializar una instancia para cada objeto al que se quiera realizar tracking, lo que aumenta considerablemente el tiempo de procesado, por lo que para aplicaciones en tiempo real resulta imprescindible implementar un método que tolere el multiprocesado. Sobre esto volveremos a profundizar en el desarrollo del modelo.

3.2.2. Deep Tracking

Podemos encontrar modelos de tracking basados en el aprendizaje profundo, existiendo uno dentro de la librería OpenCV, llamado GOTURN, basado en la aplicación de redes convolucionales en la imagen actual y la siguiente para establecer una relación entre los resultados con capas fully-connected:

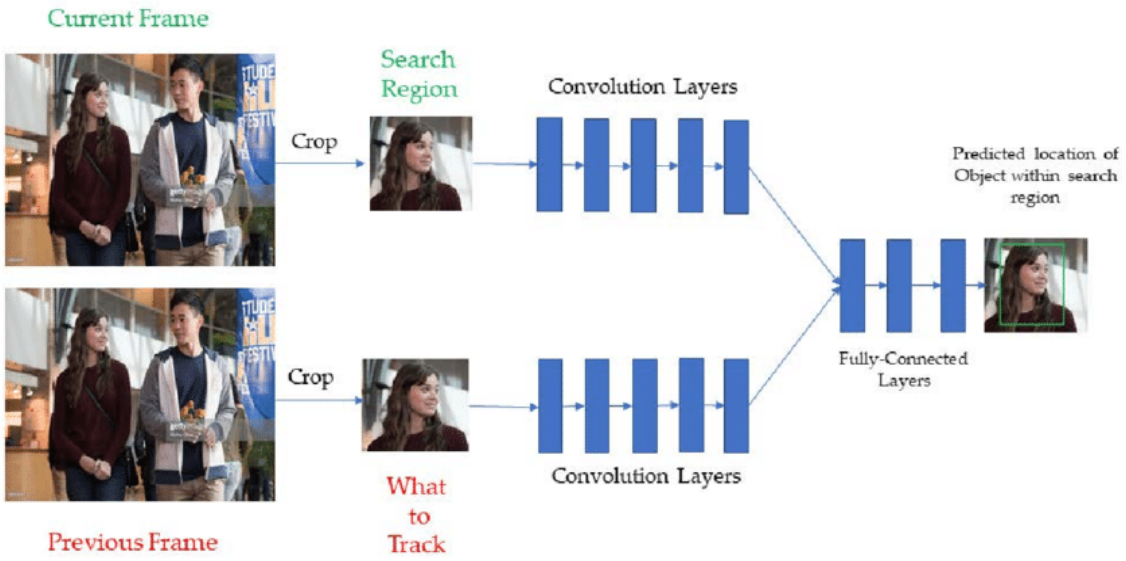


Figura 15: Esquema del algoritmo GOTURN de deep Tracking.

Como se ve en la figura 15 se utiliza como entrada la posición del objeto a seguir en la imagen anterior y determina una región de búsqueda en la cual identifica al objeto en el momento actual. Esto es práctico porque limita la cantidad de regiones a las que se le aplican las capas convolucionales respecto a la detección clásica en imágenes, pero supone una pérdida de precisión y al seguir siendo un método de aprendizaje profundo, la ganancia de velocidad de computación no será tanta como en los métodos clásicos.

Deep SORT

Ampliación del algoritmo SORT (Simple Real Time Tracker) es uno de los métodos de tracking más usados en la actualidad. Se basa en el uso conjunto de distintas métricas y parámetros como:

Filtro de Kalman

Ayuda a reducir el ruido y los errores de predicción haciendo estimaciones de la posición basándose en la posición dada por la detección (asumiendo una tasa de

error en esta) y una velocidad de cambio asumida lineal. Por tanto, el estado viene descrito por los parámetros (x, y, a ,h, x', y', a', h') siendo x e y las coordenadas del centro de la detección, h la altura y a la relación de forma, y sus respectivas tasas de cambio.

A continuación se necesita una métrica de distancia para obtener una asociación entre objetos a los que se les estaba realizando tracking y una nueva detección. Para esto se utiliza la distancia de Mahalanobis [12] incorporando incertidumbres con el filtro de Kalman [13].

Para perfeccionar este algoritmo en los casos donde el filtro de Kalman falla en la predicción se introduce una nueva métrica de distancia vinculada a la 'apariencia' del objeto. Esto se hace a partir de su vector de características, quedando la distancia resultante [14]

$$D = \lambda D_k + (1 - \lambda) D_a \tag{2}$$

Siendo D_k la distancia de Mahalanobis, D_a la distancia coseno entre los vectores y λ un factor de peso a optimizar.

4. Detectron2 para la detección en imágenes

Una vez comprendido el marco teórico de la visión artificial en sus dos principales marcos de aplicación, vamos a adentrarnos en el uso práctico y la evaluación de los distintos modelos en la detección en imágenes. Aunque realizar un modelo de redes convolucionales para este problema no resultaría complicado, el entrenamiento de este para obtener resultados óptimos requeriría de grandes volúmenes de datos etiquetados de gran calidad, los cuales son especialmente difíciles de conseguir. Por ello vamos a recurrir a librerías con modelos ya entrenados para esta tarea.

Detectron 2 es una librería de codigo abierto de facebook creado sobre PyTorch donde se encuentran implementados múltiples algoritmos de detección de objetos como Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, TensorMask, PointRend, DensePose, entre otros.

Esta será la librería principal usada en este proyecto para la detección de personas, ya que nos permite aplicar distintas detecciones en imágenes, así como detección de objetos de hasta 80 clases o como los keyPoints de personas. Incluye también la opción de segmentación semántica y de instancias. Vemos a continuación en la figura 16 el ejemplo de los principales casos de aplicación:

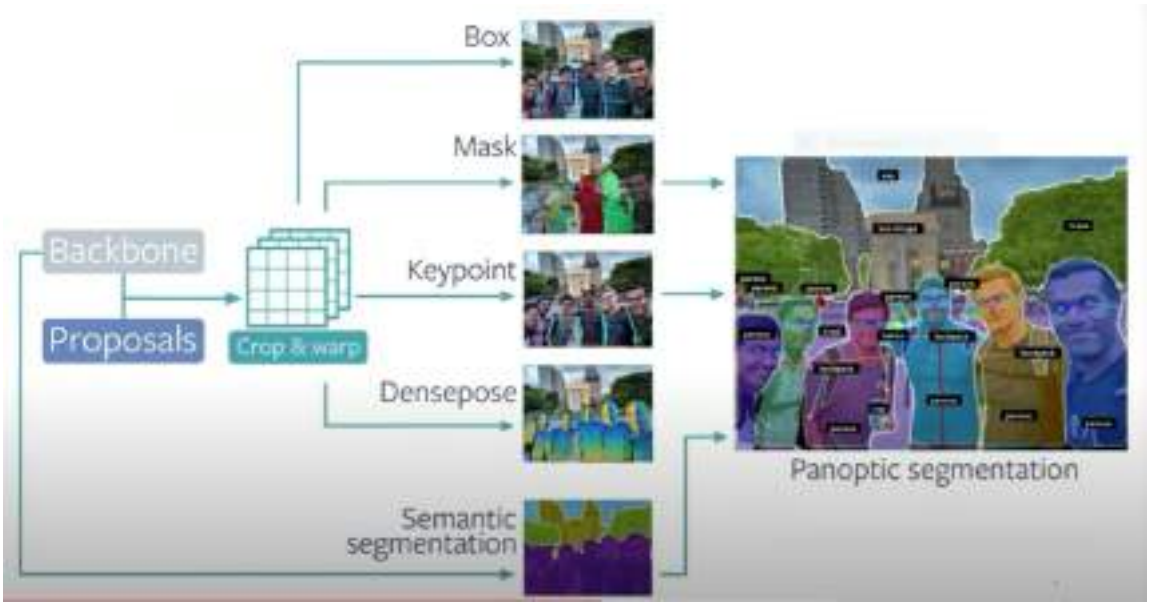


Figura 16: Esquema de las distintas funcionalidades de detectron2.

4.1. Estructura de la red

Puesto que va a ser la librería principal usada en los modelos de este trabajo vamos a entrar más en detalle en la arquitectura de la red generada por esta. En términos generales los distintos modelos de dos etapas (R-CNN, fast-RCNN, etc...) usan la misma estructura con pequeñas variaciones.

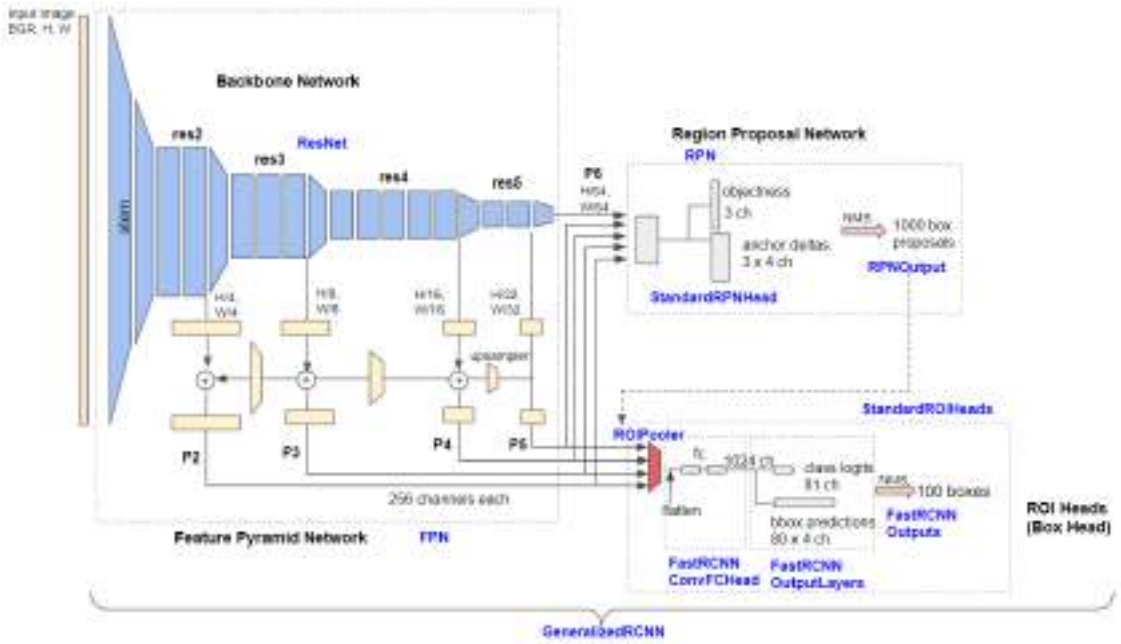


Figura 17: Arquitectura del algoritmo faster R-CNN de detectron2. [15]

En la figura 17 encontramos las tres partes principales de las que se componen los modelos generados, los cuales explicaremos a continuación en profundidad.

4.1.1. Backbone Network

Esta parte se encarga de extraer mapas de características de la imagen de entrada, usando para ello el algoritmo FPN [16] (Feature Piramide Network), el cual extrae estos mapas con múltiples escalas como vemos en la figura 18.

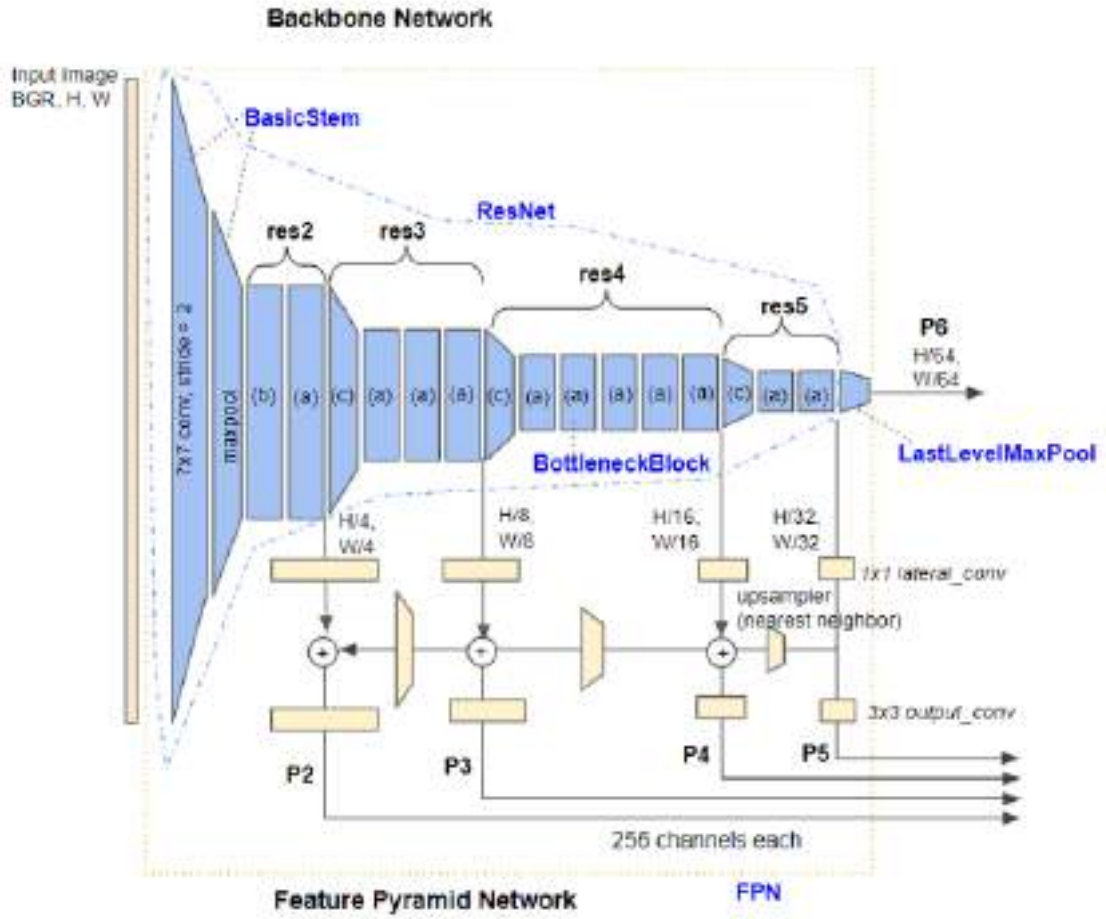


Figura 18: Feature Piramide Network [15]

Aquí se observan las distintas subsecciones que tiene el modelo donde se aplican alternadamente capas convolucionales de diferentes tamaños y con parámetros los cuales no entraremos a analizar aquí. Esto nos deja en la salida de cada subcapa (P2, P3... P6) con mapas de características con 256 canales cada uno con diferentes tamaños de submuestreo, permitiendo la detección de objetos a distintas escalas dentro de la imagen.

