Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

HPPrintingSegmentation: Extrae e imprime desde tu imaginación

Dª. Suárez Sáez, Nerea del Carmen
TUTORA: Dª. Remeseiro López, Beatriz

FECHA: Febrero 2025

AGRADECIMIENTOS

Primero de todo, quiero agradecerles a mis padres y a mi hermano, Oscar, Carmen y Rubén, por creer en mí, apoyarme y brindarme siempre los medios necesarios para poder estudiar y poder acceder a la universidad. En segundo lugar, a Mar, por no dejar que me rinda, escuchar demasiadas veces mis preocupaciones con la carrera y darme fuerzas siempre. En tercer lugar, a mi tutora académica Beatriz por el inmenso soporte dado durante todo un año, realizando revisiones periódicas cada dos semanas. Y, por último, a mis amigos y compañeros de carrera por hacer que la vida universitaria sea una tragicomedia divertida.

Índice

Figuras

Tablas

1. Hipótesis de partida y alcance

1.1. Introducción a la IA

La inteligencia artificial (IA) es un campo de la informática que se enfoca en la capacidad de las máquinas para realizar tareas simulando aspectos de la inteligencia humana tales como el aprendizaje, el razonamiento, la percepción, la solución de problemas, la interacción lingüística y hasta la creatividad [1].

Funciona mediante algoritmos y modelos matemáticos que procesan grandes cantidades de datos, permitiendo a las máquinas tomar decisiones basadas en patrones y reglas establecidas. Una de las principales ramas de la IA es el aprendizaje automático, que permite a las máquinas aprender de forma autónoma a partir de datos sin ser programadas específicamente para hacerlo. Este tipo de aprendizaje posibilita que la IA mejore su precisión y eficiencia con el tiempo, sin requerir programación específica para cada tarea [1].

Dentro del aprendizaje automático, existen diferentes enfoques, entre los que destacan el aprendizaje supervisado y el no supervisado.

- **Aprendizaje supervisado:** Se basa en entrenar un modelo en un conjunto de datos etiquetados, donde los datos de entrada se emparejan con la salida correcta. El modelo aprende a asignar etiquetas a las entradas, minimizando el error en sus predicciones [2].
- **Aprendizaje no supervisado:** Trabaja con datos no etiquetados, donde el objetivo es encontrar patrones en la información proporcionada [3].

Más recientemente, ha surgido el **aprendizaje profundo**, una rama del aprendizaje automático que utiliza redes neuronales multicapa, conocidas como redes neuronales profundas, para simular el complejo poder de toma de decisiones del cerebro humano [4].

En nuestro día a día usamos la IA en multitud de aplicaciones y servicios, algunas veces sin siquiera ser conscientes de ello (ver *Figura 1)*. Por ejemplo, Alexa, Google Assistant o Cortana en el caso de asistentes virtuales, o los programas de traducción de idiomas que dependen de la IA para mejorar continuamente sus resultados [1].

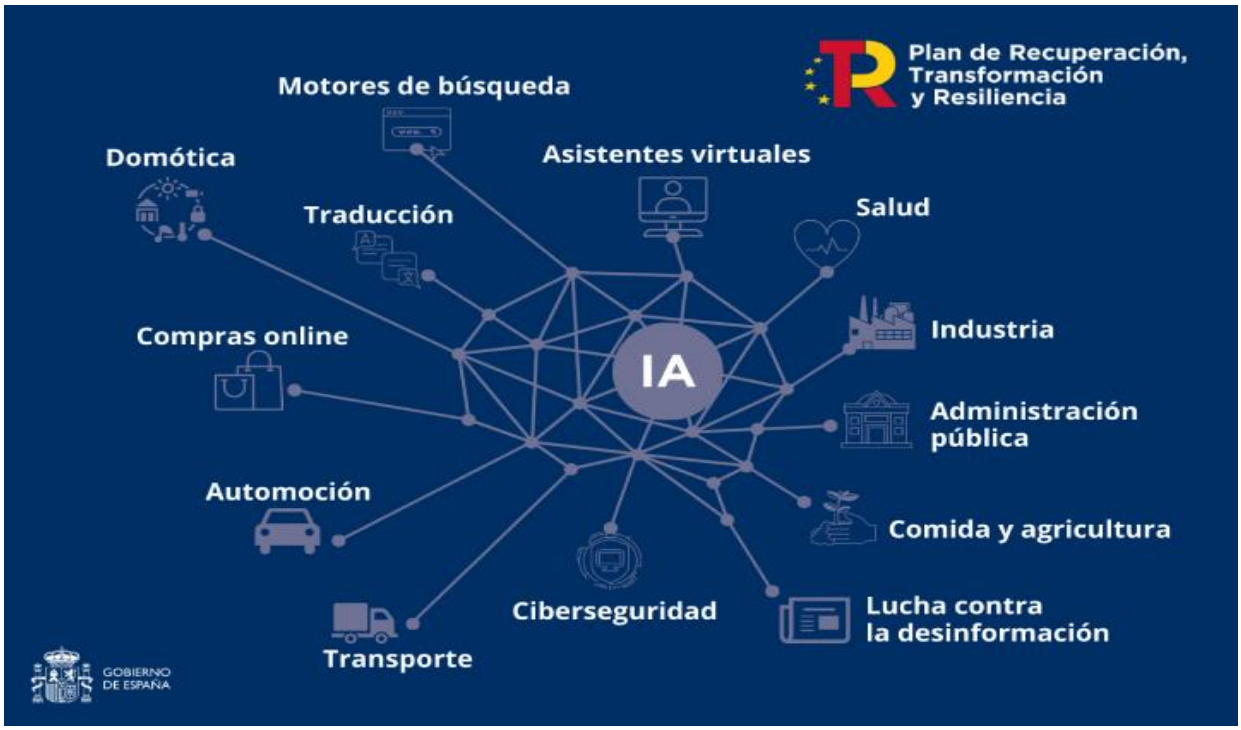

Figura 1: Aplicaciones de la IA **[1]**

Según su potencia, existen distintos tipos de IA que influyen en las principales aplicaciones de la IA (ver *Figura 2)*: la IA débil (*narrow*), la IA fuerte (*general*) y IA superinteligente (*super*) [1].

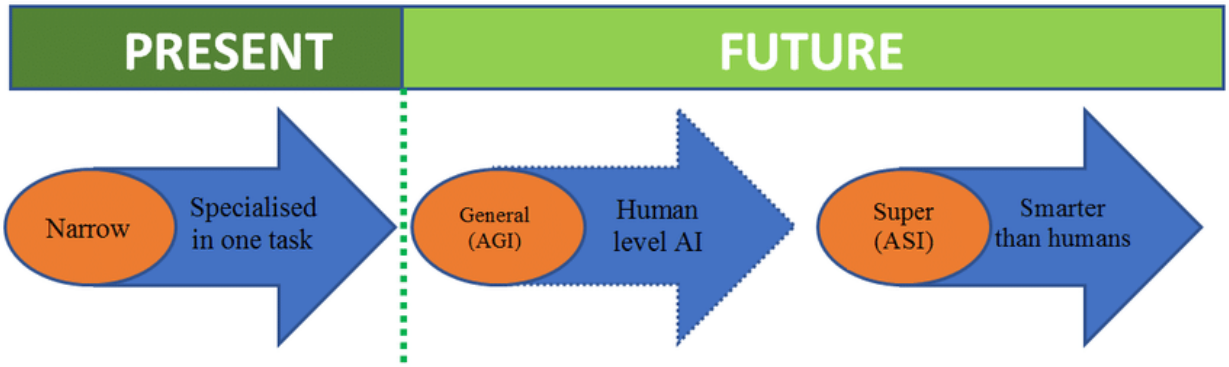

Figura 2: Progresión de los tipos de IA **[5]**

La IA débil son sistemas diseñados para realizar tareas específicas y limitadas, como el reconocimiento de voz, la identificación de imágenes o la traducción de idiomas. No tienen capacidad de aprendizaje o adaptación por sí mismos, y requieren ser programados para realizar una tarea determinada. Su alcance es limitado y no pueden realizar tareas fuera de su campo de especialización [1].

La IA fuerte está diseñada para tener una amplia gama de habilidades cognitivas y capacidad de aprendizaje autónomo. Estos sistemas pueden realizar múltiples tareas y aprenden de forma autónoma a medida que interactúan con el entorno. La IA fuerte debe tener la capacidad de razonar, planificar y tomar decisiones complejas en un amplio espectro de situaciones. Sin embargo, es un objetivo que aún no se ha alcanzado, dado que los sistemas actuales se limitan a la IA débil [1].

La IA superinteligente es un tipo de IA que superaría la inteligencia humana en todos los aspectos. Este nivel de IA sería capaz de comprender el mundo de una manera que está más allá de la capacidad humana, y de resolver problemas complejos a una velocidad y eficiencia que los seres humanos no pueden alcanzar. Por tanto, es más una hipótesis teórica que un desarrollo práctico actual [1].

1.1.1. Segmentación de imágenes

Uno de los campos específicos en los que se aplica la IA débil es la **segmentación de imágenes**, una técnica clave en la visión por computador, que es un campo de la IA

que permite a los sistemas interpretar imágenes y videos de manera automática mediante una serie de procesos fundamentales. Estos incluyen [6]:

- **El preprocesamiento de imágenes**, que mejora la calidad de los datos visuales.
- **La extracción de características**, que identifica patrones clave.
- **La predicción y reconocimiento**, que clasifica y detecta objetos o regiones de interés.

En los enfoques tradicionales, estos pasos debían diseñarse manualmente, lo que implicaba un alto costo computacional y de desarrollo. Sin embargo, con el avance del aprendizaje profundo, muchos de estos procesos han sido automatizados, permitiendo una mayor precisión y eficiencia en aplicaciones de visión artificial [6].

La segmentación de imágenes divide una imagen digital en distintos grupos de píxeles (segmentos de imagen) para facilitar la detección de objetos y otras tareas relacionadas. Las primeras técnicas de segmentación se basaban en heurísticas, analizando atributos como color e intensidad para determinar los límites entre diferentes regiones [7]. Sin embargo, la aparición del aprendizaje profundo y, en particular, las redes convolucionales, ha revolucionado la segmentación de imágenes permitiendo un reconocimiento de patrones más sofisticado, hasta el punto de que es posible resolver tres tipos de tareas diferentes: segmentación semántica, de instancias y panóptica [7]. La **segmentación semántica**, que es la tarea en la que se centra este trabajo, permite separar diferentes regiones de una imagen en base a una serie de categorías pre-definidas. Para ello, asigna una etiqueta de clase a cada píxel mediante un algoritmo de aprendizaje profundo, permitiendo a los modelos de visión por computador identificar y diferenciar con precisión áreas clave en una imagen [8]. Es muy utilizada, entre otras, en aplicaciones de conducción autónoma, donde es esencial distinguir los distintos elementos del entorno como "carretera", "persona" o "señal de tráfico".

A diferencia de la detección de objetos, que delimita elementos individuales con recuadros, la segmentación semántica clasifica todos los píxeles, proporcionando una comprensión más detallada de la composición visual (ver *Figura 3)*. Sin embargo, la

segmentación semántica presenta un enfoque limitado y altamente especializado, ya que solo clasifica píxeles en clases predefinidas, sin comprender el contexto de la imagen o transferir ese conocimiento a otras tareas. Este tipo de segmentación requiere grandes cantidades de datos etiquetados de alta calidad para entrenar modelos específicos y, aunque logra alta precisión, solo ejecuta patrones aprendidos sin verdadera comprensión. Por estas razones, la segmentación semántica es un ejemplo de IA débil, pues su rendimiento está restringido a la tarea específica para la cual fue entrenada [9].

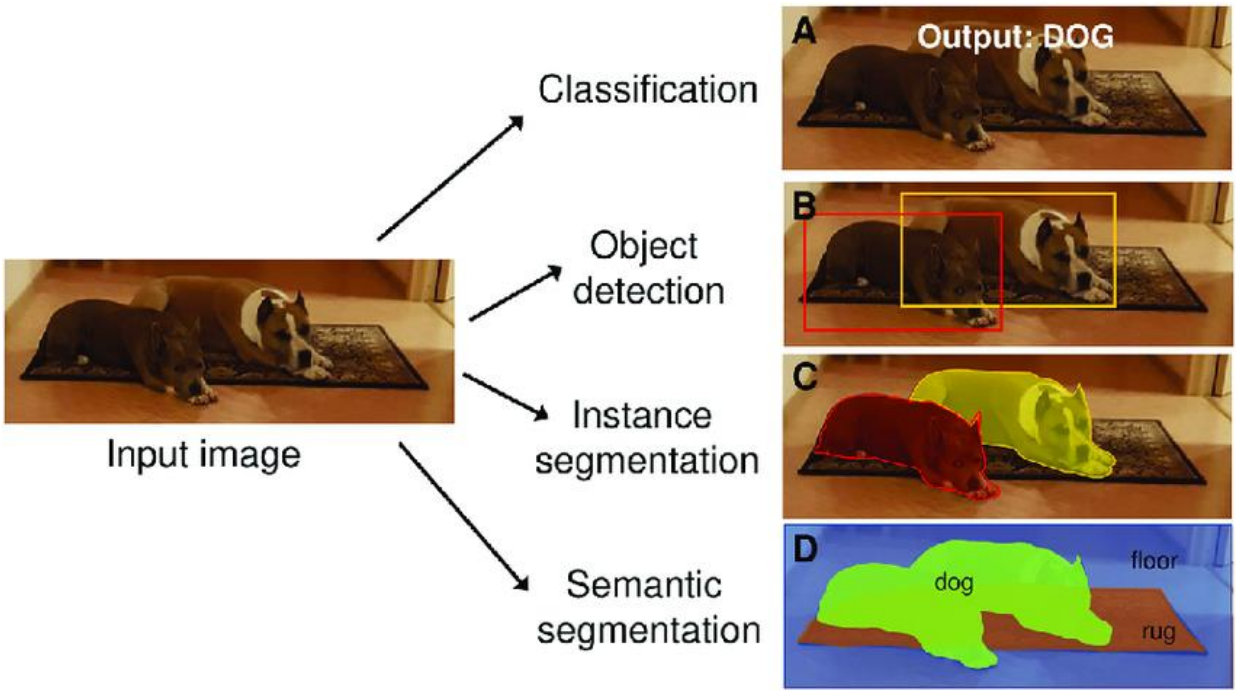

Figura 3: Comparación entre clasificación, detección de objetos, segmentación de instancia y segmentación semántica **[10]**

La segmentación de instancias y la segmentación panóptica representan enfoques más complejos, fuera del alcance de este trabajo. La segmentación de instancias no solo clasifica los píxeles de una imagen, sino que también identifica y separa objetos individuales dentro de una misma clase, distinguiendo, por ejemplo, entre diferentes individuos (clase “persona”). Sin embargo, no se aplica a clases generales que representan regiones continuas, como "cielo" o "carretera", ya que estas no requieren

una distinción entre instancias. Este enfoque es común en aplicaciones médicas, como el conteo de células en imágenes de tejidos, donde se necesita distinguir cada célula individualmente. La segmentación panóptica, por otro lado, combina las técnicas de segmentación semántica e instancias, proporcionando un análisis más completo de la imagen. Para ello, asigna etiquetas a todos los píxeles mientras distingue instancias únicas cuando es necesario, como en la realidad aumentada, donde es importante no solo saber qué objetos hay, sino también cuántos y en qué contexto (ver *Figura 4)* [8].

