VNIVERSITAT DE VALÈNCIA Escola Tècnica Superior d'Enginyeria

Máster en Ciencia de Datos

Trabajo Fin de Máster

ENSEMBLES FUNDACIONALES APLICADOS A SEGMENTACIÓN SEMÁNTICA DE IMÁGENES

Rubén Castillo Carrasco

Tutores: Valero Laparra
Pérez-Muelas

Pablo Hernández Cámara

Curso académico 2024/25

Declaración de autoría

Yo, Rubén Castillo Carrasco, con DNI 494927163W, declaro que soy el único autor del trabajo fin de máster titulado "ENSEMBLES FUNDACIONALES APLICADOS A SEGMENTACIÓN SEMÁNTICA DE IMÁGENES", que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual, y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Madrid, a 1 de septiembre de 2025

Fdo.: Rubén Castillo Carrasco

Abstract

Semantic image segmentation, the task of assigning a class label to every pixel in an image, is one of the fastest-growing tasks within the field of computer vision. In the current Master's Thesis, the improvements obtained by using ensemble-type architectures composed of different models—both foundational (including models such as *Segment Anything Model* and *YOLO*), and models trained from scratch (such as *U-Net* architectures), specialized in different tasks to perform the segmentation task—are evaluated. The results allow assessing whether this hybrid approach is capable of improving the performance of individual solutions on massive datasets such as *COCO*.

Resumen

La segmentación semántica de imágenes, consistente en el etiquetado de cada píxel de una imagen con una clase, es una de las tareas de más rápido desarrollo dentro del campo de la visión por computador. En el actual Trabajo de Fin de Máster se evalúan las mejoras obtenidas al hacer uso de arquitecturas tipo ensemble conformadas por diferentes modelos tanto fundacionales (donde se encuentran modelos como *Segment Anything Model* y *YOLO*), como modelos entrenados desde cero (como son las arquitecturas *U-Net*), especializadas en diferentes labores para realizar la tarea de segmentación. Los resultados permiten evaluar si este enfoque híbrido es capaz de mejorar los resultados de soluciones individuales en conjuntos de datos masivos como *MS COCO*.

Agradecimientos

En primer lugar, quiero expresar mi agradecimiento a la Fundación ValgrAI por su apoyo financiero, que ha sido fundamental para realizar este máster y por su labor en la promoción y el desarrollo de la inteligencia artificial en la Comunidad Valenciana.

Asimismo, me gustaría expresar mi agradecimiento a mi familia, en especial a mi hermana Aroa, por su apoyo incondicional y su confianza en mí en todo momento, y a mis amigos, especialmente a aquellos que he tenido la suerte de conocer gracias al Máster, por acompañarme en esta etapa y hacer más entretenido el día a día.

Por último y no por ello menos importante, quiero dar las gracias a mis tutores, Valero y Pablo, por su ayuda y por sus clases impartidas a lo largo del curso, fundamentales para adentrarme en el mundo del *Deep Learning* de una forma dinámica y entretenida.

Tabla de contenidos

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

La discapacidad visual afecta a millones de personas en todo el mundo, limitando su capacidad para interactuar con el entorno de manera autónoma y segura. De acuerdo con el NIH [56], la cantidad de personas que padecían de ceguera en 2020 ascendía a 43.3 millones, lo que representaba el 0.53 % de la población mundial, mientras que la cantidad de personas que padecían de discapacidad visual moderada o grave ascendía a 295 millones, equivalente al 3.6 % de la población mundial. Esta limitación tiene un fuerte impacto en la calidad de vida, restringiendo la independencia y dificultando la integración en la sociedad de las personas afectadas.

Además, se estima [6] que en los próximos años esta proporción de personas con discapacidad visual se incrementará hasta más de 100 millones de casos, debido a factores como el crecimiento y envejecimiento de la población, además del aumento de casos en enfermedades como la diabetes tipo 2. Puede observarse una estimación de la evolución en la figura 1.1.

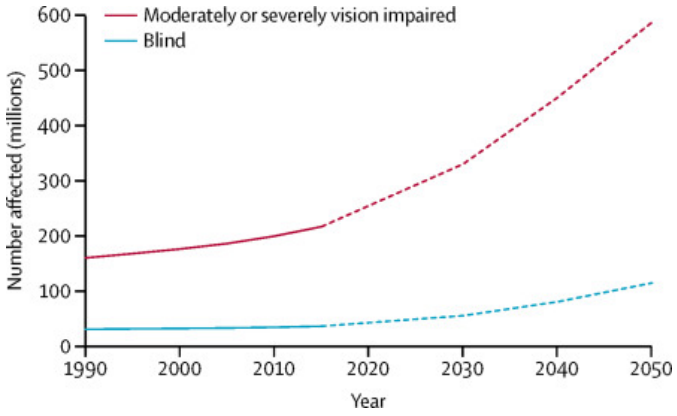

Figura 1.1: Estimación global de la evolución de los problemas de visión graves y moderados, según Bourne et al.[6]

Ante este problema, los recientes avances en visión por computador, desde el desarrollo del aprendizaje profundo (Deep Learning en adelante) hasta la aparición de modelos fundacionales, ofrecen una oportunidad única para abordar este caso de uso mediante el desarrollo de sistemas capaces de automatizar esta tarea, permitiendo así la obtención de descripciones del entorno precisas y útiles para personas con discapacidad visual y, por ende, reduciendo la dependencia de las personas afectadas.

Este proyecto surge de la motivación de mejorar la autonomía e inclusión de las personas con discapacidad visual, y busca desarrollar una arquitectura capaz de detectar elementos que representen desafíos en la vida cotidiana de estas personas.

1.2. Objetivos

El presente Trabajo de Fin de Máster tiene como objetivo principal el desarrollo de un modelo de aprendizaje profundo que permita la automatización de la segmentación de imágenes. Estas segmentaciones servirán como guía para la obtención de descripciones de las escenas, prestando especial atención a los objetos cotidianos presentes en ellas y relevantes para la mejora de la accesibilidad en personas con discapacidad visual. Es por ello que el modelo a obtener buscará la optimización de la eficiencia de la visión por computador ante los objetos específicos de mayor importancia para personas con dificultades de visión, como semáforos u otros obstáculos en el camino.

En este caso se tratará de resolver un problema de segmentación semántica de imágenes, al tenerse como objetivo la clasificación de cada píxel de la imagen como una categoría específica de un subconjunto de clases finito. Este se diferencia de otros problemas similares relativos al dominio de las imágenes en que, para el de clasificación se asigna una etiqueta o categoría a la totalidad de la imagen, en el de detección se han de localizar objetos delimitándolos por cajas o bounding boxes dentro de la imagen, y en el de segmentación de instancias se ha de identificar cada una de las instancias de cada clase de forma separada [30].

El hecho de enfocar el problema como uno de segmentación semántica radica en el contexto adicional que aporta este primero para personas con visión reducida, al aportar mayor información para casos de uso como el reconocimiento de superficies transitables, en contraste con enfoques basados en la detección. Un ejemplo práctico de cada uno de los problemas anteriores se puede observar en la figura 1.2.

Con todo lo anterior, uno de los objetivos primarios de este proyecto es el diseño, entrenamiento y evaluación de modelos y arquitecturas avanzadas de aprendizaje profundo para la segmentación de imágenes, utilizando el dataset *MS COCO* [46] como base para este entrenamiento, estudiado en profundidad en el siguiente capítulo.

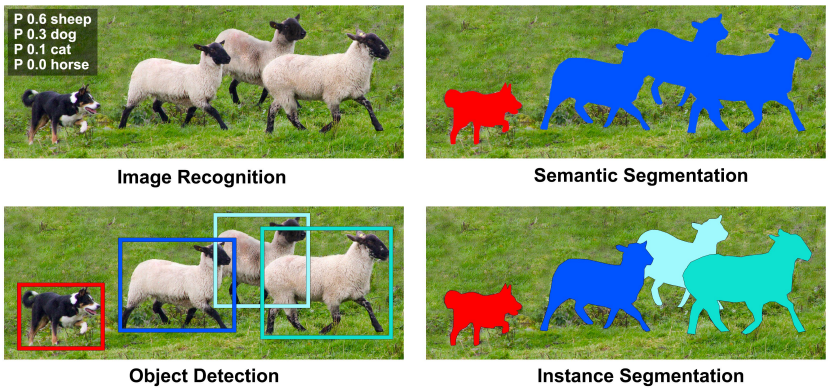

Figura 1.2: Ejemplos representativos de distintas tareas en el ámbito de la visión por computador. En la parte superior izquierda se muestra un ejemplo de clasificación, y a su derecha, uno de segmentación semántica. En la parte inferior izquierda se muestra un caso de detección de objetos, y a su derecha, un ejemplo de segmentación de instancias. Imagen obtenida en [3].

Otro de los objetivos más relevantes es el uso de modelos fundacionales (foundation models en inglés) como *Segment Anything Model* [37] para mejorar los resultados obtenidos con modelos tradicionales de segmentación, como pueden ser arquitecturas neuronales como la *U-Net* [65]. Asimismo, se valorarán posibles problemáticas en el desempeño de los diferentes modelos y arquitecturas de acuerdo con restricciones propias del problema, como el tiempo de inferencia y la falta de datos.

Además, desde un primer momento, se estudiarán y aplicarán metodologías como *CRISP-DM* [11] en todas las fases del proyecto para guiar el desarrollo y obtener resultados aceptables y válidos en el contexto de la ciencia de datos.

Capítulo 2

Estado del arte

2.1. Problema a resolver

La segmentación semántica hace referencia al problema de visión por computador consistente en la asignación a cada uno de los píxeles de una imagen de una etiqueta de una clase de un conjunto finito dado.

Formalmente, podemos definir este problema como la búsqueda de un mapeo tal que:

$$f : \mathbb{R}^{H \times W \times D} \rightarrow \mathbb{R}^{H \times W \times C}, \tag{2.1}$$

donde:

- H y W son las dimensiones de la imagen I,
- D es el número de canales de I,
- C es el número de clases posibles.

O lo que es lo mismo, se busca optimizar la función f que obtiene un mapeo de cada píxel de la imagen de entrada con la distribución de probabilidad de las C clases.

2.2. Datasets relevantes

Dado el problema a resolver definido anteriormente, es necesario seleccionar un dataset que contenga imágenes etiquetadas relevantes para la obtención de un modelo que automatice el proceso de segmentación. Es por ello que, para el entrenamiento del modelo, se evalúa la viabilidad y características de los conjuntos de datos más empleados en la literatura en la actualidad.

Entre los datasets valorados para la resolución de este problema se encuentran *MS COCO* [46], *CityScapes* [16], *ADE20K* [85] y *PASCAL VOC Dataset* [24].

Una primera comparativa de las imágenes contenidas en cada uno de los datasets anteriores se puede observar en la figura 2.1.

Figura 2.1: Ejemplos de instancias segmentadas en los datasets estudiados, obtenidos de forma directa de los conjuntos de datos *MS COCO* [46], *PASCAL VOC Dataset* [24], *ADE20K* [85] y *CityScapes* [16]

El primero de los anteriores, MS COCO (de las siglas Microsoft Common Objects in Context), destaca por contener, en su versión más reciente, hasta un total de 300.000 imágenes, entre las cuales se encuentran un total de 1.5 millones de objetos diferentes segmentados de 80 clases diferentes. Fue propuesto por Microsoft en 2014 [46] y actualizado en 2017. Puede ser utilizado para resolver problemas de detección de objetos, segmentación semántica y segmentación de instancias, además de contener *keypoints* de las 17 partes del cuerpo humano más relevantes que permiten la detección de poses. Cabe destacar que, en este conjunto de datos, conviven una gran variedad de resoluciones de las imágenes y fuentes de datos, al ser obtenidas a partir de plataformas como *Flickr* y de una amplia gama de geografías, que ayudan a que el entrenamiento de un modelo con este conjunto de datos pueda generalizar mejor.

En segunda instancia, el dataset CityScapes, propuesto por el instituto Max Planck [26] contiene imágenes segmentadas para resolver problemas de segmentación semántica y de instancias en escenas urbanas, alcanzando hasta un total de 5000 imágenes y 30 clases diferentes capturadas en 50 ciudades diferentes, siendo ampliamente usado para resolver problemas de segmentación y detección de objetos para vehículos autónomos.

En cuanto al dataset ADE20K, desarrollado por el MIT [87], destaca por contener hasta un total de más de 20.000 imágenes completamente segmentadas con la presencia de instancias de hasta 150 clases diferentes, destacando su uso como benchmark para la evaluación de modelos de segmentación semántica.

Por último, el dataset PASCAL VOC data de 2012 [22] y su uso principal ha sido el de la detección de objetos en imágenes, aunque también contiene clasificaciones por píxel para poder ser usado para resolver problemas de segmentación, al contener hasta un total de más de 10.000 imágenes segmentadas con más de 20 categorías diferentes.

Para la obtención de un modelo adecuado capaz de generalizar en instancias relativas a nuestro problema se ha decidido hacer uso del dataset MS COCO. Entre los motivos que respaldan esta decisión destacan principalmente su gran volumen de imágenes y el gran número de clases presentes en las mismas, que alcanza las 80, incluyendo instancias etiquetadas con objetos considerados importantes para personas con discapacidad visual, como lo son los semáforos, señales de tráfico, coches o muebles, que puedan obstaculizar su avance. Además, se ha considerado de relevancia que el dataset seleccionado posea diferentes fuentes y resoluciones de imágenes, favoreciendo la obtención de un modelo que generalice mejor.

2.3. Modelos de segmentación en imágenes

En términos de los modelos empleados para modelar problemas de segmentación semántica de imágenes, en la literatura destacan diferentes aproximaciones, desde soluciones más simples mediante arquitecturas encoder-decoder con capas convolucionales como la *U-Net*, desarrollada por Ronneberger et al. [66], hasta el uso de modelos fundacionales mucho más complejos como SAM, empleados por una gran cantidad de autores para diferentes casos de uso con buenos resultados, siendo un ejemplo la segmentación de PASCAL VOC 2012 propuesta por Yang et al. [83].

2.3.1. Primeras aproximaciones en la segmentación de imágenes

Antes del desarrollo del Deep Learning, las primeras aproximaciones para resolver problemas de segmentación de imágenes o tareas similares, como la detección de rostros, se basaban en modelos poco profundos (o *shallow models*). Estos combinaban técnicas de umbralización, detección de bordes y análisis de regiones, pero presentaban dificultades para capturar relaciones espaciales complejas presentes en los datos.

Ejemplos comunes en la literatura incluyen el uso de modelos como k-means y Expectation Maximization para la segmentación de regiones en imágenes, como el presentado por Moftah et al. [53], además de modelos de Markov, como los expuestos por Yuri Boykov [7], o técnicas basadas en características propias de la imagen, como las propuestas por Mamat et al. [49].

Estos enfoques enfrentaban serias dificultades para generalizar en imágenes de distintos dominios y, por ello, no se testearán en el presente trabajo de fin de Máster, optándose, en su lugar, por arquitecturas más avanzadas y recientes que se describen a continuación.

2.3.2. Redes neuronales convolucionales

La convolución es una operación matemática utilizada en el procesamiento de señales que, en el ámbito de la visión por computador, sustenta las bases de las capas convolucionales usadas por algunas de las arquitecturas neuronales más recientes para resolver problemas como el de la segmentación en imágenes. Esta operación permite la combinación de dos funciones o señales en una tercera, que expresa cómo la forma de la primera afecta a la segunda [58].

Formalmente, puede definirse tal que:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau \tag{2.2}$$

Donde:

- f es la función o señal de entrada a transformar,
- g es la función que se desplaza, modificando a f.

En imágenes, donde se aplica de forma discreta por píxeles, la convolución consiste en desplazar el kernel o filtro sobre la matriz bidimensional que representa la imagen.

De forma análoga, en este contexto discreto, la convolución de una señal o imagen $f(x)$, usando un kernel $g(x)$ puede definirse tal que [28]:

$$(f * g)(x, y) = \sum_{i=-k/2}^{k/2} \sum_{j=-k/2}^{k/2} f(x+i, y+j) \cdot g(i, j) \tag{2.3}$$

Siendo:

- $f(x, y)$: valor del píxel (x, y) de la imagen,
- $g(i, j)$: el valor del filtro para la posición (i, j),
- k: el tamaño del filtro a aplicar en la convolución.

Este proceso permite extraer características específicas de la misma, más sencillas en las primeras capas de una red, como bordes, y más complejas en capas más profundas, como texturas.

Las redes convolucionales (CNNs en adelante, por sus siglas en inglés) hacen uso de esta operación en sus capas para la resolución de problemas donde existen fuertes dependencias espaciales o temporales en los datos de entrada [59], como pueden ser todos aquellos relativos a imágenes y series temporales, donde los elementos vecinos de cada instancia guardan fuertes relaciones de dependencia.

Las CNNs están conformadas por arquitecturas que hacen uso principalmente de los siguientes tipos de capas:

- Capas convolucionales: aplican la operación de la convolución mediante filtros a lo largo del input, de forma que se puedan detectar patrones espaciales de las imágenes, como bordes, formas o texturas.
  En estas, es importante destacar la importancia de dos parámetros característicos de este tipo de capas, el *padding* y el *stride*.

  El *padding* indica los valores de relleno colocados en los bordes, añadidos con el objetivo de preservar las dimensiones y facilitar el aprendizaje en zonas cercanas a estos en las imágenes. Por otro lado, el *stride* hace referencia a la cantidad de píxeles que se desplaza el filtro en cada movimiento, permitiendo así una reducción de la dimensionalidad de la salida.

  Un ejemplo de los resultados de la aplicación de una capa convolucional sobre un input de tamaño 5x5 se observa en la figura 2.2.

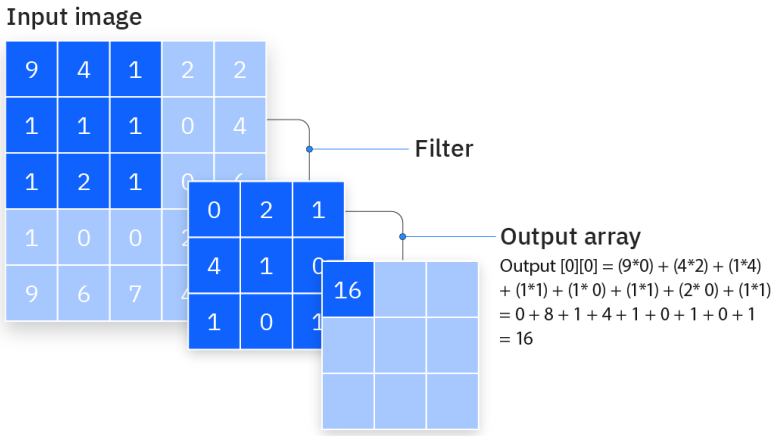

Figura 2.2: Ejemplo gráfico resultado de la aplicación de un filtro convolucional sobre el canal de una imagen 5x5 haciendo uso de un kernel 3x3 con $stride = 1$ y $padding = same$. Imagen obtenida de [34]

Con los parámetros anteriores, la dimensionalidad de la salida de esta capa tendrá unas dimensiones para cada canal tales que $H_{\text{out}} \times W_{\text{out}}$, pudiendo ser calculadas como:

$$H_{\text{output}} = \frac{H_{\text{input}} - K_h + 2P}{S} + 1, \quad W_{\text{output}} = \frac{W_{\text{input}} - K_w + 2P}{S} + 1 \tag{2.4}$$

Donde:

- K_h y K_w son las dimensiones altura y anchura del kernel,
- P es el valor del padding,
- S es el valor del stride.

- Capas de pooling: utilizadas para aplicar *downsampling*, es decir, reducir la dimensionalidad de la entrada, con el objetivo de mejorar la capacidad de generalización de la red. Para ello, estas capas aplican operaciones como la *media* o el *máximo* a una entrada. Un ejemplo del uso de esta capa está presente en la figura 2.3.

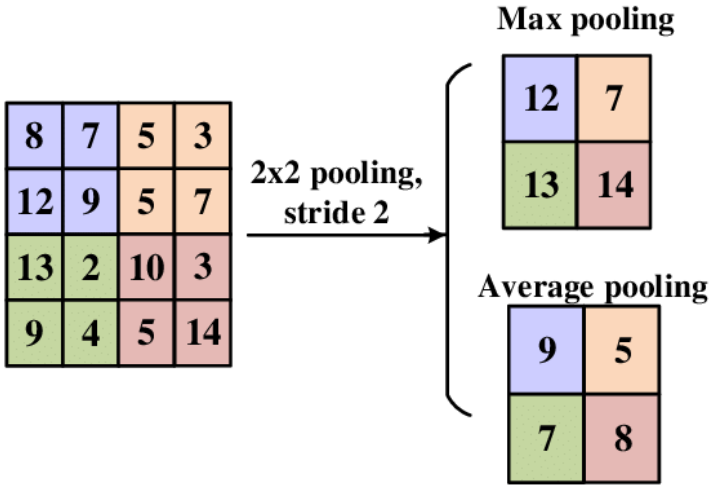

Figura 2.3: Ejemplo gráfico del resultado al aplicar *max pooling* y *average pooling* sobre una misma entrada. Imagen obtenida de [84].

- Capas de activación: introducen no linealidad en la red, permitiendo que esta aprenda patrones complejos. Dentro de las funciones de activación para este tipo de problemas destaca $ReLU$ (de las siglas en inglés Rectified Linear Unit), que aplica una transformación a la entrada tal que $ReLU(x) = \max(0, x)$ y que solventa problemas propios de otras funciones de activación [27], como la función sigmoide en redes neuronales profundas con el desvanecimiento de gradiente, donde los gradientes de los pesos tienden a volverse muy pequeños, impidiendo el entrenamiento del modelo.

- Capas fully connected: usadas en las últimas capas de la red que permiten realizar las predicciones de la red en base a las características extraídas por esta en las capas anteriores. Para ello, aprenden transformaciones lineales de la forma

$$y = f(Wx + b) \tag{2.5}$$

  Siendo W la matriz de pesos y b el vector de sesgos a optimizar por la capa.

- Capas para la normalización, que aparecen de forma opcional entre las capas de la red, generalmente entre capas de convolución y de activación. Dentro de la normalización aplicada al input destaca la normalización por *batches* [35], consistente en la aplicación de la misma a muestras de datos procesadas de forma simultánea en cada lote de entrenamiento. También destaca la normalización por capa [5], donde se aplica a toda la capa por cada instancia en lugar de a nivel de batch.

Ejemplos de uso de cada una de las anteriores pueden ser encontrados en arquitecturas como *LeNet-5* desarrollada por LeCun et al. en 1998 [39], representada en la figura 2.4 y *VGGNet*, desarrollada por Simonyan et al. [69], utilizadas para resolver problemas de clasificación en imágenes.

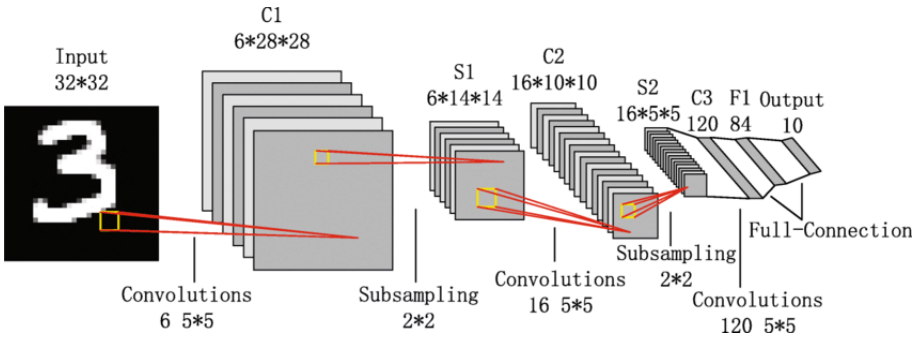

Figura 2.4: Arquitectura completa de la red convolucional para la clasificación *LeNet-5*. Imagen obtenida de [71]

2.3.3. Fully Convolutional Networks

Dentro de las arquitecturas que hacen uso de capas convolucionales para la resolución de problemas de segmentación en imágenes destacan las denominadas *Fully Convolutional Networks* (FCNs en adelante, por sus siglas en inglés), introducidas en 2015 por Long et al. [48], y caracterizadas por reemplazar las capas densamente conectadas de las últimas capas de las redes neuronales tradicionales por capas convolucionales.