4.1.2. Region proposal Network

Esta capa recibe como entrada los mapas de características para las diferentes escalas obtenidos en la 'Backbone network' y genera alrededor de 1000 propuestas de región donde potencialmente habría objetos a detectar. Para ello, esta capa consta de una red neuronal y de distintas funcionalidades complementarias y como hemos explicado antes requiere de un entrenamiento independiente. Vemos en 19 el orden del procesado para esta parte del modelo.

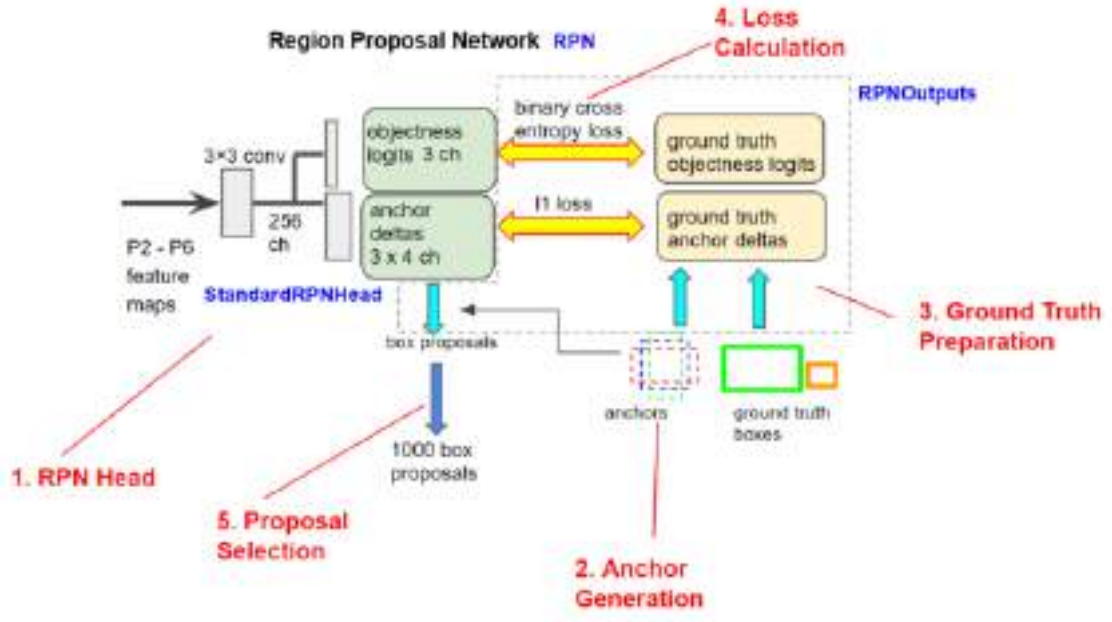


Figura 19: Region Proposal Network [15]

Esta supondría la capa más importante de los modelos de dos etapas implementados en esta herramienta, siendo el motivo por el cual pueden focalizar la detección y así mejorar la precisión. Por ello vamos a explicar las distintas componentes para entender mejor el funcionamiento.

En la primera parte se introduce como entrada los mapas de características para cada escala (P2-P6) y a partir de estos se genera un mapa de probabilidad de la existencia de objetos y la forma de las cajas relativa a estas.

Después, para relacionar esto con los objetos del 'ground truth' se generan una serie de Anclajes, para cada nivel del modelo se fija un tamaño de celda (32, 64, 128, 326, 512) y unas proporciones (1:2, 1:1, 2:1). A continuación se divide la imagen en una malla con la dimensión del mapa de características de cada nivel y en cada esquina se sitúan los 3 Anclajes.

Luego se computa la métrica de Intersección sobre la Unión (IoU), la cual definimos en el apartado 4.2, para obtener los anclajes más representativos y se implementan gradientes en las 4 direcciones para ajustarlos lo máximo posible al ground Truth. Para optimizar los parámetros se calculan dos métricas como vemos en el paso 4.

Finalmente, se preseleccionan 2000 Anclajes optimizados por cada subnivel ordenados según las métricas calculadas, para después con la técnica de 'Non-Maximum Suppresion' [17] se eligen únicamente 1000 cajas cada una con una puntuación (score). Esta técnica para filtrar las regiones propuestas se basa en los siguientes pasos:

1. Selección de la región con mayor puntuación (score) y por tanto mayor confianza en la clasificación.
2. Calculo de la métrica de IoU de la región seleccionada con todas las restantes y eliminación de todas las que superen un umbral a definir.
3. Repetición de los dos primeros procesos hasta que llegar al número de regiones filtradas deseado.

Con esto lo que se consigue es no procesar regiones solapadas que contienen la misma información.

4.1.3. ROI Heads

Esta capa es la encargada de extraer las propuestas de caja de los distintos mapas de características procedentes de la capa FPN. Para determinar de que nivel de submuestreo extraer la caja se aplica el criterio definido por la siguiente fórmula:

$$feature\ level = floor\left(4 + log_2\left(\frac{\sqrt{box\ area}}{224}\right)\right) \tag{3}$$

Una vez extraídas las regiones de interés (ROI) de los mapas de características, estas se normalizan a una dimensión 7×7 para los 256 canales y se pasan como entrada a las redes ROI (box) Head. En estas se aplican dos funciones de perdida, una para la localización de los objetos y otra para su clasificación.

Es de vital importancia comprender el funcionamiento interno del modelo, puesto que, como veremos más adelante, si se quiere complementar las opciones que este ofrece hay que tener un buen conocimiento de los componentes y su función.

4.2. Evaluación previa de los modelos de detección de personas

Llegados a este punto ya tenemos todo el conocimiento teórico necesario para empezar a implementar casos prácticos de identificación y seguimiento en vídeo. A partir de ahora nos enfocaremos, por tanto, en la detección de personas. Estos modelos se basarán en las detecciones en imágenes aportadas por los distintos modelos proporcionados por detectron2. Para aprender a usar de manera óptima esta herramienta realizamos primero una prueba de aplicación en un dataset de distinción entre personas y objetos con forma de persona.

En nuestro caso estamos interesados en obtener la caja de las personas a detectar. Para ello podemos utilizar los modelos ya pre-entrenados que nos ofrece para la detección o entrenar un nuevo modelo creando un dataset con el formalismo de COCO (Common Objects in Context) [5] para ello. Como hemos dicho, hay cerca de 80 clases cuya detección ya ha sido previamente entrenada con datasets por facebook. Para este proyecto nos limitaremos únicamente a la detección de personas usando modelos pre-entrenados por su mejor desempeño. Cabe mencionar, sin embargo, que dichos modelos han sido entrenados sin una pretensión de robustez ante objetos

[5]COCO es un conjunto de datos de detección, segmentación y subtitulación de objetos a gran escala. Consta con su formalismo de etiquetado propio y es el usado para el entrenamiento de la red de modelos de detectron2. [18]

parecidos (con forma) de personas.

Así pues, con el fin de familiarizarnos con el formato de datos de los datasets aceptados por el formato de COCO y por ende el detectron2 y evaluar el criterio de distinción de los modelos en la detección de personas, lo cual nos dará una idea de los rasgos en los que se fija, hemos entrenado un modelo (mask R-CNN) de discriminación entre personas y figuras de personas y comparado el resultado con la aplicación de la red pre-entrenada.

(a) Modelo entrenado con un dataset. (b) Modelo pre-entrenado.

Figura 20: Vemos el resultado de la detección con el modelo de detectron2 de Mask R-CNN para el modelo pre-entrenado por Facebook y un modelo entrenado con un dataset de detección de objetos similares a personas.

Como se observa en las figuras 20 y 21, el modelo pre-entrenado confunde al muñeco y los maniquis (sin cabeza) con una persona, con una probabilidad del 100 % o cercana en la mayoría de casos, lo que nos da a entender que este se basa en la detección de rasgos muy generales de las personas. Por otro lado al haber entrenado el modelo con un dataset específico para detectar estos fallos vemos que identifica correctamente al muñeco como "person-like". Aunque esto indica una simplicidad en el entrenamiento del modelo a la hora de discernir los rasgos de una persona real, para nuestro problema no resulta significativamente relevante y utilizaremos el modelo pre-entrenado por su mejor desempeño detectando personas.

Para decidir que modelo usar vamos a evaluar las distintas opciones que nos ofrece detectron2. Aunque podemos encontrar diversos artículos que comparan su

(a) Modelo entrenado con un dataset. (b) Modelo pre-entrenado.

Figura 21: Vemos el resultado de la detección con el modelo de detectron2 de Mask R-CNN para el modelo pre-entrenado por facebook y un modelo entrenado con un dataset de detección de objetos similares a personas.

desempeño y hemos comentado ya la diferencia de tendencia entre mayor velocidad frente a mayor precisión de los modelos de una etapa respecto a los de dos etapas. Para ello primero vamos a definir la métrica de acierto que se utiliza de manera generalizada en estos modelos.

Intersection Over Union

Como su propio nombre indica, esta métrica computa el área de intersección de la detección y el ground truth normalizada a la unión de ambas. Vemos una representación visual de esta en la figura 22.

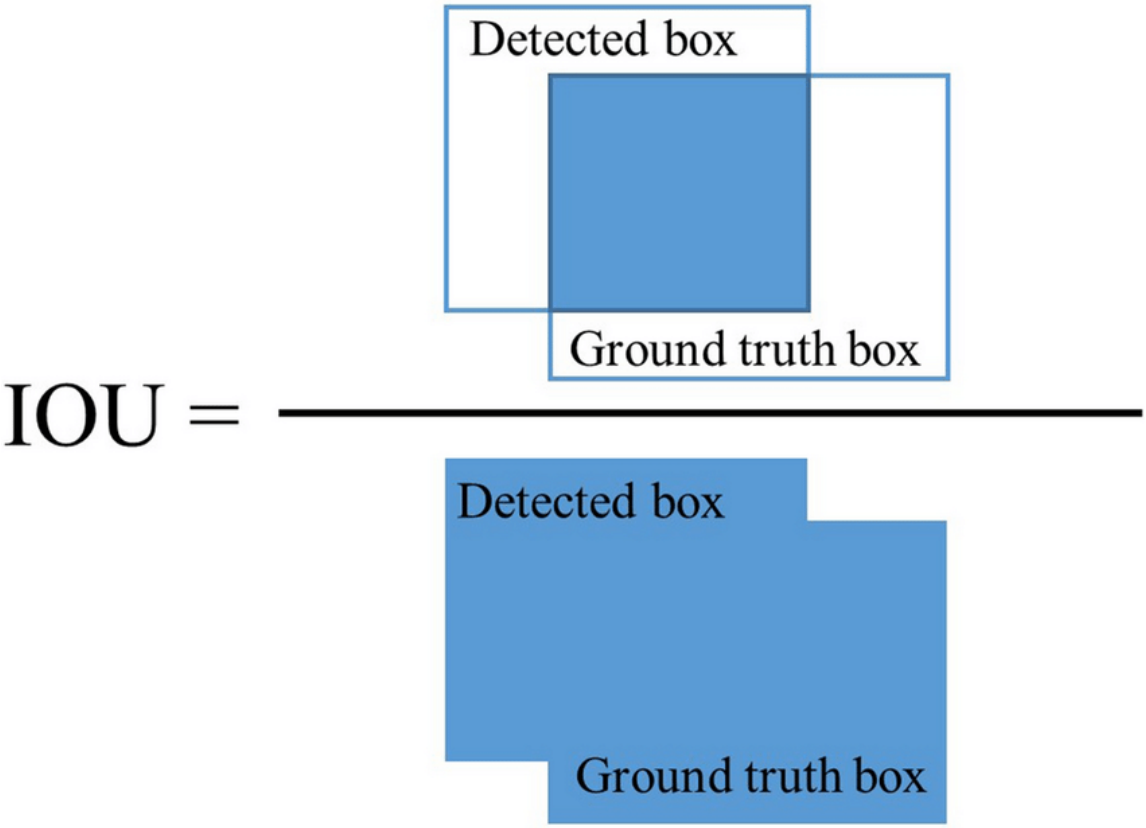


Figura 22: Ejemplo gráfico de la métrica IOU.

Por tanto, un valor de 1 de la métrica supondría una correspondencia exacta entre la detección y el ground truth mientras que si no hay coincidencia el valor mínimo sería de 0. Para penalizar objetos no detectados, el valor medio de esta métrica se calcula utilizando todos los objetos existentes. Para el caso de objetos detectados que no se corresponden con ningún objeto existente no se incluye aquí ninguna consideración, para atender a este problema necesitaremos una métrica distinta. Para ello, a parte de la métrica de acierto de la detección respecto a la real, también calculamos dos métricas adicionales correspondientes al porcentaje de detecciones que se han realizado que no se correspondían con un objeto real etiquetado, así como el número de objetos que no han sido detectados.

Para llevar a cabo esta evaluación previa utilizamos un dataset de detección de personas en imágenes [19].

Modelo	IoU	Detecciones Erróneas (%)	Personas no detectadas (%)	Tiempo Procesado
Faster rcnn	0.729	10.40	6.83	0.00182
RetinaNet	0.701	1.47	19.68	0.00114
Mask rcnn	0.731	7.94	6.82	0.000936

Tabla 1: Resultados de la evaluación de los métodos de detección en imágenes.