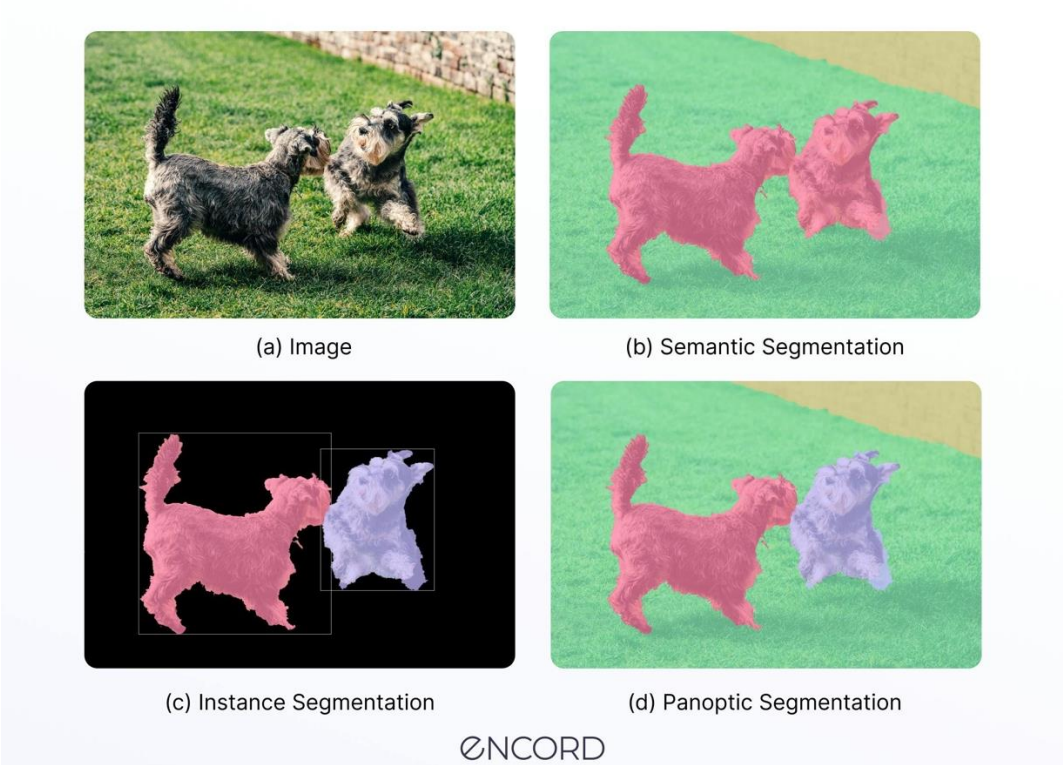

Figura 4: Comparación entre los tres tipos de tareas de segmentación de imágenes **[11]**

1.2. Motivación

A menudo, antes de imprimir una imagen, se quiere eliminar elementos no deseados, como cielos, fondos u otros objetos, y conservar el resto. Sin embargo, realizar esta edición manualmente puede ser difícil y las opciones de configuración en los sistemas de impresión suelen ser limitadas. Por ello, es necesario desarrollar una solución que permita realizar este proceso de forma eficiente y accesible.

Para lograrlo, es necesaria una segmentación semántica de alta precisión que permita diferenciar las distintas categorías de elementos de la imagen, agrupando aquellos con características similares en clases como "vegetación", “coche” o "persona". Esta técnica facilita la adaptación a la variedad de contenidos visuales de una imagen, permitiendo trabajar con categorías de objetos en lugar de cada elemento individual. Sin embargo, los métodos tradicionales de segmentación pueden ser limitados, tanto en precisión como en flexibilidad, y no siempre se adaptan adecuadamente a las diversas composiciones y tipos de agrupaciones que los usuarios desean manipular.

Este trabajo se centra en resolver este problema a través de una solución que permita a los usuarios seleccionar las clases de objetos que quieren eliminar en una imagen antes de imprimirla, conservando únicamente aquellas que no han sido descartadas, utilizando técnicas de segmentación semántica. El usuario puede elegir una clase en cada paso como por ejemplo eliminar primero la vegetación y luego el cielo, y aplicar en cada caso una acción específica. Las áreas descartadas se reemplazan con una máscara de color blanco, correspondiente al color del papel de impresión, o, de forma opcional, mediante técnicas de *in-painting* usando modelos generativos. Esto permite que el usuario pueda reemplazar una clase específica de la imagen, como un cielo nublado, por una nueva, como un cielo despejado, a partir de un texto descriptivo. La solución desarrollada facilita la segmentación precisa de imágenes, permitiendo así la eliminación o reemplazo de áreas específicas según las preferencias del usuario.

2. Objetivos concretos y relación con el estado actual

2.1. Objetivo general

El objetivo de este trabajo es desarrollar un modelo de segmentación semántica que permita a los usuarios eliminar o reemplazar clases específicas de una imagen antes de su impresión.

2.2. Objetivos específicos

Los objetivos específicos son:

1. **Estudiar la arquitectura U-Net y su funcionamiento.** Analizar los componentes y principios de operación de U-Net, incluyendo sus capas de codificación y decodificación, así como las conexiones de salto, para comprender su eficacia en la segmentación semántica de imágenes.

2. **Seleccionar y preparar un conjunto de datos de segmentación semántica adecuado para el entrenamiento y la evaluación del modelo U-Net.** Seleccionar un conjunto de datos que represente la variedad de contenidos visuales típicos en imágenes destinadas a impresión.

3. **Buscar y seleccionar un modelo basado en U-Net para segmentación semántica.** Identificar en repositorios de código público una implementación del modelo U-Net que cumpla con los requisitos de segmentación semántica y adaptarlo para garantizar que funcione correctamente en el contexto del proyecto.

4. **Optimizar el entrenamiento del modelo.** Experimentar con diferentes configuraciones de hiperparámetros y técnicas de aumento de datos para mejorar la precisión y la robustez del modelo en diversas condiciones de entrada.

5. **Desarrollar una interfaz de usuario para la selección de clases.** Crear una interfaz que permita a los usuarios seleccionar qué clases de objetos desean conservar y cuáles eliminar, facilitando la interacción con el modelo.

6. **Estudiar modelos generativos para la generación de imágenes a partir de texto.** Investigar diferentes modelos generativos que permitan crear imágenes basadas en descripciones textuales, evaluando sus capacidades, limitaciones y requisitos técnicos para identificar el más adecuado para el proyecto.

7. **Implementar una función de in-painting con el modelo generativo seleccionado.** Integrar el modelo generativo elegido para permitir la generación de contenido en áreas eliminadas, en respuesta a descripciones textuales proporcionadas por el usuario.

2.3. Relación con el estado actual

2.3.1. Trabajos relacionados

2.3.1.1. Redes neuronales

Una de las claves detrás de muchas de las innovaciones en visión por computador y procesamiento de imágenes es el uso de redes neuronales artificiales, un tipo específico de modelo matemático inspirado en las redes de neuronas biológicas. Es necesario entender sus principios básicos ya que constituyen la base de muchas de las soluciones modernas en la IA. Estas redes están inspiradas en el cerebro humano y sus procesos de aprendizaje, lo que les permite reconocer patrones y tomar decisiones de manera autónoma, emulando de alguna forma la forma en que los seres humanos aprenden. Al hacerlo, las redes neuronales son capaces de adaptarse a diferentes tareas y mejorar su rendimiento a medida que procesan más datos [12].

En la estructura neuronal biológica, en general, las neuronas constituyen procesadores de información sencillos. El sistema nervioso humano se puede clasificar como un

sistema de tres etapas (ver *Figura 5)*. El cerebro ocupa una posición central en el sistema nervioso, el cual está representado por la red neuronal. La función de la red neuronal es recibir la información de manera continua, percibirla y luego tomar la decisión apropiada. Los estímulos del entorno externo o del cuerpo humano son convertidos en impulsos eléctricos por los receptores, los cuales transmiten la información al cerebro (red neuronal). Los impulsos eléctricos generados por los receptores se convierten posteriormente en una respuesta diferenciable como salida del sistema nervioso por medio de los efectores. Este proceso de transmisión y procesamiento de señales es fundamental para entender el funcionamiento de las redes neuronales artificiales [13].

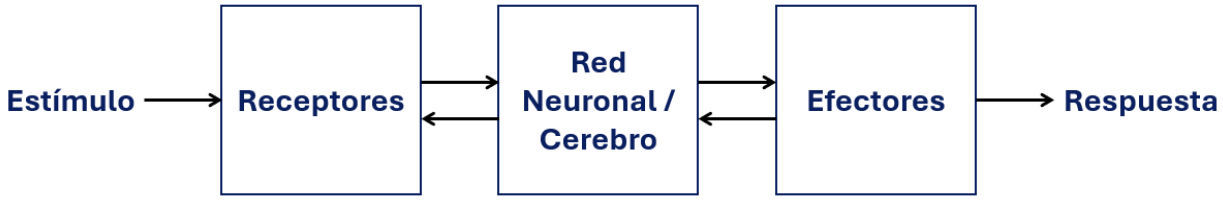

Figura 5: Representación del sistema nervioso mediante diagrama de bloques

Análogamente, en los sistemas de redes neuronales artificiales, se imita la estructura del sistema nervioso con la intención de construir sistemas con un cierto comportamiento inteligente en un esquema de procesamiento paralelo, memoria distribuida y adaptabilidad al entorno. Una neurona artificial toma un vector de entrada de variables independientes y genera una salida, calculada a partir de pesos sinápticos que determinan la importancia de cada entrada y que puede ser continua, binaria o categórica. El cálculo de la salida sigue tres pasos: una función de propagación (suma ponderada de entradas), una función de activación (para ajustar la salida) y una función de transferencia (para escalar la salida final) [12].

Cada capa de la red neuronal puede tener distintas funciones de activación, como la función rectificadora (ReLU) en capas ocultas o la función sigmoide en la capa de salida (ver *Figura 6)*. Las redes pueden aprender automáticamente a resolver problemas

específicos mediante el ajuste de sus parámetros (pesos y *bias*) en un proceso llamado entrenamiento [12].

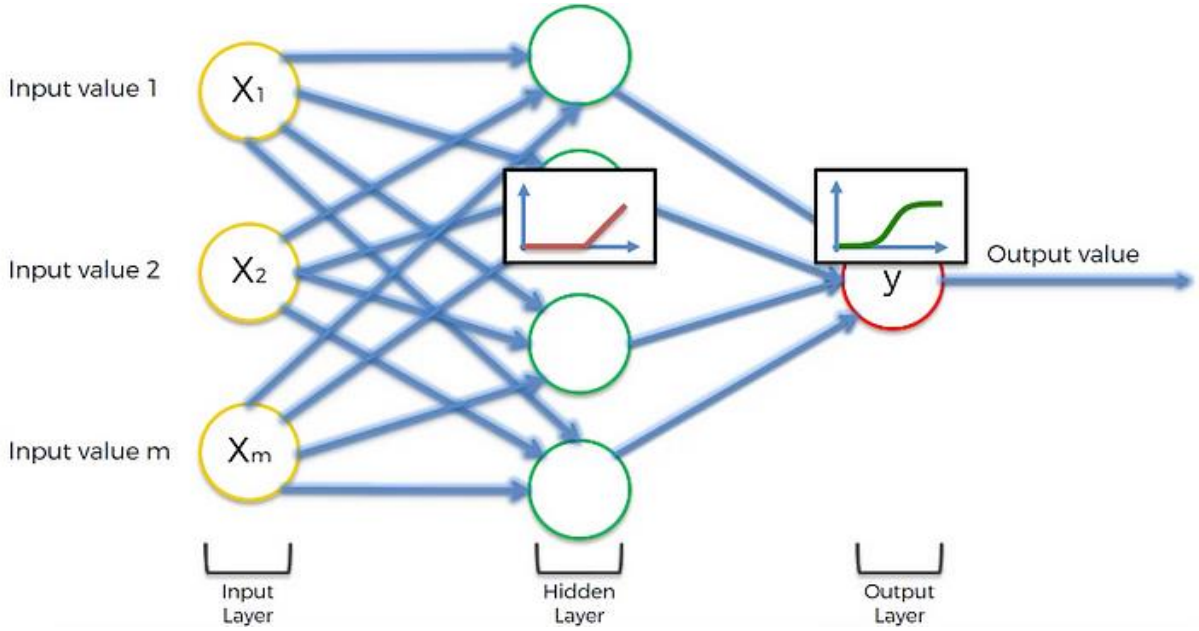

Figura 6: Red neuronal con distintas funciones de activación, la función rectificadora en la capa oculta y la función sigmoide en la capa de salida **[12]**

2.3.1.2. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNNs, por sus siglas en inglés) son un tipo de red neuronal artificial diseñada específicamente para el procesamiento y análisis de imágenes. Estas redes se inspiran en el funcionamiento del córtex visual humano, el cual identifica características visuales a diferentes niveles de detalle. Las CNNs utilizan múltiples capas que procesan las características de las imágenes de manera jerárquica. Esto significa que las primeras capas de una CNN detectan características básicas, como líneas y bordes, mientras que las capas profundas son capaces de reconocer patrones complejos, como formas y siluetas de objetos completos [14].