Este cambio en la arquitectura permite procesar imágenes de tamaño variable y generar mapas de segmentación con resolución adaptable, donde la salida conserve la estructura espacial de la entrada, a diferencia de las CNNs, donde la salida era de menor resolución espacial.

Esta adaptabilidad de la dimensionalidad de la salida las hace adecuadas para problemas como el de segmentación en imágenes, donde la salida ha de tener la misma altura y anchura, variando el número de canales en función del problema a resolver.

Esta arquitectura puede dividirse en los bloques siguientes, los convolucionales y de *downsampling*, y los de convolución transpuesta y *upsampling*.

Bloques convolucionales y de downsampling

Bloques donde se combinan capas convolucionales propias de las redes convolucionales con funciones de activación, generalmente *ReLUs*, para extraer características relevantes de los datos de entrada. Este bloque está compuesto por las siguientes capas:

- Capas convolucionales: presentes en las CNNs, tienen como objetivo detectar patrones locales en los datos de entrada. En las primeras capas estas detectan características más simples como bordes, texturas o estructuras básicas mientras que en capas más profundas capturan progresivamente patrones más abstractos y complejos.

- Capas de downsampling: encargadas de la reducción de la dimensionalidad de la entrada y la eliminación de redundancias espaciales, ayudando a la capacidad de generalización del modelo. Para ello, hacen uso de operaciones como el max-pooling o el average pooling, ya vistos en el anterior apartado.

Bloques de convolución transpuesta y de upsampling

Bloques que tienen como objetivo la reconstrucción y expansión de la salida de los bloques convolucionales y de downsampling, de vital importancia para tareas donde se requiere una dimensionalidad de la salida similar a la de la entrada, como sucede en segmentación de imágenes. Este tipo de bloques se compone, a su vez, de las siguientes capas:

- Capas de convolución transpuesta: realizan una operación inversa a la convolución estándar, al incrementar la resolución espacial y permitir la reconstrucción progresiva de los datos hacia su forma original. Al igual que en las capas convolucionales, destacan parámetros como el *padding*, que añade valores antes de aplicar la convolución transpuesta, y el *stride*, que especifica cuánto ha de moverse el kernel en cada desplazamiento. Además, destaca el parámetro de dilatación, que indica la cantidad de píxeles adicionales añadidos alrededor para ajustar el tamaño de la salida.

  Las dimensiones resultantes de la salida tras aplicar la operación son tales que [20]:

$$H_{\text{output}} = (H_{\text{input}} - 1) \cdot S_h + K_h - 2P_h + O_h, \tag{2.6}$$
$$W_{\text{output}} = (W_{\text{input}} - 1) \cdot S_w + K_w - 2P_w + O_w \tag{2.7}$$

  Siendo

  - H_{input}, W_{input} las dimensiones de la entrada,
  - S_h, S_w las dimensiones del stride aplicado,
  - K_h, K_w las dimensiones del kernel,
  - P_h, P_w el padding aplicado a cada input antes de aplicar el filtro,
  - O_h, O_w el número de píxeles añadidos para la dilatación en altura y anchura.

  Un ejemplo de la aplicación de esta capa sobre un input se puede encontrar en la figura 2.5.

- Capas de upsampling: operación opuesta al downsampling, encargada de aumentar la dimensionalidad espacial de los datos y permitir la reconstrucción progresiva de la imagen. Dentro de las técnicas utilizadas destacan desde la más sencilla, *Nearest-Neighbor*, donde se reutiliza el valor del píxel más cercano para generar el nuevo valor, hasta otras más complejas como la interpolación bilineal, donde se hace uso de los vecinos para el cálculo de píxeles intermedios [31]. Formalmente, este último método de interpolación para imágenes puede definirse de la forma:

$$y(i,j) = \sum_{k=1}^{2} \sum_{l=1}^{2} w_{kl} \cdot x(p_k, q_l) \tag{2.8}$$

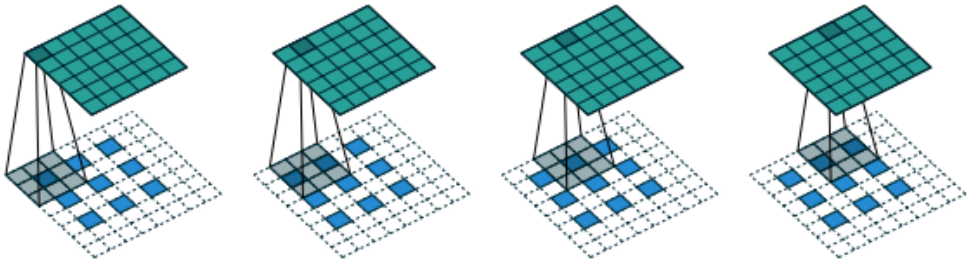

Figura 2.5: Ejemplo de la metodología para el cálculo de la salida de una capa de convolución transpuesta aplicada sobre un input de dimensiones 3x3 con un kernel de 6x6 usando *padding* de 1x1 y *stride* de 2x2 y dilatación en la salida de 1x1. Imagen obtenida de [20]

Un ejemplo concreto de la aplicación de esta capa en imágenes en función del método se observa en la figura 2.6.

Figura 2.6: Diferencias en los resultados al aplicar upsampling mediante Nearest-Neighbor e interpolación bilineal sobre una misma imagen de baja resolución. Obtenida en [31]

Como ejemplo de algunas arquitecturas FCNs presentes en la literatura que logran resolver problemas de forma exitosa, podemos destacar DeepLab v3+ [12] presentada por Chen et al. en 2018, y la U-Net [65] presentada por Ronneberger et al. en 2015.

U-Net

Arquitectura introducida en 2015 [65], tratando de mejorar otras arquitecturas *fully-convolutional* de la época como la propuesta por Ciresan et al. [14]. Esta se desarrolla con el objetivo inicial de resolver problemas de segmentación en imágenes médicas, pero ha demostrado a lo largo de los años buenos resultados para una amplia gama de tareas relacionadas con la segmentación en imágenes, más allá del ámbito médico, como la detección de estructuras en imágenes satelitales, la segmentación de imágenes para la conducción autónoma o la detección de anomalías en líneas de producción industrial.

La principal característica de la misma frente a otras arquitecturas *FCN* de la época radica en el uso de las denominadas *skip connections*, conexiones entre capas de *downsampling* y de *upsampling*, de tal forma que en la reconstrucción de la imagen se permite combinar características de bajo nivel de las primeras capas con características de alto nivel extraídas en capas más profundas.

Esta arquitectura se puede observar en detalle en la figura 2.7 constando de dos bloques diferenciables denominados contractivo y expansivo, que dan lugar a su característica forma de U, enlazados mediante las *skip connections*.

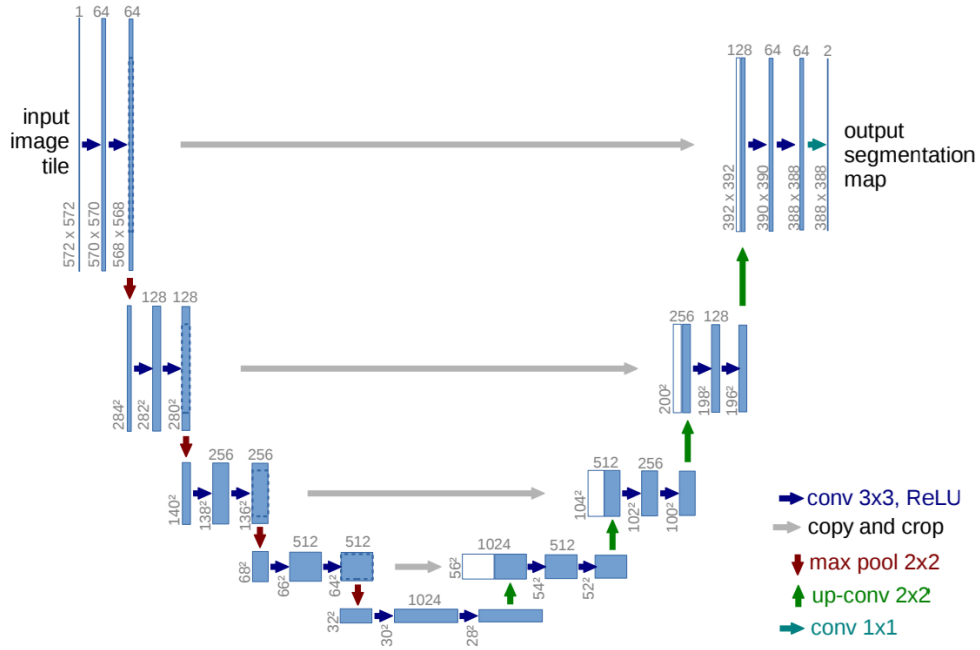

Figura 2.7: Representación gráfica de las capas que componen la arquitectura de la red U-Net. Obtenida en [65]

El bloque contractivo destaca por el uso repetido de dos capas convolucionales con kernels 3x3 y *padding = same* , precedidas de capas de activación *ReLU* y capas de max-pooling de tamaño 2x2 con stride 2, de tal forma que con cada bloque anterior se duplica el número de canales de la entrada.

En cuanto al bloque expansivo, este está compuesto por bloques de capas de upsampling precedidas por capas de convolución transpuesta, que reducen a la mitad el número de canales en cada bloque, alimentándose mediante una concatenación de la salida de un bloque del camino contractivo, y por dos capas convolucionales de dimensión 3x3.

Por último, al final del bloque expansivo se hace uso de una convolución 1x1 para mapear cada vector de características al número deseado de clases que conforman el problema de segmentación, dando lugar a una red con un total de 23 capas convolucionales y un número de parámetros que, para un problema de segmentación para imágenes a color y de un tamaño medio, asciende al orden de millones [65].

2.3.4. Arquitecturas basadas en mecanismos de atención

Los mecanismos de atención son módulos computacionales diseñados para asignar dinámicamente diferentes niveles de importancia a los elementos de entrada, lo que permite centrar la atención del modelo en la información más relevante de la misma [79].

Estos mecanismos han mejorado significativamente la capacidad de los modelos de segmentación de imágenes para enfocarse en las regiones relevantes dentro de entradas visuales complejas para una amplia gama de problemas, como indica Xie et al. [82].

Un mecanismo de atención está basado en la idea de calcular pesos de relevancia para cada uno de los elementos de entrada, siendo estos cada una de las regiones o patches de una imagen en este contexto [78]. Formalmente, dada una entrada de la forma $X = \{x_1, x_2, ..., x_n\}$, se puede calcular una representación de la misma ponderada por atención tal que:

$$\text{Atención}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.9}$$

Donde

- $Q = XW^Q$ o *query* es un vector que representa lo que se está buscando,
- $K = XW^K$ o key representa la información disponible que se compara con la query,
- $V = XW^V$ o value representa la información asociada a cada una de las keys,
- d_k es la dimensión de los vectores query y key.

Las matrices $W^Q W^K W^V$ son aprendibles a partir de la entrada X. Además, a estas se les aplica un factor de escalado $\sqrt{d_k}$ que estabiliza la distribución de probabilidades de atención, especialmente ante valores altos de d_k, que resultaría en valores extremos de *softmax*, y, por ende, en dificultades en el aprendizaje. Puede observarse un ejemplo más gráfico del proceso de aplicación de mecanismos de atención sobre una entrada en la figura 2.8.

Este tipo de mecanismos, para problemas de segmentación de imágenes, han permitido superar las limitaciones inherentes presentes en otras arquitecturas neuronales más locales como las *CNN* y *FCN*, que, a pesar de ser efectivas para capturar características locales, tienen dificultades para capturar dependencias a mayores rangos.

La integración de estos módulos en imágenes permite a los modelos ponderar la importancia de diferentes regiones espaciales y detectar el contexto de los elementos de la imagen, solventando algunos de los problemas existentes al resolver problemas de segmentación con otras arquitecturas, como los expuestos por Xie et al. [82], donde en imágenes médicas los modelos convolucionales no logran capturar bien dependencias a mayores rangos, como pueden ser tumores dispersos.

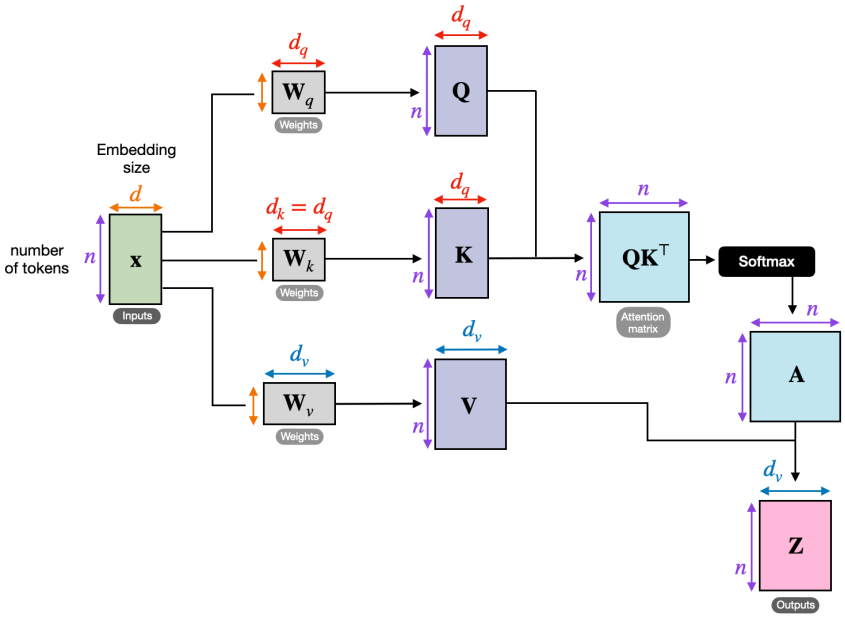

Figura 2.8: Representación gráfica de las componentes matriciales en mecanismos de atención y el procedimiento de la aplicación de atención sobre una entrada. Imagen obtenida de [62]

Los mecanismos de atención para modelos de segmentación, como los empleados en la Attention U-Net de Oktay et al. [57] y en la Pyramid Attention Network (PAN en adelante por sus siglas) de Li et al. [41] han demostrado mejoras notables en los resultados obtenidos por otras arquitecturas convolucionales más simples, como las ya vistas anteriormente.

Una aproximación similar incorporando este tipo de mecanismos en su arquitectura a nuestro modelo *baseline*, la U-Net, es la Attention U-Net, que conserva su característica forma de U. Esta incluye puertas de atención en lugar de las *skip connections* de la red original, como se muestra en la figura 2.9. De esta forma, se permite la selección de características más relevantes durante las etapas de codificación y decodificación.

2.3.5. Modelos fundacionales

Podemos definir los modelos fundacionales como aquel subconjunto de modelos con gran capacidad de generalización y escalabilidad que actúan como base para aplicaciones complejas [9]. Su principal característica que los diferencia del resto de modelos radica en su versatilidad, pudiendo ser adaptados para una amplia gama de tareas al haber sido entrenados a partir de grandes volúmenes de datos. A su vez, esta capacidad de generalización adquirida durante su entrenamiento les permite resolver problemas de diversos dominios.

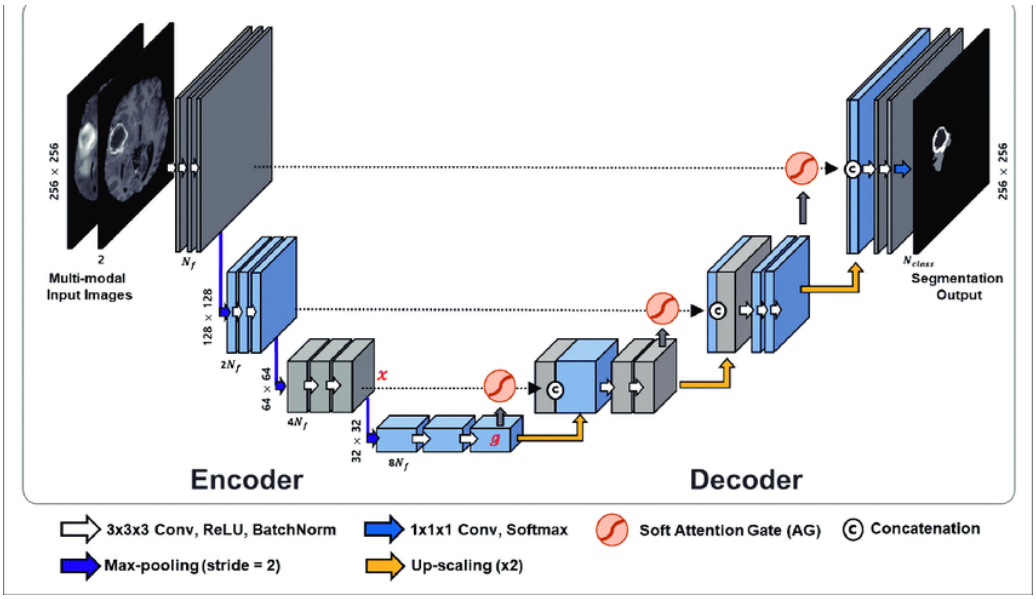

Figura 2.9: Arquitectura de la red Attention U-Net propuesta por Oktay et al. Imagen obtenida de [68]

Entre sus características fundamentales, destacan [9]:

- Alta capacidad de generalización: una vez entrenados, estos se caracterizan por la obtención de buenos resultados ante muestras distintas a las vistas durante su entrenamiento (también denominado *out-of-sample performance*).
- Entrenamiento mediante un paradigma auto-supervisado: que implica que para su entrenamiento no se hace uso de un conjunto de datos etiquetado de forma manual por humanos, sino que es el propio modelo en que genera etiquetas de forma automática ante datos sin etiquetar. Esto tiene como consecuencia una reducción significativa de los costes, tanto económicos como de tiempo de desarrollo, asociados al proceso de etiquetado tradicional. Además, permite hacer uso de un mayor volumen de datos frente al paradigma tradicional, donde el conjunto de datos etiquetado es mucho más limitado.
- Adaptación mediante técnicas como el fine-tuning: estos modelos permiten hacer uso de esta técnica, que permite adaptarse a tareas específicas sin la necesidad de un re-entrenamiento desde cero. Esto reduce el tiempo de entrenamiento necesario para adaptar el modelo a esta nueva tarea, en especial ante un entrenamiento desde cero para la resolución de la misma.
- Adaptabilidad multi-tarea: un amplio conjunto de estos modelos están adaptados para tanto responder de forma correcta como producir salidas de diferentes modalidades, incluyendo texto, imágenes o vídeo, entre otros.

En el contexto de la segmentación semántica de imágenes, dentro de los modelos fundacionales destacan tanto las grandes redes convolucionales preentrenadas con grandes volúmenes de datos como arquitecturas con mecanismos de atención.

A su vez, dentro de las redes convolucionales, destacan arquitecturas como EfficientNet, desarrollada por Google en 2019 [72], que introduce un método escalado automático y permite abordar problemas de segmentación de más de 1.000 clases de objetos de la vida cotidiana.

En cuanto a las arquitecturas fundacionales con mecanismos de atención, como las Vision Transformers (ViTs en adelante), caracterizadas por la buena conservación de dependencias de las entradas a largo plazo a lo largo de la imagen, destacan arquitecturas como *Segment Anything Model* y *Mask2Former* [67]. Estas permiten la realización de anotaciones píxel a píxel de imágenes de datos que salen de los dominios del conjunto de datos con el que han sido entrenados, gracias a su capacidad de generalización ante muestras no vistas (o *Zero-shot learning*).

CLIP

CLIP (de las siglas Contrastive Language–Image Pretraining) es un modelo desarrollado por OpenAI [61] que tiene como objetivo vincular inputs en un espacio de representación compartido, permitiendo así la comparación de embeddings n-dimensionales generados de datos de distinta modalidad, en este caso, texto e imagen.

En este caso, este modelo, por sí mismo, no permite la segmentación de imágenes, pero combinado con otros modelos segmentadores, permite resolver problemas de segmentación semántica de forma precisa. Un ejemplo de esto es su aplicación para la segmentación de imágenes médicas propuesta por Aleem et al. [2].

Para su entrenamiento, al igual que con otros modelos fundacionales, se hace uso del aprendizaje *auto-supervisado*, en este caso, contrastivo. Este consiste en el uso de instancias agrupadas por pares para el entrenamiento. Para ello, se hizo uso de hasta 400 millones de pares imagen-texto obtenidas de internet y permitiendo así la asociación de datos de diferentes modalidades sin necesidad de un etiquetado explícito de gran costo temporal y económico.

Para la arquitectura del modelo, CLIP utiliza dos redes neuronales separadas [29]:

- Un transformer entrenado a partir de conjuntos de textos, encargado de obtener un embedding n-dimensional a partir del prompt introducido por el usuario, generando un vector que encapsula el significado semántico.
- Un Vision Transformer encargado de obtener características visuales profundas de las imágenes y relaciones espaciales complejas, dando como salida otro embedding n-dimensional comparable con el textual empleando medidas de similitud como el producto escalar.

La arquitectura detallada del mismo se muestra en la figura 2.10.

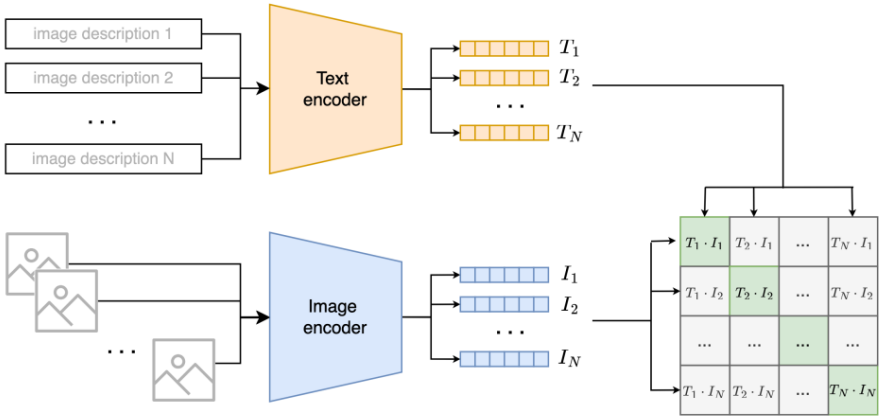

Figura 2.10: Arquitectura del modelo CLIP. Fuente [29]

Para su entrenamiento, tanto el transformer que genera el embedding de imágenes como el de textos fueron entrenados de manera conjunta para optimizar la similitud entre pares relacionados, obteniendo así una representación compartida de imágenes y descripciones textuales que se refieren al mismo concepto, alineadas en el espacio latente del modelo.

Segment Anything Model

Segment Anything Model (SAM en adelante) es un modelo creado por Meta en 2023 [37] para tareas de segmentación de imágenes que destaca frente a otros modelos por su gran capacidad de generalización y su capacidad para generar segmentaciones de una amplia variedad de dominios.

Dentro de sus características destacan:

- Flexibilidad: posibilita tanto la segmentación automática de imágenes como la segmentación interactiva, permitiendo tanto la identificación sin intervención de los objetos contenidos en la entrada como la segmentación a partir de inputs del usuario, como pueden ser descripciones textuales, puntos o cajas en la imagen, lo que da lugar a un enfoque multimodal.

- Capacidad de generalización: obtenida al haber sido entrenado con un conjunto de datos masivo y diverso llamado *SA-1B* [50], que contiene más de mil millones de máscaras segmentadas a partir de más de 11 millones de imágenes, de una amplia gama de geografías y temáticas variadas. Hasta la fecha, se trata de uno de los conjuntos de datos más grandes y diversos jamás usados para segmentación de imágenes. Cabe destacar que para el etiquetado de las imágenes se ha seguido un paradigma auto-supervisado, donde modelos de ML obtienen anotaciones y validaciones manuales, que, de obtenerse de forma manual, supodría un incremento tanto temporal como económico.

- *Zero-shot performance*: este modelo genera buenos resultados para la gran mayoría de tareas sin la necesidad de aplicar *fine-tuning*, habiendo sido validado su funciona-

miento para una amplia gama de tareas, donde se encuentran desde la segmentación de imágenes médicas, el análisis de imágenes satelitales, la edición de imágenes y vídeos mediante la segmentación rápida de objetos específicos o el análisis visual en vehículos autónomos para detectar y segmentar peatones entre otros [37].