Para optimizar el funcionamiento de cada modelo bajo un criterio común hemos ajustado el umbral de la detección según el valor del parámetro de IoU. Viendo que el valor óptimo del método de One Stage (RetinaNet) es significativamente menor que para los métodos de Two Stages (Faster RCNN y Mask RCNN).

De los resultados recogidos en la tabla 1 se puede ver que por lo general los métodos basados en dos etapas muestran una mayor precisión, a pesar de qué realizan un mayor número de detecciones que no se corresponden con un objeto. Por otro lado, el método RetinaNet de una etapa genera pocas detecciones erróneas pero, tiene un porcentaje significativamente más alto de personas no detectadas, llegando casi al veinte por ciento. Por último apreciamos que en cuanto a tiempo de procesado, también Mask R-CNN presenta el mejor desempeño, viendo que Faster R-CNN se queda significativamente atrás a los otros dos. Atendiendo a los valores obtenidos se desprende que no hay un método universalmente superior, en nuestro caso debido a que el objetivo principal es la detección en vídeo y las detecciones se realizaran sobre objetos de cuerpo entero con perfiles mejor delimitados elegimos el modelo Mask R-CNN para ello. De hecho, fijándonos únicamente en el tiempo que tarda en realizar la detección se podrían procesar vídeos de hasta casi 1068 imágenes por segundo.

5. Definición de los modelos

Describimos en este apartado la arquitectura utilizada en el desarrollo de los modelos. En el apartado teórico hemos llevado a cabo una revisión de las distintas posibilidades que encontramos actualmente para trasladar la detección en imágenes al formato vídeo. Por completitud implementaremos un modelo de ejemplo de las dos principales vertientes, la detección en cada imagen, complementada con modelos de deep learning (LSTM) y el tracking. Puesto que este último tiene como principal ventaja la mayor velocidad de procesado, utilizaremos modelos clásicos para ello. Aunque algunos de los modelos de deep learning orientados al tracking están pensados para el procesado en tiempo real, como Deep SORT, no hemos encontrado este pre-entrenado de manera consistente, como si pasa para los algoritmos en imágenes.

Por tanto, desarrollaremos el modelo de tracking clásico basándonos en funciones de correlation tracking y compararemos la precisión y acierto de ambos modelos.

5.1. Modelo de detección y tracking clásico

Teniendo en cuenta la información sobre el estado del arte de los algoritmos de detección y tracking vamos a realizar un modelo sencillo que permita identificar a personas a lo largo de este, manteniendo la información de estas y enumerándolas.

Mostramos en la figura 23 un diagrama básico del modelo:

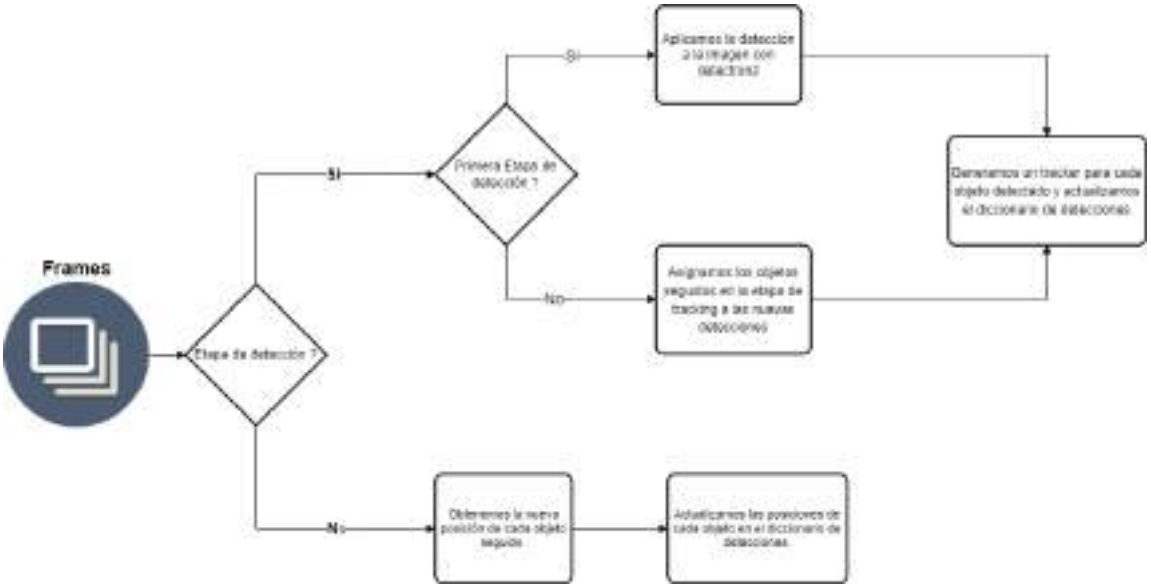


Figura 23: Diagrama de flujo del modelo de detección.

A continuación, pasamos a detallar el papel y funcionamiento de cada una de las etapas del modelo.

Etapa de detección

Cada un número N de imágenes realizamos la detección con el modelo de detectron2 seleccionado, filtrando únicamente a la clase persona. En la primera iteración guardaremos la información en un diccionario ordenado directamente de los objetos detectados con sus correspondientes identificadores. En las etapas de detección consiguientes se asociará dichas detecciones con las personas registradas, como explicaremos en la etapa de reidentificación.

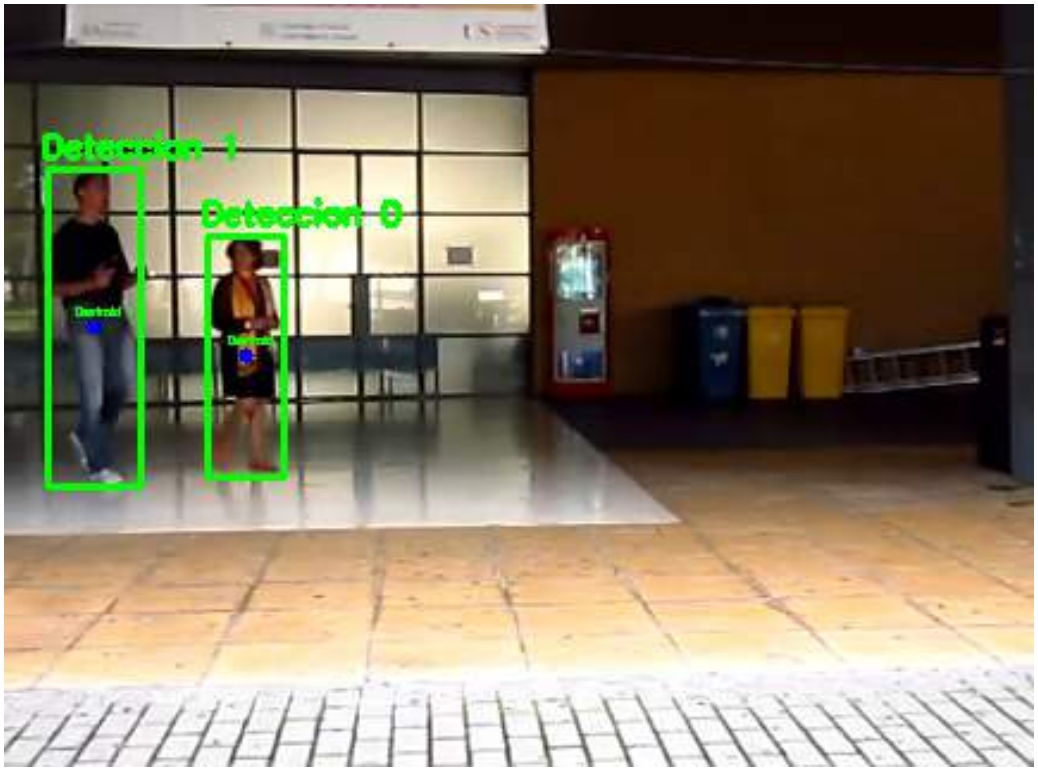


Figura 24: Ejemplo de detección.

En la figura 24 vemos la información correspondiente a cada objeto detectado que almacenamos en forma de Bounding Box, centroide e identificador.

Etapa de tracking

Una vez hemos detectado los objetos, las siguientes etapas consistirán en generar un tracker de dicho objeto e ir representando la nueva posición aportada por dichos trackers para el objeto en cada nueva imagen a lo largo de esta etapa. Para ello únicamente necesitamos como entrada la posición (bounding box) del objeto en el frame anterior y su identificador.

Como comentamos en el apartado de la explicación de los distintos métodos de tracking, necesitaremos un método que nos permita implementar técnicas de multiprocesado, ya que, al aumentar el número de objetos sobre los que se realiza el

seguimiento aumentaría significativamente el tiempo de computación. Para llevar a cabo esto hemos creado una clase llamada tracker en las que pasamos como referencia un **InputQueue** y un **OutputQueue**, y generamos un objeto de esta clase para todas las personas detectadas sobre las que vamos a hacer tracking. Al inicio de una nueva imagen a cada objeto de esta clase se le pasa dicha imagen a través del metodo put del **Inputqueue**, y al recibir esta se calculan las nuevas posiciones y se obtienen a través del método get aplicado al **OutputQueue**. Explicamos mejor el funcionamiento en el apéndice A. Una vez generados todos los trackers y actualizadas sus posiciones en el diccionario de detecciones durante toda la etapa de tracking llegamos al momento de tener que relacionar la última posición registrada con la siguiente detección.

Etapa de reidentificación

Puesto que nuestro objetivo es mantener identificadas durante el largo del proceso a las personas detectadas, tenemos que establecer una relación entre la posición final de la última imagen en el que se realizó tracking y las nuevas detecciones. Esto es lo que supone la mayor parte de la algoritmia del modelo y se realiza de la siguiente manera:

1. Primero se calcula la distancia entre los nuevos centroides detectados y los objetos a los que se hacía tracking en la etapa previa, los cuales tienen ya asignado su correspondiente id y están almacenados en un diccionario ordenado.

2. A continuación se asignan los nuevos objetos detectados a los correspondientes objetos existentes con los cuales sus centroides presenten una distancia mínima. Por completitud se establece un máximo al valor de esta distancia a partir del cual no se realizará la asignación. De la misma manera objetos cuya posición final en la etapa de tracking no se encuentre dentro de los márgenes de la imagen serán eliminados.

3. Los objetos detectados que no hayan sido asignados serán añadidos al diccionario de objetos registrados, asignándoles automáticamente el próximo id disponible.

4. Por último, los objetos de los que se estaba realizando tracking que no hayan sido asignados a ninguna nueva detección son eliminados.

Figura 25: Ejemplo sencillo de reidentificación, donde vemos, las puntos rojos representando los valores de los centroides registrados para la última imagen de la etapa de tracking, los puntos azules los centroides de la detección en la imagen actual, y las flechas muestran la distancia entre los puntos existentes y nuevos, siendo la menor la de color azul.

Vemos en la figura 25 un ejemplo sencillo de como funcionaría la reidentificación, siendo las flechas azules las que indican como se realizaría la asociación siempre que esta distancia esté por debajo del umbral fijado. Una vez determinado con qué persona existente asociar cada nueva detección se actualiza el diccionario de detecciones con las nuevas posiciones.

Debemos considerar que el error cometido por los métodos de tracking se acumulará haciendo más imprecisa la asignación, por lo que dejaremos la duración (en imágenes) de esta como un parámetro a optimizar, así como la distancia máxima para realizar la asignación.

5.2. Modelo de detección con redes recurrentes

Por contraposición, vamos a implementar también un modelo basado en redes recurrentes a entrenar a partir de las salidas de detectron2.

Como hemos dicho en el desarrollo teórico, el principal reto de la detección en vídeo respecto a la detección en imágenes es conseguir inferir la lógica temporal para detectar objetos incluso cuando no son visibles(oclusión). Puesto que nuestro objetivo también pasa por mantener una relación entre las personas identificadas con este fin, deberíamos definir una función de coste personalizada que tenga esto en cuenta.

Dado que este se trata de un problema con una gran casuística y complejidad, nos vamos a limitar en este caso únicamente a abordar casos sencillos que ejemplifiquen la capacidad del modelo. Esto se debe en parte a que para generar un modelo completo para este problema, aparte de la estructura de la red adecuada para tal fin, se requeriría una cantidad de datos capaz de cubrir de manera amplia y descriptiva la gran variedad de casos que como hemos dicho se podrían dar. Y, como comentaremos en el apartado siguiente, la obtención de estos es una tarea tan laboriosa o más que la propia construcción del modelo.

Centrándonos ahora ya en el modelo, nos encontramos con la primera decisión relevante, referente a las entradas que le proporcionamos a la red. Hay dos posibilidades principales dependiendo del objetivo que tengamos.

Posición de los objetos

Simplemente utilizar como entrada la posición de los objetivos y basar la proximidad entre estos como el criterio de identificación entre objetos en las distintas imágenes. Con esto principalmente el modelo podrá aprender a seguir el movimiento del objetivo a lo largo del marco de la imagen, pudiendo predecir la siguiente posición en base a esto.

Posición de los objetos y mapa de características

Aparte de únicamente la posición de los objetos, podríamos utilizar paralelamente una métrica de proximidad entre el mapa de características de las detecciones para realizar dicha asociación como hemos visto que se lleva a cabo en los métodos de Deep Tracking. En este caso, además de seguir el movimiento, se podría usar dicho mapa tanto para relacionar las detecciones en las distintas imágenes como para tener una referencia de la completitud de la detección y con ello poder mejorarla.