Para que una CNN aprenda a clasificar correctamente diferentes tipos de objetos, como por ejemplo perros y gatos, necesita un gran volumen de datos de entrenamiento. Estas imágenes de entrenamiento permiten que la red capture las características únicas de

cada clase y aprenda a generalizar, de forma que pueda reconocer variaciones dentro de la misma clase. Por ejemplo, la red aprenderá a identificar un gato en distintas posiciones, colores y contextos [14].

Las redes neuronales convolucionales reciben como entrada una imagen representada por una matriz de píxeles con valores que oscilan entre 0 y 255. Esta matriz tiene un solo canal para imágenes en escala de grises y tres canales para imágenes en color RGB. Antes del procesamiento, es habitual normalizar estos valores dividiéndolos por 255 para ajustarlos al rango [0, 1] (Ver *Figura 7*) [14].

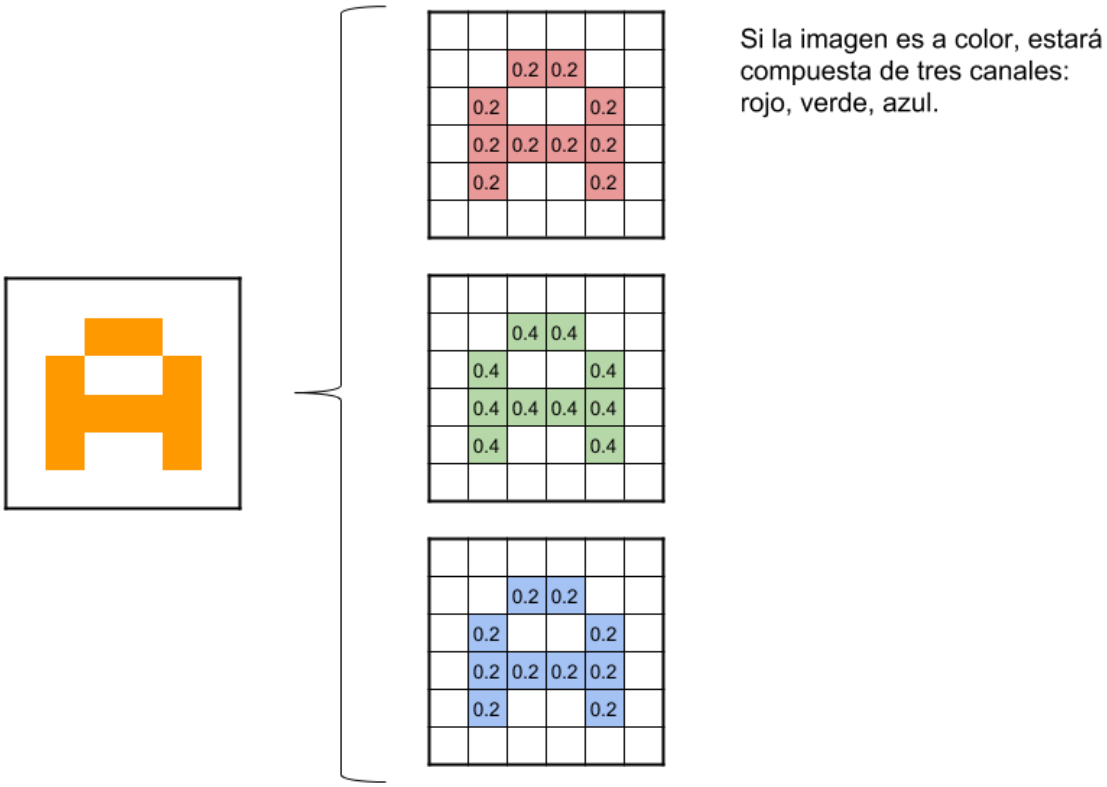

***Figura 7: Canales de imagen pre-procesados para la capa de entrada de una CNN* [14]**

Una CNN se compone de varias operaciones clave que permiten extraer características relevantes de las imágenes y reducir su tamaño progresivamente sin perder información importante.

Cada capa en una red convolucional procesa volúmenes de dimensiones h×w×d, donde h y w son las dimensiones espaciales y d es el número de canales (capa de entrada) o mapas de características [15]. El procesamiento en una capa convolucional puede expresarse mediante la ecuación:

$$y_{ij} = f_{ks}\left(\{x_{si+\delta i,sj+\delta j}\}0 \leq \delta i, \delta j \leq k\right),$$

donde $x_{si+\delta i,sj+\delta j}$ representa los valores de una ventana de tamaño k×k de la entrada, desplazada según el stride s. La función f_{ks} aplica un filtro convolucional sobre esta ventana para generar un único valor en la salida.

A continuación, se describen las principales operaciones utilizadas en una CNN:

- **Convolución:** En una CNN, la convolución es el proceso mediante el cual se aplican pequeños filtros, llamados *kernels*, a la imagen de entrada. Estos filtros son matrices que recorren la imagen, horizontal y verticalmente, y realizan un producto escalar sobre grupos de píxeles cercanos, generando mapas de características. Cada uno de estos mapas refleja las características detectadas por el filtro correspondiente en diferentes partes de la imagen, y cada uno de estos filtros se especializa en captar diferentes tipos de patrones, como bordes o texturas (ver *Figura 8)*. En las primeras capas, estos patrones son sencillos, pero en capas más profundas los filtros capturan estructuras más complejas [14].

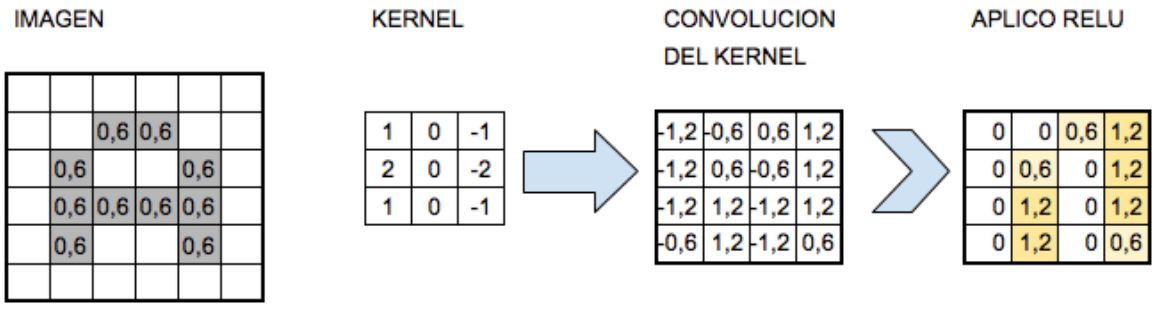

***Figura 8: Canales de imagen pre-procesados para la capa de entrada CNN* [14]**

- **Padding:** La convolución reduce el tamaño de la imagen, ya que el filtro no puede procesar los píxeles de los bordes de la imagen. Para evitar esta pérdida de información, se utiliza el *padding*, que consiste en agregar bordes, típicamente de ceros, alrededor de la imagen original (ver *Figura 9)*. Esto permite que la imagen de salida conserve el mismo tamaño que la de entrada, facilitando la detección de características en los bordes y permitiendo la construcción de redes más profundas [16].

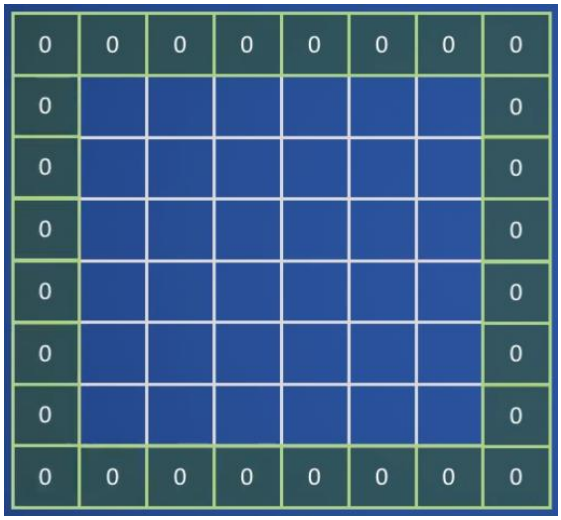

Figura 9: Imagen original (en el centro) y el padding con ceros (en los bordes) **[16]**

- **Stride:** El paso o *stride* controla el desplazamiento del filtro en cada iteración. Un stride de 1 implica que el filtro se desplaza un píxel a la vez, mientras que un stride mayor reduce el tamaño de la imagen de salida al saltar más píxeles en cada paso. Esto ayuda a reducir la cantidad de datos a procesar en cada capa de la red [16].

- **Pooling:** El *pooling* es un método de *subsampling* que se utiliza para reducir el tamaño de los mapas de características y seleccionar solo la información más relevante. En esta operación, la imagen se divide en pequeñas regiones, generalmente de 2x2 píxeles, y se selecciona un valor dentro de cada región. Este proceso no solo reduce el número de datos, sino que también hace que la red sea más robusta ante pequeñas variaciones en la imagen, preservando las

características principales [14]. El valor seleccionado es, habitualmente, el máximo (*max-pooling*) o el promedio (*average-pooling*).

- **Stacking:** En lugar de aplicar un solo filtro, las CNNs utilizan múltiples filtros en cada capa de convolución para capturar distintas características de la imagen. El *stacking* o apilamiento consiste en combinar estos mapas de características generados por distintos filtros en una estructura tridimensional. Esto permite a la red analizar múltiples aspectos de la imagen al mismo tiempo y construir una representación más completa y detallada de sus características [16].

Después de las capas de convolución y pooling, los mapas de características obtenidos se transforman en un vector de características a través de una operación de aplanamiento o pooling global. Este vector se procesa mediante una o más capas totalmente conectadas, que procesan la información agregada de todos los filtros y producen la clasificación final.

La capa final de la red CNN contiene una función de activación que convierte las salidas en probabilidades [14]. La elección de esta función depende del tipo de problema a resolver [17]:

- **Función sigmoide (sigmoid):** Se emplea para tareas de clasificación binaria o multietiqueta, generando valores entre 0 y 1 de forma independiente para cada clase.
- **Función softmax:** Se emplea para tareas de clasificación multiclase, asignando una probabilidad a cada clase de manera que la suma total de probabilidades sea igual a 1.

El entrenamiento de la red se realiza mediante el algoritmo de retropropagación (*backpropagation*), que ajusta los pesos de los filtros para minimizar el error en las predicciones, independientemente de la función de activación de salida utilizada [14].

2.3.1.3. Técnicas de segmentación semántica

En el contexto de la segmentación semántica, las CNNs han revolucionado el campo al permitir una clasificación precisa a nivel de píxel en imágenes. Los dos enfoques predominantes son las primeras aproximaciones basadas en redes totalmente convolucionales y las arquitecturas codificador-decodificador, que mejoran la precisión en los bordes y la capacidad de detección multiescala.

2.3.1.3.1. Primeras aproximaciones: Redes totalmente convoluciones

Las redes totalmente convolucionales (FCNs, por sus siglas en inglés) son un tipo de red neuronal convolucional diseñada específicamente para tareas de predicción densa. Su capacidad para mapear características de salida de baja resolución hacia los píxeles originales mediante capas de *upsampling* permite obtener segmentaciones precisas (ver *Figura 10*). La operación de upsampling permite aumentar la resolución espacial de los mapas de características generados por la red, devolviéndolos a su tamaño original. Para ello, las FCNs utilizan convoluciones transpuestas o deconvoluciones, que expanden los mapas de características a una resolución mayor aprendiendo patrones de reconstrucción de los datos [15].

A diferencia de las CNNs como VGG, AlexNet o GoogLeNet, que generan salidas no espaciales como clasificaciones, las FCNs producen mapas de características de tamaño variable que conservan la correspondencia espacial con la entrada [15].

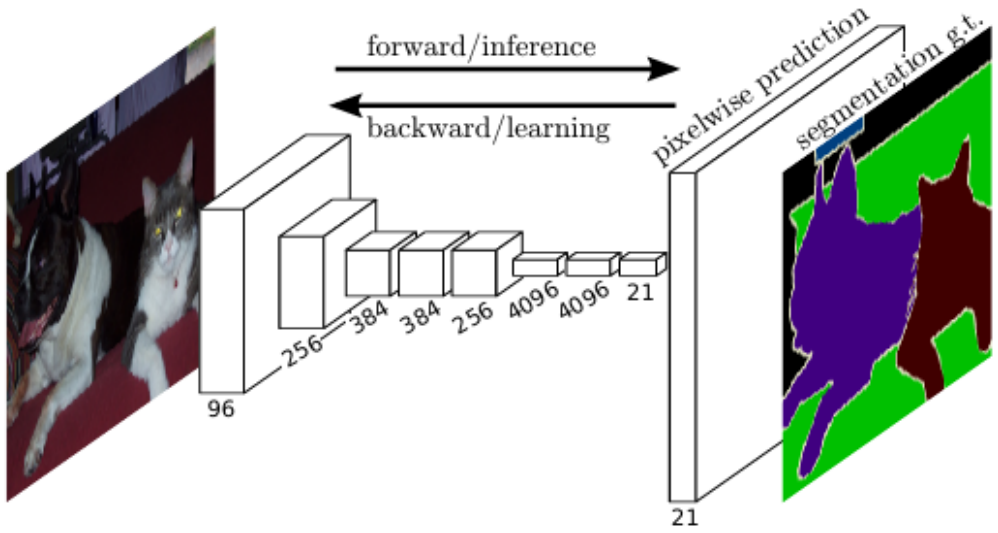

Figura 10: FCN generando predicciones densas para tareas a nivel de píxel como la segmentación semántica **[15]**

Las FCNs están compuestas exclusivamente por capas convolucionales, de pooling y de upsampling. Estas redes no incluyen capas completamente conectadas, lo que les permite procesar entradas de tamaño arbitrario. Además, se entrenan de extremo a extremo (*end-to-end*) para tareas de predicción píxel a píxel, ajustando directamente los pesos mediante el algoritmo de retropropagación. La salida de una FCN es una imagen segmentada del mismo tamaño que la original, lograda mediante una interpolación bilineal que escala los mapas de predicción a la resolución deseada [15].Aunque las FCNs lograron segmentaciones eficientes, las características generadas en las capas profundas pierden detalles espaciales debido a operaciones como max-pooling y al hiperparámetro stride, lo que impacta la precisión en los bordes y en estructuras pequeñas [15].

2.3.1.3.2. Arquitecturas codificador-decodificador

Con el fin de abordar las limitaciones de las FCNs, surgieron arquitecturas basadas en un esquema codificador-decodificador, diseñadas específicamente para segmentación. A continuación, se detallan algunas de las más populares.