En lo relativo a su arquitectura, una visión general se puede observar en la figura 2.11.

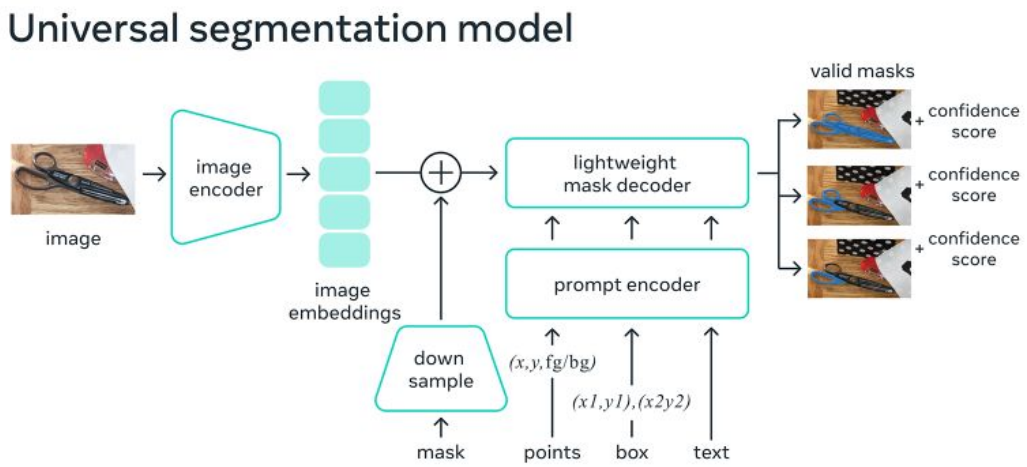

Figura 2.11: Bloques que componen la arquitectura del modelo SAM aplicado a problemas de segmentación. Obtenida en [21]

Los bloques que componen este modelo para realizar la tarea de segmentación son los siguientes:

1. Encoder de imágenes: procesa la imagen de entrada al modelo mediante un ViT con más de 632 millones de parámetros, lo que permite la obtención de un embedding latente que contiene las características visuales representativas de la imagen, al capturar el contexto y características de alto nivel de las distintas partes de la misma.

2. Encoder de entrada del prompt: permite al modelo incorporar una entrada por parte del usuario, de forma que este pueda guiar la tarea de la segmentación. Estas permiten entradas de dos tipos de modalidades:

   - Entradas de modalidad textual: obteniendo como salida un embedding semántico mediante un modelo de lenguaje derivado de *CLIP*.
   - Entradas de modalidad espacial: obtiene como salida un embedding que contiene la información de la entrada espacial, pudiendo ser o bien puntos o *bounding boxes*.

3. Decoder de máscaras: módulo encargado de fusionar las características generadas por los encoders de imágenes y de prompts del usuario y producir las máscaras de segmentación finales, haciendo uso de mecanismos de atención que permiten enfocar las áreas relevantes de la imagen según el prompt proporcionado.

Una variante a destacar de este modelo es *fast-SAM*, desarrollada por Zhao et al. en 2023 [86] con el objetivo de optimizar el tiempo de inferencia sin degradar el rendimiento en la medida de lo posible. Su uso destaca por ser algo menos eficiente en tareas de segmentación complejas, aunque proporciona resultados decentes en casos de uso donde la velocidad de inferencia es una restricción, como lo es la segmentación de vídeos en tiempo real.

Como conjunto de datos para su entrenamiento, se hizo uso del mismo que su variante, SA-1B, aunque en este caso usando un subconjunto menor del mismo. En cuanto a la velocidad de inferencia, ante determinados benchmarks es capaz de reducirla en un factor de 50, pudiéndose observar una comparativa del *tradeoff* tiempo de inferencia - rendimiento en la figura 2.12.

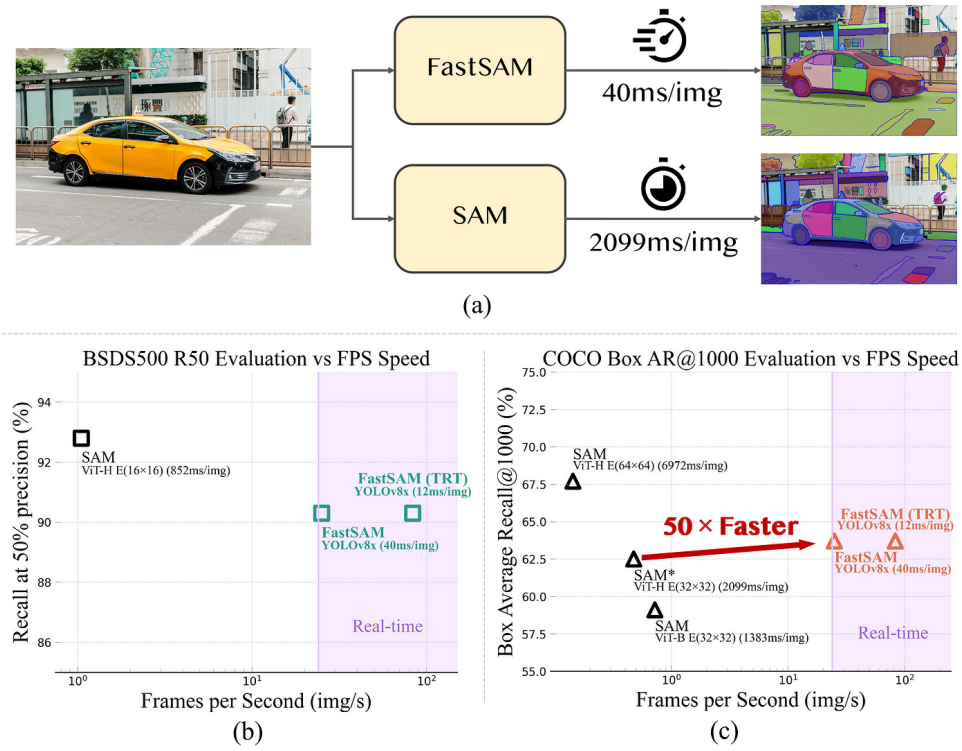

Figura 2.12: Comparativas entre SAM y fast-SAM sobre una misma imagen del dataset COCO MS. Imagen obtenida del paper de fast-SAM [86]

En cuanto a la arquitectura del modelo, este difiere de SAM en algunos de sus componentes, estando conformado principalmente por los siguientes:

- Un primer bloque principal CNN como *backbone*, elemento que actúa como extractor de características en la red, obteniendo de píxeles crudos características de alto nivel.
- Un bloque FPN que permite manejar características visuales de diferentes escalas [42], lo que permite que el modelo las procese de forma eficiente y evitando así el uso de transformers, que aumentan de forma significativa el tiempo de inferencia. Como salida de la capa se obtiene un conjunto de mapas de características donde cada nivel contiene información tanto semántica como espacial, pero a diferentes resoluciones.
- Un módulo *Mask Decoder* encargado de la generación de máscaras que utiliza convoluciones para generar máscaras binarias de las características obtenidas en capas anteriores.
- Un componente denominado *Prompt encoder* que codifica inputs del usuario como puntos o bounding boxes en el mismo espacio de características.
- Un módulo de fusión que combina la información proveniente del backbone, la FPN y el prompt-encoder para generar predicciones de salida.

Un gráfico detallado de esta arquitectura se puede observar en la figura 2.13.

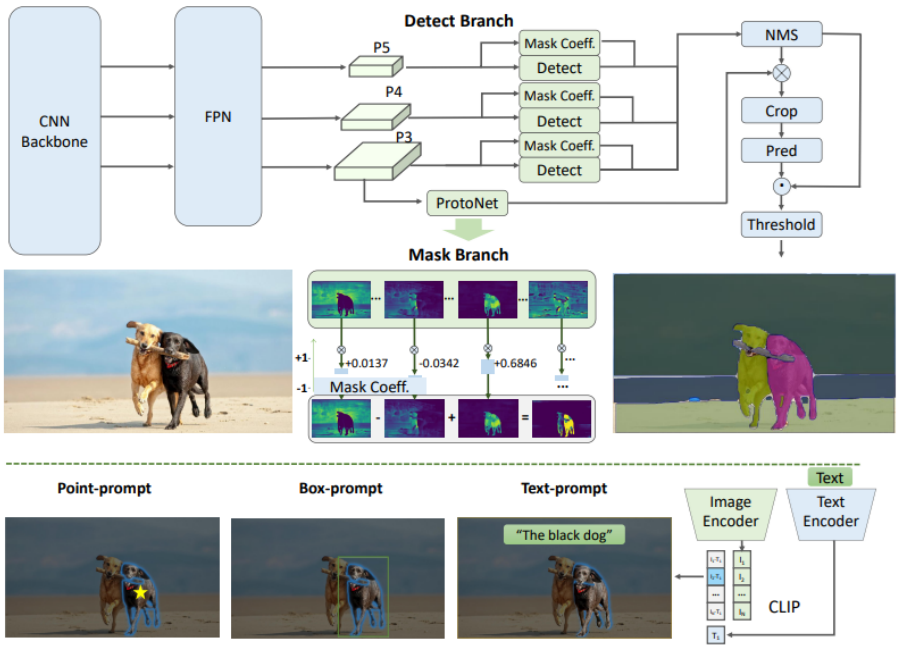

Figura 2.13: Arquitectura del modelo fast-SAM. Obtenida en [86]

YOLOv8

YOLO (de las siglas en inglés You Only Look Once) es una familia de modelos de detección de objetos en imágenes introducida en 2016 por Redmon et al. [63], que en sus versiones más recientes permite tanto la detección como la segmentación de instancias. Este destaca frente a otros modelos por su alta velocidad de inferencia, dado que realiza la fase de detección en una única pasada, permitiendo así detecciones rápidas en tiempo real.

YOLOv8 es un modelo de la familia YOLO introducido en 2023 y mantenido por ultralytics [77] que permite abarcar tareas de segmentación, detección y clasificación, empleando una única arquitectura.

YOLOv8 no se trata de un modelo fundacional por sí mismo, dado que ha sido entrenado para resolver problemas de segmentación y detección del dataset COCO en específico, y por ende no ha aprendido segmentaciones y detecciones generales, sino propias de este conjunto de datos y sus clases. En cambio, permite la realización de un entrenamiento sencillo ante otros conjuntos de datos de forma sencilla, de manera que pueda adaptarse para resolver problemas de detección y segmentación de otros dominios específicos.

En cuanto a la arquitectura de YOLOv8, esta destaca por estar conformada por tres bloques diferenciados, denominados *Backbone*, *Neck* y *Head* respectivamente. Un esquema más detallado de la arquitectura global se puede observar en la figura 2.14.

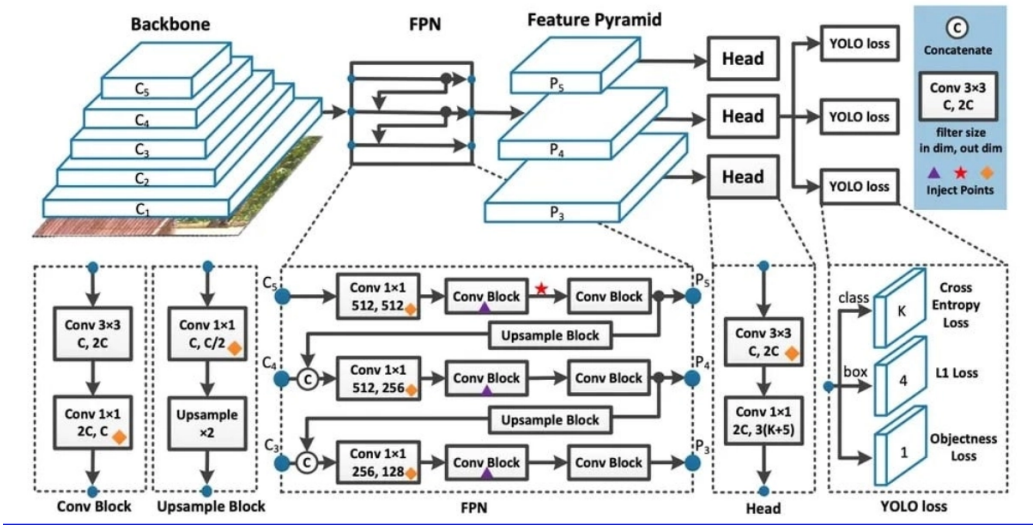

Figura 2.14: Arquitectura del modelo YOLOv8 [75]

Backbone

Es la capa encargada de extraer características de cada imagen de entrada, en el caso de YOLOv8 mediante una combinación de convoluciones, capas de batch normalization, capas de activación *SiLU* y capas *CSP*.

La función de activación por el modelo para su entrenamiento es *SiLU*. A diferencia de *ReLU*, [8] hace uso de una función sigmoide multiplicada por la entrada, formulándose tal que:

$$\text{SiLU}(x) = x \cdot \sigma(x) = \frac{x}{1 + e^{-x}} \tag{2.10}$$

Esta destaca por ser una función de activación más suave que *ReLU*, que mantiene el gradiente incluso para valores negativos pequeños, al igual que *ReLU* aunque, por otro lado, su cálculo es algo más costoso computacionalmente.

Por otro lado, los bloques CSP, de las siglas *Cross Stage Partial*, introducidas en 2019 por Wang et al. [80], se tratan de bloques que dividen el flujo en dos ramas. En una de ellas, propagan el flujo de forma más directa, mientras que en la segunda el flujo de características atraviesa varias capas de procesamiento. Al final del bloque, ambas direcciones se concatenan, dando lugar a un bloque que mejora la eficiencia computacional del modelo sin sacrificar la capacidad de representación. Un ejemplo gráfico de un bloque CSP se encuentra en la figura 2.15.

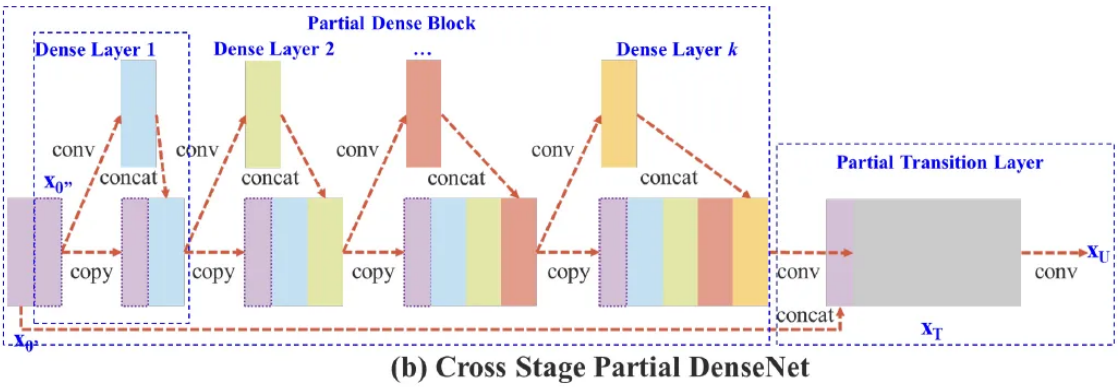

Figura 2.15: Representación gráfica de un módulo CSP. Obtenido en [76]

Neck

Bloque responsable de intermediar entre el *Backbone*, que extrae características, y el bloque *head* que genera la salida del modelo, bien con las predicciones por píxel en el caso de segmentación o con los *bounding-boxes* en el caso de detección. Este bloque está conformado por dos tipos de capas, las *Feature Pyramid Networks* (FPN en adelante) y las *Path Aggregation Network*.

Los bloques *Path Aggregation Network* tienen como objetivo mejorar la agregación de características de diferentes rutas de la red y permiten fusionar las características de diferentes resoluciones obtenidas por la FPN de manera más eficiente, mejorando la precisión en la detección de objetos de tamaño variado. Para esto, se hace uso de capas convolucionales, funciones de agregación que ponderan la importancia de cada camino, y el uso de conexiones de largo alcance, que aplican funciones de agregación como pueden ser sumas o concatenaciones a salidas de capas no adyacentes, mejorando las dependencias a largo plazo.

Head

Bloque encargado de realizar las predicciones, clasificaciones o detecciones finales. Este bloque hace uso de capas de convolución, capas de normalización por batch y capas de activación sigmoide y softmax para la obtención de las coordenadas asociadas a la puntuación de confianza y las probabilidades de cada clase, respectivamente.
La entrada se divide en diferentes cuadrículas, y, como salida final de la capa, para cada una de las celdas de la capa se obtiene:

- Las coordenadas del *bounding box* $(x, y, x + w, y + h)$,
- La confianza, que indica la probabilidad de que un objeto esté presente en el cuadro delimitador,
- Las probabilidades correspondientes a cada clase.

Función de pérdida

Por último, como función de pérdida para su entrenamiento, YOLOv8 hace uso de una compuesta que combina distintos tipos de errores para equilibrar su importancia. Formalmente, se tiene:

$$\mathcal{L} = \lambda_{box} \cdot \mathcal{L}_{box} + \lambda_{obj} \cdot \mathcal{L}_{obj} + \lambda_{cls} \cdot \mathcal{L}_{cls} \tag{2.11}$$

Siendo:

- $\mathcal{L}_{box}$: el error asociado entre la caja predicha y la real (medido usando IoU, explicado en detalle a continuación),
- $\mathcal{L}_{obj}$: el error asociado a la confianza de presencia o ausencia de un objeto en una celda dada (medido mediante *Binary Cross-Entropy*, explicado en detalle en el siguiente apartado),
- $\mathcal{L}_{cls}$: el error asociado por clasificar con una clase diferente un objeto, usando entropía cruzada para ello.

2.3.6. Ensembles

El uso de ensembles hace referencia a la combinación de varios modelos para obtener la predicción final en lugar de realizar la predicción mediante un único modelo, de manera que la predicción combinada mejore a la realizada por cada uno de los modelos de forma individual, al aprovechar las fortalezas de los modelos y compensar sus debilidades [51].

En la literatura se pueden encontrar soluciones mediante ensembles a problemas de segmentación con buenos resultados, como la propuesta de Loreggia et al. [54], donde se combinan modelos como DeepLabV3+, HarDNet-MSEG, y un modelo basado en mecanismos de atención, que empleados conjuntamente, mejoran los resultados individuales de los modelos que componen la arquitectura.

Los predictores ensembles pueden agruparse en tres categorías principales, según el enfoque utilizado [51]:

- Bagging o bootstrap aggregating: consiste en el entrenamiento de diversos modelos haciendo uso de distintos subconjuntos de los datos obtenidos mediante técnicas de submuestreo como el bootstrapping. Esto permite la obtención de diferentes modelos, que, a partir de técnicas como la votación mayoritaria o el promedio, buscan la obtención de un rendimiento superior al de un modelo individual entrenado con el conjunto de datos completo. Este muestreo y el uso de varios modelos permiten reducir la varianza del modelo final, destacando dentro de este conjunto modelos como Random Forest [33].
- Boosting: enfoque consistente en el entrenamiento de modelos de forma secuencial, de forma que cada modelo aprenda a corregir los errores de su modelo antecesor. En este proceso se asignan mayores pesos a aquellas instancias mal predichas por los modelos anteriores, de forma que los modelos posteriores enfocan su esfuerzo en aprender las áreas más problemáticas del conjunto de datos. Dentro de estos, destacan modelos como AdaBoost y Gradient Boosting [33].
- Stacking: consiste en hacer uso de un conjunto de modelos base para obtener un nuevo conjunto de datos, a partir del cual se entrena un nuevo modelo, también denominado *meta-learner*, que realiza las predicciones finales. Este enfoque destaca por su gran flexibilidad para incorporar modelos heterogéneos, como pueden ser los modelos fundacionales, y explotar sus diferentes características. Se muestra un diagrama de este enfoque en la figura 2.16.

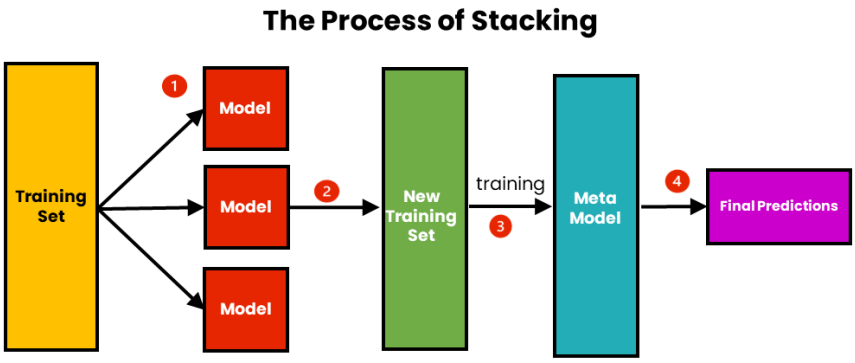

Figura 2.16: Arquitectura enseble tipo stacking y la representación de su flujo de entrenamiento. Obtenida en [70]

2.4. Métricas de rendimiento y funciones de pérdida

En el contexto del aprendizaje profundo, la selección adecuada de la función de pérdida y métricas de rendimiento es crucial para la optimización efectiva y la evaluación eficaz del modelo.

2.4.1. Métricas de rendimiento

Las métricas de rendimiento permiten la comparación entre modelos y la justificación del correcto aprendizaje de los mismos. En el contexto de la segmentación semántica, en la literatura [32] destacan la *intersección sobre la unión*, la *similitud Dice* y la *mean average precision (mAP en adelante)*.

Intersección sobre la unión

La intersección sobre la unión (IoU en adelante) o índice de Jaccard [36] indica en problemas de segmentación y detección de objetos la relación entre la máscara predicha y el *ground truth* de la imagen. En el contexto de la segmentación semántica, representa, en escala $[0, 1]$, la proporción entre la intersección de la predicción y el ground truth y su unión media para cada una de las clases.

Esta, a diferencia de otras métricas como la precisión, donde solo se tienen en cuenta el porcentaje de píxeles predichos como verdaderos positivos calculándose como $(\frac{TP}{TP+FP})$, proporciona una mejor representación del funcionamiento del modelo en problemas con desbalanceo de clases en el conjunto de datos. Esto se debe a que pondera por los mismos pesos las clases que ocupan poco espacio o están menos representadas que las clases mayoritarias. Un ejemplo gráfico de su cálculo se encuentra en la figura 2.17.

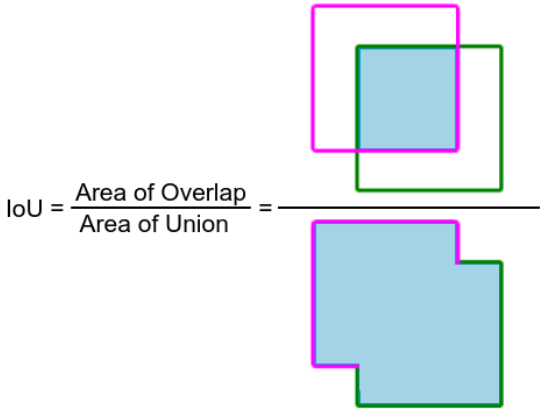

Figura 2.17: Ejemplo gráfico del cálculo de la métrica IoU en segmentación de imágenes. Obtenida en [15]

Formalmente, para un problema de segmentación de una máscara binaria, esta métrica se define de la forma:

$$\text{IoU} = \frac{|T \cap M|}{|T \cup M|} \tag{2.12}$$

Siendo:

- T el ground truth para una clase,
- M la máscara predicha para una clase.

Coeficiente de similitud Dice

El coeficiente de similitud Dice o F1-Score, propuesto inicialmente por Dice en 1945 [19], se emplea como métrica de rendimiento tanto para problemas de detección como de segmentación.

Esta métrica mejora a IoU en datasets donde las dimensiones de las clases son pequeñas frente a la clase fondo, siendo un ejemplo problemas de segmentación de imágenes médicas, al aportar una mayor importancia a los píxeles identificados como verdaderos positivos. Por otro lado, se trata de una métrica menos intuitiva que IoU [52]. Se muestra una representación gráfica del cálculo de esta métrica en la figura 2.18.

2.4.2. Mean average precision (mAP)

Mean Average Precision (mAP en adelante) es una métrica de rendimiento empleada en segmentación de instancias y adaptable a segmentación semántica, cuyo uso se popularizó en 2007 en el PASCAL VOC Challenge [23].