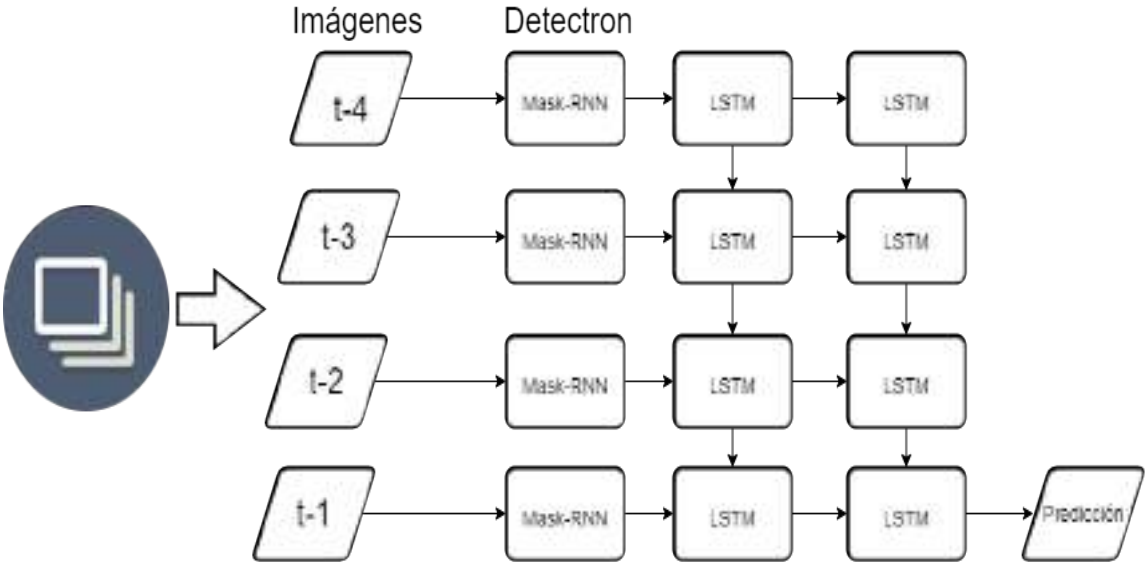


Figura 26: Esquema de la estructura de nuestra red LSTM.

Vemos en la figura 26 la arquitectura de la red, que se resume en generar un vector con las salidas proporcionadas por el modelo de detectron2 y usarlas como entrada para las capas LSTM. Como hemos dicho, esta buscará mostrar un ejemplo del

desempeño del uso de redes recurrentes en estos problemas. Por tanto, la estructura tomará como entrada las detecciones de una única persona a la vez y tendrá un registro temporal de las últimas n imágenes para predecir la siguiente. Este número de registros históricos para la predicción tendrá que ser optimizado, pero debemos considerar que cuanto mayor cantidad de datos más se podrá ajustar a una evolución temporal con mayor margen

5.2.1. Función de pérdida

Como hemos comentado en la sección 4.2, la métrica más habitual y descriptiva para el acierto de las detecciones en imágenes es la intersección sobre la unión (IoU) y sería la ideal para el proceso de entrenamiento. Una de las principales ventajas de esta métrica respecto a la diferencia absoluta(o cuadrática) entre los diferentes parámetros se debe a que las cajas que localizan las personas no tienen lados con la misma medida. En el caso de detección de personas en concreto la anchura suele ser significativamente menor que la altura. Por lo tanto, el mismo error absoluto en el eje x(ancho de la imagen) provocaría una discrepancia en la intersección entre la caja predicha y la real mayor que el mismo en el eje y.

Sin embargo, hay diversos problemas derivados con el uso de esta métrica. El primero sería que toma un valor mayor conforme mejor es el ajuste, por lo que no se podría minimizar el valor directamente. Esto es fácilmente resoluble haciendo la diferencia del valor respecto a uno (máximo valor posible). Otro problema más significativo es que esta métrica no penaliza la diferencia entre detecciones más haya de que no haya unión, por lo que para nuestro problema una detección que se encuentre a 1 píxel de distancia o a 100 tendría el mismo error. Por tanto, habría que dividir la métrica en un primer proceso para asegurar la proximidad de los ejes de la predicción y a partir de que se dé el solapamiento entre la caja predicha y real se podría empezar a optimizar en función de esta métrica.

Todo esto hace que no sea el caso más sencillo de implementar y probablemente no el más eficiente. En el apartado de evaluación de los modelos entraremos en detalle sobre el resultado obtenido utilizando esta métrica.

5.2.2. Estructura de nuestra red

Como mostramos en la figura 26 la red final consta de dos capas de neuronas únicamente, suficiente teóricamente para establecer una relación entre las anteriores posiciones y la siguiente. En la capa de entrada las detecciones serán pasadas de manera bidireccional para que la tendencia se pueda interpretar en ambas direcciones y la importancia de las distintas posiciones sea igual. Como vemos en la figura 14 y hemos comentado en el apartado 5.2 sería como tener dos capas de neuronas, una que recibe las entradas en el orden temporal correcto y otra que las recibe en dirección inversa.

La primera capa se trata de una capa many-to-many, lo que significa que devuelve a su vez una secuencia con distintos registros en la dimensión 'temporal' de la red que son pasados uno por uno análogamente a la última capa, que genera la salida buscada.

En cuanto a las funciones de activación utilizadas, hemos decidido emplear la función ReLu (Rectified Lineal Unit), detallada en la figura 27, para ambas capas, debido a sus características y a su rango de valores, ya que nuestras salidas deben poder ser mayores de 1 y serán siempre positivas.

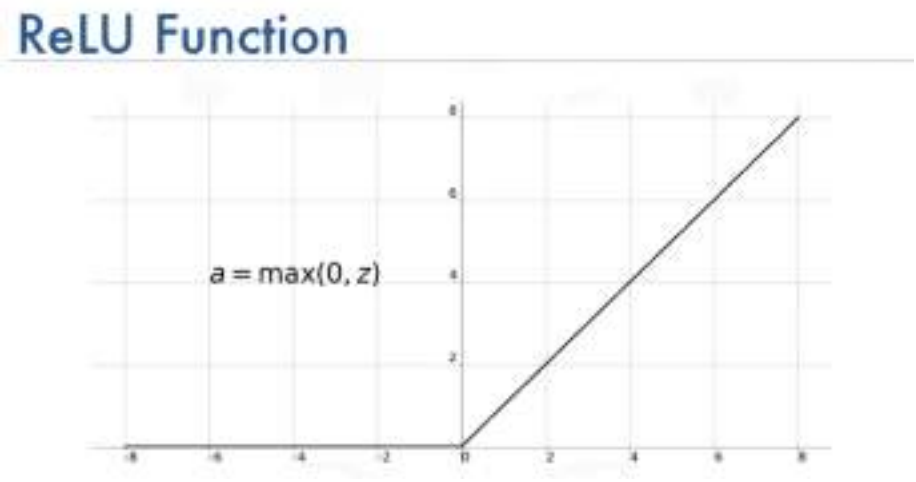


Figura 27: Función de activación ReLu

6. Resultados de los modelos

Uno de los puntos más difíciles y cruciales del proyecto ha sido tratar de obtener conjuntos de datos sobre los que poder realizar una evaluación efectiva y consistente de los distintos modelos, sobre todo en el caso del uso de capas recurrentes donde teníamos que entrenar la última parte del modelo. Esto se debe a la complejidad del problema y la dificultad de encontrar de manera públicamente accesible datos etiquetados de calidad y que se ajusten al problema que queremos tratar.

Aunque existen herramientas avanzadas de etiquetado para realizar nuestra propia base de datos, hemos intentado en la medida de lo posible trabajar con datos obtenidos de otras fuentes para evitar sesgos. En este apartado vamos a hacer un recorrido por la evolución de los modelos conforme los testeamos/entrenamos con las fuentes de datos encontradas, y como las diversas dificultades encontradas nos conducen a la decisión final de generar datos propios.

6.1. Modelo clásico

6.1.1. Desarrollo del modelo

En un primer momento para la evaluación cualitativa del modelo clásico trabajamos con vídeos sin etiquetar, como mostraremos a continuación, dado a la mayor accesibilidad de estos. En este caso esto proporciona una manera rápida y bastante sencilla de evaluar el desempeño del modelo sin necesidad de un cálculo concreto de métricas de error.

Como se ve en las imágenes de la figura 28 logramos el funcionamiento buscado del modelo de detección y tracking. Gracias a esta primera aproximación pudimos mejorar de manera iterativa defectos del modelo. Puesto que en este caso la mejor manera de analizar el funcionamiento de los modelos es viendo las secuencias de vídeo donde se aplican estos, se pueden encontrar los vídeos de los que se extraen los distintos ejemplos que mostraremos en el siguiente enlace `https://github.com/Heclope/TFM/tree/main/videos`.

Observamos que tanto el tracking como la asociación con nuevas detecciones

(a) Imagen inicial. (b) Imagen Intermedia. (c) Imagen Final.

Figura 28: Detecciones del modelo de tracking clásico a lo largo de una secuencia para una frecuencia de detección cada 6 imágenes.

funciona correctamente; sin embargo, el método de tracking en ocasiones conduce a un error que se va acumulando al pasar las imágenes aunque no supone un gran problema para las frecuencias evaluadas. Lo que si resulta más significativo es el error que comete dicho tracker aparentemente al cambiar la textura del fondo, como se aprecia al pasar la persona uno de la imagen vista en 28b a la observada en 28c. Para minimizar dicho error se puede optimizar la frecuencia con la que se realizar la etapa de detección, buscando el equilibrio entre acierto y tiempo de ejecución. Para ello necesitamos un vídeo etiquetado. Así pues, tomamos en primer lugar un dataset del MOTChallenge [21], uno de los principales retos de tracking de personas, que proporcionan distintos vídeos etiquetados para ello. Debido a la alta cantidad de personas a detectar presentes en estos, hemos escogido únicamente uno de ellos para evaluar nuestro modelo.

Vamos a mostrar también gráficamente el resultado obtenido para este dataset en la figura 29, para una frecuencia de detección cada 6 imágenes.

Podemos ver que tras la persona detectada número 3 pasar por detrás de la señal no puede ser detectada y al volver a identificar se le asigna un nuevo id. La optimización de los valores de la frecuencia con la que se realiza una nueva detección y la máxima distancia para la identificación dependerán de la tipología de cada vídeo, pero para acumular el menor error posible durante la etapa de tracking esta debe ser lo más corta posible, por regla general, lo que aumenta la probabilidad de realizar una detección con alguno de los objetivos ocluido.

(a) Imagen inicial. (b) Imagen Intermedia. (c) Imagen Final.

Figura 29: Detección del modelo de tracking clásico a lo largo de una secuencia para una frecuencia de detección cada 6 imágenes.

Sin embargo, también podemos ver como cuando dicha oclusión es parcial y no afecta a un gran número de imágenes el modelo funciona como se muestra en la figura 30. Por otro lado, aquí se confirma lo que será uno de los principales problemas del modelo. Al igual que pasaba en el caso anterior, vemos que cuando la persona número 2 pasa por detrás de la farola, la posición que marca el algoritmo se detiene en esta, debido a que está basado en filtros de correlación.

(a) Imagen inicial. (b) Imagen Intermedia. (c) Imagen Final.

Figura 30: Detección del modelo de tracking clásico a lo largo de una secuencia para una frecuencia de detección cada 6 imágenes.

De la exploración visual del desempeño de nuestro modelo en dicho vídeo se desprende que es capaz de distinguir eficazmente personas cercanas pero no si estas se llegan a solapar debido al error cometido por el tracking, y que en casos de variación brusca de la dirección u oclusión prolongada la reidentificación puede fallar. Sin embargo, a pesar de la complejidad que presenta este vídeo, debido al elevado número de objetivos, la cercanía entre estos y el tamaño relativo al plano, vemos que en fragmentos donde se cumplen unas determinadas condiciones el funcionamiento es correcto. En cuanto al tiempo de procesamiento en este caso, aunque las etapas

de tracking son relativamente rápidas para el número de objetivos, del orden de 0.1 s, las etapas de detección y reidentificación suponen un tiempo de procesado significativamente mayor, llegando al orden del segundo, debido a los cálculos que este supone y a la nueva definición de los trackers.

Durante el entrenamiento de la red recurrente nos dimos cuenta de que el etiquetado de estos datos consta de significativos errores generalizados, como la falta de etiquetado de los objetos en los márgenes de las imágenes, como explicaremos en el apartado siguiente.

Todo esto nos conduce a la conclusión de que, aunque gracias a la evaluación en dichos vídeos se desprenden capacidades del modelo, las cuales comentaremos en el apartado siguiente, necesitamos casos más específicos y más consistentemente etiquetados para poder aplicar las métricas de precisión de la detección correctamente. Dado que la mayor dificultad para el método de tracking escogido es cuando la persona se ve ocluida o el fondo cambia, vamos a incluir casos de esta naturaleza para determinar los límites de este. Para el etiquetado nos serviremos de la herramienta roboflow [22].

6.1.2. Evaluación cuantitativa y cualitativa

Presentamos en la tabla 2 las métricas de error para el modelo clásico en función de la frecuencia con la que se realiza la detección.

Detección única con oclusión parcial.

Mostramos en la figura la tendencia de los resultados para las distintas métricas para este caso, hasta una frecuencia de detección de cada 9 imágenes. Como se observa en la tabla, no hemos incluido ahora la métrica de detecciones erróneas, ya que hemos optimizado el umbral de la detección para no cometer ninguna, puesto que esto comprometería el desempeño de nuestro modelo y de esta forma tenemos un criterio de comparación equitativo con el modelo basado en redes recurrentes.

Conviene aclarar aquí que de estas gráficas podemos considerar únicamente la tendencia general, ya que se tratan de muestras estadísticas relativamente pequeñas (alrededor de las 200 imágenes), y nuestro modelo debido a sus parámetros tiene una fuerte componente aleatoria.

Frecuencia de detección	Métrica IoU	Tiempo por imagen (s)	Detecciones correctas(%)
1	0.815	0.104	99.5
2	0.781	0.198	99.5
3	0.752	0.142	99.5
4	0.712	0.108	100
5	0.680	0.0898	100
6	0.649	0.0802	99.5
7	0.643	0.0711	100
8	0.613	0.0631	100
9	0.640	0.0607	100

Tabla 2: Métricas de error y tiempo de procesamiento en función de la frecuencia de la detección en el caso con oclusión parcial.