- **Redes de deconvolución**

Las redes de deconvolución (*deconvolution networks*) son arquitecturas que combinan capas de deconvolución y *unpooling* con el objetivo de identificar las etiquetas de clase por píxeles y predecir las máscaras de segmentación. Estas redes pueden manejar variaciones de escala y capturar detalles finos que las FCNs no logran debido a la pérdida de detalles espaciales, reconstruyendo la estructura y los detalles del objeto a partir de las características extraídas [18].

Los procesos principales son [18]:

- **Unpooling:** Restaura la resolución espacial perdida utilizando información guardada durante el max-pooling de las capas convolucionales, reconstruyendo la localización original de las activaciones.
- **Deconvolución:** Densifica las activaciones ampliadas mediante filtros aprendidos, permitiendo capturar detalles de forma y mejorar la precisión en los bordes.

Al combinar una red de convolución basada en VGG16 con una red de deconvolución, se forma una arquitectura codificador-decodificador capaz de generar mapas de predicción densos por píxeles (ver *Figura 11*) [18].

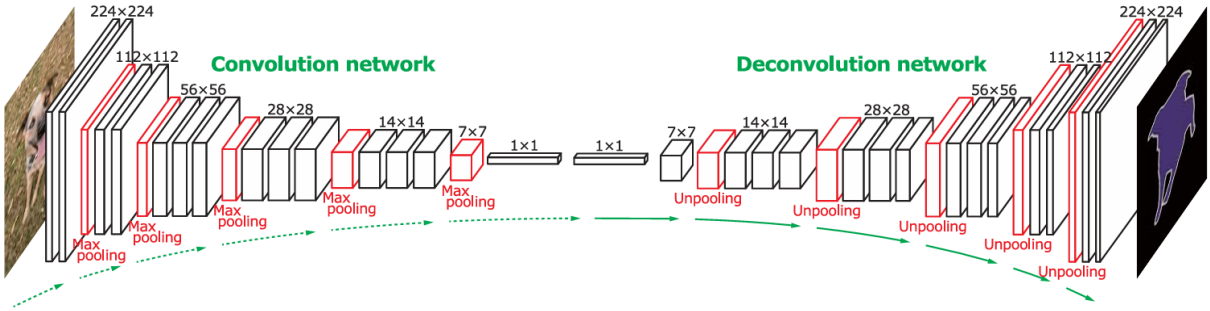

Figura 11: En la parte superior de la red de convolución basada en la red VGG de 16 capas, se coloca una red de deconvolución multicapa para generar el mapa de segmentación preciso de una propuesta de entrada. Dada una representación de características obtenida de la red de convolución, se construye un mapa de predicción de clases densas por píxeles a través de múltiples series de operaciones de deconvolución y unpooling **[18]**

- **SegNet**

SegNet es una red neuronal convolucional totalmente entrenable diseñada para la segmentación semántica a nivel de píxel. Su estructura se basa en una arquitectura codificador-decodificador que se distingue por cómo maneja la reconstrucción de la resolución espacial en el proceso de decodificación [19].

Los procesos principales son [19]:

- **Codificador:** Basado en las 13 capas convolucionales de VGG16, el codificador extrae características a resoluciones reducidas mediante operaciones de convolución y max-pooling. Este proceso conserva los índices de las operaciones de pooling, que son claves para el decodificador.
- **Decodificador:** Emplea los índices de pooling almacenados para realizar un upsampling no lineal, lo que evita la necesidad de aprender cómo interpolar las características. Las características ampliadas son densificadas mediante convoluciones con filtros entrenables, lo que permite generar mapas densos para la clasificación por píxel.

- **Clasificación a nivel de píxel:** Utiliza un clasificador softmax para generar probabilidades por clase a nivel de píxel, produciendo una segmentación precisa y eficiente.

Al almacenar solo los índices de pooling en lugar de todos los mapas de características, SegNet reduce significativamente los requerimientos de memoria durante la inferencia y mejora la precisión en la delineación de bordes, lo cual es esencial para clases pequeñas o con formas complejas. Comparado con arquitecturas anteriores, SegNet logra un buen equilibrio entre precisión y velocidad, especialmente en tareas de comprensión de escenas de carretera e interiores [19].

A diferencia de las redes tradicionales que incluyen capas completamente conectadas, SegNet utiliza únicamente capas convolucionales. El decodificador realiza el upsampling a partir de los índices de pooling transferidos por el codificador, generando un mapa de características disperso. Este mapa se densifica posteriormente mediante convoluciones con filtros entrenables y, finalmente, los mapas de salida del decodificador son procesados por un clasificador softmax para obtener las probabilidades por píxel (ver *Figura 12*) [19].

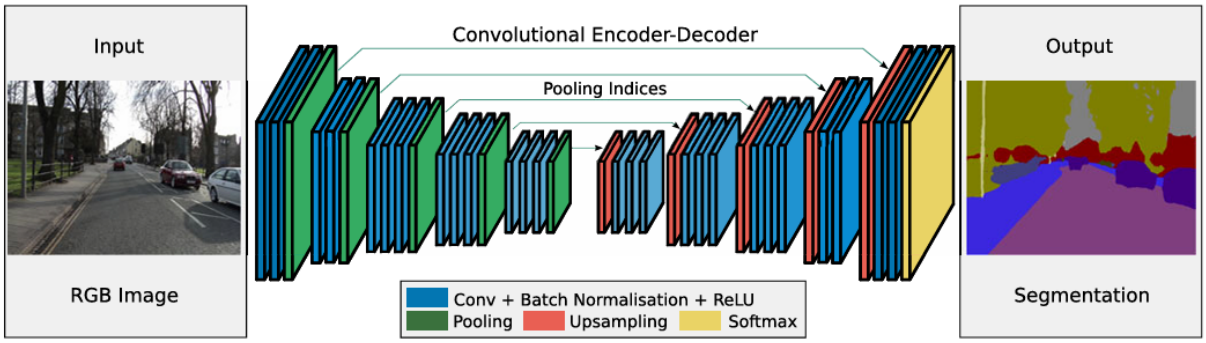

Figura 12: Arquitectura de SegNet. No incluye capas completamente conectadas y opera únicamente mediante capas convolucionales. El decodificador realiza operaciones de upsampling utilizando los índices de pooling transferidos por el codificador y densifica el mapa con filtros entrenables antes de la clasificación por píxel **[19]**

- **U-Net**

U-Net es una red neuronal convolucional diseñada específicamente para tareas de segmentación semántica. Destaca por su capacidad para entrenarse de extremo a extremo con un conjunto limitado de datos anotados, logrando segmentaciones precisas mediante un esquema simétrico de codificación y decodificación. La arquitectura de U-Net combina de manera efectiva la localización precisa con el contexto global, lo que la hace adecuada para imágenes donde la cantidad de datos de entrenamiento es limitada pero los detalles locales son críticos [20].

La red se estructura en dos componentes principales (ver *Figura 14*) [20]:

- **Codificación:** Este componente reduce las dimensiones espaciales mientras aumenta el número de canales de características, capturando información de contexto global. Cada nivel de este camino consta de:
  - Dos convoluciones 3x3 (sin padding), seguidas de una activación ReLU.
  - Una operación de max-pooling 2x2 con un stride de 2, que reduce la resolución espacial.
  - En cada nivel, el número de canales se duplica para enriquecer la representación de características.

  La salida del último nivel contiene un conjunto comprimido de características con información contextual global.

- **Decodificación:** Este componente recupera las dimensiones espaciales mientras reduce el número de canales. Sus componentes incluyen:
  - Operaciones de upsampling mediante convoluciones transpuestas 2x2, que reducen a la mitad el número de canales.
  - Concatenación de los mapas de características del camino de codificación, lo que ayuda a combinar detalles locales con contexto global.
  - Dos convoluciones 3x3 seguidas de activaciones ReLU para refinar las características combinadas.

La última capa es una convolución 1x1 que genera mapas de probabilidad por clase, adaptándose al número deseado de clases.

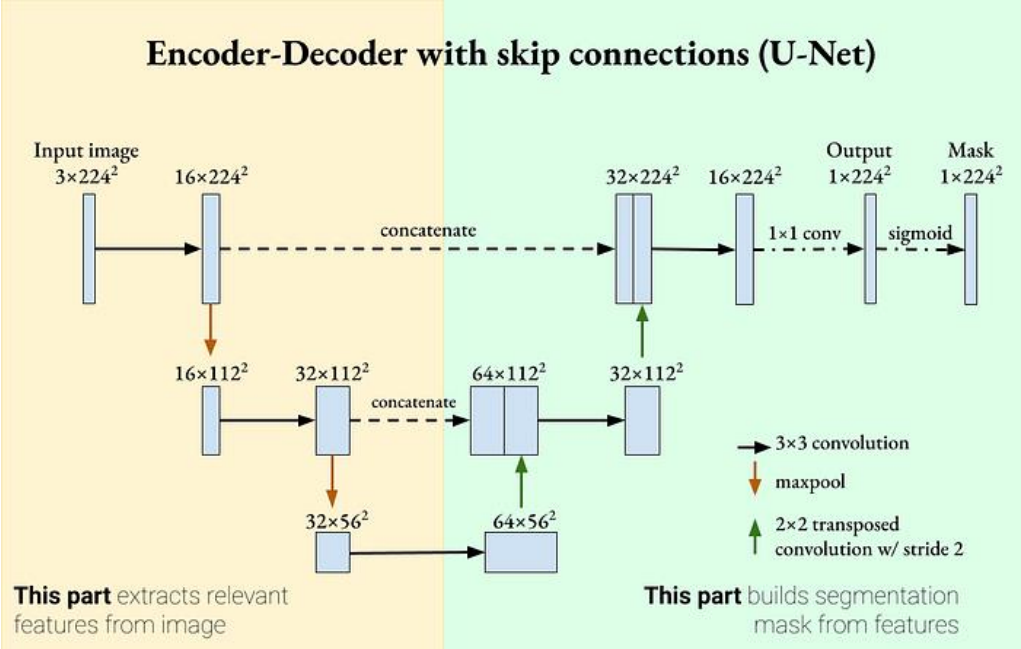

Figura 13:: Arquitectura U-Net con estructura de codificación-decodificación. La fase de codificación extrae características clave reduciendo la resolución mediante convoluciones y max-pooling, mientras que la fase de decodificación la recupera mediante convoluciones transpuestas y conexiones de salto, permitiendo combinar detalles locales con contexto global. Su última capa aplica una convolución 1x1 para generar mapas de probabilidad por clase **[21]**

U-Net utiliza deformaciones elásticas para aumentar el conjunto de datos de entrenamiento, crucial en imágenes biomédicas con datos limitados. Estas transformaciones generan robustez frente a variaciones en forma y desplazamiento. Los pesos iniciales se seleccionan de una distribución normal con desviación estándar $\sqrt{(2/N)}$, donde N es el número de nodos de entrada, asegurando una varianza uniforme entre mapas de características [20].

- **DeepLab**

DeepLab es una arquitectura de código abierto diseñada por Google. Está basada en redes convolucionales profundas (DCNNs, por sus siglas en inglés) y ha sido diseñada

para abordar desafíos clave en la segmentación semántica, como la resolución reducida de características, la variabilidad de escalas en los objetos y la precisión limitada en los bordes. Para ello, introduce innovaciones fundamentales [22]:

- **Convoluciones dilatadas:** Esta técnica permite aumentar la resolución de las características sin incrementar los parámetros del modelo o el coste computacional. Las convoluciones dilatadas amplían el campo de visión de los filtros insertando huecos en los valores del filtro, lo que facilita capturar un mayor contexto espacial en las imágenes.
- **Pirámide espacial con convoluciones dilatadas (ASPP, por sus siglas en inglés)**: Este método aplica filtros convolucionales dilatados con diferentes tasas de muestreo de forma paralela, logrando captar tanto detalles finos como contexto global en múltiples escalas. Esto es crucial para segmentar objetos grandes y pequeños de manera eficiente.
- **Campos aleatorios condicionales (CRF, por sus siglas en inglés):** DeepLab combina los resultados de la red convolucional con un CRF completamente conectado, lo que mejora la segmentación al afinar los bordes de los objetos y capturar relaciones espaciales a largo alcance entre píxeles. Este enfoque ayuda a superar las limitaciones de localización en las DCNN profundas.

DeepLab utiliza una CNN como VGG-16 o ResNet-101, adaptando las capas completamente conectadas a capas convolucionales para generar predicciones densas por píxel, optimizando la segmentación semántica (ver *Figura 14*) [22].

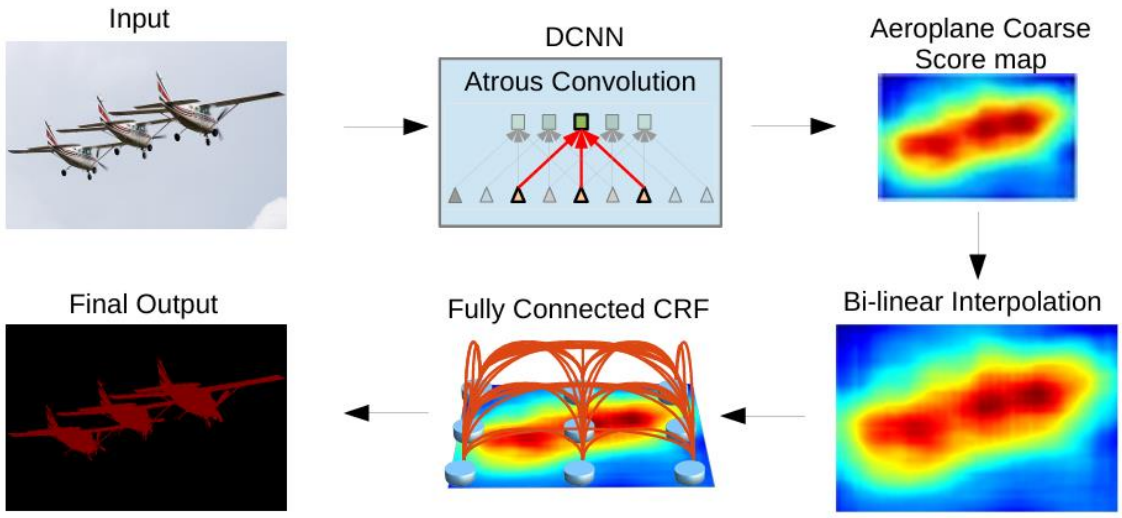

Figura 14: Arquitectura de DeepLab que combina redes convolucionales profundas, convoluciones dilatadas, interpolación bilineal y CRF para obtener segmentaciones semánticas precisas **[22]**

2.3.1.4. Modelos generativos e In-Painting

En los últimos años, los modelos generativos han revolucionado el campo de la visión por computador al introducir nuevas capacidades para la edición avanzada de imágenes. Entre estos, destacan modelos como Stable Diffusion, GLIDE y DALL-E, desarrollados para realizar tareas de generación y edición de imágenes condicionadas por texto. Estos modelos utilizan técnicas de difusión y aprendizaje profundo para transformar imágenes a partir de instrucciones textuales, lo que permite aplicaciones como la sustitución de objetos, la creación de contenido realista en áreas eliminadas (*in-painting*) y la modificación de fondos.