En el contexto de la segmentación semántica, la mAP mide la calidad de la segmentación predicha, considerando la precisión ($\frac{TP}{TP+FP}$) y la sensibilidad o recall ($\frac{TP}{TP+FN}$) de las predicciones a distintos umbrales de IoU, considerando como TP las máscaras predichas si su IoU supera el umbral, FP si no corresponde a ningún objeto real, y FN si se omite un objeto presente.

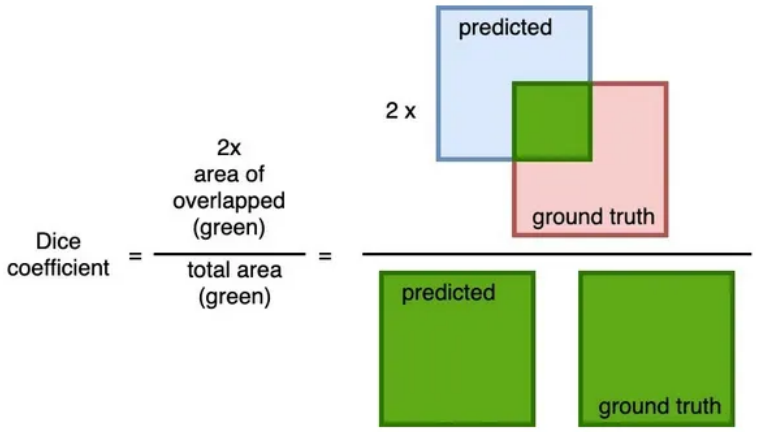

Figura 2.18: Ejemplo del cálculo de la métrica Dice Loss. Obtenida en [55]

El cálculo de mAP se realiza en las siguientes fases:

1. Cálculo del IoU para cada una de las predicciones del modelo en el conjunto de datos.

2. Definición de un conjunto de umbrales para los que se evaluará el IoU de cada predicción. En general, en la literatura para datasets como COCO se emplean umbrales equiespaciados que van de 0.5 a 0.95 de IoU [81].

3. Cálculo de la precisión y del recall para cada uno de los umbrales definidos, para evaluar si cada predicción cumple el umbral de IoU para ser considerada verdadera positiva.

4. Cálculo del área bajo la curva (AUC) que relaciona precisión y recall de cada uno de los umbrales y clases, obteniendo así el Average Precision (AP) de cada uno de estos. Formalmente:

$$\text{AP} = \int_0^1 P(r)\, dr \tag{2.13}$$

   Siendo P la precisión de la predicción y r el valor del recall.

5. Cálculo final del mAP empleando la media del AP del conjunto de clases y umbrales definidos. Formalmente:

$$\text{mAP} = \frac{1}{N_{umbrales} \times N_{\text{clases}}} \sum_{t=0}^{umbrales} \sum_{c=1}^{clases} AP^{(c)}_{\text{IoU}(0{,}5+0{,}05t)} \tag{2.14}$$

La mAP permite tanto evaluar cómo el modelo realiza predicciones precisas de las instancias de las imágenes como su habilidad para la detección de todos los objetos relevantes, suponiendo un buen indicador del funcionamiento global del modelo. Por contra, cabe destacar que se trata de una métrica que, frente a otras como IoU o el coeficiente de similitud Dice, es más costosa computacionalmente y difícil de interpretar [25].

2.4.3. Funciones de pérdida

La función de pérdida es aquella encargada de guiar el proceso de optimización de los parámetros de la red neuronal durante su entrenamiento. Esta es la empleada para actualizar de forma óptima cada uno de los parámetros de la red mediante su gradiente en cada iteración del entrenamiento.

Entre las funciones de pérdida más comunes en la literatura para resolver problemas de segmentación semántica se encuentran la *entropía cruzada*, *Dice Loss* y *Focal Loss*, además de las funciones fruto de la combinación ponderada de varias funciones.

2.4.4. Categorical Cross-Entropy

La entropía cruzada o Categorical Cross-Entropy (CCE en adelante) es una función de pérdida más usada en problemas de clasificación multiclase. En segmentación semántica multiclase, compara las distribuciones de probabilidad asignadas por el modelo a cada píxel con su ground truth, definiéndose de forma formal tal que:

$$\mathcal{L}_{CCE} = -\sum_{c=1}^{C} y_c \log(p_c) \tag{2.15}$$

Siendo:

- C el número total de clases,
- $y_c \in \{0, 1\}$ el ground truth del píxel para máscaras binarias,
- $p_c \in [0, 1]$ la probabilidad predicha para la clase c.

Su principal desventaja es su sesgo hacia la optimización de clases mayoritarias, al aportar la misma importancia a cada uno de los píxeles. Esto implica que no es adecuada para problemas donde las clases no estén balanceadas.

2.4.5. Dice Loss

La Dice Loss es una función de pérdida derivada del coeficiente Dice, que enfatiza las clases minoritarias a diferencia de la entropía cruzada, al ponderar la superposición de píxeles predichos y reales, enfatizando clases minoritarias, haciéndola de gran utilidad en conjuntos de datos desbalanceados.

Formalmente, para un problema de segmentación semántica se puede expresar de la forma:

$$\text{Dice Loss} = 1 - \frac{2\sum_i p_i y_i + \epsilon}{\sum_i p_i + \sum_i y_i + \epsilon} \tag{2.16}$$

Siendo:

- p_i la probabilidad predicha para el píxel i,
- y_i la etiqueta del píxel i,

- ϵ, un pequeño valor para evitar división por cero.

Cabe destacar también la métrica Log-Cosh Dice Loss [4], que añade una función log-cosh a la anterior para suavizar su derivada en el intervalo $[0, 1]$, definida como:

$$L_{\text{lc-dice}} = \log\big(\cosh(\text{Dice\_Loss})\big) \tag{2.17}$$

2.4.6. Focal Loss

Focal Loss es una función de pérdida propuesta por Lin et al. en 2017 [45] en el contexto de detección de objetos, siendo la función de pérdida aplicada a redes como RetinaNet. Esta ha sido adaptada a problemas de segmentación semántica, proporcionando mejores resultados en problemas con clases desbalanceadas, al focalizar el aprendizaje en píxeles difíciles.

Mejora los resultados frente a otras funciones de pérdida vistas anteriormente en problemas con fondos dominantes. Además, está dotada de un factor de modulación γ, que le permite ajustar el peso y ponderar con mayor importancia clases difíciles de aprender, particularmente para objetos pequeños o poco visibles en las imágenes.

Formalmente, puede definirse tal que:

$$\mathcal{L}_{Focal}(p_t) = -\sum_{c=1}^{C} \alpha_c (1 - p_c)^{\gamma} y_c \log(p_c) \tag{2.18}$$

Siendo:

- p_t la probabilidad predicha para la clase verdadera,
- γ el parámetro de focalización, que ajusta el enfoque en máscaras de instancias difíciles,
- $\alpha_c \in [0, 1]$ el factor de ponderación de cada clase, que evitar el desbalanceo de clases.

Algunos ejemplos de casos de uso de éxito de esta función de pérdida en la literatura son Lin et al. [44] con la RetinaNet y Crasta et al. [17], mediante la arquitectura 3D-VNet.

2.4.7. Combinación de funciones de pérdida

La combinación de funciones de pérdida es una de las estrategias más utilizadas en la literatura para la resolución de problemas de segmentación mediante modelos fundacionales. Una combinación repetida es la combinación de Focal Loss y Dice Loss, donde Dice Loss corrige el desbalance y maximiza la superposición, mientras que Focal Loss penaliza con mayor peso los errores en píxeles difíciles de predecir. Formalmente, se puede expresar de la forma:

$$\mathcal{L} = \alpha \cdot \text{FocalLoss} + \beta \cdot \text{DiceLoss} \tag{2.19}$$

Podemos encontrar ejemplos exitosos de esta práctica en la literatura en arquitecturas como YOLOv8 [13], donde se combinan ponderaciones de *BCE* e *IoU Loss*.

Capítulo 3

Experimentos realizados

A lo largo de este capítulo se explicará de forma detallada todo el proceso completo que se ha seguido de cara a la resolución del problema de segmentación, guiado por una metodología, abarcando así todas las fases de un problema real de ciencia de datos.

3.1. Metodología aplicada

Como metodología a aplicar para la estructuración de las fases del proyecto se ha decidido hacer uso de *CRISP-DM*. Esta tiene su origen en empresas dedicadas a la minería de datos y fue formalizada en 1999 con la publicación de la *guía CRISP-DM* [11] y diseñada con el objetivo de proporcionar un framework reutilizable aplicable en proyectos de minería de datos. Dentro de los objetivos de la misma, destacan facilitar la organización de tareas, mejorar la comunicación y colaboración entre equipos, estructurar el proyecto de forma iterativa y flexible, responder de forma adecuada ante nuevos descubrimientos y garantizar una calidad de los resultados obtenidos por el proceso.

Las fases que conforman este framework son las siguientes [10]:

1. Comprensión del negocio: incluye tanto la definición del problema a resolver como la priorización de los objetivos del proyecto.

2. Comprensión de los datos: abarca tanto la carga del conjunto global de datos usados de partida hasta la exploración de los mismos, de cara a la detección de posibles problemáticas como la falta de valores, presencia de anomalías o errores en los mismos.

3. Preparación del conjunto de datos: tiene como objetivo la obtención de los datos en un formato adecuado para alimentar el entrenamiento del modelo usado a posteriori. Esta fase comprende la limpieza e imputación de valores en los datos, la selección y extracción de características y la partición de los conjuntos de datos.

4. Modelado del conjunto de datos: incluye tanto la selección del modelo como el ajuste hiperparamétrico del mismo, de forma que a partir de este se puedan ofrecer soluciones a los problemas planteados en las fases anteriores.

5. Evaluación del modelo desarrollado: se estudian los resultados obtenidos por el modelo entrenado en la anterior fase, teniendo en cuenta diferentes métricas, la interpretabilidad del mismo, y la alineación de los resultados con el problema a resolver.

6. Despliegue: puesta en marcha del modelo en un entorno de producción que sea accesible y monitorizable para su uso.

Un esquema de esta metodología y el carácter iterativo de la misma pueden observarse en la figura 3.1.

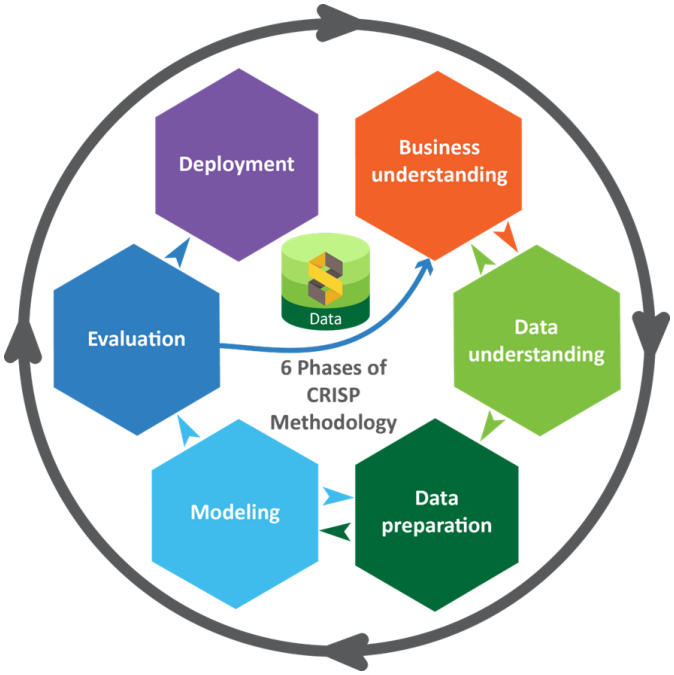

Figura 3.1: Esquema de las fases que componen la metodología CRISP-DM. Imagen obtenida en [18]

3.2. Entorno de desarrollo e infraestructura

Los resultados han sido obtenidos haciendo uso de dos entornos diferentes. En primer lugar, para el desarrollo y testeo inicial de cada uno de los scripts de análisis del conjunto de datos y modelado, se ha hecho uso de un portátil con procesador AMD Ryzen 5 7000 series con GPU AMD RADEON y 16 GB de memoria RAM. En segunda instancia, para la obtención de los modelos convolucionales, que requieren recursos computacionales mayores para su entrenamiento, se ha hecho uso de los servidores de Google Colab con GPU Tesla T4.

El código que muestra en detalle el procedimiento seguido puede encontrarse en el repositorio de GitHub público del siguiente enlace:
https://github.com/rucasca/TFM.

3.3. Carga del conjunto de datos

En este caso se ha decidido hacer uso del dataset MS COCO en su más reciente actualización, que data del año 2017, siendo posible acceder al mismo en el siguiente enlace [1]. La decisión de hacer uso de este dataset frente a otros datasets como *ADE20K* radica en su alto volumen de datos y mayor variedad de clases etiquetadas, además de ser más útil de cara al modelado del problema a tratar.

Este dataset destaca por contener, en su última versión, más de 300.000 imágenes, entre las cuales se encuentran más de 1,5 millones de objetos diferentes segmentados de hasta un total de 80 clases. Fue propuesto inicialmente por Microsoft en 2014 y actualizado en 2017 y puede ser utilizado para resolver problemas tanto de detección de objetos como de segmentación semántica y segmentación de instancias. Además, este contiene keypoints de las 17 partes del cuerpo humano más relevantes para la detección de poses.

Cabe destacar una gran variedad de resoluciones y orígenes geográficos de las imágenes, al ser obtenidas a partir de plataformas como *Flickr* de forma automática, que ayudan a que el entrenamiento de un modelo con este conjunto de datos generalice mejor.

3.4. Análisis exploratorio

Antes de proceder con el entrenamiento de modelos que resuelvan el problema a tratar, se decide realizar un breve análisis exploratorio de nuestro conjunto de datos que permita identificar patrones en las imágenes y posibles problemáticas que afectarán al entrenamiento del modelo, como el desbalanceo de clases.

En primera instancia, antes de realizar un análisis más elaborado, se visualizan las imágenes y máscaras de muestras aleatorias.

Tanto las máscaras de segmentación como el diccionario que asocia el identificador de la imagen con el path en el que se encuentra alojada están almacenadas en formato JSON, implicando un tratamiento adicional para su uso en el entrenamiento del modelo. Una primera inspección de su estructura muestra que los datos de las segmentaciones se representan mediante una lista de coordenadas (x, y) que las definen, junto con la clase correspondiente. Este formato dificulta no solo su uso directo en el entrenamiento del modelo, sino también la realización de tareas más sencillas, como su representación gráfica.

Para una manipulación de los datos más fluida se hace uso de la API de COCO [43], que facilita métodos para el muestreo de imágenes y generación de las máscaras.

[1] `https://cocodataset.org/#home`

En lo relativo a las imágenes, se puede determinar que, a simple vista, tienen temáticas muy variadas, siendo un ejemplo el de la figura 3.2.

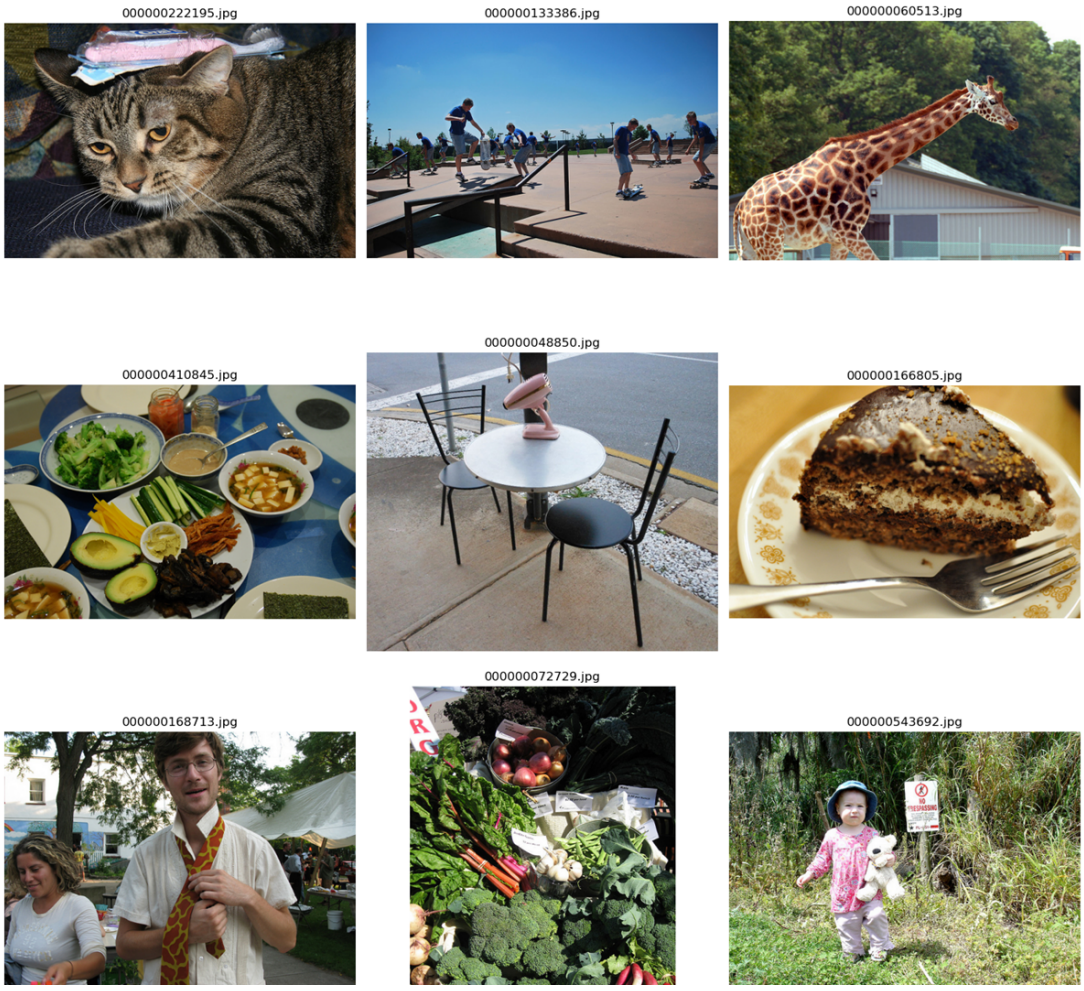

Figura 3.2: Primera visualización de nueve instancias aleatorias del conjunto de datos

En cuanto a las máscaras, como las presentes en la figura 3.3, se pueden detectar varias características en las mismas que dificultarán el entrenamiento. La primera de ellas se trata del solapamiento entre objetos, haciendo que ciertas instancias queden divididas en varias partes. Adicionalmente, destaca la presencia de máscaras de tamaño ínfimo, difíciles de apreciar por el ojo humano, que tendrán implicaciones importantes de cara a la selección de la función de pérdida. Esto hará que podamos considerar el dataset COCO como un dataset difícil, que requerirá el uso de un modelo complejo para su modelización correcta.

Tras esta primera visualización de instancias, el análisis exploratorio podrá ser dividido en dos fases, una primera donde se estudian las distribuciones individuales de cada clase y una posterior donde se estudian relaciones entre clases.

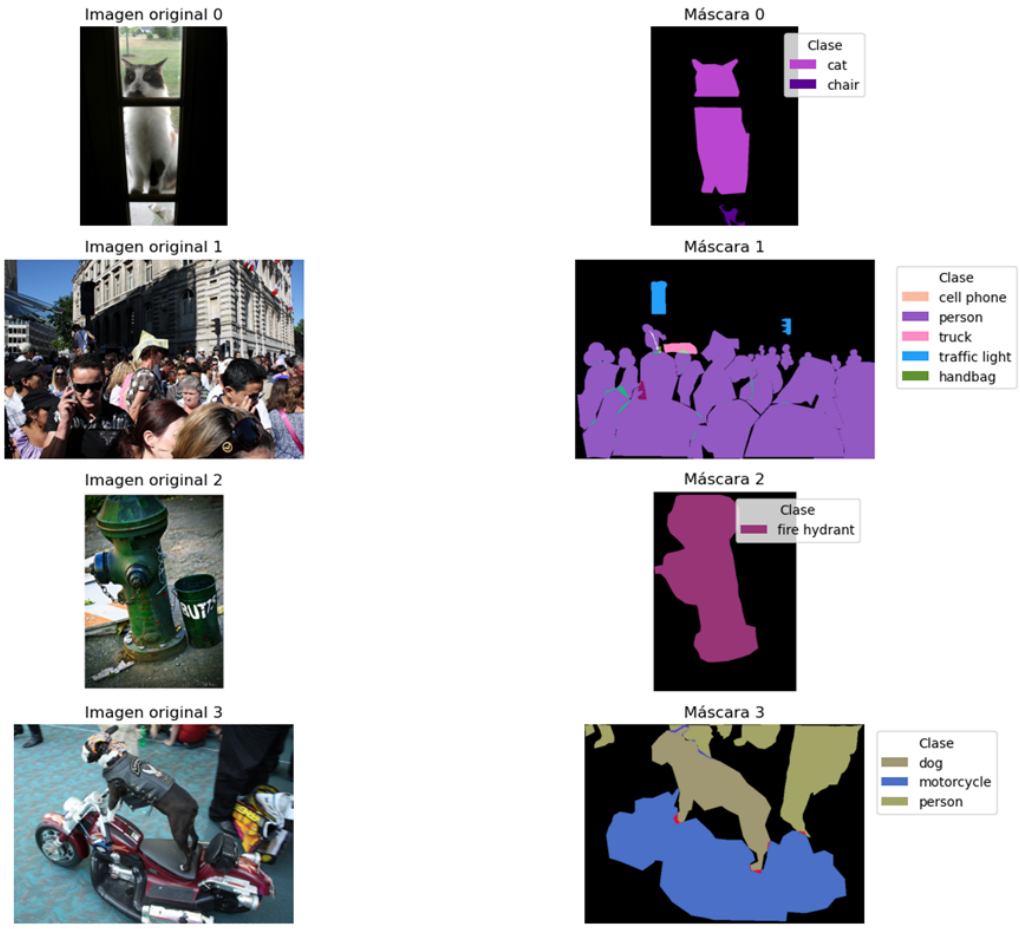

Figura 3.3: Primera visualización de máscaras del conjunto de datos

3.4.1. Análisis exploratorio monoclase

En primera instancia se ha estudiado la distribución de cada clase de forma individual en el conjunto de imágenes, que permitirá sacar conclusiones en cuestiones como el balanceo de las clases.

La distribución de instancias de cada clase en el conjunto de imágenes queda reflejada en la figura 3.4, donde se observa una presencia mayoritaria de la clase *persona* frente a otras clases, existiendo clases muy poco representadas dentro del conjunto de datos.

En lo relativo a la cantidad de píxeles promedio por clase, se tiene que la distribución de cada una de las clases es, de nuevo, diferente, como se observa en la figura 3.5. Esto implica que, pese a realizar un muestreo balanceado para entrenar un modelo no sesgado a ciertas clases, la función de pérdida empleada para su entrenamiento ha de aportar la misma importancia a cada instancia, independientemente de su tamaño en píxeles.

Esto implica que funciones de pérdida como *Categorical Cross Entropy* per se no sean adecuadas para entrenar el modelo.