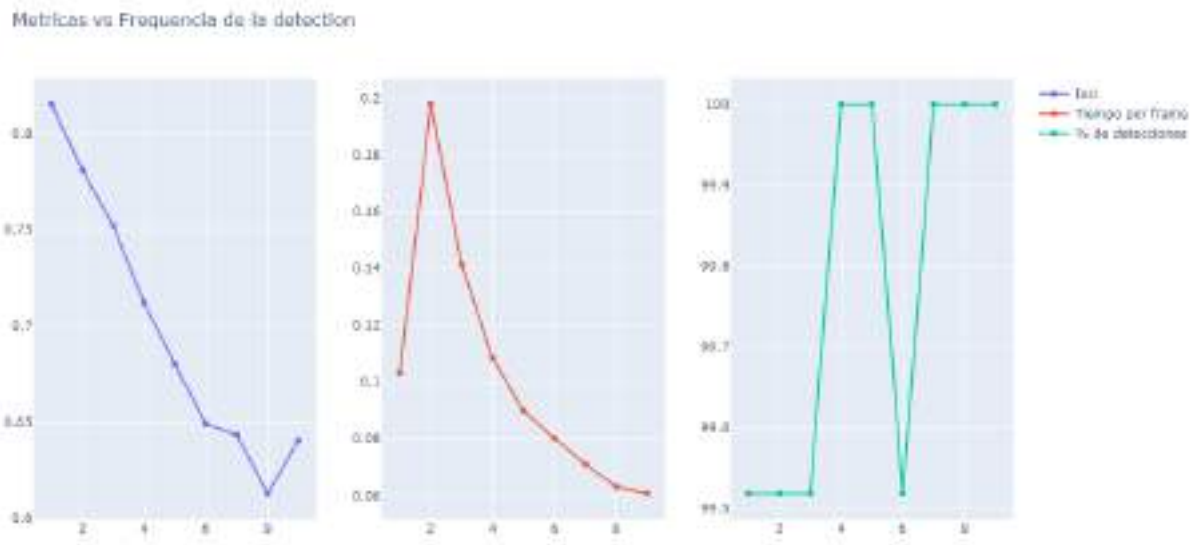


Figura 31: Evolución de las métricas de una secuencia con oclusión parcial.

De la primera gráfica de la figura 31 se puede observar que la métrica de acierto que considera la precisión de la detección (IoU) disminuye significativamente conforme esta se aplica cada un número mayor de imágenes. Cabe destacar que el valor obtenido para la frecuencia 1 indica una precisión alta de la detección. En cuanto al tiempo de ejecución por imagen, se aprecia que presenta una caída con tendencia exponencial conforme la detección se realiza cada un número mayor de imágenes.

Sin embargo, vemos que el tiempo correspondiente a únicamente realizar detecciones en todas las imágenes es menor que las primeras frecuencias sucesivas. No es hasta a partir de 4 imágenes de tracking consecutivo cuando este tiempo se reduce por debajo de este primer valor. Esto es debido fundamentalmente al tiempo empleado en definir los trackers en el modelo. Por último, también es conveniente mostrar en cuantas imágenes se ha registrado un objeto en cada caso. Vemos que en este ejemplo el algoritmo del detectron2 es capaz de detectar el objeto correctamente en un 99.5 por ciento de las imágenes. Por tanto, la aplicación del algoritmo de tracking en este sentido no es de gran utilidad salvo que se busque pura velocidad de procesado sin atender tanto a la precisión de la detección, mientras esta permita localizar al objeto.

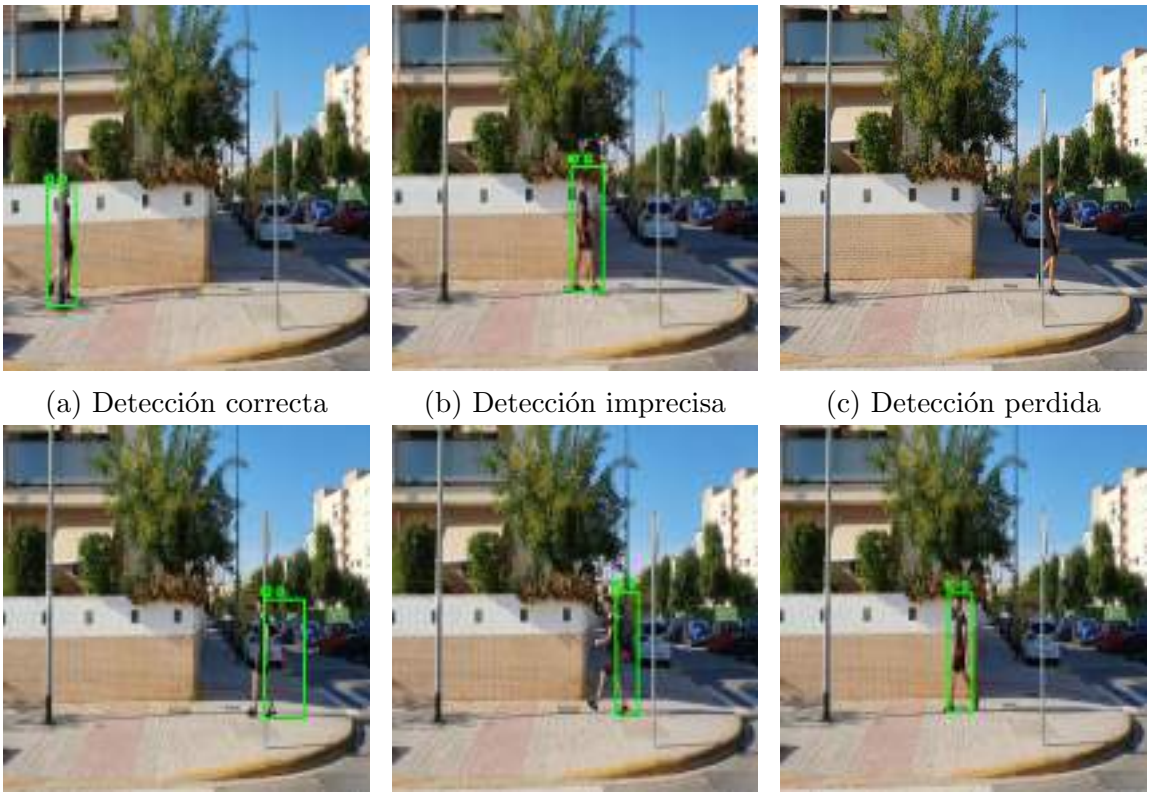
Una peculiaridad de este caso es que, debido al fondo que contenía una alta variedad de figuras y formas y la baja calidad de las imágenes hemos tenido que optimizar el umbral de confianza de la detección para evitar detecciones erróneas como vemos en la figura 32. En ella se puede observar como se identifica la parte de atrás de un coche como una persona de manera recurrente(hasta 12 veces a lo largo de la secuencia) en la subfigura 34a, mientras todavía hay imágenes en las que no se detecta la persona objetivo como se aprecia en 34c.

(a) Detección incorrecta. (b) Persona no detectada.

Figura 32: Ejemplos de detecciones erróneas para un umbral de confianza de 0.7

Vemos también en la figura 33 algunos ejemplos que nos ayudan a comprender

el comportamiento del modelo. Se puede observar que realizando la detección en todas las imágenes, la precisión de esta es significativamente alta, incluso en casos de oclusión parcial. También se aprecia en la subfigura 33b como el caso donde el fondo es difuso, los márgenes de la detección pueden fallar ligeramente. Por el contrario, se puede observar también en la figura 33 los problemas del algoritmo de tracking para lidiar con la oclusión parcial e incluso con la presencia de objetos que se puedan confundir con la persona sobre la que se realiza seguimiento, como vemos en la subfigura 33e.



(a) Detección correcta (b) Detección imprecisa (c) Detección perdida

(d) Tracking impreciso (e) Tracking impreciso (f) Tracking correcto

Figura 33: Ejemplos de detección a lo largo de una secuencia para distintas frecuencias.

A pesar de estos defectos, cabe decir que la precisión del algoritmo de tracking para imágenes con fondos bien definidos es significativamente buena incluso para frecuencias altas. Vemos en la figura 34 un ejemplo de esto para una frecuencia de detección de cada 9 imágenes.

(a) Imagen inicial. (b) Imagen intermedia. (c) Imagen intermedia.

Figura 34: Aplicación del modelo para frecuencias altas en zonas donde no hay oclusión.

Puesto que en este caso los ejemplos gráficos no son muy representativos recalculamos la métrica de IoU únicamente para esta zona, como podemos observar en la figura 35.

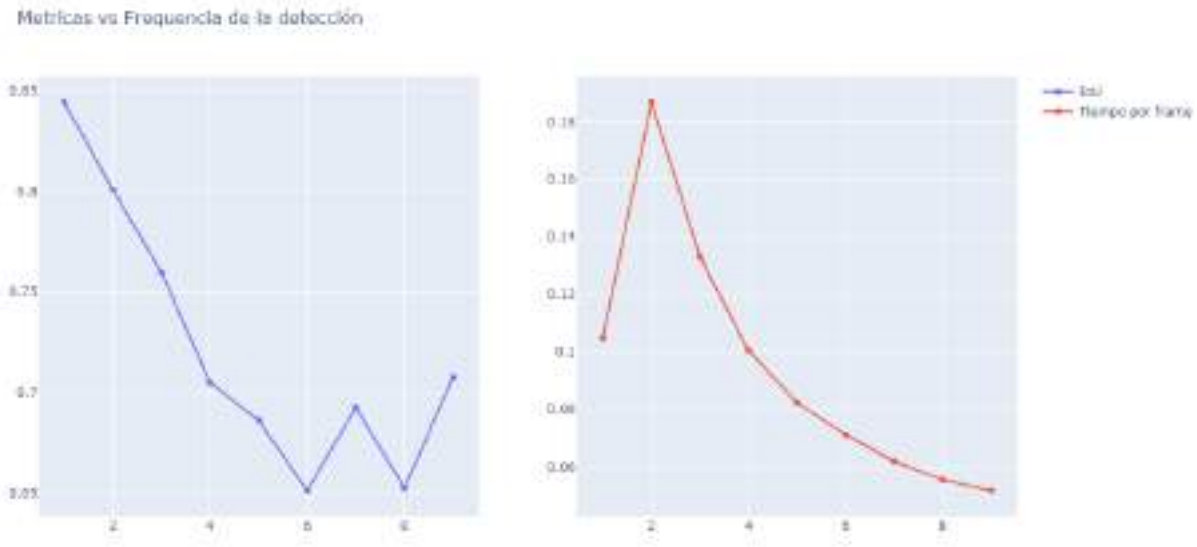


Figura 35: Evolución de los valores de las métricas para una zona sin oclusión..

Obteniendo **un valor de 0.708** y un tiempo de procesado de 0.0518 para dicha

frecuencia. Además se aprecia que, al contrario que en el caso mostramos en la figura 31 aquí el valor de la métrica IoU del modelo no llega a bajar de 0.65, siendo cercano a 0.7 en dos resultados de frecuencia elevada (7,9). Por tanto podemos asumir que en casos sin oclusión sería factible aplicar este modelo, aumentando la velocidad de procesamiento al doble respecto a las detecciones únicas.

Detección única con oclusión "total"

Entendemos con oclusión total aquí la imposibilidad de detectar a la persona objetivo en dicho frame. Aunque a simple vista no se produzca una oclusión total al uso de la persona, ya que esto implicaría un alto número de imágenes en las que no se podría realizar ni tracking ni detección debido a la frecuencia de muestreo.

Mostramos los valores obtenidos para las métricas en este caso en la tabla 3.

Frecuencia de detección	Métrica IoU	Tiempo por imagen (s)	Detecciones correctas(%)
1	0.708	0.107	90.1
2	0.694	0.94	92.7
3	0.667	0.137	93.7
4	0.640	0.111	97.3
5	0.614	0.0928	96.4
6	0.593	0.0832	93.7
7	0.571	0.0712	95.5
8	0.541	0.0636	99.1
9	0.496	0.0634	96.4
10	0.517	0.0549	96.4
11	0.425	0.0501	95.5
12	0.495	0.0480	100
13	0.493	0.0466	97.3
14	0.487	0.0420	100

Tabla 3: Métricas de error y tiempo de procesamiento en función de la frecuencia de la detección en el caso con oclusión parcial.

En este caso vemos como, al aumentar el número de imágenes cada cuanto se aplica la detección crece de manera considerable el porcentaje de detecciones . Esta es la mayor diferencia con el caso anterior, ya que aquí la oclusión es más significativa y afecta a un mayor número de imágenes, llegando hasta un 10 por ciento de detecciones perdidas. La otra principal diferencia se trata del valor de la métrica de acierto, significativamente más bajo debido al mayor número de detecciones perdidas y de

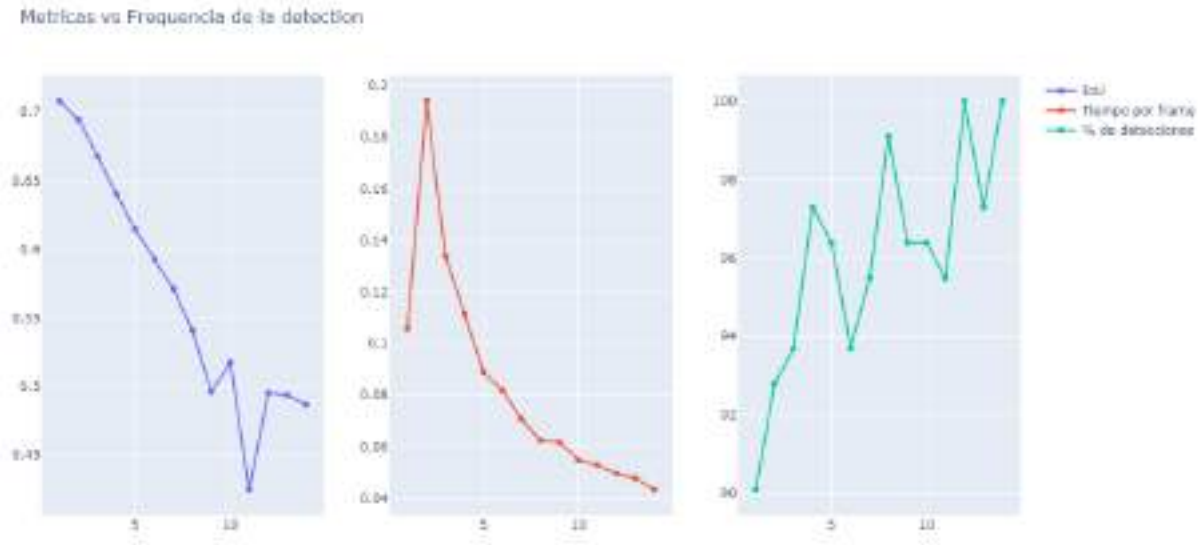


Figura 36: Evolución de las métricas del primer vídeo etiquetado que se ha generado.

imágenes donde se da una oclusión parcial.