Stable Diffusion: Es un modelo basado en técnicas de difusión latente, diseñado para generar imágenes realistas a partir de ruido aleatorio condicionado por texto. Su capacidad para realizar in-painting permite editar regiones específicas de una imagen mediante máscaras y *prompts* textuales [23]. Esto lo hace ideal para tareas avanzadas de edición visual, como la sustitución de objetos en una escena o la creación de contenido coherente en áreas eliminadas.

GLIDE: Es un modelo generativo basado en la técnica de difusión guiada por texto, diseñado para la síntesis de imágenes más fotorrealistas. Además de generar imágenes desde cero, GLIDE ofrece capacidades avanzadas de edición, como el in-painting, que permite rellenar áreas específicas de una imagen de manera coherente con el contexto visual y acorde a una descripción textual proporcionada [24]. Por ejemplo, puede transformar un "cielo nublado" en un "cielo despejado" siguiendo una instrucción escrita. Esta combinación de generación y edición iterativa amplía las posibilidades en la creación y modificación de contenido visual con alta precisión.

DALL-E: Desarrollado por OpenAI, DALL-E es un modelo generativo diseñado para crear imágenes a partir de descripciones textuales detalladas. Con su evolución a DALL-E 2 y posteriormente a DALL-E 3, estas versiones han ampliado sus capacidades, incluyendo herramientas avanzadas como el in-painting, que permite editar o reemplazar elementos específicos de una imagen manteniendo la coherencia visual general. Esto facilita tanto la generación de imágenes completamente nuevas como la personalización de contenido visual existente, adaptándose a necesidades creativas y comerciales con gran precisión [25].

Estos modelos generativos permiten abordar el in-painting con precisión y flexibilidad, lo que resulta ideal para la edición avanzada en la que el usuario necesita reemplazar o complementar elementos visuales de manera natural y realista. En este proyecto, los modelos generativos se exploran como una opción para la sustitución de contenido visual en áreas segmentadas, una vez que el usuario ha decidido qué elementos conservar o eliminar.

3. Metodología de trabajo

3.1. Componentes software

Esta sección incluye una descripción de todos los componentes de software utilizados durante este trabajo.

3.1.1. Git

Git es un sistema de control de versiones distribuido, gratuito y de código abierto, diseñado para gestionar proyectos de cualquier tamaño con rapidez y eficiencia. Inicialmente, fue desarrollado para el kernel de Linux y escrito en C para minimizar la sobrecarga. Su enfoque se centra en la velocidad, la flexibilidad y la integridad de los datos [26].

Entre sus características más destacadas, Git permite trabajar con múltiples ramas locales de forma eficiente. Además, no es necesario compartir todas las ramas en un repositorio remoto, lo que fomenta la experimentación local sin riesgo. Por otro lado, cada clon de un repositorio Git contiene la totalidad de su historial, lo que actúa como una copia de seguridad completa y elimina el riesgo de depender de un único punto de fallo [26].

En términos de seguridad, Git garantiza la integridad criptográfica de cada archivo y confirmación mediante sumas de comprobación. De este modo, se asegura que el historial de un proyecto no pueda ser modificado sin alterar todos los identificadores relacionados [26].

Finalmente, Git introduce una innovadora área de preparación (o índice), que permite seleccionar con precisión qué cambios incluir en cada confirmación. Gracias a esto, se facilita un control más granular y se evita mezclar cambios no relacionados en una sola operación [26].

3.1.2. GitLab

GitLab comenzó como una plataforma para alojar repositorios gestionados con el sistema de versiones Git [27]. Sin embargo, ha evolucionado para convertirse en una plataforma DevOps completa y de código abierto que no solo administra repositorios, sino que también permite a los equipos de desarrollo cooperar de manera productiva, automatizar tareas repetitivas y entregar software más rápido y con mejor calidad. Esta evolución permite a los equipos de desarrollo gestionar de manera eficiente todo el ciclo de vida de sus proyectos, desde la planificación hasta la implementación [28].

Para facilitar la organización y colaboración, GitLab incluye funcionalidades clave [28]:

- **Seguimiento de incidencias:** Proporciona un sistema integrado que permite crear, asignar, priorizar y rastrear tareas, errores y solicitudes de características. Esto fomenta una comunicación eficaz entre los miembros del equipo y mejora la gestión de tareas.
- **Wiki integrado:** Actúa como un repositorio central de conocimiento, permitiendo documentar información clave del proyecto de forma accesible para todos los colaboradores.
- **Solicitudes de fusión (*merge requests*):** Permiten a los desarrolladores proponer cambios al código base, que deben ser revisados y aprobados por sus compañeros antes de ser integrados en la rama principal. Este proceso asegura el cumplimiento de estándares de calidad y mejora la colaboración en equipo.
- **GitLab runners:** Son agentes encargados de ejecutar las tareas definidas en las canalizaciones de integración y despliegue continuo (CI/CD), una práctica que automatiza la integración, prueba y despliegue del software para garantizar entregas rápidas y confiables. Los runners pueden configurarse para proyectos específicos o compartirse entre varios, ejecutándose en diferentes entornos como Linux, Windows, macOS o contenedores Docker.
- **Gestión de grupos y proyectos:** GitLab organiza repositorios en grupos y proyectos, lo que facilita el control de accesos y la asignación de permisos. Esto

permite a los equipos colaborar y compartir recursos de manera eficiente entre proyectos relacionados.

3.1.3. Google Colab

Google Colab, también conocido como Colaboratory, es una plataforma gratuita basada en la nube que permite programar y ejecutar código en Python directamente desde el navegador. Diseñado inicialmente como una extensión de los cuadernos de Jupyter, Colab ofrece múltiples ventajas, entre las que destacan [29]:

- No requiere configuración previa: No es necesario instalar software ni configurar entornos de desarrollo.
- Acceso a GPUs sin coste adicional.
- Permite compartir contenido fácilmente: Los cuadernos se almacenan en Google Drive, lo que permite compartirlos fácilmente con otros usuarios.

Colab permite combinar en un único documento código ejecutable, texto enriquecido, gráficos e incluso imágenes, lo que lo convierte en una herramienta ideal para documentar experimentos y presentar resultados. Además, integra la capacidad de importar datos desde diversas fuentes como Google Drive o GitHub. Las bibliotecas más populares de Python para análisis y visualización de datos, como NumPy y Matplotlib, están preinstaladas, simplificando el flujo de trabajo [29].

Además, permite ejecutar código en servidores de Google, lo que facilita el uso de GPUs y TPUs para tareas intensivas en computación, como el aprendizaje profundo y la segmentación semántica. Por ejemplo, en proyectos de procesamiento de imágenes, se puede importar un conjunto de datos, entrenar un modelo de clasificación o segmentación, y evaluar los resultados con tan solo usar unas pocas líneas de código, superando las limitaciones de hardware local [29].

Por estas razones, y debido a las restricciones de computación locales y a la alta capacidad computacional del procesamiento de modelos de segmentación semántica,

se ha optado por un enfoque cliente/servidor utilizando Google Colab. Esto permite aprovechar máquinas con GPUs más potentes que las disponibles localmente, optimizando el tiempo de ejecución y el rendimiento del modelo.

3.1.4. PyTorch

PyTorch es un framework de aprendizaje profundo de código abierto que ofrece interfaces en Python y C++. Diseñado para facilitar el desarrollo de modelos de inteligencia artificial, sus características principales son las siguientes [30]:

- **Tensores:** Son arreglos multidimensionales similares a los de NumPy, pero con la ventaja de que pueden aprovechar GPUs para cálculos más rápidos. PyTorch incluye una amplia variedad de operaciones tensoriales, como álgebra lineal, reducción de dimensiones y transposición, lo que lo convierte en una herramienta versátil para cálculos científicos y modelado de redes neuronales.
- **Gráficos computacionales dinámicos:** El gráfico se crea sobre la marcha a medida que se realizan las operaciones, lo que permite cambios dinámicos durante el tiempo de ejecución, permitiendo mayor flexibilidad para ajustar y depurar modelos.

Además, PyTorch dispone de cuatro módulos clave [30]:

- **Módulo Autograd:** Calcula automáticamente gradientes de tensores con respecto a algún valor escalar. Esto es crucial para entrenar redes neuronales mediante algoritmos de optimización basados en gradientes. Durante el entrenamiento, la función *loss.backward()* calcula los gradientes de manera automática.
- **Módulo nn:** Incluye capas predefinidas, funciones de pérdida y otros componentes para crear y entrenar redes neuronales. También se pueden definir arquitecturas de redes neuronales personalizadas mediante la creación de subclases de la clase *torch.nn.Module*.

- **Módulo Optim:** Implementa algoritmos de optimización para ajustar los pesos del modelo. Contiene los algoritmos más utilizados, como Adam, SGD y RMS-Prop. Para usar *torch.optim*, se crea un objeto *Optimizer* que mantendrá los parámetros y los actualizará en consecuencia. Antes de la retropropagación es necesario reiniciar los gradientes con *optimizer.zero_grad()*, y tras el cálculo de los mismos, se actualizan los parámetros mediante *optimizer.step()*.
- **Módulo torch.utils.data:** Contiene las clases Dataset y DataLoader, que simplifican la carga y manipulación de datos, incluyendo opciones para procesamiento paralelo y creación de lotes.

PyTorch es compatible con plataformas de computación en la nube, como Google Colab, permitiendo a los usuarios acceder a GPUs sin necesidad de hardware local.

3.1.5. Paquetes de Python para la generación de conjuntos de datos

Una de las tareas de este trabajo fue generar o crear conjuntos de datos que se utilizaron para alimentar los modelos de segmentación semántica. Algunos de los paquetes que se utilizaron son:

- **NumPy:** Es el paquete fundamental para la computación científica en Python, que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas [31]. En este trabajo se utiliza para realizar operaciones matriciales eficientes, como el mapeo de etiquetas en la segmentación de imágenes y la manipulación de matrices de etiquetas para convertirlas a formatos visualizables.
- **Python imaging library (PIL):** Añade capacidades de procesamiento de imágenes al intérprete de Python. Proporciona soporte para una amplia variedad de formatos de archivo, una representación interna eficiente y herramientas poderosas para procesar imágenes. Su núcleo está diseñado para un acceso rápido a datos en formatos básicos de píxeles, sirviendo como base sólida para desarrollar herramientas generales de procesamiento de imágenes [32]. En este trabajo se utiliza para cargar, procesar y convertir imágenes en diferentes

formatos y modos, como la conversión a RGB o el guardado de imágenes procesadas.

- **torchvision:** Esta biblioteca es parte del proyecto PyTorch. Contiene conjuntos de datos populares, arquitecturas de modelos y transformaciones comunes de imágenes para visión por computadora [33]. En este trabajo se utiliza para realizar transformaciones de imágenes, como redimensionamiento y conversión a tensores, que son aplicadas a las imágenes de entrada y sus correspondientes máscaras de etiquetas.
- **Matplotlib:** Es una biblioteca completa para crear visualizaciones estáticas, animadas e interactivas en Python. Permite crear gráficos de calidad profesional, generar figuras interactivas con zoom, panorámica y actualizaciones, personalizar estilo y diseño visual, exportar en múltiples formatos de archivo, integrarse con JupyterLab e interfaces gráficas, y usar una amplia variedad de paquetes de terceros basados en Matplotlib [34]. En este trabajo se utiliza para generar y visualizar gráficos que muestran la evolución de la función de pérdida y las métricas de rendimiento durante las fases de entrenamiento y validación del modelo a lo largo de las épocas.
- **tqdm:** Proviene de la palabra árabe "taqaddum", que significa "progreso". Es una librería de Python que facilita la creación de barras de progreso de manera sencilla. Permite agregar barras de progreso a los bucles e iterar sobre elementos mostrando el avance en tiempo real con solo unas pocas líneas de código [35]. En este trabajo se utiliza para visualizar el progreso del entrenamiento y la validación durante el ciclo de épocas, mostrando barras de progreso que indican el avance en cada batch dentro de de cada época.

3.2. Arquitectura

De las diversas arquitecturas presentadas en el estado del arte, se ha seleccionado U-Net debido a que es muy popular y reconocida por su eficacia en tareas de segmentación semántica. Aunque se introdujo inicialmente en el ámbito de la medicina, también ha sido aplicada con éxito en otros ámbitos como, por ejemplo, en la

segmentación semántica de imágenes satelitales [36] o en el análisis de escenas urbanas, facilitando la detección de elementos como carreteras, peatones y vehículos [37]. Además, su estructura de codificación-decodificación permite procesar imágenes manteniendo detalles importantes gracias a sus conexiones de salto. En esta sección se describen más en detalle sus componentes principales y su funcionamiento.

U-Net es una red neuronal convolucional diseñada para tareas de segmentación semántica. Su estructura, basada en un esquema de codificación-decodificación, permite procesar imágenes manteniendo detalles importantes gracias a sus conexiones de salto entre las etapas de codificación y decodificación (ver *Figura 13)* [21].

La primera etapa de codificación se basa en reducir la dimensionalidad espacial mediante capas de pooling combinadas con convoluciones sucesivas, extrayendo características relevantes y reduciendo el tamaño de los datos. La segunda etapa de decodificación es completamente simétrica a la primera, aumentando la dimensionalidad de la imagen hasta llegar al tamaño original mediante la combinación de up-sampling y convoluciones transpuestas. Hay que destacar que las capas de la etapa de codificación están conectadas a sus respectivas capas de la etapa de decodificación mediante conexiones de salto, para mantener el contexto de la clasificación y así poder reducir la cantidad de información perdida durante el proceso de compresión de la imagen [21].