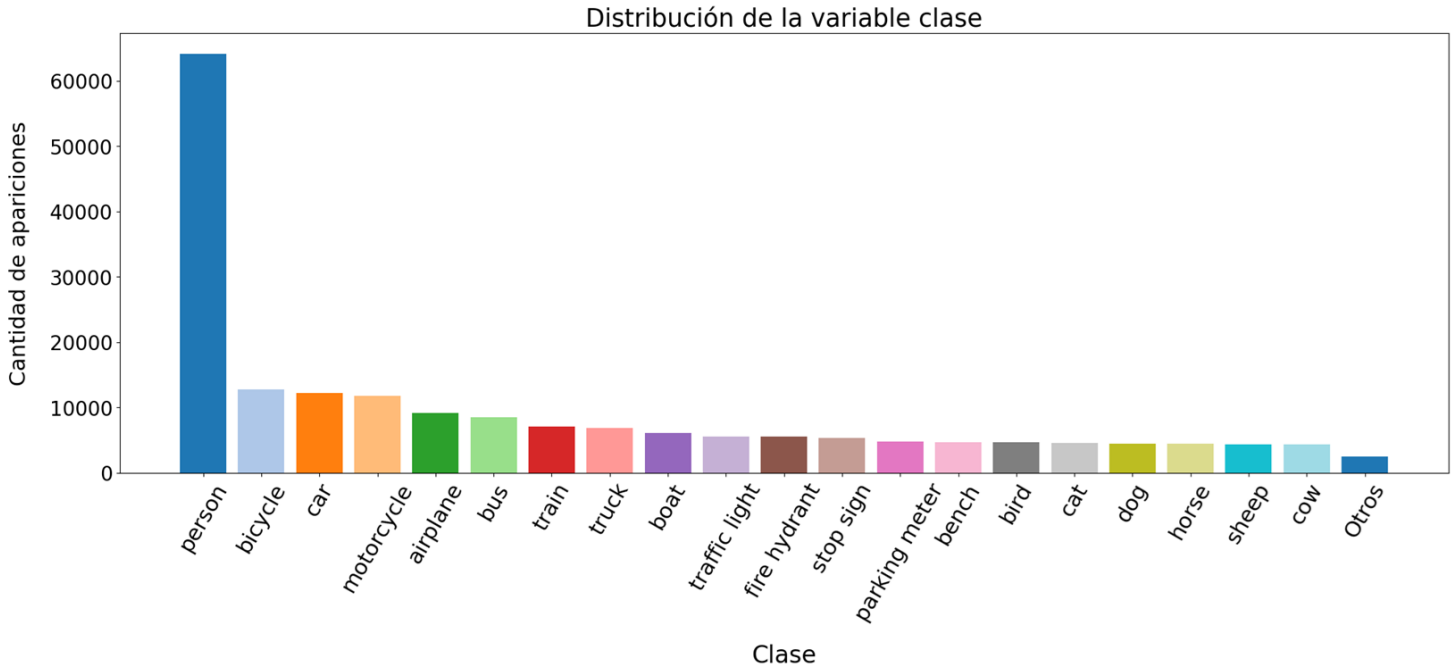

Figura 3.4: Visualización de la distribución de instancias de cada clase en el conjunto de datos

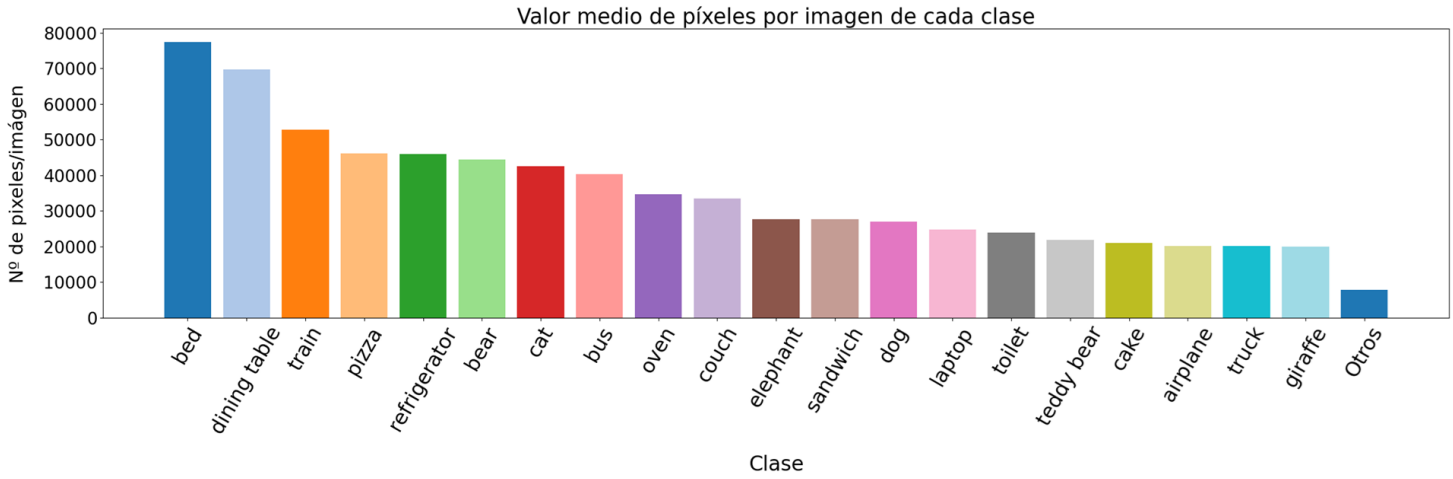

Figura 3.5: Visualización de la distribución de los píxeles por clase en las imágenes

Además, se estudia la distribución de la frecuencia de aparición promedio de cada clase, representada en la figura 3.6, dado que clases con una frecuencia alta estarán más fragmentadas en las imágenes, dificultando su aprendizaje al modelo. La figura nos permite determinar que clases como *sheep* o *person* tienen tendencia a aparecer de forma más fragmentada en las imágenes.

3.4.2. Análisis exploratorio multiclase

En este caso, se ha considerado relevante estudiar de forma breve las frecuencias de coocurrencia de cada una de las clases en el conjunto de datos. Para ello, se ha considerado el uso de grafos no dirigidos.

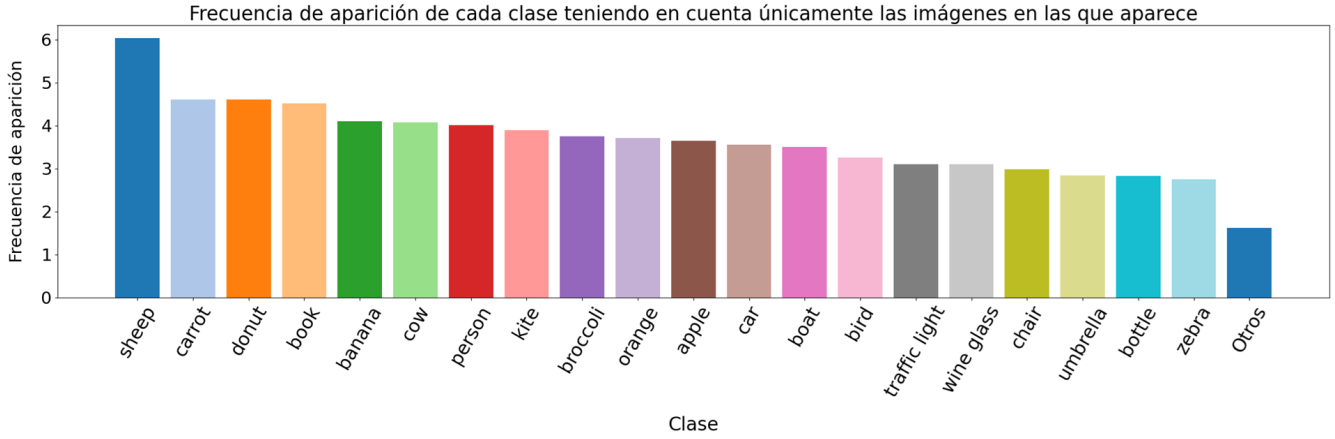

Figura 3.6: Visualización de la distribución de frecuencias de aparición por imágenes donde al menos hay una instancia presente de la clase

Un grafo que muestra las frecuencias de coocurrencia, generado a partir de la matriz de coocurrencia de las clases, se muestra en la figura 3.7. En la figura podemos observar que la presencia de algunas clases implica con una mayor probabilidad la aparición de otras, algo a tener en cuenta al hacer el muestreo de un conjunto balanceado, dado que existen combinaciones muy poco frecuentes que imposibilitarían el muestreo de un conjunto de entrenamiento completamente balanceado.

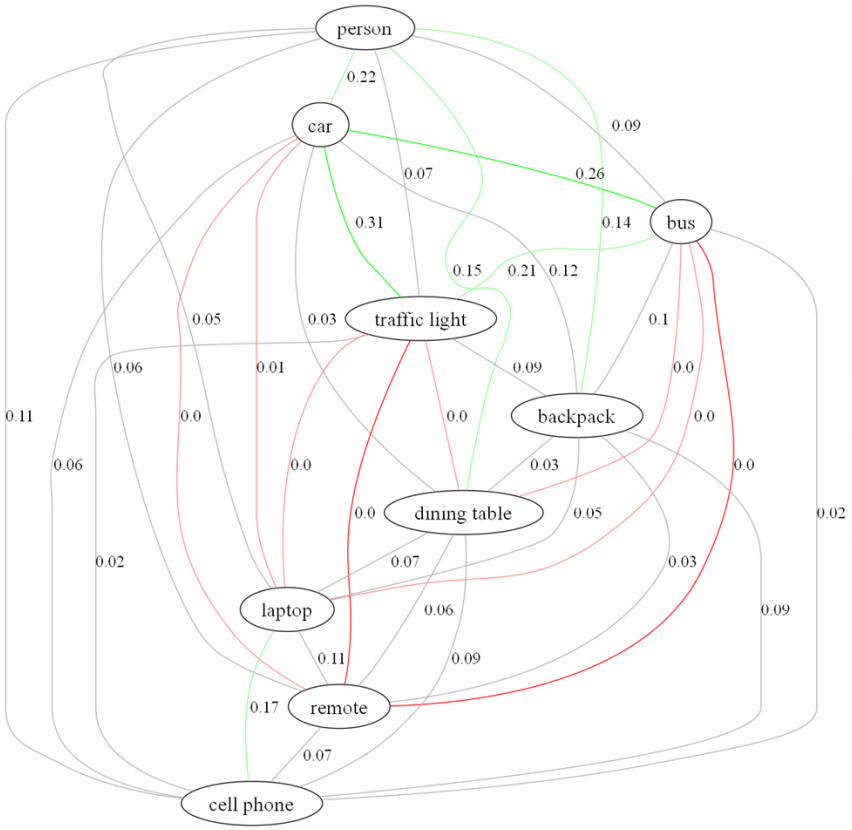

Figura 3.7: Visualización de las frecuencias de coocurrencia en un subconjunto de las clases del dataset de entrenamiento

3.5. Preparación del conjunto de datos

Antes del entrenamiento del modelo, se llevan a cabo algunos ajustes en el conjunto de datos que nos permitirán la obtención de un mejor modelo capaz de generalizar y ajustarse al caso de uso dado en función de las problemáticas encontradas en la fase de análisis exploratorio del conjunto de datos.

Es por ello por lo que, para la resolución del problema, las decisiones tomadas en lo relativo al conjunto de datos han sido las siguientes:

- Eliminación de clases: dado que en el dataset están presentes clases escasamente representadas y que son redundantes para nuestro problema, eliminamos las máscaras de estas y centramos nuestro modelo en el aprendizaje de 10 clases relevantes para problemas de movilidad de personas con discapacidad visual. Estas clases se ha decidido que sean: *person, car, motorcycle, bus, traffic light, backpack, handbag, cell phone, chair y dining table*.

- Uso de cropping aleatorio: una de las grandes problemáticas del dataset es la frecuencia de coocurrencia entre las diferentes clases que conforman el dataset no está equilibrada, como se mostró en la figura 3.7. Es por ello que, con el objetivo de solventar este problema, se han generado imágenes artificiales mediante recortes de instancias de otras imágenes aleatorias. Esta práctica logra obtener, de un dataset tan limitado en este aspecto como este, imágenes con frecuencias de coocurrencia equilibradas que facilitarán el aprendizaje del modelo.

- Particionamiento en train y test: se emplean las particiones proporcionadas por el propio dataset, conteniendo en el conjunto de train hasta 200.000 instancias. En lo relativo a la generación de conjuntos de entrenamiento para modelos ensemble tipo *stacking*, se muestrearán estos para la obtención de conjuntos de entrenamiento más pequeños, computacionalmente abordables y que no incurran en fuga de datos.

La aplicación de este *random cropping* se realiza de forma uniforme entre las clases objetivo, seleccionando en primer lugar de forma uniforme la clase a colocar y posteriormente de forma aleatoria la máscara perteneciente a la clase. Cabe destacar que, en esta fase, para cada imagen se incluyen entre 3 y 5 instancias aleatorias que superan un umbral mínimo de píxeles, que en este caso se ha definido a 100 para evitar elementos de tamaño mínimo.

La aplicación de random cropping sobre una imagen aleatoria del conjunto de train da como resultado la imagen de la figura 3.8. En esta destaca que, si bien el resultado no es completamente natural, elimina por completo el problema del desbalanceo de clases en el conjunto de datos.

Figura 3.8: Resultado de la aplicación de random cropping sobre una imagen del conjunto de de datos

Como resultado final, se obtiene un conjunto de datos prácticamente balanceado, donde continúa teniendo una ligera mayor presencia la clase persona, pero no de una forma muy significativa frente al resto de clases, lo que posibilita un entrenamiento efectivo de los modelos.

Algo a tener en cuenta para el entrenamiento es la cantidad de píxeles que ocupa en promedio cada instancia de cada objeto dentro de la imagen, que continúa estando desbalanceada y cuya solución no es trivial. Esto se solventará en el propio entrenamiento, con la selección de la función de pérdida adecuada, como se verá posteriormente en cada modelo.

3.5.1. Pipeline de procesamiento

Uno de los grandes problemas presentes debido a la gran dimensionalidad del dataset es su gran tamaño, debido al amplio volumen de imágenes que contiene y la variabilidad y tamaño de las mismas. Esto hace inabordable para las capacidades de cómputo del equipo de trabajo su carga completa en memoria de forma simultánea.

Es por ello que, para el entrenamiento del modelo, se ha visto necesario hacer uso de una estructura de datos que permita la carga de imágenes *on-the-fly*, además de implementar un preprocesamiento previo que permita adaptar las características de las mismas de forma consistente para poder ser empleadas por un modelo de tamaño de entrada fijo.

Para la obtención de imágenes y máscaras con una resolución fija, en este caso fijado a (256×256), se ha empleado como método para el redimensionamiento la interpolación bilineal, implementada en la librería *opencv* [73].

Para el almacenamiento y carga del conjunto de datos de forma dinámica, se ha optado por utilizar como estructura de datos *tf.tfrecord* cuya documentación se encuentra en la web de TensorFlow [74]. Esta estructura de datos destaca por almacenar los datos en formato binario, ocupando menos espacio en disco y acelerando la lectura en la fase de entrenamiento. Además, permite la construcción de pipelines de carga optimizados, al permitir el *prefetching* de instancias y el paralelismo. Tfrecord permite ayudar a superar las dos principales limitaciones asociadas a la resolución del problema: la restricción espacial por la gran dimensionalidad de los datos y la restricción temporal del entrenamiento del modelo.

Data augmentation

Con el objetivo de mejorar la capacidad de generalización del modelo y mitigar el riesgo de sobreajuste, se ha implementado una estrategia de data augmentation dentro del pipeline de entrenamiento. Esta etapa parte de un conjunto de datos ya preprocesado y almacenado como *tf.tfrecord*.

Asimismo, otro de los principales retos del presente trabajo es la limitación de recursos computacionales y el alto coste temporal que conlleva el entrenamiento del modelo. Por ello, se opta por una estrategia de data augmentation en tiempo real, mediante la cual se generan versiones alteradas de las imágenes originales dinámicamente durante cada iteración del entrenamiento. De esta forma, no se incrementa el tamaño del conjunto de datos en disco, pero sí se aumenta la diversidad de los ejemplos que el modelo observa en cada época.

Las transformaciones aplicadas durante esta fase tienen como propósito introducir variaciones en las imágenes que permitan al modelo aprender características más robustas y generales, evitando que memorice patrones específicos del conjunto de entrenamiento, es decir, sobreajuste. Estas transformaciones incluyen:

- Movimientos de traslación horizontal y vertical aleatorios con un 0,3 de probabilidad para cada tipo de movimiento, aplicando una traslación uniforme en el rango $[-10\,\%, 10\,\%]$.
- Volteo horizontal con probabilidad de 0,4.
- Movimientos de rotación de 90° en sentido tanto horario como antihorario con probabilidad de 0,05.
- Cambios en el brillo de la imagen, con una probabilidad de 0.4 y variaciones, aumentando o disminuyendo este en un intervalo uniforme de $[-10\,\%, 10\,\%]$.
- Cambios en el contraste de la imagen, con las mismas características y probabilidad que el cambio de brillo.
- Inclusión de ruido gaussiano, tomando $\mu = 0$ y $sd = 0{,}05$ para su aplicación a cada uno de los píxeles con una probabilidad de 0,3.
- Adicionalmente, para el modelo final tipo embedding, cuya capa intermedia utiliza un canal adicional derivado de la salida de sus modelos anteriores, se incluye con una probabilidad de 0.1 para cada píxel la activación como 0 de la salida del modelo.

Un ejemplo de la aplicación de la función de data augmentation definida sobre una imagen aleatoria del conjunto de datos, tras la aplicación de random cropping, se encuentra en la figura 3.9

Figura 3.9: Resultados de la aplicación de data augmentation tras random cropping en una imagen del dataset

Un diagrama completo de la pipeline completa donde se representa el flujo de entrada del modelo en el entrenamiento, incluyendo random cropping y data-augmentation, se puede observar en la figura 3.10

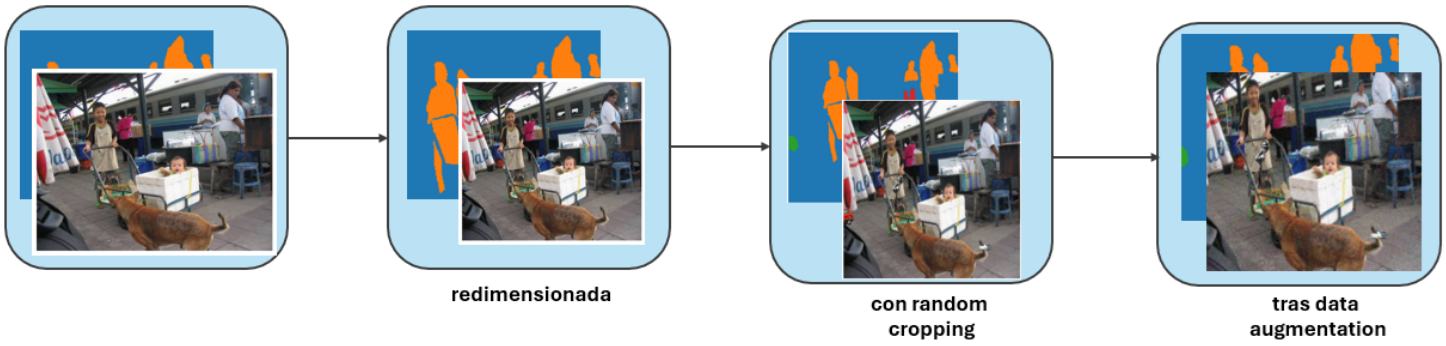

Figura 3.10: Representación del flujo seguido por cada instancia al aplicar data augmentation previo a la ingesta por el modelo

3.6. Modelos propuestos

En cuanto a los modelos utilizados, se han explorado diversas aproximaciones con el objetivo de evaluar los resultados obtenidos ante diferentes arquitecturas.

Entre ellas, se incluye el uso de modelos más simples, como variantes de la red U-Net, entrenada desde cero con el conjunto de datos específico del estudio, en contraste con enfoques que emplean modelos preentrenados sobre otros conjuntos de datos. Asimismo, se han considerado arquitecturas que abordan el problema en múltiples etapas, comenzando con una fase de detección de objetos, seguida posteriormente por una fase de segmentación.

3.6.1. Modelos baseline U-Net

Como primer modelo, considerado como *baseline* por su simplicidad frente al resto, se hace uso de la U-Net, adaptada a la dimensionalidad de nuestras imágenes de entrada. En este caso, debido a los problemas de cómputo limitantes por parte de Google Colab, se hace uso de una variante de la U-Net ligera, donde se elimina un bloque de profundidad y se adapta el número de filtros a aplicar por capa.

Es por ello que como dimensión de entrada se tienen imágenes de (256×256) y 3 canales, procedentes de la pipeline previamente definida alimentada con el preprocesamiento de los datos y data augmentation. En el primer bloque de contracción se emplean sucesiones de hasta 3 bloques conformados por dos capas convolucionales 2D, seguidos de un max pooling y una capa de activación $ReLU$.

En el bottleneck del modelo se tienen dos capas convolucionales 2D que se corresponderán con las capas de mayor profundidad del modelo y que trabajarán con características de la imagen más complejas, en este caso de dimensionalidad 64x64 y con un total de 128 canales.

En la fase expansiva del modelo se suceden bloques de convolución transpuesta precedidos de una concatenación con una capa del bloque expansivo y, posteriormente, dos capas de convolución 2D.

Para las capas convolucionales del modelo se hace uso de kernels de tamaño 3x3, con un *stride* de 1 píxel, *padding=same*, y función de activación $ReLU$.

Además, se incluye una capa convolucional 2D que dará lugar a la salida final, en este caso de un kernel de tamaño 1x1, siendo su salida la imagen segmentada de tantos canales como número de clases se usan en el entrenamiento, en este caso 11, dado que la primera de ellas se tratará del fondo.

Todo esto da lugar a un modelo que usa un total de más de 400,000 parámetros entrenables, que ocupan un total de $1,78MB$ en memoria principal. Una imagen de la misma se puede encontrar en la figura 3.11.

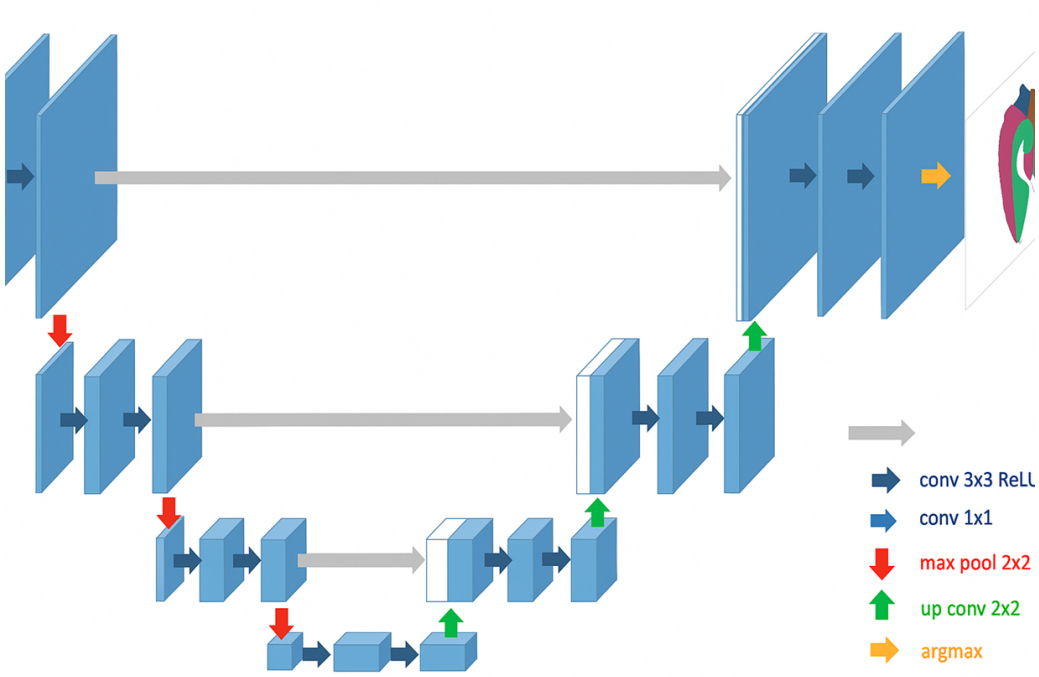

Figura 3.11: Arquitectura de la variante ligera del modelo U-Net, con cuatro niveles de profundidad. Imagen obtenida a partir de [40].

Por último, como optimizador se hace uso de Adam con un learning rate inicial de 10^{-3}. Se decide hacer uso de este por su convergencia rápida y estable, además de por su ajuste optimizado del *learning-rate*. Cabe destacar que, para un entrenamiento más efectivo, se implementan diversas *callback* encargadas de guiar y monitorizar el mismo. Entre estas se encuentran *ModelCheckpoint* para guardar los pesos del modelo que optimiza los resultados, *EarlyStopping* para detener el entrenamiento del modelo cuando no se producen mejoras en el conjunto de validación en 5 épocas y *ReduceLROnPlateau*, encargada de reducir el learning rate tras 3 épocas sin mejoras y así facilitar la convergencia.