Vemos en la figura 37 de manera gráfica las diferencias entre los resultados en la misma secuencia para los distintos valores de la frecuencia de detección. En la última fila tenemos la detección presente en todas las imágenes, en el cual la precisión es obviamente la más alta pero al haber oclusión la cantidad de imágenes en las que no se detecta, según el umbral que optimizamos para evitar detecciones erróneas, es alto. Vemos un ejemplo de esto en la imagen izquierda de la última fila de la figura 37, donde no se ha podido detectar al objetivo, en contraposición a los dos casos de las filas superiores. Por otro lado vemos que nuestro tracker al basarse en la correlación no es capaz de seguir al objeto si el cambio visual es significativamente alto, pero si disminuye el número de imágenes en el que el objeto no está localizado para frecuencias relativamente bajas (5 en el caso de la fila intermedia). Para frecuencias más altas, como en la fila superior, vemos que al pasar un objeto que tapa la persona a seguir la posición se queda estanca en dicho objeto, como se aprecia en la subfigura superior central.

Sin embargo, la diferencia si que llega a ser significativa en casos donde el objetivo no se ve solapado totalmente. Como mostramos en la figura 38 a continuación.

Figura 37: Detección del modelo de tracking clásico a lo largo de una secuencia para los valores 8-5-1 de la frecuencia de detección por fila, respectivamente.

Por último, aquí apreciamos como ante este tipo de oclusión el algoritmo de tracking optimizado a la frecuencia óptima para este caso consigue, a pesar de la ligera

(a) Frame inicial. (b) Frame Intermedio. (c) Frame Final.

(d) Frame inicial. (e) Frame Intermedio. (f) Frame Final.

Figura 38: Deteccion del modelo de tracking clásico a lo largo de una secuencia.

pérdida de precisión, mantener detectado al objetivo durante todas las imágenes.

6.2. Modelo de redes recurrentes

En esta sección hablaremos de todo el proceso de entrenamiento que realizamos en el apartado 6.2.1, explicando los pasos y las conclusiones a las que llegamos durante este proceso que nos llevaron a probar una aproximación distinta hasta llegar al modelo definitivo. También mostraremos en el apartado 6.2.2 los resultados obtenidos de aplicar dicho modelo a ambos vídeos y las métricas de error.

6.2.1. Entrenamiento del modelo

Para entrenar este modelo primero generamos todas las detecciones del detectron2 en el vídeo seleccionado del MOT [21]. Una vez hecho esto intentamos entrenar un modelo secuencial sencillo con varias capas LSTM de la manera explicada en el apartado 5.2. Sin embargo, no conseguimos a partir de estos datos que la red aprendiera la lógica del modelo.

Después de analizar las etiquetas de nuestro vídeo nos dimos cuenta de que había bastantes imágenes en los que no etiquetaba detecciones que el detectron2 sí que generaba, lo cual nos llevó a pensar que los datos podían ser incorrectos. Tras realizar comprobaciones llegamos a la conclusión de que en efecto, en casos de objetos cercanos a los límites de la imagen el etiquetado fallaba. Intentamos subsanar esto pero sobre todo para el caso de nuestro modelo de entrenamiento, este efecto sumado a la complejidad del problema y el bajo número de datos dificultaba un correcto entrenamiento.

Por esta razón decidimos servirnos de los datos etiquetados por nosotros mismos para el modelo clásico para entrenar la red para los casos más simples y significativos.

En primer lugar, para iniciar el entrenamiento convenía definir el conjunto de datos utilizado para cada uno de los fines. Para ello decidimos utilizar definitivamente los dos datasets de los vídeos etiquetados, para tener dos tendencias y rangos de valores distintos y evitar que la red sobre ajuste en base a los parámetros de sesgo, como comprobamos que ocurre al entrenar con un solo conjunto.

En cuanto al modelo en primer lugar tratamos de implementar la métrica IoU como función de pérdida customizada con el vector con las 4 entradas, pero debido a la pequeña cantidad de datos y a los problemas derivados de la naturaleza de esta no fue posible conseguir resultados óptimos a partir de ella, por lo que la dejamos únicamente como métrica de evaluación y no función de perdida. Posteriormente, empezamos a entrenar el modelo con todas las entradas con una métrica de perdida cuadrática. En este caso obtuvimos buenos resultados para el error y la mayoría predicciones tanto en test como en entrenamiento eran coherentes pero parecía haber tendencias incorrectas y un sobre ajuste generalizado que daba lugar a resultados incorrectos e ilógicos, sobre todo en el set de test. Para evitar que esto pasara y que las predicciones de los ejes sean influenciadas por las entradas de otros ejes separamos el entrenamiento para cada uno de ellos.

A continuación, entrenando las variables por separado, obtenemos un ajusto mucho mejor, mostrando un error cuadrático tanto en entrenamiento como en test comprendido entre la decena y las 5 decenas (para rangos de valores del orden de las centenas). Sin embargo, al analizar los datos de salida seguían habiendo errores que dan a entender que la red no está pudiendo aprender la sencillez del problema. Observando más detalladamente los resultados vemos que los casos más relevantes de errores se dan cuando intentamos predecir un frame donde alguna de las últimas detecciones están perdidas. Analizándolo llegamos a la conclusión de que una única red no puede optimizar los pesos para entradas con una información cambiante, y, por tanto, si el ajuste predominante se basa en usar todas las últimas n detecciones para este, si una o más se encuentra perdida este fallará.

Considerando esto y teniendo en cuenta que nuestro conjunto de datos llega a tener 5 detecciones perdidas consecutivas, implementamos una solución en la que el modelo se base en predecir la posición para la imagen siguiente dadas las últimas n detecciones, y si una de ellas está perdida se utiliza el valor de la predicción correspondiente para que, recursivamente se pueda aplicar este a secuencias con un número indeterminado de detecciones perdidas. Teniendo en mente esta nueva lógica extraemos todas las secciones con detecciones perdidas de los data sets y sobre los conjuntos restantes dividimos entre entrenamiento (80 %) y validación/test (20 %).

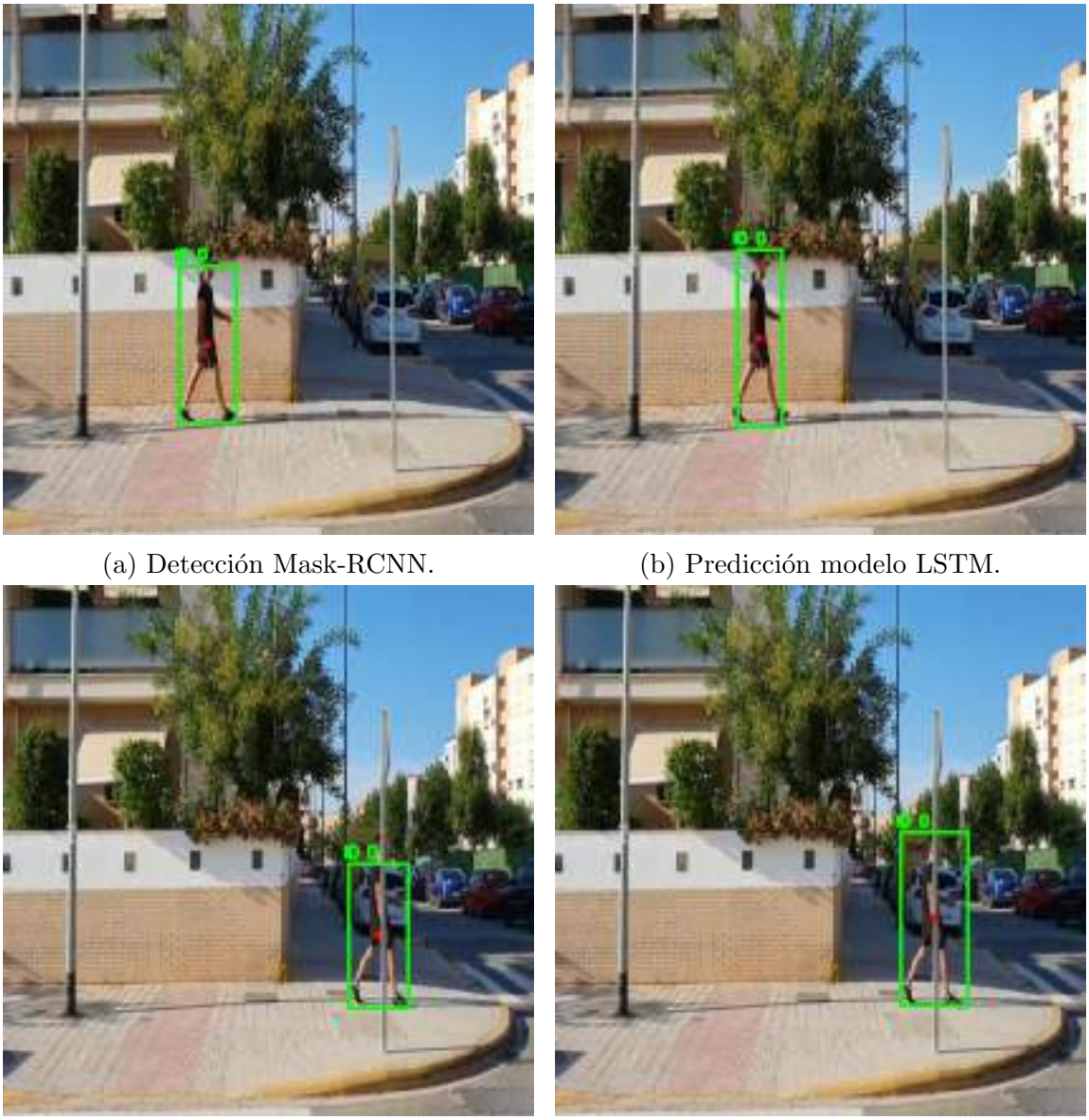
Por último, una vez hecha la división, entrenamos la red asegurándonos en base al conjunto de test que no se esté produciendo un sobre-entrenamiento. Como explicamos en el apartado 5.2, al solo usar la posición de las detecciones, aunque la predicción pueda, y de hecho mejore la predicción en diversos casos, no podemos tener certeza de esto sin analizar el vector de características y otras características propias de la imagen. Por esto, el foco principal de la capa recurrente es sustituir a la detección cuando esta falle, y la evaluación más relevante se dará en dichas imágenes.

6.2.2. Evaluación cuantitativa y cualitativa

Mostramos aquí algunos de los resultados más relevantes que se han obtenido para cada uno de los tipos de vídeo.

Detección única con oclusión parcial

Primero mostramos en la figura 39 un par de casos donde la predicción mejora



(a) Detección Mask-RCNN. (b) Predicción modelo LSTM.

(c) Detección Mask-RCNN. (d) Predicción modelo LSTM.

Figura 39: Ejemplos de predicción del modelo de redes recurrentes y detecciones.

la detección, ya que ya sea el fondo o el primer plano, hay objetos que dificultan la visibilidad del objeto. Esto sucede en distintos casos, sin embargo no podemos confiar con la profundidad del entrenamiento que por regla general las predicciones supongan

una corrección a las detecciones sin atender a otras características de la imagen. Si este tuviera un acierto mucho mayor debido a la robustez del entrenamiento, o si se incluyera en las entradas al modelo el mapa de características que diera a entender que la detección no está completa la fiabilidad de estas sería mayor, de nuevo, si se tuviera la cantidad suficiente de datos etiquetados.

Por ello, aunque nuestro modelo de predicción simple, siguiendo la tendencia de los datos, en ocasiones mejora la detección existente, en general no consigue un mayor desempeño, como vemos en los valores de las métricas evaluadas a continuación. Esto nos lleva a la decisión de complementar las detecciones con estas predicciones únicamente cuando el objeto no sea detectado en una imagen.

- Métrica IoU de detecciones y predicciones: 0.823
- Métrica IoU únicamente de predicciones: 0.707
- Tiempo de procesamiento de detecciones y predicciones: 0.101 s

Si comparamos con los valores obtenidos en el apartado del modelo clásico para detecciones en todas las imágenes observamos que el valor de la métrica ha aumentado muy ligeramente, debido a que en este caso solo había una predicción perdida. Por otro lado, confirmamos que la precisión de nuestro modelos de predicción únicamente en sensiblemente menor que las detecciones. Mostramos a continuación en la figura 40 una de las principales tendencias incorrectas que hemos observado de dicho modelo para poder explicar entenderlo mejor. Como se observa, la predicción de nuestro modelo hace cada vez más estrecha la caja de la persona seguida lo cual es un claro mal funcionamiento, pero analizándolo de manera lógica vemos que en las imágenes previas de la secuencia, el margen derecho de la persona detectada no presenta variación, lo que hace que la progresión lógica del modelo, siguiendo sobre todo la secuencia para pocas imágenes, reproduzca esta tendencia. Para ser capaces de considerar estos casos y llevar a cabo predicciones adecuadas deberíamos entrenar el modelo con una gran cantidad de casos, de manera que pueda ajustar mejor la aceleración/desaceleración del movimiento, así como implicaciones lógicas entre la variable del margen izquierdo y derecho de ambos ejes. Para todo esto tener la información del mapa de características y demás información espacial de la imagen también sería de utilidad.

Figura 40: Ejemplos de predicción del modelo de redes recurrentes únicamente.