Aunque U-Net se definió inicialmente para segmentación binaria, puede adaptarse para segmentar múltiples clases, demostrando su capacidad para abordar problemas de segmentación semántica con un número mucho mayor de clases, como persona, cielo, edificio o coche. Gracias a esta capacidad de adaptación, U-Net puede aplicarse a una mayor variedad de escenarios y conjuntos de datos, incluyendo imágenes médicas o de tráfico. Esto se logra mediante una función de pérdida de entropía cruzada categórica, combinada con la activación softmax en la capa de salida, permitiendo al modelo clasificar cada píxel en una de las múltiples categorías posibles, ampliando su aplicabilidad a tareas más complejas.

Softmax es una función que convierte las activaciones de los canales de características en probabilidades por clase para cada píxel x. Esta función se define como [20]:

$$p_k(x) = \frac{\exp\left(a_k(x)\right)}{\sum_{k'=1}^{K} \exp\left(a_{k'}(x)\right)},$$

donde:

- $a_k(x)$ es la activación en el canal de características k en la posición x del mapa de características.
- K es el número total de clases.
- $p_k(x)$ es la probabilidad que el modelo asigna al píxel x perteneciente a la clase k.

La función softmax asegura que las probabilidades para cada píxel suman 1, y que la probabilidad $p_k(x)$ es mayor para los canales con activaciones $a_k(x)$ más altas, favoreciendo la clase más probable [20].

La función de pérdida de entropía cruzada es una función de energía E que mide el error entre las probabilidades predichas $p_k(x)$ y las etiquetas verdaderas de cada píxel $l(x)$. Se define como: [20]

$$E = \sum_{x \in \Omega} w(x) \log(p_{l(x)}(x)),$$

donde:

- Ω es el conjunto de píxeles de la imagen.
- $l(x)$ representa la etiqueta verdadera del píxel x, indicando la clase a la que pertenece.
- $w(x)$ es un peso que puede ajustarse para dar mayor importancia a ciertos píxeles, como los bordes entre objetos o clases infrarrepresentadas.
- $p_{l(x)}(x)$ es la probabilidad predicha para la clase verdadera del píxel x, calculada usando el softmax.

- $\log(p_{l(x)}(x))$ evalúa cuánto se desvía la predicción del modelo para el píxel x respecto a la etiqueta verdadera $l(x)$. Si $p_{l(x)}(x)$ es cercana a 1 (alta confianza en la predicción correcta), el logaritmo será cercano a 0 y la contribución al error será baja, mientras que si $p_{l(x)}(x)$ es baja (confianza en una predicción incorrecta), el logaritmo será negativo y la contribución al error será alta.

Para mejorar el desequilibrio de clases, la precisión en la segmentación de bordes y separar objetos tocantes, U-Net añade un peso adicional a la función de pérdida en $w(x)$ [20]:

$$w(x) = w_c(x) + w_0 * \exp\left(-\frac{(d_1(x)+ d_2(x))^2}{2\sigma^2}\right),$$

donde $w_c(x)$ equilibra las frecuencias de clases, $d_1(x)$ y $d_2(x)$ son las distancias al borde más cercano y al segundo más cercano, respectivamente; y w_0 y σ son constantes ajustables. Este ajuste ayuda a priorizar las áreas críticas de la imagen, como los bordes entre objetos [20].

3.3. Metodología propuesta

La metodología planteada a continuación describe los pasos detallados seguidos para implementar un modelo de segmentación semántica utilizando la arquitectura U-Net. Cada fase de este proceso ha sido diseñada con el objetivo de maximizar la precisión del modelo, mejorar la calidad de las imágenes generadas y asegurar la facilidad de uso para el usuario final.

- **Elección del conjunto de datos:** Selección de un conjunto de datos adecuado para segmentación semántica teniendo en cuenta una gran variedad de categorías de objetos presentes en el conjunto de datos y un gran número de datos para asegurar que el modelo pueda realizar un buen aprendizaje.

- **Visualización de los datos:** Visualización de las imágenes del conjunto de datos para entender su diversidad y complejidad y de las etiquetas de segmentación para comprender como están representadas las clases.
- **Adaptación del modelo:** Selección y adaptación de un modelo basado en la arquitectura U-Net que aprenda de los datos de entrada y sea capaz de predecir las salidas correctas.
- **Preprocesamiento de los datos:** Conversión de las imágenes y sus etiquetas a un formato adecuado para el modelo, asegurando que el modelo pueda procesar correctamente los datos sin comprometer la calidad de los resultados.
- **Entrenamiento del modelo:** Utilización del conjunto de datos preprocesado para entrenar el modelo adaptado, permitiendo que aprenda a partir de las características de los datos de entrada. Durante este proceso, el modelo ajusta sus parámetros internos mediante técnicas de optimización, con el objetivo de minimizar el error entre las predicciones y las salidas reales.
- **Evaluación del modelo con datos de validación:** Una vez que el modelo ha sido entrenado, se valida utilizando un conjunto de datos que no ha sido utilizado en el entrenamiento para evaluar su rendimiento y detectar posibles problemas de generalización o sobreajuste. Se comparan las predicciones del modelo con los datos reales mediante métricas específicas, asegurando que el aprendizaje sea efectivo y que las predicciones sean consistentes en datos nuevos.
- **Ajuste de hiperparámetros:** Ajuste de hiperparámetros del modelo en base a los resultados obtenidos en el conjunto de validación para evitar el sobreajuste y conseguir una mejora en las predicciones.
- **Evaluación final con datos de prueba:** Tras seleccionar los hiperparámetros óptimos, se realiza una evaluación final del modelo utilizando un conjunto de datos de prueba independiente. Esto asegura que el modelo final cumple con los objetivos deseados.
- **Eliminación de clases:** Implementación de la funcionalidad que permite al usuario seleccionar y eliminar categorías específicas de objetos dentro de una imagen.

- **Implementación de in-painting utilizando un modelo generativo:** Generación de nuevos objetos en las áreas eliminadas a partir de descripciones textuales proporcionadas por el usuario.

4. Trabajo realizado y resultados obtenidos

4.1. Elección y procesado de datos

En este apartado se describen las decisiones tomadas durante el proceso de selección y preparación del conjunto de datos que se utilizará para el entrenamiento y la validación del modelo de segmentación semántica.

4.1.1. Selección del conjunto de datos

El primer paso consistió en identificar los conjuntos de datos más relevantes y utilizados en tareas de segmentación semántica. Las opciones evaluadas fueron PASCAL VOC, ADE20K, NYU Depth y Cityscapes.

4.1.1.1. Criterios para la elección del conjunto de datos

Para garantizar que el conjunto de datos elegido fuera adecuado para el objetivo de segmentación semántica, se tuvieron en cuenta los siguientes criterios:

- **Conjunto de datos de imágenes:** El conjunto de datos debe contener imágenes reales, necesarias para entrenar y evaluar el modelo.
- **Enfoque de segmentación semántica:** El conjunto de datos debe proporcionar máscaras de etiquetas que asignen una clase a cada píxel, permitiendo segmentar correctamente diferentes objetos o regiones.
- **Calidad de las máscaras:** Las etiquetas deben ser precisas y detalladas para un entrenamiento robusto.
- **Variedad de escenarios:** Las imágenes deben representar una diversidad de escenarios y categorías, de modo que el modelo no se limite a un único tipo de entorno o contexto visual.
- **Tamaño adecuado:** El conjunto de datos debe contar con un volumen suficiente de imágenes y estar estructurado en particiones claras para entrenamiento, validación y prueba.

4.1.1.2. Conjuntos de imágenes analizados

4.1.1.2.1. PASCAL VOC

PASCAL VOC es un conjunto de datos clásico en tareas de segmentación semántica, con imágenes y máscaras bien definidas para múltiples clases. Sin embargo, su tamaño limitado y la menor variedad de escenarios lo hacen menos adecuado para este proyecto.

ADE20K

Las máscaras proporcionadas están orientadas principalmente a la segmentación de instancias, lo que lo hace más adecuado para otras tareas como la segmentación de objetos individuales.

4.1.1.2.2. NYU Depth

Este conjunto de datos incluye imágenes interiores con etiquetas detalladas para segmentación semántica. Aunque es útil, su enfoque en entornos interiores no se ajusta al objetivo de segmentación en escenarios urbanos.

4.1.1.2.3. Cityscapes

El conjunto de datos Cityscapes destaca por su enfoque específico en la segmentación semántica de imágenes urbanas. Este conjunto de datos cumple con los criterios definidos al presentar las siguientes características:

- **Estructura clara:** Cityscapes organiza sus datos en carpetas separadas, una para las imágenes reales (leftImg8bit) y otra para las máscaras de etiquetas (gtFine). Esto permite un acceso rápido y organizado a las imágenes y sus correspondientes máscaras.
- **División predefinida:** Las imágenes están previamente divididas en subconjuntos de entrenamiento, validación y prueba. Cada subconjunto contiene carpetas organizadas por ciudades, lo que facilita el manejo de datos y asegura que las imágenes de diferentes escenarios estén equilibradas en las particiones.

- **Variedad de escenarios:** Al estar basado en imágenes de entornos urbanos de diferentes ciudades, Cityscapes proporciona una gran variedad de escenas, lo que ayuda al modelo a generalizar mejor.
- **Máscaras de etiquetas de alta calidad:** Ofrece máscaras con categorías bien definidas, asignando un valor o clase a cada píxel, lo que garantiza un entrenamiento más preciso y alineado con los objetivos del proyecto.

Debido a que cumple con los criterios definidos, Cityscapes ha sido el conjunto de datos seleccionado.

4.1.2. Exploración de los datos

Antes de comenzar con el preprocesamiento de los datos se ha realizado una exploración sobre los mismos.

4.1.2.1. Características de los datos

El conjunto de datos Cityscapes incluye 5000 imágenes con anotaciones detalladas, organizadas en 35 clases, que a su vez se agrupan en 8 categorías principales:

Categoría	Clases
ground (suelo)	road (carretera), sidewalk (acera), parking (estacionamiento), rail track (vía férrea)
construction (construcción)	building (edificio), wall (muro), fence (valla), guard rail (barrera de seguridad), bridge (puente), tunnel (túnel)
object (objeto)	pole (poste), polegroup (grupo de postes), traffic light (semáforo), traffic sign (señal de tráfico)
nature (naturaleza)	vegetation (vegetación), terrain (terreno)

Categoría	Clases
sky (cielo)	sky (cielo)
human (humano)	person (persona que no conduce, mayoritariamente peatones), rider (conductor, mayoritariamente ciclistas y motoristas)
vehicle (vehículo)	car (coche), truck (camión), bus (autobús), caravan (caravana), trailer (remolque), train (tren), motorcycle (motocicleta), bicycle (bicicleta), license plate (placa de matrícula)
void (vacío)	unlabeled (sin etiquetar), ego vehicle (vehículo propio), rectification border (borde de rectificación), out of roi (fuera de la región de interés), static (estático), dynamic (dinámico), ground (suelo)

Tabla 1: Categorías y clases del conjunto de datos Cityscapes

Las imágenes fueron capturadas en 50 ciudades diferentes durante varios meses, cubriendo distintas condiciones de iluminación y horarios. Sin embargo, las condiciones climáticas se limitaron a buen tiempo para garantizar claridad en las capturas [39].

El conjunto de datos fue originalmente grabado como secuencias de video. De estos videos, los fotogramas fueron seleccionados manualmente para incluir [39]:

- Un gran número de objetos dinámicos, asegurando la presencia de vehículos en movimiento, personas y otros elementos cambiantes.
- Variabilidad en la disposición de la escena, priorizando imágenes con diferentes configuraciones urbanas para maximizar la diversidad de datos.
- Fondo variado, incluyendo en las imágenes distintas arquitecturas, vegetación y elementos urbanos para proporcionar un contexto visual grande y diverso.

4.1.3. Repositorios utilizados

En el desarrollo de este trabajo, se hizo uso de dos repositorios clave que fueron fundamentales para la implementación del modelo y el preprocesamiento de los datos:

- **Repositorio del modelo basado en U-Net:** Se utilizó como punto de partida para la implementación del modelo. Contiene un modelo de segmentación semántica multiclase con una arquitectura U-Net y un conjunto de imágenes de Cityscapes. Tanto la arquitectura como los datos fueron adaptados y ajustados para cumplir con los objetivos del trabajo. Está disponible públicamente en: https://github.com/hamdaan19/UNet-Multiclass/tree/main/scripts.
- **Repositorio del TFG:** El código desarrollado para este trabajo, que incluye todas las fases del proyecto (preprocesamiento, entrenamiento, visualización de resultados y generación de contenido), se encuentra almacenado en un repositorio público. Está disponible en: https://gitlab.com/UO265230/tfg-hpprintingsegmentation

4.1.4. Preprocesamiento de los datos

El preprocesamiento de datos en este trabajo se enfoca principalmente en la preparación de imágenes y máscaras de etiquetas para entrenar el modelo de segmentación semántica basado en U-Net.

4.1.4.1. Carga de los datos

Los datos se descomprimen desde un archivo ZIP subido a Drive y se organizan en carpetas según las imágenes originales (leftImg8bit) y sus etiquetas (gtFine). Esto asegura que cada imagen de entrada tenga su correspondiente máscara de etiquetas correctamente asociada, lo cual es crucial para el entrenamiento supervisado del modelo de segmentación semántica.

4.1.4.2. Mapeo de clases

Las máscaras de etiquetas originales, que contienen identificadores de clase (id), se convierten a un esquema de *trainId*, un conjunto de identificadores simplificados usados específicamente para el entrenamiento, mediante la función *encode_segmap*. Este mapeo reasigna los identificadores originales a un conjunto reducido de clases (trainId) para simplificar la complejidad del modelo y descartar clases no relevantes. Estas clases irrelevantes se marcan con el índice ignorado 255, que indica al modelo que las excluya durante el cálculo de la función de pérdida. Aunque el índice 255 no representa una clase específica, se suele asociar al color negro ((0, 0, 0)) como convención en las máscaras visualizadas. Este proceso asegura que las clases relevantes sean interpretadas correctamente durante el entrenamiento y la evaluación del modelo.