En lo relativo a la función de pérdida empleada, de forma inicial se hizo uso de IoULoss, pero, tras diversas pruebas, se logró obtener mejores resultados mediante una función de pérdida combinada. En este caso, la función de pérdida pondera CCE e IoU Loss, dándole una importancia de $0,6$ sobre 1 a esta última. Las mejoras de los resultados de esta función radican en que se balancea la precisión por píxel de CCE con la calidad y la obtención de formas razonables y detección de objetos pequeños y menos representados de IoU Loss. Cabe destacar que el intervalo de los valores que puede tomar durante el entrenamiento pueden superar los del intervalo $[0, 1]$ por esta primera función, a diferencia del uso de IoU Loss únicamente.

La curva de entrenamiento de este modelo puede observarse en la figura 3.12. En esta destaca que podría haberse aumentado en alguna época el valor del *early stopping* hasta que la convergencia fuera más clara, usando un valor mayor a 5. En cambio, dado que la ganancia sería ínfima, el entrenamiento es computacionalmente costoso y los recursos, aun usando Google Colab Pro, son limitados, se ha mantenido el entrenamiento al tratarse del modelo baseline.

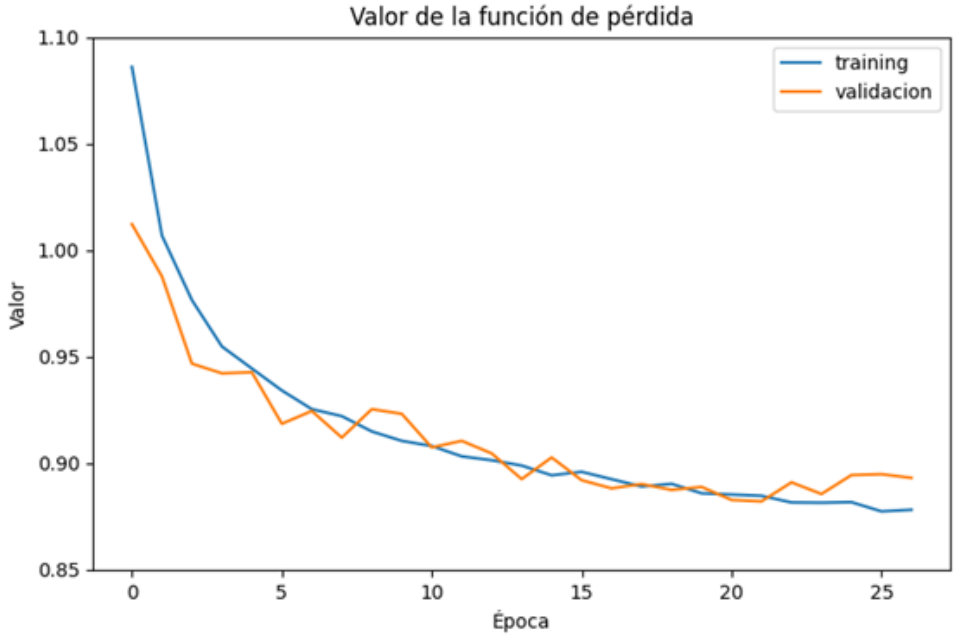

Figura 3.12: Curva de entrenamiento del modelo Baseline U-Net, donde se representa el valor de la función de pérdida a lo largo de las épocas

En cuanto a una primera visualización de los resultados sobre una imagen del conjunto de test, observable en la figura 3.13, muestra la capacidad de detección del modelo de la mayoría de las instancias, aunque con fronteras muy difusas, en especial para objetos como la motocicleta, todo ello pese a usar una función de pérdida parcialmente conformada por IoU Loss, debido al uso de una arquitectura de mayor simplicidad que la U-Net original.

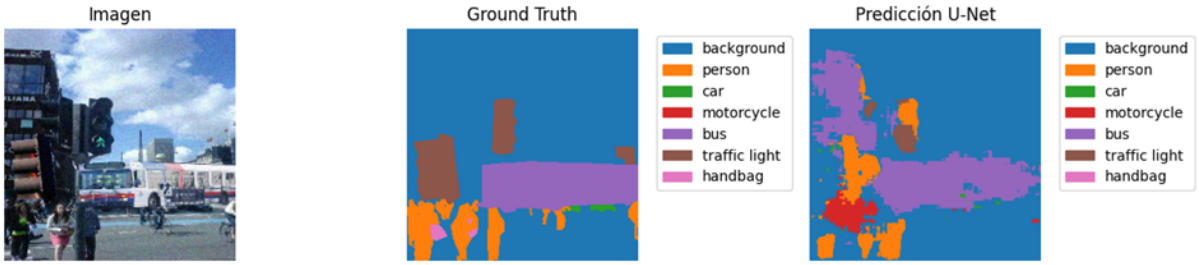

Figura 3.13: Primera visualización de la inferencia de la red U-Net sobre una imagen del conjunto de test frente a su ground truth.

Asimismo, cabe destacar la gran dificultad y el mal etiquetado del propio dataset, donde no se etiquetan vehículos como la motocicleta de la izquierda de la imagen, que sí es detectada por nuestro modelo, lo que cuestiona la calidad de los datos del dataset. Una vez comprobado esto, se pasa a definir modelos de mayor complejidad que tratarán de solventar este problema de forma más eficaz.

3.6.2. Ensembles de modelos

Con el objetivo de mejorar los resultados obtenidos por el modelo *baseline* en tareas de segmentación de imágenes en este dataset, se ha explorado el uso de enfoques ensemble basados en modelos preentrenados con grandes volúmenes de datos. Con ello, se busca mejorar los resultados obtenidos en el dataset COCO sin la necesidad de un entrenamiento desde cero, que sería inviable dado los recursos computacionales disponibles, que ya suponen una limitación en el modelo *baseline*.

Se plantearán dos tipos de arquitecturas ensemble tipo *stacking* diferentes en función de su complejidad, y, para cada una de estas, distintas variantes obtenidas por hacer uso de diferentes modelos como sus componentes. Algunos de los modelos empleados en la composición de estas arquitecturas son *YOLOv8*, *RetinaNet*, *SAM* y *CLIP*. Los dos tipos de arquitecturas propuestas son:

- Arquitectura de dos bloques: conformada por dos modelos que realizan la segmentación de forma conjunta en dos fases, como puede ser mediante primero una detección y posteriormente una segmentación de las detecciones.
- Arquitectura de tres bloques: conformada por un segmentador, un detector y un modelo corrector.

Arquitectura ensemble con YOLOv8 y SAM

Como primer ensemble de la primera aproximación se estudia el uso de un modelo compuesto por un modelo detector de objetos, YOLOv8, y un segmentador, siendo este último SAM.

Un esquema de la arquitectura puede encontrarse en la figura 3.14, donde se muestra un primer bloque, encargado de la obtención de las detecciones de objetos, dando lugar a un *bounding-box* que delimita cada una de las instancias identificadas como clases objetivo. Esta salida se hace uso como entrada para SAM, dado que uno de sus parámetros aceptados se trata de los cuadros delimitadores, dando lugar como salida final a las máscaras de segmentación semánticas.

En este caso, como única parametrización se tiene el ajuste de θ, que indica el grado de sensibilidad del modelo final para determinar si una máscara de segmentación ha de ser aceptada de acuerdo a la probabilidad de salida de SAM. El resto de parámetros no requieren un ajuste dado que ambos modelos que componen la pipeline de inferencia tienen sus pesos preentrenados.

Para la parametrización de este valor θ se ha hecho uso de *grid search* con un conjunto de valores fijos, debido a la fuerte restricción temporal de un testeo más exhaustivo.

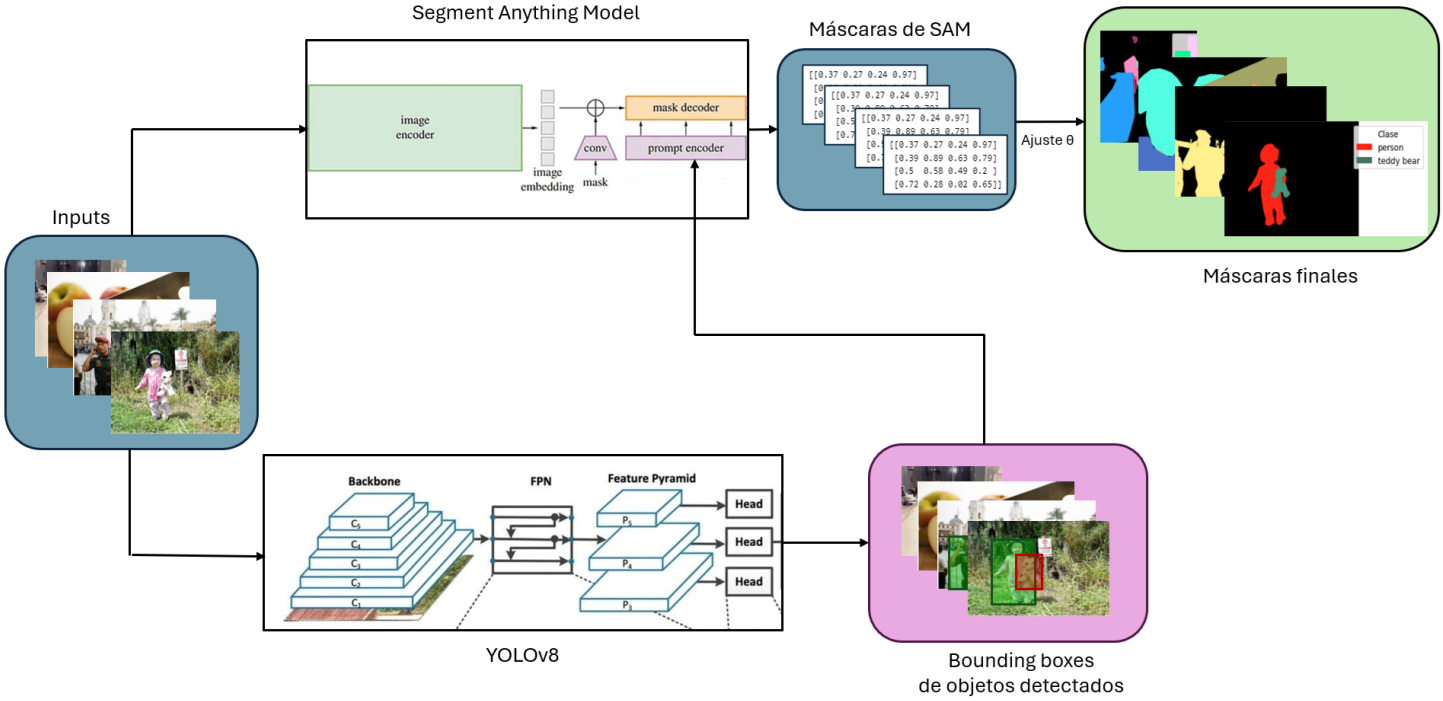

Figura 3.14: Esquematización de la primera arquitectura ensemble compuesta por los modelos YOLOv8 y SAM

Al igual que con el primer modelo baseline, se obtiene una primera visualización para sacar unas primeras conclusiones del funcionamiento del modelo, mostrándose en la figura 3.15.

En base a esta, se puede destacar que el funcionamiento del ensemble es bastante bueno, aunque cabe destacar que quizá instancias pequeñas no son fácilmente detectadas por el modelo. Como siguiente aproximación, se tratarán de evaluar arquitecturas similares con modelos diferentes, para estudiar si variaciones en estos suponen cambios significativos.

Figura 3.15: Primera visualización de la inferencia del ensemble SAM + YOLOv8 frente al ground truth

Arquitectura ensemble con RetinaNET + SAM

Una de las problemáticas presentes en la arquitectura anterior se basa en la dificultad de la detección de objetos pequeños en la primera capa del ensemble.

Dado que el funcionamiento global del modelo es muy sensible al funcionamiento de esta primera capa, en especial para este conjunto de datos, donde es habitual que la gran mayoría de las instancias de algunas clases sean pequeñas, este error tiene un impacto muy negativo en las métricas de rendimiento.

Esta arquitectura propuesta busca evaluar los resultados con el uso de un modelo cuya función de pérdida empleada para su entrenamiento ha sido *Focal Loss*, que presta mayor atención al desbalanceo y a la predicción de este tipo de clases. Una representación gráfica de la arquitectura de esta aproximación se encuentra en la figura 3.16.

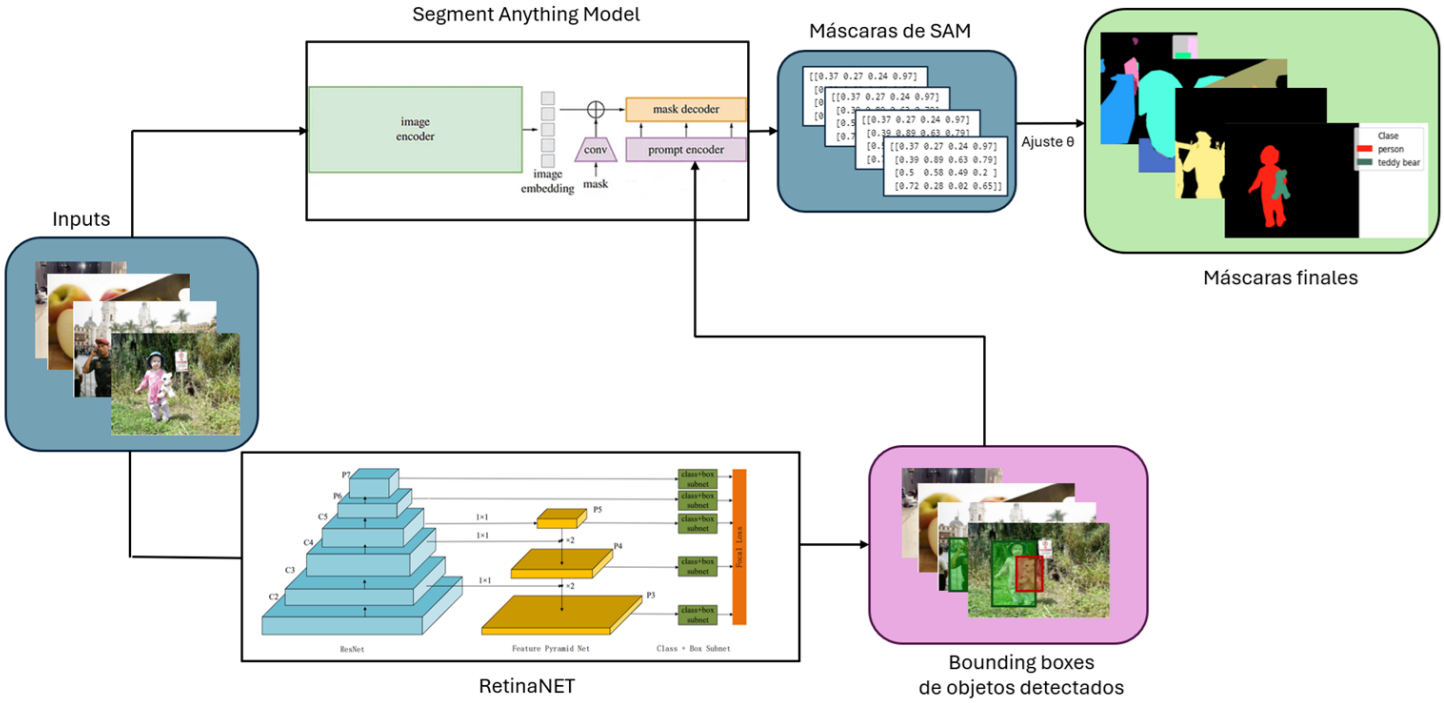

Figura 3.16: Esquematización de la segunda arquitectura ensemble compuesta por los modelos RetinaNet y SAM

En este caso, se hace especial énfasis en el correcto funcionamiento del primer bloque, cuya salida se puede observar en la figura 3.17. Cabe destacar la correcta detección de la gran mayoría de objetos, incluso en ocasiones para instancias difíciles de apreciar inclusive para el ojo humano, aunque en muchos casos con salidas con probabilidades bajas.

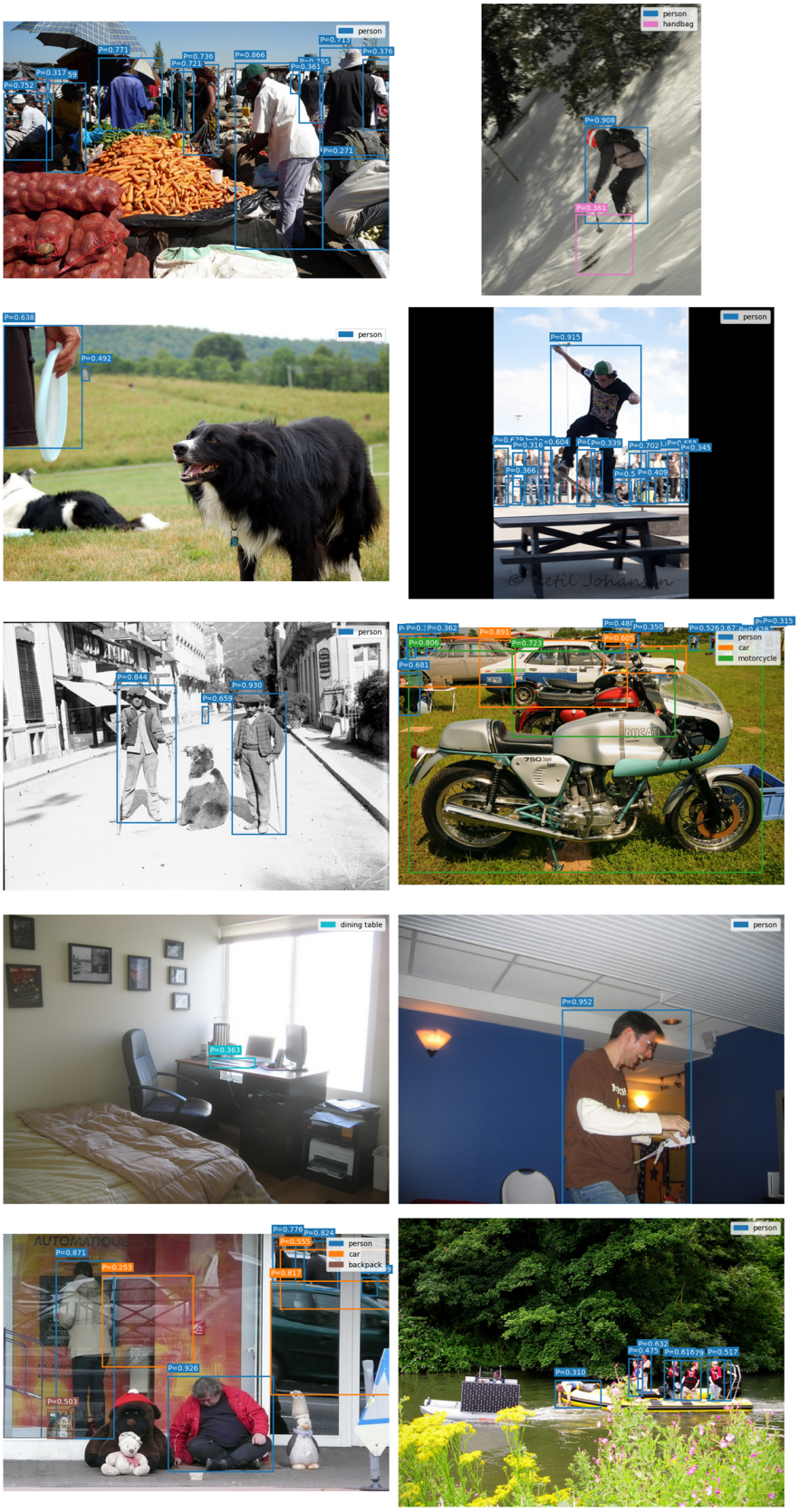

Figura 3.17: Output del primer bloque del modelo RetinaNet, donde se muestra el buen funcionamiento de este modelo para la predicción de instancias difíciles.

En cuanto a los resultados finales del modelo, podemos obtener los resultados de la inferencia sobre las primeras imágenes en la figura 3.18. Se puede apreciar que los resultados en general son muy buenos, teniendo a priori una capacidad de detección de objetos pequeños mayor a la de la arquitectura anterior. Al igual que sucede con el modelo que hace uso de YOLOv8, también se da que las segmentaciones de salida del modelo son mucho más ajustadas a la realidad que las presentes en *ground truth* del dataset.

Figura 3.18: Resultado de la inferencia de la arquitectura compuesta por RetinaNet y SAM sobre las primeras muestras del conjunto de datos

Arquitectura ensemble SAM + CLIP

Como última arquitectura de modelos de dos capas se testea una variante que usa SAM como primer bloque del modelo, en este caso para la producción de todas las máscaras que componen la imagen de forma independiente de su clase. Una vez segmentados todos los objetos de la imagen, se usa un modelo como CLIP para clasificarlas. Esto es posible dado que el modelo es capaz de obtener las similitudes de estas máscaras con cada una de las clases, al tener un espacio de representación compartida multimodal, en este caso entre imágenes y textos.

Una representación gráfica de la pipeline de procesamiento propuesta para esta tercera arquitectura de dos bloques se encuentra en la figura 3.19. En esta, cabe destacar el uso de SAM con la imagen como único input, a diferencia de cómo sucedía con las anteriores arquitecturas.

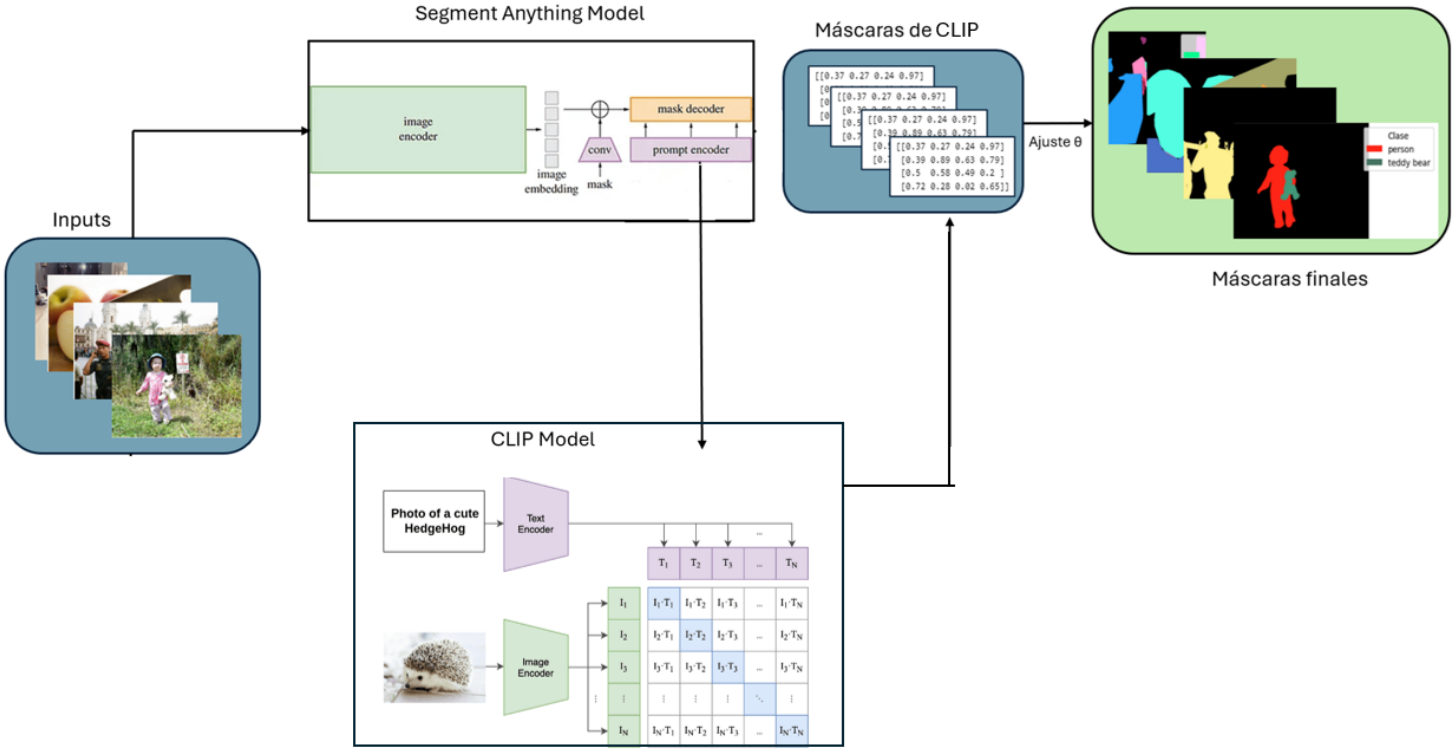

Figura 3.19: Esquematización de la segunda arquitectura ensemble compuesta por los modelos SAM y CLIP

Como input para este segundo modelo, CLIP, se hace necesario el uso de imágenes rectangulares en lugar de máscaras de formas irregulares como las obtenidas por el modelo SAM. Es por ello que, como preprocesamiento para alimentar este segundo modelo, se mantienen los valores en RGB de las segmentaciones obtenidas por SAM, dejando en negro el resto de la imagen y recortando la misma en el cuadrado donde se presenta la máscara.