A continuación, para finalizar la documentación gráfica del modelo para este vídeo, mostramos el comportamiento a lo largo de la secuencia donde encontramos la detección perdida, que se encuentra en 41c. Siendo esta la única diferencia con la aplicación única de detecciones en todas las imágenes. La unión de estas dos aproximaciones nos permite tener localizado al objetivo en toda la secuencia.

(a) Imagen inicial. (b) Imagen intermedia. (c) Imagen final.

Figura 41: Ejemplos de predicción del modelo de redes recurrentes y detecciones a lo largo de una secuencia.

Detección única con oclusión "total"

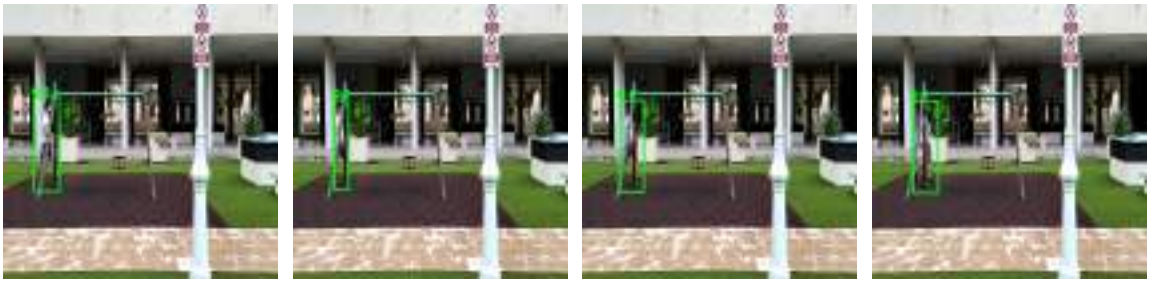
Empezamos este apartado mostrando las métricas obtenidas para la aplicación de las detecciones complementadas por las predicciones del modelo, y de solo estas últimas.

- Métrica IoU de detecciones y predicciones: 0.754

- Métrica IoU únicamente de predicciones: 0.658
- Tiempo de procesamiento de detecciones y predicciones: 0.116 s

Estos resultados suponen una mejora significativa respecto a los resultados obtenidos de aplicar únicamente la detección a cada imagen, recogido en la tabla 3. Esto se debe al alto número de detecciones perdidas, cercano al 10 %, el cual además ahora se reduce a 0 %. Por otro lado vemos que la precisión de las predicciones, aún en este caso siguen siendo sensiblemente menor que la que se obtiene con las detecciones.

Ahora vamos a mostrar gráficamente los resultados para el vídeo con oclusión total, donde, debido al número de detecciones perdidas, puede tener más sentido el uso de este modelo. Puesto que ya hemos mostrado propiedades del modelo en el caso anterior aquí nos centramos en los casos donde había detecciones perdidas.



(a) Detección Mask-RCNN. (b) Predicción modelo LSTM. (c) Predicción modelo LSTM. (d) Predicción modelo LSTM.

Figura 42: Ejemplos de predicción del modelo de redes recurrentes y detecciones a lo largo de una secuencia.

Mostramos en la figura 42 una secuencia de imágenes donde en la primera de ellas 42a se ha podido realizar la detección, mientras que en las tres consiguientes, debido a la oclusión, no se ha detectado al objetivo y se ha aplicado el modelo de predicción. Observamos que se localiza a la persona con una precisión elevada durante 3 imágenes consecutivas, comprobando que, a pesar de los pocos datos con los que se contaba, la recursividad de las predicciones funciona y no introduce un error acumulado demasiado grande.

Para acabar la descripción gráfica vemos en 43 el otro caso de detecciones perdidas debido a la oclusión total, de manera más clara aquí. Las imágenes de los extremos

(a) Imagen inicial. (b) Imagen intermedia. (c) Imagen final.

Figura 43: Ejemplos de predicción del modelo de redes recurrentes y detecciones a lo largo de una secuencia.

de la secuencia se corresponden con las últimas detecciones correctas antes y después de que la persona a detectar se vea eclipsada prácticamente en su totalidad. En el centro podemos apreciar la predicción realizada, la cual, pese a que no muestra toda la precisión posible, localizada inequívocamente a la persona.

7. Comparativa de los modelos

Aunque ya hemos observado el desempeño de los distintos modelos por separado, en este apartado vamos a comparar directamente los modelos según el enfoque de la velocidad de procesado y la precisión de estos, en cada uno de los tres casos estudiados. Puesto que ya sabemos que en cuanto a precisión los modelos de detección y redes recurrentes muestran un mejor desempeño en cualquier caso, mientras que los métodos de tracking presentan una mayor velocidad de procesado a partir de cierta frecuencia, vamos a comparar con el primer valor de esta que suponga un menor tiempo de procesado, así como con el modelo que mejor ratio de ganancia de tiempo de procesado respecto a perdida de ganancia muestre en cada caso, poniendo un límite mínimo al valor de la métrica de acierto de 0.50 (mitad de la detección coincidente). Este criterio es arbitrario y podría escogerse otro según la tipología del problema a tratar.

Pero antes de entrar a evaluar cada caso en concreto, conviene mencionar que en cuanto a la métrica de porcentaje de imágenes en el que se detecta al objeto, el único modelo que nos garantiza que podamos mantener identificado al objeto cuando la detección falla es el basado en redes recurrentes, ya que, por más que se aumente la frecuencia de detección siempre habrá una probabilidad no nula de realizar esta en una imagen donde falle. Esto ya puede ser un criterio muy significativo para discriminar entre ambos modelos.

7.1. Oclusión parcial

Siguiendo el criterio mencionado anteriormente, vamos a comparar el modelo de redes recurrentes con el modelo de tracking clásico para las frecuencias 5 y 9, tanto visualmente como cuantitativamente.

Mostramos en la tabla a continuación el modelo más rápido y el más preciso y las diferencias porcentuales de los otros dos respecto a este, para tener una idea de la discrepancia entre estos.

Métrica	Detección + Redes recurrentes	Modelo tracking clásico (F5)	Modelo tracking clásico (F9)
IoU	**0.823**	17.4 %	22.2 %
Tiempo de ejecución	71.3 %	47.9 %	**0.0607**

Tabla 4: Comparativa entre el desempeño de los modelos, mostrando el valor del mejor según cada métrica y la discrepancia relativa respecto al resto

Aquí se ve de manera clara como, a pesar de la pérdida de precisión por parte de los modelos clásicos, la ganancia porcentual de velocidad de procesado es mayor. Pero todos estos valores se tratan de medias para todas las imágenes, vamos a realizar un análisis más detallado del acierto en las distintas imágenes para poder tener una visión más clara a nivel individual.

Figura 44: Distribución del valor de la métrica IoU para los distintos modelos.

Podemos observar en la figura 44 como el valor de la métrica para el modelo de detecciones y redes recurrentes está significativamente más colimado que los casos de tracking clásico. Sin embargo estos muestran una mediana mayor que la media y una densidad de valores máxima en torno al 0.8 en el caso de frecuencia 5 y un poco más baja en el caso de frecuencia 9. También se observa que para frecuencias más altas los valores significativamente bajos, son más frecuentes, pero la densidad por debajo de 0.4 se reduce sensiblemente en ambos casos. Por tanto confirmamos así que el

acierto no toma valores extremos y hay pocos casos por debajo de 0.2, en concreto un 1.4 % para el caso de frecuencia 5 y 5.7 % en el de frecuencia 9. Para interpretar esto de manera gráfica vamos a mostrar un ejemplo con este nivel de acierto.

(a) Frecuencia 5. (b) Frecuencia 9. (c) Redes recurrentes.

Figura 45: Ejemplos gráficos del acierto en el percentil 10 para los modelos clásicos y el modelo basado en redes recurrentes.

En ambos ejemplos del modelo clásico se observa que dichos casos de detecciones con acierto bajo están relacionados con oclusión o cambios en el fondo de la imagen, siendo más frecuentes y notorios para frecuencias más altas, ya que se prolongan durante más imágenes. Por su parte, en la subfigura 45c se puede afirmar que hasta para las detecciones con un acierto en el rango más bajo la persona está perfectamente localizada. Aquí hemos comparado distintas imágenes de la secuencia simplemente buscando que se encontraran en el mismo percentil para cada distribución, dando así una idea gráfica de los rangos de acierto más bajos. Vamos a tratar en el siguiente ejemplo de comparar imágenes en el mismo tramo de la secuencia.

Por último, comparamos ejemplos de acierto situados alrededor del valor con mayor densidad de cada modelo para ver el desempeño típico.

Resulta interesante comprobar como la diferencia gráfica entre los modelos aquí no es fácilmente apreciable, ya que en todos casos con este nivel de acierto la persona está perfectamente localizada. La principal diferencia es que los márgenes en el modelo basado en detecciones y redes recurrentes están mejor ajustados al perfil mientras que en los otros casos hay un ligero margen. Cabe destacar que para los modelos de tracking clásico este acierto medio se da principalmente en zonas despejadas.

(a) Frecuencia 5. (b) Frecuencia 9. (c) Redes recurrentes.

Figura 46: Ejemplos gráficos del acierto más frecuente para cada modelo.

7.2. Oclusión total

Aquí la diferencia del desempeño entre los modelos al incrementar la frecuencia es más evidente, siendo el más preciso de nuevo el modelo de redes recurrentes, pero esta vez con un 15.5 % de diferencia respecto al primer modelo más rápido de tracking, de frecuencia 4, llegando hasta más del doble (31.7 %) de diferencia para el caso de frecuencia 10.

Métrica	Detección + Redes recurrentes	Modelo tracking clásico (F4)	Modelo tracking clásico (F10)
IoU	**0.757**	15.5 %	31.7 %
Tiempo de ejecución	107.6 %	107.61 %	**0.0549**

Tabla 5: Comparativa entre el desempeño de los modelos, mostrando el valor del mejor según cada métrica y la discrepancia relativa respecto al resto

Observamos en este caso que para la frecuencia 10 la velocidad de procesado llega a ser más del doble que para los otros dos modelos, vamos ahora a ver la distribución de acierto para los tres modelos en este caso.

Al contrario que el caso anterior, a pesar de estar igualmente bastante concentrado, encontramos una cola hacia valores bajos de la métrica de acierto. Por otro lado vemos que para la frecuencia 10 la varianza de los valores es muy alta, estando más simétricamente distribuidos. Pero vamos a mostrar visualmente como antes que suponen estos valores para el rango incluido en el percentil 10. La primera evidencia

Figura 47: Distribución del valor de la métrica IoU para los distintos modelos.

de la diferencia en este caso es que para la frecuencia 10 el percentil 10 se encuentra en valores de acierto menores a 0.04, mientras que para el caso de frecuencia 4 este umbral aumenta hasta 0.35, y en el modelo de redes recurrentes baja ahora hasta 0.56, ya que entra más en juego el papel de las predicciones del modelo, debido al mayor número de detecciones perdidas.

(a) Frecuencia 4. (b) Frecuencia 10. (c) Redes recurrentes.

Figura 48: Ejemplos gráficos del rango de acierto más frecuente para cada modelo.

Este caso permite una mejor comparación de los distintos modelos, como se ve en la figura 48, donde se recoge el comportamiento posible del modelo de tracking en casos de oclusión total. Como se ve en el ejemplo 48a puede darse un fallo de la detección y que por tanto no se registre la persona directamente, o como se ve en el caso 48b puede corresponder con la etapa de tracking y mostrar una precisión significativamente baja por el error de este al lidiar con la oclusión. Por otro lado vemos que, aún siendo uno de los casos del percentil de acierto más bajo de este, el modelo basado en redes recurrentes y detección es capaz de identificar al objetivo aún cuando este no se ha podido detectar.

Ahora vamos a comparar como antes la apariencia visual de los resultados en el rango más frecuente de estos modelos.

(a) Frecuencia 4. (b) Frecuencia 10. (c) Redes recurrentes.

Figura 49: Ejemplos gráficos del rango de acierto más frecuente para cada modelo.

Por último, y debido a que se aprecia un funcionamiento significativamente más parejo entre los distintos modelos en zonas despejadas vamos a representar la distribución del acierto para la subregión seleccionada del ejemplo de oclusión parcial con estas características.

Figura 50: Frecuencia 4.

Figura 51: Distribución del acierto para zonas sin oclusión.

Como vimos ya en el apartado de resultados, el acierto en este caso es más cercano al de las detecciones, mostrando una densidad más concentrada, sobretodo para valores altos de la frecuencia. Podemos resumir la comparativa realizada aquí en los siguientes puntos para cada modelo, empezando por el de tracking.

1. Menor precisión general.
2. Mayor velocidad de procesado.
3. Capacidad de gestionar la oclusión parcial.
4. Incapacidad de gestionar casos de oclusión total.
5. Capacidad de localización del objetivo de manera continua a lo largo de la secuencia en casos sin oclusión.

Para el caso del modelo basado en detecciones y redes recurrentes:

1. Mayor precisión general.
2. Menor velocidad de procesado.
3. Capacidad de gestionar la oclusión parcial.
4. Capacidad de gestionar casos de oclusión total para varias detecciones perdidas consecutivas, hasta 5 en nuestro caso.
5. Capacidad de localización del objetivo de manera continua a lo largo de la secuencia en cualquier caso.

8. Conclusiones

Después de hacer una extensa revisión de los distintos métodos de detección en imágenes y en vídeos que conforman el estado del arte, podemos afirmar que, como mencionábamos desde la introducción, es un campo apasionante que aún tiene un gran horizonte de desarrollo por delante. Nos centraremos aquí en recapitular la información más relevante y descriptiva de las distintas partes tratadas en este trabajo. Empezando por el ámbito de la detección en imágenes comprobamos que se encuentra en un estado significativamente avanzado, con modelos y tecnologías muy optimizadas para dicho fin, pudiendo procesar un gran número de imágenes por segundo (1000 en nuestro sistema).

Centrándonos en la detección de personas cabe destacar que la red utilizada (detectron2), siendo una de las herramientas más punteras, no presenta una gran robustez, pareciendo centrarse en rasgos muy generales y no pudiendo discernir entre objetos parecidos a personas y personas reales, como mostramos en las figuras 20 y 21. Esto lleva también a detecciones erróneas, y objetos que no se detectan, como se ha podido ver de forma cuantitativa en la tabla 1.