Después del mapeo, las clases finales utilizadas en el modelo de segmentación semántica son las siguientes:

trainId	Clase	Categoría	Color
0	Road (carretera)	Ground	(128, 64, 128)
1	Sidewalk (acera)	Ground	(244, 35, 232)
2	Building (edificio)	Construction	(70, 70, 70)
3	Wall (muro)	Construction	(102, 102, 156)
4	Fence (valla)	Construction	(190, 153, 153)
5	Pole (poste)	Object	(153, 153, 153)
6	Traffic light (semáforo)	Object	(250, 170, 30)

trainId	Clase	Categoría	Color
7	Traffic sign (señal de tráfico)	Object	(220, 220, 0)
8	Vegetation (vegetación)	Nature	(107, 142, 35)
9	Terrain (terreno)	Nature	(152, 251, 152)
10	Sky (cielo)	Sky	(70, 130, 180)
11	Person (persona que no conduce, mayoritariamente peatones)	Human	(220, 20, 60)
12	Rider (conductor, mayoritariamente ciclistas y motoristas)	Human	(255, 0, 0)
13	Car (coche)	Vehicle	(0, 0, 142)
14	Truck (camión)	Vehicle	(0, 0, 70)
15	Bus	Vehicle	(0, 60, 100)
16	Train (tren)	Vehicle	(0, 80, 100)
17	Motorcycle (motocicleta)	Vehicle	(0, 0, 230)
18	Bicycle (bicicleta)	Vehicle	(119, 11, 32)

Tabla 2: Clases finales utilizadas en el modelo tras el mapeo

El índice 255 se asigna a todas las clases no relevantes, como "license plate", "polegroup ", "parking", y otras incluidas en la categoría *void*.

La categoría *void* se emplea para etiquetar píxeles que no pertenecen a ninguna de las clases principales del conjunto de datos. Incluye [39]:

- **Elementos del fondo indistinguibles:** Partes visibles del vehículo propio (“ego vehicle”), montañas, farolas y la parte trasera de señales de tráfico.
- **Objetos en movimiento o temporales:** Contenedores de basura móviles, cochecitos, bolsas, sillas de ruedas y animales, que pueden no estar presentes en la misma ubicación en diferentes momentos.
- **Estructuras planas no clasificables:** Zonas compartidas entre coches y peatones, rotondas elevadas, islas de tráfico y superficies de agua.

Este esquema garantiza que el modelo se enfoque únicamente en las clases significativas para la tarea de segmentación, evitando el ruido generado por elementos ambiguos o irrelevantes.

4.1.4.3. Transformaciones de las imágenes

Para garantizar que las imágenes estén en un formato adecuado para el modelo se realizan las siguientes transformaciones:

- **Redimensionamiento:** Las imágenes y máscaras de etiquetas se ajustan a un tamaño fijo de 256x512 píxeles para uniformidad.
- **Conversión a tensores:** Las imágenes se convierten a tensores normalizados en un rango [0, 1]. La normalización escala los valores de píxeles entre 0 y 1, lo que facilita el entrenamiento al reducir la variabilidad entre las entradas.
- **Interpolación vecino más próximo:** Las máscaras de etiquetas se redimensionan utilizando interpolación de vecino más cercano para preservar la integridad de los valores categóricos. Este método es muy importante en el preprocesamiento de máscaras de etiquetas, ya que estas contienen valores

enteros que representan clases discretas (por ejemplo, 0 para carretera, 11 para persona, 13 para coche). Al utilizar la interpolación vecino más próximo, cada píxel en la máscara redimensionada toma el valor del píxel más cercano en la máscara original, asegurando que los valores categóricos permanezcan consistentes. Esto evita problemas que podrían surgir con otros métodos de interpolación, como el promedio de valores (bilineal o bicúbico), que introducirían valores no enteros. De esta forma, se garantiza que los datos de las etiquetas sean interpretados correctamente por el modelo, manteniendo su precisión y consistencia durante el entrenamiento y la evaluación.

4.1.4.4. Transformaciones de las máscaras de etiquetas

Las máscaras de etiquetas pasan por dos transformaciones específicas dentro de *label_transform*, un objeto de *transforms.Compose* que encapsula una secuencia de transformaciones dentro del pipeline de preprocesamiento:

- **Codificación (*encode_segmap*):** Asigna a cada píxel de las máscaras su trainId correspondiente según el diccionario id_to_trainid. El índice 255 asignado a las clases irrelevantes indica al modelo que ignore estos píxeles durante el cálculo de la función de pérdida (*CrossEntropyLoss*), lo que permite concentrarse en las clases relevantes.
- **Conversión a tensor:** Una vez que las máscaras de etiquetas son procesadas mediante la función *encode_segmap*, se convierten en tensores de tipo entero (*long*), que son compatibles con la función de pérdida (*CrossEntropyLoss*).

4.2. Desarrollo y entrenamiento

4.2.1. Modelo base

El modelo base seleccionado es una red neuronal basada en la arquitectura U-Net, descrita en el apartado 3.2. La implementación del modelo fue adaptada para procesar imágenes del conjunto de datos Cityscapes, en formato RGB y con unas dimensiones

espaciales de 256x512 píxeles, y generar como salida un mapa de probabilidad con 19 clases correspondientes a las categorías del conjunto de datos.

El modelo está organizado en dos partes principales: un codificador que reduce la dimensionalidad espacial para capturar características globales y un decodificador que la recupera, reconstruyendo la resolución de la imagen mientras integra detalles locales. Las conexiones de salto entre estas partes permiten preservar la información espacial detallada, asegurando la precisión de las predicciones.

El codificador emplea bloques de convoluciones con activaciones ReLU y operaciones de max-pooling, reduciendo la resolución de la imagen a medida que se capturan patrones más complejos. Por su parte, el decodificador utiliza convoluciones transpuestas para realizar upsampling, combinando esta información con las características extraídas por el codificador. Finalmente, una capa de convolución 1x1 genera un mapa de probabilidades por píxel, permitiendo asignar una clase a cada píxel.

La implementación en PyTorch permite estructurar el modelo de forma modular y flexible. Las capas se definieron utilizando torch.nn.Module, incluyendo convoluciones, normalización de batch para mejorar la estabilidad del entrenamiento y funciones auxiliares como *__double_conv* para definir bloques de convolución. El flujo de datos se organiza mediante el método *forward*, que integra el codificador, las conexiones de salto y el decodificador.

Para reducir el riesgo de sobreajuste durante el entrenamiento, se ha incorporado dropout con una tasa del 20% en los bloques de convolución. Esta técnica desactiva aleatoriamente algunas neuronas, estableciendo su valor en cero con una probabilidad $p = 0.2$ en cada paso de *forward*. La selección de neuronas ocurre de manera independiente en cada iteración, siguiendo una distribución de Bernoulli [40].

Al aplicar dropout, el modelo evita depender excesivamente de patrones específicos del conjunto de entrenamiento, lo que mejora su capacidad de generalización en datos no

vistos. Además, para mantener la magnitud esperada de las activaciones, las salidas se escalan por un factor de $\frac{1}{1-p}$ durante el entrenamiento [40].

En la fase de evaluación y prueba, dropout no se aplica, permitiendo que todas las unidades estén activas y el modelo utilice todos los valores de activación sin ninguna probabilidad de ser cero para realizar predicciones estables y deterministas [40].

Además, el modelo está optimizado para ejecutarse en GPU, aprovechando las capacidades de cálculo paralelo para acelerar los entrenamientos en grandes volúmenes de datos. Esta implementación básica, caracterizada por su simplicidad y flexibilidad, sirve como punto de partida para los ajustes y optimizaciones descritos en secciones posteriores.

4.2.2. Experimentación

4.2.2.1. Gestión de Checkpoints

Para facilitar la continuidad del entrenamiento, especialmente en sesiones interrumpidas en Google Colab, se implementó un sistema de guardado y carga de *checkpoints*. Cada checkpoint almacena el estado del modelo, el optimizador y las métricas, permitiendo reanudar el entrenamiento sin pérdida de progreso.

La función creada *save_checkpoint* guarda periódicamente el estado del modelo en tres situaciones clave: cuando la pérdida de validación alcanza un nuevo mínimo, cuando se activa la parada temprana o al completar el número total de épocas. Estos *checkpoints* se almacenan automáticamente en Google Drive para garantizar su disponibilidad en futuras sesiones.

Por otro lado, la función creada *load_model_checkpoint* permite recuperar el mejor modelo disponible, cargando tanto sus parámetros como los del optimizador para asegurar que el entrenamiento continúe en las mismas condiciones en las que se detuvo.

4.2.2.2. Parada temprana

La técnica de parada temprana (*early stopping*) se utiliza para prevenir el sobreajuste durante el entrenamiento del modelo. Esta técnica supervisa el rendimiento del modelo en el conjunto de validación y detiene el entrenamiento automáticamente si no se observa mejora después de un número predefinido de épocas consecutivas, denominado paciencia.

En este proyecto, la parada temprana se configuró con una paciencia de 7 épocas. Esto significa que, si la pérdida en el conjunto de validación no disminuye durante 7 épocas seguidas, el entrenamiento se detiene para evitar un sobreentrenamiento del modelo y ahorrar tiempo de computación. Este enfoque es especialmente útil en Google Colab, donde los recursos computacionales están limitados en tiempo y capacidad.

4.2.2.3. Elección de hiperparámetros

Los hiperparámetros de una red neuronal son valores que se definen antes de iniciar su entrenamiento y regulan el comportamiento del modelo durante el proceso de aprendizaje. Los hiperparámetros influyen directamente en cómo el modelo procesa los datos, ajusta sus pesos y optimiza su rendimiento. La selección inicial de los hiperparámetros suele realizarse mediante valores predeterminados o experimentales, pero posteriormente se ajustan de forma iterativa para mejorar los resultados y maximizar la precisión y la capacidad de generalización del modelo.

4.2.2.3.1. Definición de los hiperparámetros

A continuación, se definen los principales hiperparámetros utilizados:

- **Tamaño de batch:** Número de muestras procesadas en cada iteración del entrenamiento.
- **Número de épocas:** Cantidad de veces que el modelo recorrerá el conjunto de datos de entrenamiento.
- **Optimizador:** Algoritmo encargado de ajustar los pesos del modelo en cada iteración para minimizar la función de pérdida.

- **Tasa de aprendizaje:** Valor que regula la magnitud de los cambios en los pesos del modelo en cada actualización.
- **Función de pérdida:** Métrica utilizada para cuantificar el error entre las predicciones del modelo y los valores reales.
- **Número de capas:** Profundidad de la red neuronal, determinando la capacidad del modelo para extraer características en distintos niveles de abstracción.
- **Funciones de activación:** Transformaciones aplicadas a las salidas de las neuronas para introducir no linealidad en la red y permitir el aprendizaje de patrones complejos.

4.2.2.3.2. Configuración inicial

A continuación, se detallan los valores iniciales establecidos:

- **Tamaño de batch:** Se fijó en 16 debido a las limitaciones de memoria de la GPU utilizada y la necesidad de mantener un equilibrio entre velocidad y estabilidad durante el entrenamiento.
- **Número de épocas:** Se estableció un límite máximo de 100 épocas para permitir que el modelo aprenda las características del conjunto de datos.
- **Optimizador:** Se seleccionó Adam, conocido por su capacidad de adaptarse dinámicamente a diferentes tasas de aprendizaje, lo que facilita una convergencia más rápida y estable en problemas complejos como la segmentación semántica.
- **Tasa de aprendizaje:** Se estableció una tasa de aprendizaje inicial de 0.00005 para asegurar una convergencia estable del modelo.
- **Función de pérdida:** Se utilizó la función de entropía cruzada categórica. Esta función es adecuada para tareas de clasificación multiclase como la segmentación semántica, donde cada píxel pertenece a una clase específica.
- **Número de capas:** Se definieron cuatro niveles para el codificador y el decodificador, siguiendo la estructura típica de U-Net.

- **Funciones de activación:** Se seleccionó ReLU en las capas ocultas, por su efectividad en evitar el problema de gradientes desvanecidos y acelerar la convergencia.

Estos valores iniciales representan un punto de partida para los experimentos.

4.2.2.3.3. Configuraciones intermedias

Para evaluar si la configuración inicial era óptima, se probaron varias modificaciones en los hiperparámetros. A continuación, se detallan algunas de las configuraciones exploradas:

La primera configuración explorada:

- **Tasa de aprendizaje:** Se incrementó la tasa a 0.0001 para estudiar posibles mejoras en la estabilidad y velocidad de aprendizaje.

La segunda configuración explorada respecto a la configuración inicial:

- **Tasa de aprendizaje:** Se incrementó a 0.0005 con el objetivo de acelerar la convergencia y evaluar su impacto en la estabilidad del modelo.

La tercera configuración explorada respecto a la configuración inicial:

- **Tamaño de batch:** Se redujo a 8 para evaluar el impacto en la estabilidad del gradiente y la eficiencia computacional.
- **Tasa de aprendizaje:** Se incrementó a 0.001 para analizar su efecto en la velocidad de convergencia.

En todas las configuraciones exploradas, se mantuvieron constantes el número de épocas, el optimizador, la función de pérdida y la arquitectura del modelo. Esto se debe a que:

- No fue necesario aumentar el número de épocas, y con la implementación de la parada temprana tampoco hubo necesidad de reducirlo.

- El optimizador Adam ya ajusta automáticamente los pesos del modelo, por lo que no se consideraron cambios en este aspecto.
- La función de pérdida de entropía cruzada categórica es adecuada para la tarea de segmentación semántica.
- El número de capas de la arquitectura U-Net se mantuvo, ya que estaba diseñado para capturar tanto características de alto nivel como detalles locales.

Estos ajustes se realizaron con el objetivo de encontrar una configuración que maximizara la precisión del modelo y optimizara su capacidad de generalización. La comparación de estas configuraciones se detallará en una tabla posterior, donde se analizará el rendimiento de cada una y se justificará la selección final.