En cuanto a los inputs textuales a comparar con la imagen preprocesada, estos siguen el formato *.ª photo of a - {clase} + "surrounded by dark"*, siendo clase cada una de las clases a segmentar por el modelo, dado que el espacio de representación compartido del modelo CLIP empleado se entrenó con textos en inglés.

Una primera visualización de los resultados de la salida de CLIP permite identificar claros problemas del modelo, incapaz de clasificar de forma correcta fragmentos de algunas clases como *person*. Un ejemplo de estos errores de clasificación se muestra en la figura 3.20. En esta, cabe destacar la confusión de objetos (como mochilas o prendas de ropa) con la clase persona.

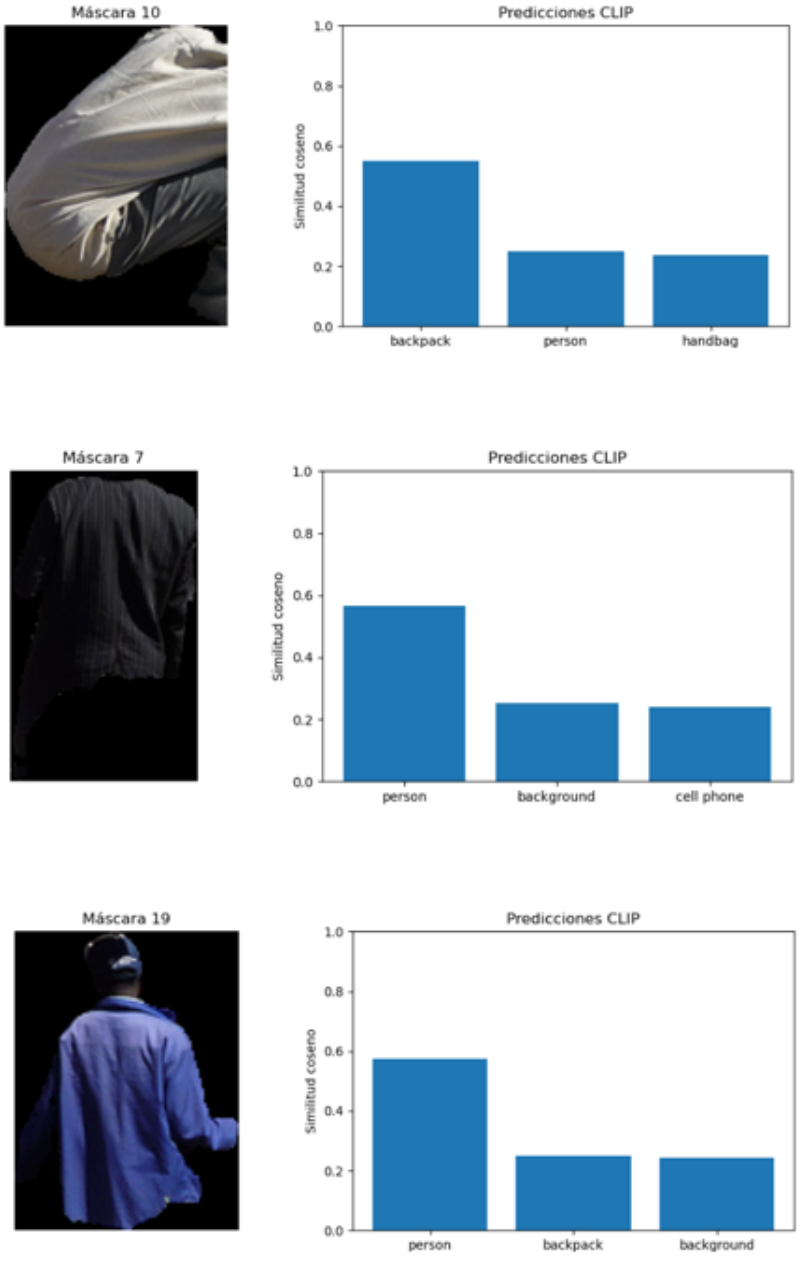

Figura 3.20: Ejemplos de clasificaciones por CLIP de diferentes máscaras generadas por SAM en la primera imagen del dataset.

Como salida final del modelo, cabe destacar que una primera visualización de los resultados evidencia malos resultados. Estos errores son principalmente debido a la segmentación de instancias como personas por parte de SAM en múltiples fragmentos, separando elementos como la ropa de las caras. Estos fragmentos son, posteriormente, mal clasificados por parte del segundo modelo, como se evidenció anteriormente. Un ejemplo de esta salida final se muestra en la figura 3.21.

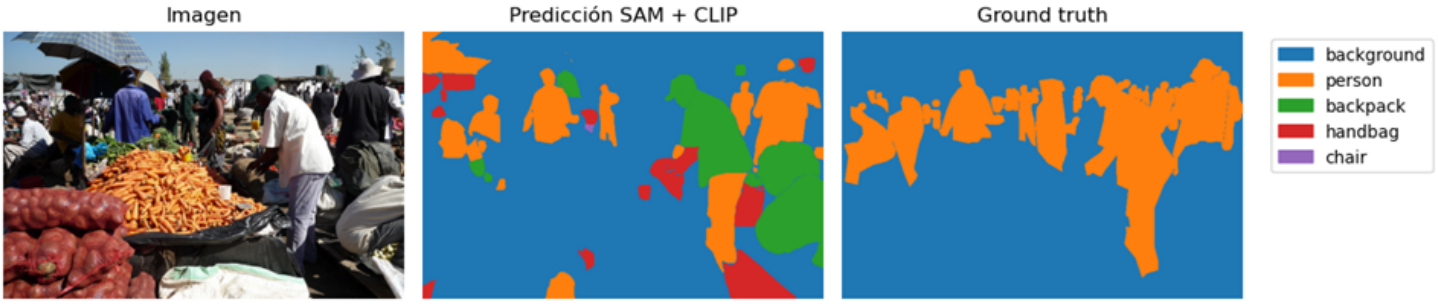

Figura 3.21: Resultado de la inferencia del ensemble SAM+CLIP en una primera muestra del dataset.

Arquitectura ensemble con RetinaNET + SAM + U-Net

Por último, trataremos de mejorar la mejor combinación encontrada hasta la fecha, compuesta de los modelos RetinaNet para la detección y SAM para la segmentación, incluyendo una última etapa a la pipeline que, a partir de la imagen de entrada y las diferentes salidas de probabilidades por píxel de pertenencia a la máscara de segmentación, genere una salida mejorada.

Como modelo para la refinación de la salida y la corrección de errores en el proceso de segmentación se ha decidido hacer uso de otra variante ligera de U-Net. Esta decisión se fundamenta en su rapidez durante la inferencia, lo cual resulta especialmente relevante dado que ya se emplean dos modelos con altos requerimientos computacionales. Además, destaca por la solidez de sus resultados incluso cuando se utilizan conjuntos de datos reducidos. Esto último es importante, ya que la preparación de un conjunto de datos preprocesado para el entrenamiento del modelo *metalearner* exige un tiempo considerable.

Respecto a la arquitectura de esta variante de la U-Net, se replica el número de niveles, filtros, callbacks y configuraciones adicionales aplicadas en el modelo baseline, con el objetivo de entrenar un modelo lo más complejo posible con los recursos presentes. Cabe destacar como diferencia principal que el input hará uso de 4 canales en lugar de 3, lo que hace que el modelo requiera el entrenamiento de un total de más de 460,000 parámetros entrenables.

Un diagrama de esta arquitectura final de tres fases está presente a grandes rasgos en la figura 3.22.

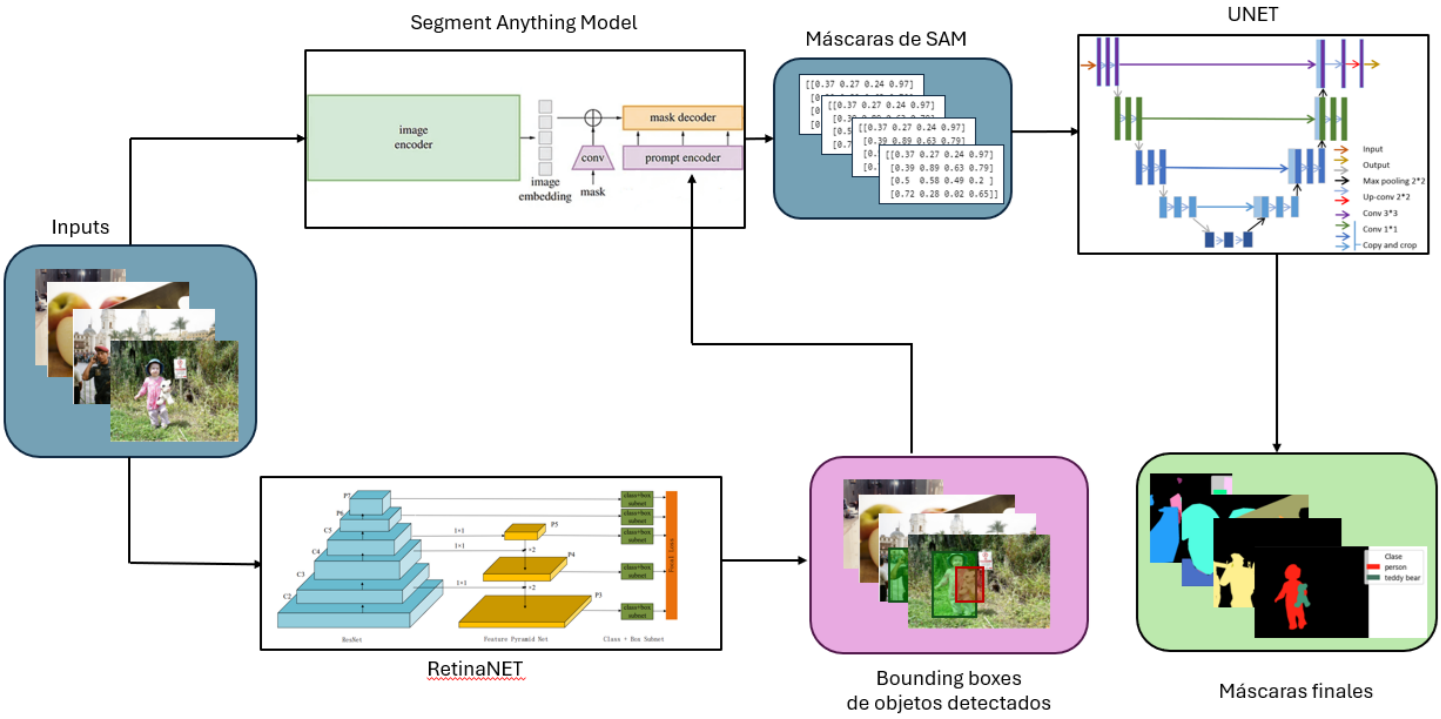

Figura 3.22: Representación de la pipeline final compuesta por los modelos RetinaNET para la detección, SAM para la segmentación de las detecciones y U-Net para el refinamiento final de los resultados

En lo relativo a la función de pérdida, al igual que sucede con el modelo Baseline, se toma la combinación de una función de pérdida combinada que pondera *Categorical Cross Entropy* e *IoU Loss*, teniendo la segunda de ellas un peso de 0,6 frente a 1.

Los resultados de la evolución de la función de pérdida durante las épocas del entrenamiento se muestran en la figura 3.23. En esta figura se destaca la convergencia del modelo al final de su entrenamiento, además de la presencia de un ligero pero aceptable error mayor en el conjunto de validación frente al de train. Esto se debe a factores como el ruido de las segmentaciones de test, que, como se ha visto, no son especialmente de buena calidad, o la influencia de data augmentation.

En cuanto a unos primeros resultados a nivel cualitativo de la inferencia, se muestran los resultados de la misma sobre una imagen aleatoria en la figura 3.24.

A priori destaca la tendencia del modelo a generar máscaras más similares al ground truth, independientemente de que estas se ajusten en mayor o menor medida a la realidad. Asimismo, destaca la mayor capacidad de detección de objetos pequeños, gracias a la función de pérdida utilizada. Al mismo tiempo, destaca la presencia de errores más frecuentes de falsos positivos, resultado del uso parcial de IoU Loss: si predice correctamente la clase *background* la ganancia es mínima, pero si falla, la pérdida es muy significativa.

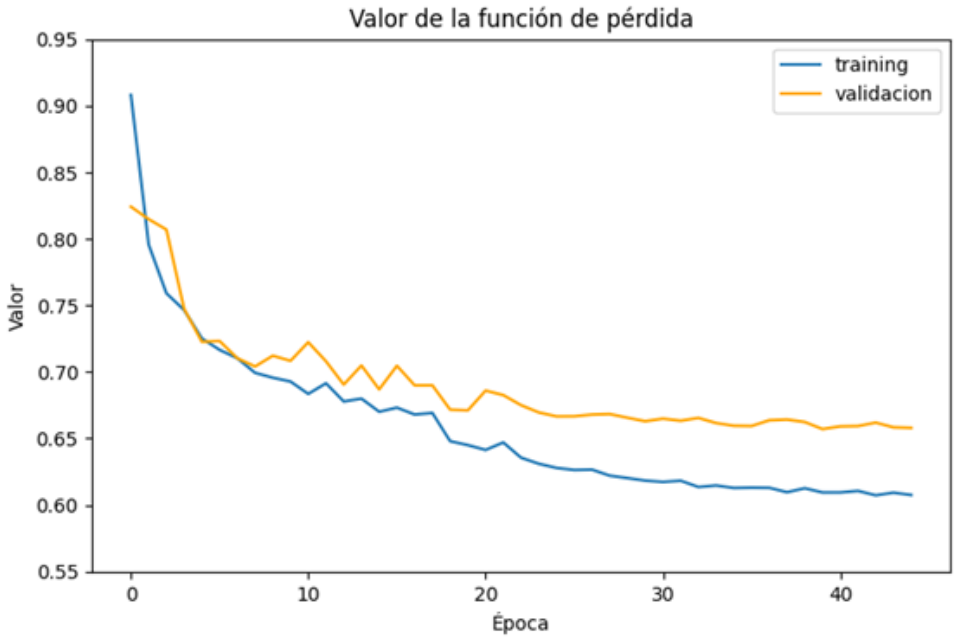

Figura 3.23: Evolución del valor de la función combinada usada como loss para el conjunto de entrenamiento y validación

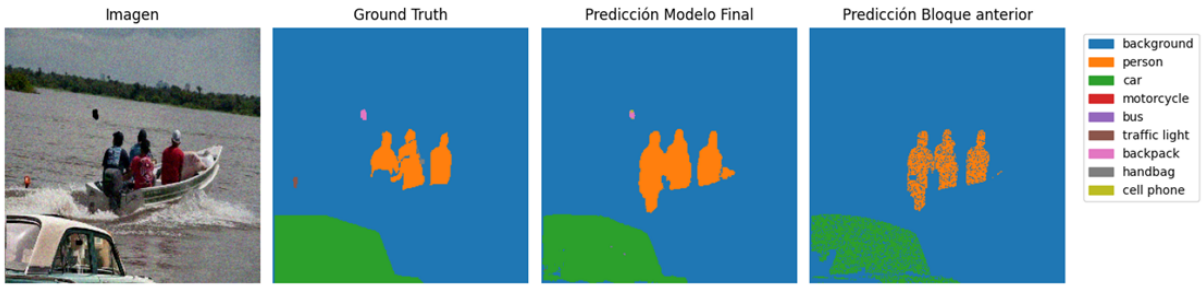

Figura 3.24: Comparativa entre las diferentes fases del modelo final, donde se incluye la imagen inicial, el ground truth, la salida del primer bloque con la aplicación de data augmentation y la salida final del modelo.

3.7. Evaluación de los resultados

La fase de evaluación de los resultados obtenidos por los modelos y la comparativa de los mismos se realizará en dos fases diferentes:

1. Evaluación cuantitativa: durante la que se comparan rendimientos ante diferentes métricas de evaluación, en este caso contemplando la Mean average precision, el IoU y el valor de mAP.

2. Evaluación cualitativa: se evaluará el rendimiento en las imágenes, visualizando tanto máscaras como la salida de cada modelo, realizando una comparativa de estas. Esto es importante, dado que, como se ha visto, el etiquetado del conjunto de datos no es el idóneo.

3.7.1. Evaluación cuantitativa

Para la comparativa de estos modelos se toman tres de las métricas ya estudiadas en el anterior capítulo, en este caso:

- Mean pixel precision: da información de que porcentajes promedios de a imagen se predicen de forma correcta. Por contra, es necesario complementarla con otras métricas al existir desbalanceo de clases y diferencias en los tamaños promedios de píxeles por instancias.
- Intersección sobre la unión media: soluciona el problema de los tamaños diferentes por clase, proporcionando información de las superposiciones entre máscaras y ground truth.
- Mean average precision: óptima para evaluar escenarios multiclase como este donde existen clases más difíciles de predecir, ya sea por su tamaño o por su complejidad. Combina precisión y recall en vez de usar solo una de estas, proporcionando información no solo sobre si el modelo acierta en cada predicción de cada clase, sino también sobre si identifica todos los elementos relevantes.

Para esta evaluación, se realiza la inferencia con todos los modelos sobre un conjunto de 500 imágenes, dado que un mayor número sería inabordable por tiempos de computación. Los resultados se muestran mediante un plot de seaborn en la figura 3.25, indicando los valores promedio de cada métrica por cada tipo de modelo.

En lo relativo al tiempo de inferencia, destaca que el modelo U-Net baseline tiene un tiempo de inferencia mínimo, mientras que los modelos que usan SAM presentan una velocidad considerablemente menor. En particular, destaca negativamente el conjunto SAM+CLIP, cuyo tiempo de inferencia es más de diez veces superior al de los demás modelos, alcanzando más de 200 segundos por imagen. Esto es debido al uso de SAM para la segmentación de todas las máscaras de la imagen en lugar de un subconjunto de las mismas, como sucede en el resto de los modelos. En cuanto al modelo final, el tiempo de inferencia no es significativamente mayor que su arquitectura sin la U-Net correctora o a la arquitectura que combina YOLO y SAM, cercano a los 20 segundos de media.

En cuanto a *pixel accuracy*, destacan inicialmente los malos resultados de SAM+CLIP y, de manera secundaria, los del modelo baseline. Por otro lado, los modelos que usan un detector seguido de SAM como segmentador logran resultados más estables. Esto es debido a la función de pérdida usada en el modelo corrector, que aporta mayor importancia a la detección de clases minoritarias, haciendo que las clases que influyen en mayor medida, en este caso principalmente *background*, no dominen la optimización y mantengan la misma importancia relativa que el resto.

Por último, cabe destacar la mejora del modelo final en métricas que mitigan el sesgo hacia clases mayoritarias, como lo son IoU y mAP. En este caso, el modelo final sí que mejora de manera consistente al resto de modelos, mostrando una mayor robustez en la segmentación equilibrada de clases y una mejor capacidad de generalización. Esta mejoría puede atribuirse principalmente a varios factores, el refinamiento de las siluetas, que reduce los falsos positivos de acuerdo con el ground truth del conjunto de datos, y la suma de importancia en el entrenamiento a clases con menor representación, que mejora

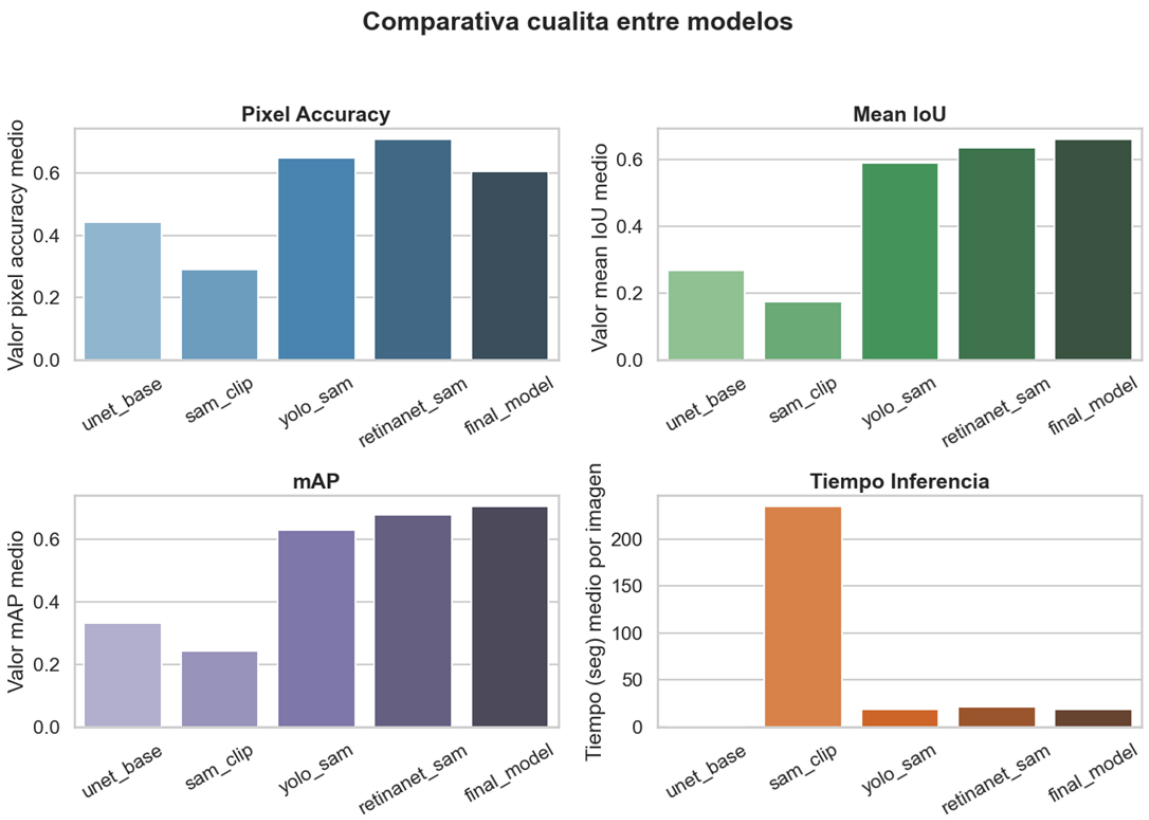

Figura 3.25: Comparativa entre las diferentes métricas a estudiar para cada uno de los modelos estudiados

la segmentación de clases menos frecuentes.

3.7.2. Evaluación cualitativa

Para la evaluación cualitativa de cada uno de los modelos, se aplica la inferencia en un subconjunto común de las imágenes del conjunto de entrenamiento y se comparan de forma visual.