Poniendo el foco en la detección en vídeo hemos comprobado que, al contrario que en el caso de imágenes, no hay un marco unificado en cuanto al estado del arte, ni herramientas entrenadas de referencia. También hemos podido comprobar que el acceso a datos etiquetados de calidad de forma gratuita es una tarea casi imposible, lo cual dificulta el entrenamiento de modelos propios.

A pesar de esto, hemos obtenido resultados satisfactorios, pudiendo explorar las limitaciones de estos métodos. Gracias a ello estamos en posición de poder concluir que no existe un modelo único que muestre mejor desempeño en todos los aspectos (velocidad de procesado y precisión), pudiendo elegir uno u otro dependiendo de las características del problema. Sin embargo, sí que hemos comprobado que para conseguir un funcionamiento correcto en casos generales conviene utilizar métodos con mayor flexibilidad como el basado en redes recurrentes. Por tanto, de cara a mejorar el desempeño de esta rama de la visión artificial a futuro, conviene focalizar los esfuerzos en algoritmos que permitan establecer una lógica temporal entre imágenes, complementando así las detecciones en imágenes individuales y no arrastrando los errores cometidos en estas.

Analizando las limitaciones y errores observados en el caso del modelo basado en redes recurrentes podemos afirmar que la mayoría se deben a que no se disponía de la cantidad de datos necesarios para un modelo óptimo que pudiera generalizar de manera correcta el comportamiento. Debido a esto nos hemos basado solo en la posición de los objetos, aspirando únicamente a predecir la tendencia extrayendo información sobre los movimientos habituales. Aún así, debido a que el movimiento siempre tendrá una componente aleatoria no se puede basar la predicción únicamente en esta información. Las posibles soluciones a este problema las desarrollaremos en el siguiente apartado 9 referentes al trabajo futuro para mejorar el modelo.

9. Trabajo futuro

Como hemos comentado en el apartado de conclusiones este es un campo con un amplio horizonte desarrollo y por ello durante el transcurso del proyecto han surgido una cantidad significativa de mejoras o variaciones que se podrían implementar. En cuanto al modelo clásico una de las mejoras más significativas sería emplear un método de tracking basado en deep learning como los mencionados en el apartado 3.2.2. La complicación principal es la falta de algoritmos de este estilo pre-entrenados. Por ello deberíamos entrenar nuestro propio modelo con la complejidad que ello conlleva. Para hacer esto es necesario extraer el mapa de características de las distintas detecciones, por ello es de vital importancia comprender el funcionamiento interno de las distintas capas del detectron2. Con vistas a implementar esta mejora llevamos a cabo la extracción de dichas características, como se detalla en el apéndice B. Esto conseguiría tener un método de detección y seguimiento en vídeo más potente que con los métodos clásicos y a su vez significativamente más rápido y preciso que detectar en todos las imágenes.

Ahora bien, puesto que nuestro objetivo es incrementar al máximo la precisión de la detección e identificación en vídeos, debemos centrarnos en el desarrollo de un método que aprenda la lógica temporal. Para ello hay que avanzar en el caso de nuestra red LSTM. En este ámbito la posibilidad de mejora es enorme.

Primero se podría trabajar en un modelo que directamente asocie las detecciones entre las distintas imágenes y prediga su próxima posición. Para ello además de entrenar el modelo con una métrica de error para la precisión de las detecciones habría que introducir una métrica de asociación, encontramos en [20] una propuesta de modelo de este tipo, con un ejemplo teórico de como implementar una métrica de estas característica y la arquitectura de la red necesaria. Esto permitiría ampliar el modelo a un número indeterminado de predicciones sin mayor esfuerzo.

Centrándonos en la predicción para un objetivo único cabe decir que en nuestro caso hemos utilizado un modelo únicamente basado en las últimas detecciones para predecir la siguiente. Pero si nuestro objetivo es, además, mejorar la predicción en cada imagen, estaríamos eliminando la información de la detección más relevante. Por ello, sería interesante utilizar dos modelos, uno entrenado con las últimas n imágenes incluyendo la actual con el fin de mejorar las detecciones existentes, y el que hemos implementado, basado en las últimas detecciones sin considerar la actual,

para predecir esta.

Por otro lado, como hemos dicho, se podría introducir información de la imagen. Las características de esta que podrían mejorar la predicción se deberían determinar de manera empírica, pero por lógica el mapa de características sin duda sería de utilidad. También podría serlo la máscara predicha por el detector(mask-RCNN) para determinar si la detección está completa, el tamaño de esta e incluso la confianza de la detección, todo con el mismo fin.

Obviamente para incluir cualquiera de estas propiedades en el modelo se necesitaría una cantidad mayor de datos con una alta variedad, algo que costaría un esfuerzo considerable y quedaba fuera del abarque de este proyecto. Sin embargo, incluimos en el apéndice B un ejemplo de como extraer el vector de características a partir de las funciones que tiene desarrolladas detectron2 para recuperar salidas intermedias de las distintas capas.

A. Implementación del Multiprocesado

Mostramos una versión resumida de las partes del código que sirven a este propósito para que sea más aclaratorio, pero posteriormente explicaremos en más detalle el funcionamiento.

```
# Define input and output Queues lists
inputQueues = []
outputQueues = []


# Define the tracking process for each detection
iq = multiprocessing.Queue()
oq = multiprocessing.Queue()
inputQueues.append(iq)
outputQueues.append(oq)
if __name__ == "__main__":
      # spawn a daemon process for a new object tracker
  p = multiprocessing.Process(target=track.tracker,
                      args=(counter,bb,  frame, iq, oq))
  p.daemon = True
  p.start()



# Execute the tracker for each detection
# in the new frame
for iq in inputQueues:
      iq.put(frame)

for oq in outputQueues:
      # grab the updated bounding box coordinates for the
      ((counteri,startXi, startYi, endXi, endYi)) = oq.get()
```

Aunque este código no sea especialmente claro la lógica de la función de multiprocesado es bastante sencilla y consta de tres pasos:

1. Primero en la etapa de detección definimos un inputQueue y un outputQueue para cada detección, y definimos e iniciamos para cada una de ellas un proceso de multiprocesado, el cual tiene como argumentos la función de tracker a ejecutar y los argumentos de esta, incluyendo el identificador de la detección.

2. A continuación, en las etapas de tracking, a cada objeto que hemos definido en la etapa anterior se le pasa mediante el método iq.put la nueva imagen, desencadenando esto la ejecución de esta con este nuevo frame como entrada.

3. Por último, una vez ejecutada la actualización de la posición de cada persona mediante el tracker se recuperan todas ellas con el método oq.get y se actualizan los correspondientes registros basándose en el id de la detección.

Mostramos a continuación también como afecta esto a la función de tracking para ayudar a entender el funcionamiento.

```
def tracker(counter,box,  frame, inputQueue, outputQueue):
    # construct a dlib rectangle object from the bounding box
    # coordinates and then start the correlation tracker
    t = dlib.correlation_tracker()
    rect = dlib.rectangle(box[0], box[1], box[2], box[3])
    t.start_track(frame, rect)

# loop indefinitely -- this function will be called
# as a daemon  process so we don't
    # need to worry about joining it
    while True:
        # attempt to grab the next frame from the input queue
        rgb = inputQueue.get()
        # if there was an entry in our queue, process it
        if rgb is not None:
            # update the tracker and grab the position
            # of the tracked object
            t.update(rgb)
            posit = t.get_position()
            # unpack the position object
            startX = int(posit.left())
            startY = int(posit.top())
```

```
                endX = int(posit.right())
                endY = int(posit.bottom())
                # add the label + bounding box coordinates
                #  to the output queue
                outputQueue.put((
                  (counter,startX, startY, endX, endY)))
```

Vemos que se define el correlation tracker de la librería dlib y se inicia con la posición inicial de la persona y la imagen inicial. Después esta función se ejecuta de manera permanente a la espera de una nueva imagen que se recupera aquí con el inputQueue.get(), y por tanto al pasarle en el código principal la nueva imagen se ejecuta en ella la actualización del tracker y se devuelve la salida, al igual que en identificador (counter) con el que se definió la función, a través de la función put() del objeto outputQueue

B. Extracción del vector de características

Mostramos aquí como extraer el vector de características de una detección gracias a los distintos backbones que hemos definido en el apartado 4.1, ya que es una información que cuesta localizar en la documentación de detectron2 y, aunque al final no hayamos encontrado factible utilizarla en los modelos, puede resultar útil para la mejorar de estos.

```
# We will use the DefaultPredictor
from detectron2.engine import DefaultPredictor

# Define what model we want to use
# Mask R-CNN in this case
cfg = get_cfg()

cfg.merge_from_file(model_zoo.get_config_file(
  "COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))

# set threshold for this model
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5

# Find a model from detectron2's model zoo.
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(
  "COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")

# Define the predictor
predictor = DefaultPredictor(cfg)


# Load an example image:
image = cv2.imread("./fotoprueba.jpg")
height, width = image.shape[:2]

# Transform the image to the accepted format
image2 = torch.as_tensor(
        image.astype("float32").transpose(2, 0, 1))
inputs = [{"image": image2, "height": height, "width": width}]
```

```
with torch.no_grad():
    image = model.preprocess_image(inputs)


# Get the features for each FPN scale (P2–P6)
features = model.backbone(image.tensor)

# Divide them in the correct format
mask_features_fpn =
        [features[f] for f in model.roi_heads.in_features]

# Obtain our target characteristics from the general ones
mask_features = model.roi_heads.mask_pooler(
  mask_features_fpn, [x.pred_boxes for x in [instances]])
```

Pasamos a clarificar cada término para que se entienda mejor tanto el proceso como las componentes que explicamos en el apartado 4.1. Primero necesitamos convertir el formato de la imagen a un tensor ya que es este el que utiliza el detectron en sus transformaciones internas que nosotros estamos reproduciendo. A partir de aquí recurrimos a funciones internas del modelo que procedemos a explicar.

- backbone: Recupera el resultado obtenido en la capa backbone, donde se aplican capas convolucionales a distintas escalas según los principios del Feature Piramid Network para extraer los mapas de características. En concreto se computan 6 escalas distintas, devolviendo un diccionario para cada una de estas escalas con tensores de 256 canales y dimensión decreciente dependiendo de la dimensión de la imagen.
- roi_heads.mask_pooler: La capa ROI Head es la encargada de extraer las propuestas de detecciones, y para ello extrae las regiones de interes de los mapas de características obtenidos antes. En esta clase tenemos integrados que permiten extraer las cajas deseadas de los mapas de características de manera análoga, devolviendonos así el vector de características deseado.

Conviene mencionar que en este caso, por consistencia, hemos mostrado como recuperar el mapa de características de la zona delimitada por la mascara del ob-

jeto, pero se podría, por ejemplo, recuperar el de la zona completa, con la función **roi_heads.mask_pooler**

Una vez entendido el papel de las distintas funciones de las que dispone la librería para recuperar salidas intermedias aclaramos que, como resulta evidente, el vector de características de las distintas escalas para la imagen se calcular dentro del código del detectron al llamar a la predicción, y con un sencillo cambio en el fichero **defaults.py** en el directorio detectron2.engine, en la definición de la clase DefaultPredictor podemos recuperarlo. Este vector se calcula en el método ' *_call_* ' de dicha clase y sirve con comentar el '[0]' de la linea 316. En el código de este proyecto ya hemos adaptado el funcionamiento para considerar este cambio.

Mostramos el código completo, con un ejemplo real en el enlace de GitHub adjuntado en el resumen.

Referencias

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik., *Rich feature hierarchies for accurate object detection and semantic segmentation.*, In CVPR, 2014.

[2] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders., *Selective search for object recognition.*, IJCV, 2013.

[3] Cortes, Corinna; Vapnik, Vladimir (1995). *"Support-vector networks"(PDF). Machine Learning. 20 (3): 273–297.* CiteSeerX 10.1.1.15.9362. doi:10.1007/BF00994018. S2CID 206787478

[4] R. Girshick., *Fast R-CNN.*, In ICCV, 2015.

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick, *Mask R-CNN.*

[6] https://lamaquinaoraculo.com/computacion/segmentacion-de-instancia-mask-r-cnn/

[7] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, *Focal Loss for Dense Object Detection.*

[8] Trong Huy Phan, Kazuma Yamamoto. *Resolving Class Imbalance in Object Detection with Weighted Cross Entropy Losses*

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi , *You Only Look Once: Unified, Real-Time Object Detection.*

[10] Hochreiter, Sepp and Schmidhuber, Jürgen. (1997)., *Long Short-term Memory. Neural computation.*, 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[11] S. Wong, *.Advanced Correlation Tracking of Objects in Cluttered Imagery,"*, presented at Acquisition, Tracking, and Pointing XIX, Orlando, FL, USA, 2005.

[12] P.C. Mahalanobis. *On the generalised distance in statistics, Proceedings of the National Institute of Science of India 12 (1936) 49-55.*

[13] Kalman, R. E.. *A New Approach to Linear Filtering and Prediction Problems, Transactions of the ASME - Journal of Basic Engineering Vol. 82: pag. 35-45 (1960).*

[14] https://nanonets.com/blog/object-tracking-deepsort

[15] https://medium.com/@hirotoschwert/digging-into-detectron-2-47b2e794fabd

[16] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie. *Feature Pyramid Networks for Object Detection*

[17] Navaneeth Bodla* Bharat Singh* Rama Chellappa Larry S. Davis, *Improving Object Detection With One Line of Code*

[18] https://cocodataset.org

[19] N J Karthika, Chandran Saravanan, *International Conference on Electronic Systems and Intelligent Computing (ESIC 2020), Addressing False Positives in Pedestrian Detection*

[20] Yongyi Lu, Cewu Lu,Chi-Keung Tang, *Online Video Object Detection using Association LSTM*

[21] Yongyi Lu, Cewu Lu,Chi-Keung Tang

[22] https://roboflow.com/