En primera instancia, la inferencia sobre la primera imagen del dataset da lugar a los resultados de la figura 3.26

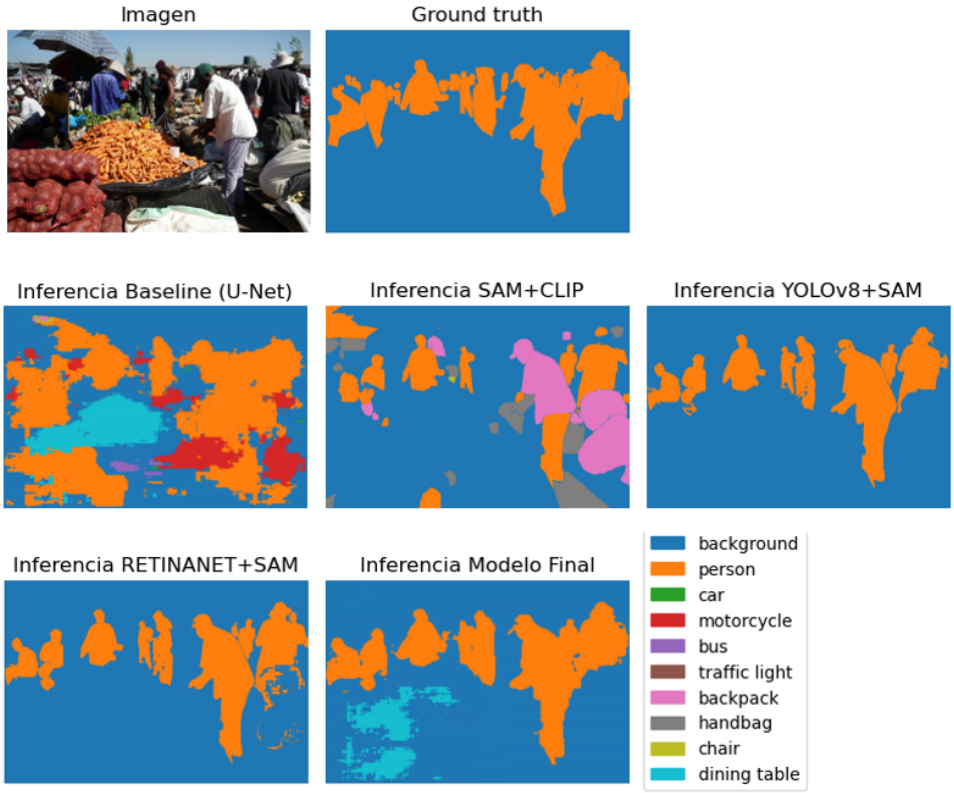

Figura 3.26: Representación de la inferencia sobre una primera imagen empleando cada uno de los modelos a evaluar de forma cualitativa

En cuanto a la comparativa visual con el *ground truth* del conjunto de datos, en líneas generales, se puede observar que los resultados obtenidos por los modelos de dos bloques son menos similares al ground truth del dataset de COCO.

En cambio, por otro lado se tienen que los resultados de modelos como SAM+YOLO son más fieles a la realidad, debido a un etiquetado impreciso por parte del dataset, lo que hace ajustar el entrenamiento del modelo final a formas más abruptas propias de este último.

Una mayor cantidad de imágenes para explorar los resultados de diferentes modelos se han incluido en la web, explicada en profundidad a continuación, permitiendo comparar de manera interactiva los resultados cualitativos de cada modelo.

3.8. Productivización

Para la productivización, no solo del modelo final sino de la totalidad de los modelos entrenados, se ha decidido llevar a cabo la implementación de un servicio web. En este caso, se ha decidido hacer uso del framework de Python *Dash* [1], para la obtención de resultados lo más profesionales posibles.

En primera instancia, se prueba a realizar el despliegue en plataformas gratuitas como Plotly Cloud. En cambio, estas plataformas tienen limitaciones en el tamaño de las

aplicaciones a hostear, que no han de superar los 80MBs. Dado que el tamaño de los modelos y el aplicativo son de tamaño menor, se almacena el código en un repositorio y se incluyen las instrucciones para su despliegue en local. El código donde el servicio web queda almacenado puede encontrarse en el repositorio de GitHub público del siguiente enlace: `https://github.com/rucasca/deployment_stacking_ensemble_segmentation_model`

Para su uso, se han de seguir los siguientes pasos:

1. Generación de un entorno virtual de `Poetry` [60] para poder hacer uso de las librerías y sus respectivas versiones almacenadas en el fichero `pyproject.toml`. Para ello, primero se descarga Poetry mediante *pip install poetry* y luego se ejecuta *poetry install*, que descargará las librerías correspondientes. Una vez hecho esto, se selecciona el entorno creado como entorno de ejecución mediante poetry shell.

2. Descarga de los modelos a emplear que requieran descargas manuales en las siguientes direcciones:

   - SAM: https://github.com/facebookresearch/segment-anything
   - U-Net baseline: `https://github.com/rucasca/TFM/tree/master/models/trained_models`
   - U-Net final: `https://github.com/rucasca/TFM/tree/master/models/trained_models`

3. Edición de las variables del entorno del fichero *.env* con las direcciones del sistema donde se han descargado los archivos.

4. Ejecución del servicio web mediante el comando *python app.py*

5. Acceso al navegador en la dirección.

En este sistema web, se incluyen funcionalidades que permiten hacer uso de todos los modelos vistos en el capítulo, desde los más simples como la U-Net baseline hasta el modelo de final de tres capas conformado por SAM, RetinaNet y U-Net. Asimismo, se ha considerado relevante la inclusión de la opción de realizar segmentación en ensembles que utilicen SAM mediante un modelo más ligero, en este caso fast-SAM, mediante un input en la sección de inferencia. Esto se debe a que ejecutar la aplicación web junto con la inferencia del modelo en un ordenador con solo 8 GB de RAM provocaba que se agotara la memoria, deteniendo el framework y sus respectivos procesos. Una imagen de la vista que permite el cálculo de esta inferencia se muestra en la figura 3.27

Esta permite el cálculo de las segmentaciones en cualquier imagen en formato *png* o *jpg*, aplicando redimensionamientos en aquellos modelos donde es necesario por la necesidad de una entrada de tamaño fijo, y permitiendo así la evaluación de los modelos en imágenes fuera del dataset COCO. Esto permite estudiar los modelos sin sesgos y obtener resultados lo más fieles a la realidad, al poder evaluarse con cualquier tipo de imagen.

Figura 3.27: Visualización de la vista de realización de inferencia en el servicio web y los diferentes inputs contemplados

Además, se implementan dos vistas adicionales, una donde se almacenan los resultados históricos de inferencia en cada imagen y otra donde se muestran los resultados de los modelos en un subconjunto de las imágenes del dataset, pudiendo visualizarse esta última en la figura 3.28.

Figura 3.28: Visualización de la vista que contiene ejemplos de la inferencia de los modelos sobre imágenes del dataset COCO

Un resultado de la aplicación de la inferencia mediante la web sobre una imagen aleatoria se muestra en la figura 3.29.

Figura 3.29: Visualización de la inferencia aplicada a una imagen descargada de forma aleatoria en el aplicativo web implementado.

Capítulo 4

Conclusiones

Como conclusión primaria de este Trabajo de Fin de Máster se puede determinar que la automatización de la segmentación semántica de imágenes es más que factible mediante el uso de modelos de Machine Learning con el estado del arte de los modelos actuales.

Es importante destacar los notables resultados obtenidos por modelos fundacionales potentes como SAM, que alcanza generalización *out-of-sample*. Es remarcable el uso de arquitecturas en formato ensemble más elaboradas, donde se combinan modelos de detección como YOLO y de segmentación como Segment Anything Model, para la obtención eficiente de máscaras y clasificaciones píxel a píxel.

Asimismo, cabe destacar la posibilidad de mejoras añadiendo capas correctoras, como la implementada con la arquitectura U-Net, que añade mayor complejidad al modelo y mejora los resultados. Del mismo modo, cabe subrayar que el hecho de incluir esta última capa, al estar entrenada con un conjunto de datos limitado, podría influir negativamente ante las capacidades *out-of-sample* del modelo, y se sobreajustaría a la distribución generadora de la forma de anotar. Esto último implica que, si las etiquetas del dataset son de mala calidad, las salidas del modelo también lo serán.

Otra conclusión importante del Trabajo de Fin de Máster es la gran dificultad para el entrenamiento de modelos para problemas complejos como el de segmentación semántica. Esto se debe a la necesidad de arquitecturas complejas como U-Net para abordar el problema de forma efectiva. Este tipo de arquitecturas, con un gran número de pesos a entrenar, requieren conjuntos de datos amplios, tiempos de entrenamiento y capacidades de cómputo elevadas, lo que ha supuesto un obstáculo dadas las limitaciones de recursos.

También cabe destacar la gran importancia en la selección del conjunto de datos, tanto para la evaluación del funcionamiento de modelos fundacionales como para el entrenamiento de nuevos modelos o bloques del ensemble. En este caso, se puede considerar que la selección de COCO como conjunto de datos ha sido muy positiva dado su gran volumen de imágenes y la diversidad de su procedencia. De la misma forma, es necesario realizar un proceso de balanceo de clases en los datos, complejo para problemas como el de segmentación en imágenes, que permita obtener resultados buenos y no sesgados a las clases mayoritarias.

Es muy importante destacar el auge de modelos fundacionales con generalización *out-of-sample*, que supondrán desde ahora en adelante soluciones más que factibles para tareas tanto de detección como de segmentación. El uso de estas arquitecturas es adecuado para este tipo de problemas, ya que permite obtener resultados oportunos sin la necesidad de un entrenamiento costoso computacionalmente. Esto sucede en escenarios donde la recopilación y el etiquetado son procesos complejos, como sucede con las imágenes etiquetadas píxel a píxel, permitiendo el ahorro de costes temporales y económicos para la resolución de este problema. Esto ha dado lugar a un cambio en el paradigma con una tendencia hacia modelos generalistas que se adapten a escenarios y tareas sin necesidad de ajustes.

En líneas generales, este trabajo me ha permitido abordar una gran cantidad de líneas de trabajo diferentes dentro del mundo de la ciencia de datos, aprendiendo conceptos que resultaban novedosos, como la segmentación semántica. Este trabajo me ha concedido la posibilidad de enfrentarme a un problema aplicable en el mundo real, mediante el uso de modelos propios, fundacionales y la combinación de estas arquitecturas, y dando lugar a soluciones complejas que permiten abordar el problema de segmentación de forma exitosa.

Capítulo 5

Trabajo futuro

Como trabajo a futuro, caben destacar las dos líneas que principalmente han supuesto problemáticas en la realización del presente Trabajo de Fin de Máster.

En primera instancia, una de las grandes problemáticas encontradas ha sido el bajo rendimiento de la arquitectura que combinaba SAM para la obtención de todas las segmentaciones con CLIP, para la clasificación de las mismas.

El problema de esta solución se debe a la gran cantidad de segmentaciones generadas por SAM en su versión *segment-everything*, que da como salida un número muy elevado de máscaras de tamaño mediano y pequeño. Esto implica que, ante elementos como personas, queden segmentados en manos, ropa, entre otros. Estas segmentaciones, al ser clasificadas mediante CLIP, dan lugar a clasificaciones erróneas, al ser la probabilidad de similitud asociada por CLIP de la imagen de ropa frente al input textual de persona baja.

Para solventar este problema, puede hacerse uso de otros modelos out-of-distribution como Grounding Dino, presentado por Liu et al. en 2024 [47], permitiendo así la detección de elementos sin limitarse a un conjunto de etiquetas, a diferencia de modelos como YOLOv8 o RetinaNet. Este ya muestra buenos resultados en la actualidad al combinarse con modelos de segmentación como SAM [64]

La segunda línea de trabajo futuro se trata de la mejora en la calidad de las segmentaciones presentes en el conjunto de datos empleado, en este caso MS COCO.

A lo largo del entrenamiento y evaluación de los modelos, se han detectado instancias dentro del conjunto de datos donde las segmentaciones no se corresponden con la realidad, al no ajustarse las formas al contenido de las imágenes. Esto tiene como consecuencia sesgos en el entrenamiento de los modelos, que replicarán la generación de estas segmentaciones erróneas.

Además, el etiquetado propuesto como ground truth no es homogéneo en clases como *dining table*, donde en determinadas imágenes queda clasificado como tal, mientras que en otras no.

Un ejemplo visual de estas problemáticas propias de COCO puede encontrarse en la figura 5.1.

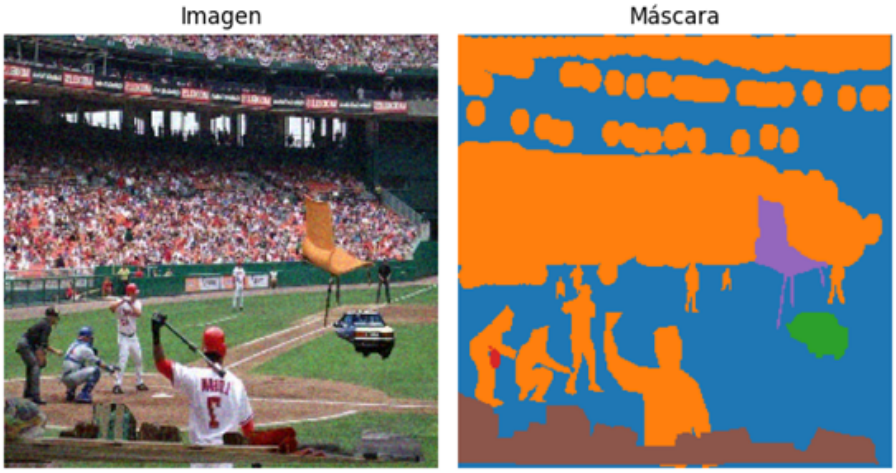

Figura 5.1: Ejemplos de una imagen del dataset MS COCO con segmentaciones imprecisas

Por ello, en el futuro, de cara a abordar este problema, se podrían considerar aproximaciones mediante un conjunto de datos más moderno, obtenido de forma autosupervisada y que presente segmentaciones más eficaces de las imágenes, para así poder obtener un mejor modelo final.

Un posible dataset podría tratarse de alguna variante anotada derivada de *SA-1B* [38], el dataset empleado para el entrenamiento de SAM, que si bien no contiene las etiquetas de cada segmentación, se trata del dataset más masivo hasta la fecha y que permite una mayor generalización.

Referencia bibliográfica

[1] Dash python user guide. `https://dash.plotly.com/`, n.d. "Dash is the original low-code framework for rapidly building data apps in Python.".

[2] Sidra Aleem, Fangyijie Wang, Mayug Maniparambil, Eric Arazo, Julia Dietlmeier, Guenole Silvestre, Kathleen Curran, Noel E. O'Connor, and Suzanne Little. Test-time adaptation with salip: A cascade of sam and clip for zero shot medical image segmentation, 2024.

[3] Yassine Alouini. All segmentation metrics! `https://yassinealouini.medium.com/all-segmentation-metrics-be65e0653529`, April 2021. Accessed: 2025-01-14.

[4] Reza Azad, Moein Heidari, Kadir Yilmaz, Michael Hüttemann, Sanaz Karimijafarbigloo, Yuli Wu, Anke Schmeink, and Dorit Merhof. Loss functions in the era of semantic segmentation: A survey and outlook. *arXiv preprint arXiv:2312.05391*, 2023. License: CC BY 4.0.

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[6] Rupert R. A. Bourne, Seth R. Flaxman, Tasanee Braithwaite, Maria V. Cicinelli, Aditi Das, Jost B. Jonas, Jill Keeffe, John H. Kempen, Janet Leasher, Hans Limburg, Kovin Naidoo, Konrad Pesudovs, Serge Resnikoff, Alex Silvester, Gretchen A. Stevens, Nina Tahhan, Tien Y. Wong, and Hugh R. Taylor. Magnitude, temporal trends, and projections of the global prevalence of blindness and distance and near vision impairment: a systematic review and meta-analysis. *The Lancet Global Health*, 5(9):e888–e897, 2017.

[7] Yuri Boykov, Olga Veksler, and Ramin Zabih. Markov random fields with efficient approximations. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001. Computer Science Department, Cornell University, Ithaca, NY 14853.

[8] Zhenguan Cao, Haixia Yang, Liao Fang, Zhuoqin Li, Jinbiao Li, and Gaohui Dong. Research on improved yolov7-sswd digital meter reading recognition algorithms. *Review of Scientific Instruments*, 95, 09 2024.

[9] Jeanina Casusi. What is a foundation model? an explainer for non-experts, May 2023. Accessed: 2024-12-31.

[10] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, and Thomas Reinartz. Crisp-dm 1.0: Step-by-step data mining guide. Technical report, NCR, SPSS, DaimlerChrysler, 2000. Available at: `https://www.the-modeling-agency.com/crisp-dm.pdf`.

[11] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. *CRISP-DM 1.0: Step-by-step Data Mining Guide*. SPSS Inc., 2000. CRISP-DM Consortium.

[12] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[13] Bowen Cheng, Ross B. Girshick, Piotr Dollár, Alexander C. Berg, and Alexander Kirillov. Boundary iou: Improving object-centric image segmentation evaluation. *CoRR*, abs/2103.16562, 2021.

[14] Dan Ciresan, Alessandro Giusti, Luca Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[15] Computer Vision Wiki. Intersection over union (iou), n.d. Accessed: 2025-06-29.

[16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[17] Lavina Jean Crasta, Rupal Neema, and Alwyn Roshan Pais. A novel deep learning architecture for lung cancer detection and diagnosis from computed tomography image analysis. *Healthcare Analytics*, 5:100316, 2024.

[18] Data Science (WiDS) La Paz. ¿estás empezando en ciencia de datos? acá te damos un pequeño resumen de la metodología crisp-dm para que inicies en este apasionante mundo., 2025. Accessed: 2025-01-07.

[19] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[20] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2018. FMILA, Université de Montréal; AIRLab, Politecnico di Milano.

[21] Encord. Meta ai's segment anything model (sam) explained: The ultimate guide. `https://encord.com/blog/segment-anything-model-explained/`, December 2024. Accessed: 2025-01-27.

[22] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge 2012 dataset, 2012. Dataset available at `http://host.robots.ox.ac.uk/pascal/VOC/`.

[23] Mark Everingham, Luc Van Gool, Christopher Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 06 2010.

[24] Mark Everingham, Luc Van Gool, Christopher K I Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. In *International Journal of Computer Vision (IJCV)*, volume 88, pages 303–338, 2010.

[25] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[26] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation, 2017.

[27] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[28] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson Prentice Hall, 3rd edition, 2008.

[29] Maja Gornik. Bridging the gap between text and images in computer vision with clip. `https://www.pareto.si/blog/computer-vision-with-clip/`, 2024. Accessed: 2025-02-01.

[30] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.

[31] Michael Guerzhoy, Steve Marschner, Alexei Efros, and Noah Snavely. Csc320: Introduction to visual computing - upsampling and interpolation, 2023. Accessed: 2025-01-19.

[32] Yuhang Guo, Yu Liu, Theodoros Georgiou, and Michael S. Lew. A review of semantic segmentation using deep neural networks. *International Journal of Multimedia Information Retrieval*, 7(2):87–93, June 2018.

[33] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[34] IBM. What are convolutional neural networks?, 2025. Accessed: 2025-02-17.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[36] Paul Jaccard. Etude de la distribution florale dans une portion des alpes et du jura. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 37:547–579, 01 1901.

[37] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[38] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.

[39] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[40] Bo Li, Yong Zhang, Yunhan Ren, Chengyang Zhang, and Baocai Yin. Lite-unet: A lightweight and efficient network for cell localization. *Engineering Applications of Artificial Intelligence*, 129:107634, 2024.

[41] Hanchao Li, Pengfei Xiong, Jie An, and Lingxue Wang. Pyramid attention network for semantic segmentation. *CoRR*, abs/1805.10180, 2018.

[42] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[43] Tsung-Yi Lin et al. Coco api - common objects in context, 2015.

[44] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[45] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[46] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[47] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding dino: Marrying dino with grounded pre-training for open-set object detection, 2024.

[48] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[49] Abd Mamat, Fatma Mohamad, Norkhairani abdul rawi, Mohd Khalid Awang, Mohd Isa Awang, and Mohd Fadzil Abdul Kadir. Region based image retrieval based on texture features. 92:9–19, 10 2016.

[50] Meta AI. Segment anything dataset (sa-1b). `https://ai.meta.com/datasets/segment-anything/`, 2023.

[51] Domor Mienye and Yanxia Sun. A survey of ensemble learning: Concepts, algorithms, applications, and prospects. *IEEE Access*, PP:1–1, 09 2022.

[52] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *Proceedings of the 2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571. IEEE, 2016.

[53] Hossam Moftah, Ahmad Azar, Eiman Al-Shammari, Neveen Ghali, Aboul Ella Hassanien, and Mahmoud Shoman. Adaptive k-means clustering algorithm for mr breast image segmentation. *Neural Computing and Applications*, 24, 06 2013.

[54] Loris Nanni, Alessandra Lumini, Andrea Loreggia, Alberto Formaggio, and Daniela Cuza. An empirical study on ensemble of segmentation approaches. *Signals*, 3(2):341–358, 2022.

[55] Neville. Dice loss in medical image segmentation, July 2023. Accessed: 2025-06-29.

[56] Vision Loss Expert Group of the Global Burden of Disease Study, GBD 2019 Blindness, and Vision Impairment Collaborators. Global estimates on the number of people blind or visually impaired by cataract: a meta-analysis from 2000 to 2020. *Eye (London, England)*, 38(11):2156–2172, 2024.

[57] Ozan Oktay, Jo Schlemper, Loïc Le Folgoc, Matthew C. H. Lee, Mattias P. Heinrich, Kazunari Misawa, Kensaku Mori, Steven G. McDonagh, Nils Y. Hammerla, Bernhard Kainz, Ben Glocker, and Daniel Rueckert. Attention u-net: Learning where to look for the pancreas. *CoRR*, abs/1804.03999, 2018.

[58] Alan V. Oppenheim, Alan S. Willsky, and with S. Hamid Nawab. *Signals and Systems*. Prentice Hall, Englewood Cliffs, NJ, 1st edition, 1983.

[59] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[60] Poetry Contributors. Poetry documentation, 2025. Accessed: 2025-08-31.

[61] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021.

[62] Sebastian Raschka. Understanding and coding self-attention, multi-head attention, cross-attention, and causal-attention in llms, 2024. Imagen.

[63] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.

[64] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, Zhaoyang Zeng, Hao Zhang, Feng Li, Jie Yang, Hongyang Li, Qing Jiang, and Lei Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024.

[65] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[66] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[67] Bo-Kai Ruan, Hong-Han Shuai, and Wen-Huang Cheng. Vision transformers: State of the art and research challenges, 2022.

[68] Hyungseob Shin, Jeongryong Lee, Taejoon Eo, Yohan Jun, Sewon Kim, and Dosik Hwang. The latest trends in attention mechanisms and their application in medical imaging. *Journal of the Korean Society of Radiology*, 81:1305, 11 2020.

[69] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[70] Brijesh Soni. Stacking to improve model performance: A comprehensive guide on ensemble learning in python. *Medium*, May 2023. Accessed: 2024-12-28.

[71] Abis Hussain Syed. Hands on guide to lenet-5 (the complete info). *Medium*, March 2023. Accessed: 2025-01-19.

[72] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[73] OpenCV Team. opencv-python. `https://pypi.org/project/opencv-python/`, July 2025. Version 4.12.0.88. Python wrapper for OpenCV.

[74] TensorFlow. Tfrecord y tf.train.example. `https://tensorflow.org/tutorials/load_data/tfrecord?hl=es-419`, n.d. TensorFlow Core Tutoriales: Carga de datos.

[75] Jane Torres. What is yolov8? exploring its cutting-edge features, 2024. Accessed: 2025-04-05.

[76] Sik-Ho Tsang. Review — cspnet: A new backbone that can enhance learning capability of cnn. *Medium*, 2021. Accessed: 2025-04-05.

[77] Ultralytics. Explore ultralytics yolov8: A computer vision model architecture for detection, classification, segmentation, and more. `https://yolov8.com/`, 2025. Accessed: 2025-04-05.

[78] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[79] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[80] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. CSPNet: A New Backbone That Can Enhance Learning Capability of CNN. November 2019. A Preprint.

[81] Luke Wood and Francois Chollet. Efficient graph-friendly coco metric computation for train-time model evaluation, 2022.

[82] Yutong Xie, Bing Yang, Qingbiao Guan, Jianpeng Zhang, Qi Wu, and Yong Xia. Attention mechanisms in medical image segmentation: A survey. 2023.

[83] Xiaobo Yang and Xiaojin Gong. Foundation model assisted weakly supervised semantic segmentation, 2023.

[84] Muhamad Yani, S Irawan, and Casi Setianingsih. Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics: Conference Series*, 1201:012052, 05 2019.

[85] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything, 2023.

[86] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything, 2023.

[87] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ADE20K dataset. *CoRR*, abs/1608.05442, 2016.