4.2.2.4. Cross-Entropy Loss

La entropía cruzada, también conocida como *logarithmic loss* o *log loss*, es una función de pérdida ampliamente utilizada en aprendizaje automático para evaluar el rendimiento de un modelo de clasificación. Proviene de la teoría de la información y se relaciona con el concepto de entropía de Shannon, que mide la incertidumbre o desorden en un sistema. La fórmula de la entropía de Shannon es [41]:

$$H = -\sum p(x) * \log p(x).$$

La entropía cruzada, por su parte, mide el número promedio de bits necesarios para identificar un evento de una distribución de probabilidad p utilizando el código óptimo de otra distribución q. En otras palabras, la entropía cruzada cuantifica la diferencia entre la distribución de probabilidad descubierta por el modelo y los valores reales. Durante el entrenamiento de un modelo, la función de pérdida de entropía cruzada sirve para encontrar la solución óptima ajustando los pesos del modelo. El objetivo es minimizar el error entre los resultados reales y los predichos, de tal manera que un valor bajo de entropía cruzada indica un mejor rendimiento del modelo. En términos prácticos, esta función guía al modelo para minimizar el error ajustando sus pesos, mejorando así su precisión. La fórmula de la entropía cruzada es [41]:

$H(p,q) = -\sum p(x) * \log q(x)$.

En el caso de llevar a cabo una tarea de clasificación binaria, se debe usar la entropía cruzada binaria, que es el promedio de la entropía cruzada en todas las muestras de datos. Su fórmula es: [41]

$L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j + \log\left(p_j\right) + \left(1 - t_j\right) * \log(1 - p_j)\right]\right]$,

donde:

- L es la pérdida promedio para N muestras.
- N es el número total de muestras en el conjunto de datos.
- t_j es la etiqueta real para la muestra j, que toma valores en {0,1}.
    - $t_j = 1$: La muestra pertenece a la clase positiva.
    - $t_j = 0$: La muestra pertenece a la clase negativa.
- p_j es la probabilidad predicha por el modelo para que la muestra j pertenezca a la clase positiva ($t_j = 1$). p_j es el valor predicho por la función sigmoide y siempre está en el rango [0,1].

Por el contrario, si se calcula la pérdida de un solo punto de datos donde el valor correcto es y=1, la fórmula sería [41]:

$l = -(1 * \log p) + (1 - 1) * \log(1 - p)$,
$l = -(1 * \log p)$,

donde muestra que el valor de pérdida, l, dependerá de la probabilidad predicha, p. Si el valor de p es alto, el modelo será recompensado con un valor bajo de l. En cambio, una probabilidad p baja implicará un mayor valor de l, reflejando que el modelo fue incorrecto en su predicción [41].

Para una tarea de clasificación multiclase, la entropía cruzada (o entropía cruzada categórica) para una sola instancia se extiende fácilmente, calculando la pérdida para cada clase por separado y sumando para obtener la pérdida total. La fórmula es [41]:

$-\sum_{c=1}^{N} y_c * \log p_c,$

donde:

- N es el número total de clases.
- y_c es el valor real (etiqueta) para la clase c.
    - Es un valor binario: $y_c = 1$ si la muestra pertenece a la clase c; $y_c = 0$ en caso contrario.
    - Solo una clase tiene $y_c = 1$ para cada muestra (codificación *one-hot*).
- p_c es la probabilidad predicha por el modelo para la clase c.

En este trabajo se utiliza la entropía cruzada categórica, que es ideal para problemas de clasificación multiclase. En PyTorch, esta función de pérdida ya incluye internamente log-softmax y negative log likelihood loss (NLLLoss), por lo que no es necesario aplicar softmax manualmente antes de calcular la pérdida, se usa torch.max(outputs, 1) para seleccionar directamente la clase con mayor activación, lo que simplifica el proceso. Log-softmax convierte las activaciones en valores logarítmicos normalizados, estabilizando los cálculos y evitando problemas numéricos con exponentes grandes, mientras que NLLLoss penaliza la probabilidad predicha de la clase correcta si es baja. Como resultado, la entropía cruzada categórica optimiza directamente el modelo sin necesidad de aplicar transformaciones adicionales sobre la salida.

Gracias a estas propiedades, la entropía cruzada ayuda al modelo a minimizar el error entre los valores reales y los predichos, mejorando su precisión en la clasificación y guiando el ajuste de parámetros en el proceso de optimización. Aplicada al modelo U-Net, la entropía cruzada permite aprovechar la arquitectura simétrica del modelo, con

su conexión de salto entre capas de codificación y decodificación, manteniendo un buen contexto espacial. Este enfoque es particularmente beneficioso para la segmentación precisa de imágenes, ya que reduce la pérdida de información durante el proceso de compresión y mejora la capacidad del modelo para distinguir entre múltiples clases de manera efectiva.

4.2.2.5. Métricas

En este trabajo, se han seleccionado cuatro métricas para evaluar el rendimiento del modelo de segmentación semántica: la intersección sobre unión, el coeficiente Dice, la precisión y la sensibilidad. Estas métricas son ampliamente utilizadas en tareas de segmentación a nivel de píxel y proporcionan una visión integral de la precisión y consistencia del modelo en la clasificación de múltiples categorías.

En todas las métricas calculadas, se ignora explícitamente el índice 255, que corresponde a las clases no relevantes o descartadas tras el mapeo de las máscaras de etiquetas (clase *void* en el conjunto de datos Cityscapes). Este índice se asigna a píxeles que no pertenecen a las clases relevantes para la tarea de segmentación y, por tanto, no deben influir en la evaluación del modelo. La exclusión de estos píxeles asegura que las métricas reflejen únicamente el rendimiento en las clases significativas. Este mismo tratamiento se aplica también en la función de pérdida, donde el índice 255 es ignorado durante la optimización del modelo.

4.2.2.5.1. Intersección sobre unión

El coeficiente de intersección sobre unión (IoU, por sus siglas en inglés), o índice Jaccard, mide la superposición entre el área predicha por el modelo y el área real de una categoría específica, normalizada por la unión de ambas áreas (ver *Figura 15)*. Su fórmula es [42]:

$$IoU = \frac{TP}{TP+FP+FN},$$

donde:

- TP son los verdaderos positivos.
- FP son los falsos positivos.
- FN son los falsos negativos.

Un valor de IoU más alto indica que la predicción del modelo está más alineada con las etiquetas reales [42].

Figura 15: Representación gráfica del cálculo de IoU para una clasificación a nivel de píxel **[43]**

4.2.2.5.2. Coeficiente Dice

El coeficiente Dice es una métrica que evalúa la similitud entre dos conjuntos, con un mayor énfasis en los verdaderos positivos. Se calcula mediante la siguiente fórmula [44]:

$$Dice = \frac{2*TP}{2*TP+FP+FN}.$$

En comparación con la métrica IoU, el coeficiente Dice penaliza menos los errores de sub- y sobre-segmentación [45]. La métrica IoU aplica una penalización mayor cuando hay áreas mal segmentadas, mientras que el coeficiente Dice tiende a favorecer la detección de regiones coincidentes.

4.2.2.5.3. Precisión

La precisión mide la proporción de predicciones positivas que son correctas, enfocándose en los verdaderos positivos con respecto a los falsos positivos. Su fórmula es [44]:

$Precision = \frac{TP}{TP+FP}$.

Esta métrica es útil para evaluar cuán confiable es el modelo al etiquetar un píxel como positivo.

4.2.2.5.4. Sensibilidad

La sensibilidad, también conocida como recall, mide la proporción de elementos positivos correctamente identificados con respecto a los que realmente son positivos. Su fórmula es [44]:

$Recall = \frac{TP}{TP+FN}$.

4.2.3. Preparación y generación de contenido visual

4.2.3.1. Eliminación de áreas

En esta sección se describe el proceso mediante el cual se identifican y eliminan áreas específicas de la imagen, utilizando el modelo de segmentación semántica desarrollado previamente y un sistema interactivo que permite al usuario seleccionar clases a eliminar (ver *Figura 16*).

1. **Generación de máscaras de segmentación:** El modelo U-Net genera máscaras de segmentación semántica, asignando una clase a cada píxel de la imagen. Estas máscaras permiten identificar las áreas correspondientes a cada clase, como “coche”, “edificio” o “persona”.
2. **Selección de clases con interfaz interactiva:** Se implementó un widget interactivo utilizando la librería ipywidgets. Este widget presenta una lista de clases disponibles (correspondientes a las clases de Cityscapes), y permite al usuario ingresar el identificador de la clase (trainId) que desea eliminar. Tras la selección, el sistema actualiza la máscara de predicción, asignando un valor especial (255) a los píxeles de la clase seleccionada. Este valor indica que esos píxeles serán considerados como áreas eliminadas en pasos posteriores. En

caso de que el trainId seleccionado no exista en la imagen o que ya haya sido eliminado previamente, aparecerá un mensaje con el texto “trainId no encontrado en la imagen. Prueba otro”.

3. **Actualización visual de la imagen:** Una vez eliminadas las áreas seleccionadas, la imagen original se actualiza para reflejar los cambios. Esto incluye cambiar el color de las áreas eliminadas a blanco, indicando que esas regiones están listas para ser rellenadas, y mostrar una versión actualizada de la máscara con las áreas eliminadas resaltadas.
4. **Exportación de las máscaras y datos actualizados:** La máscara final, junto con la imagen modificada, se guarda en el sistema de archivos. Esto asegura que las áreas eliminadas estén listas para ser procesadas por los modelos generativos en el siguiente paso. Si se desea eliminar más clases, el usuario puede repetir el paso 2, seleccionando nuevos identificadores de clase (trainId). El sistema actualizará la máscara y la imagen modificada, reflejando las eliminaciones acumuladas. Finalmente, la nueva máscara y la imagen se guardan, sobrescribiendo los archivos anteriores para mantener consistencia.

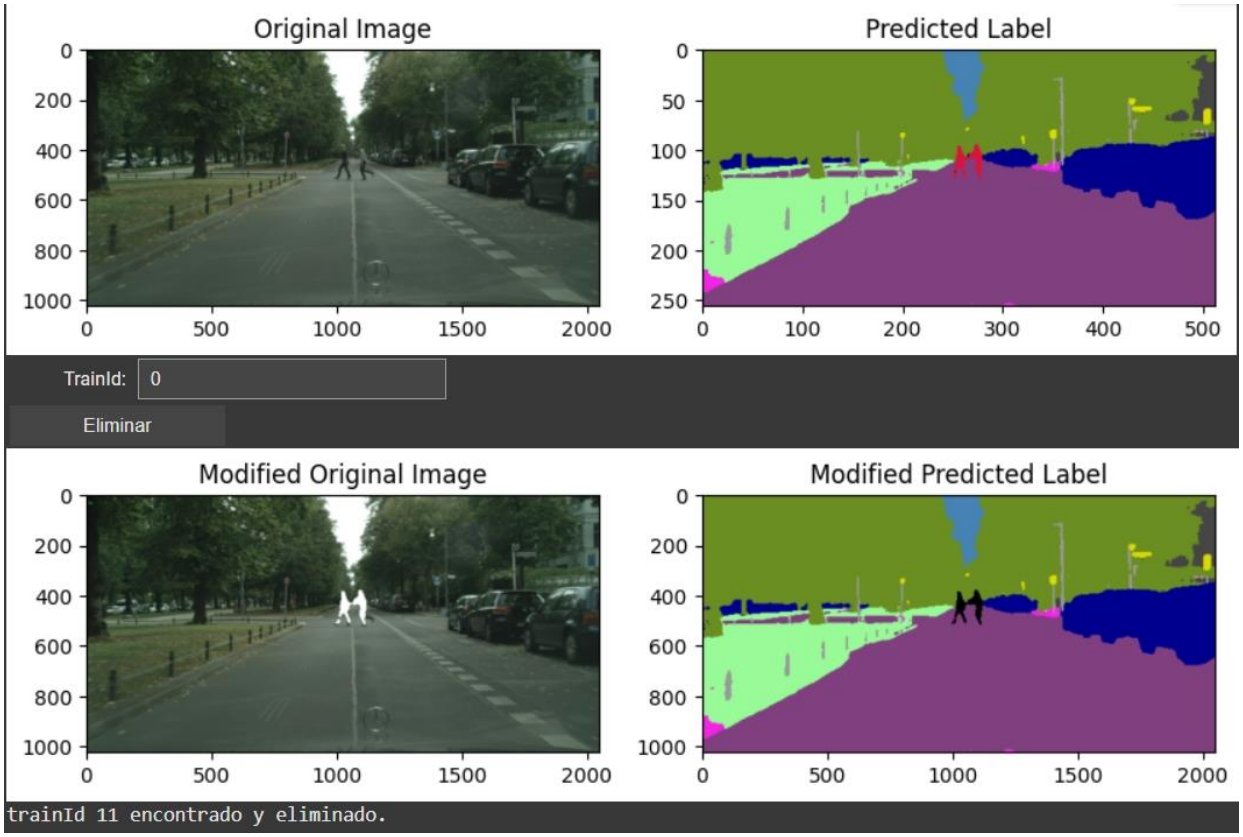

Figura 16: Eliminación de la clase persona. Arriba izquierda (Original Image): Imagen original antes de realizar cualquier modificación. Arriba derecha (Predicted Label): Etiqueta predicha por el modelo, donde cada color representa una clase diferente en la imagen. Abajo izquierda (Modified Original Image): Imagen original con la clase seleccionada (en este caso, la clase con trainId = 0) eliminada, y con los píxeles correspondientes reemplazados por un área blanca. Abajo derecha (Modified Predicted Label): Etiqueta predicha por el modelo, modificada tras la eliminación de la clase seleccionada, donde los píxeles de esa clase ahora aparecen en negro, indicando su ausencia.

4.2.3.2. Modelo generativo para In-Painting

Selección y objetivos

Con el objetivo de realizar las tareas de in-painting, se seleccionaron dos modelos generativos, Stable Diffusion y GLIDE, explicados con más detalle en la Sección 2.3.

El objetivo principal fue analizar las diferencias en términos de calidad visual, coherencia contextual y flexibilidad de cada modelo en tareas de in-painting, empleando el mismo conjunto de máscaras y prompts.

4.2.3.2.1. Flujo de trabajo del in-painting

El procedimiento implementado para realizar las tareas de in-painting es el siguiente:

1. **Predicción inicial y creación de máscaras:** El modelo U-Net genera las máscaras de segmentación que permiten identificar las áreas a modificar. Estas máscaras se preprocesan para ajustarse a los formatos requeridos por los modelos generativos. En particular, se redimensionan ajustando las máscaras a resoluciones específicas, y se normalizan convirtiendo las máscaras a valores binarios, donde 0 representa las áreas modificables y 255 las áreas a conservar.
2. **Definición del prompt textual:** Se utilizan descripciones textuales idénticas para ambos modelos, permitiendo una comparación directa entre los resultados generados por cada modelo.
3. **Generación de contenido con Stable Diffusion:** Se procesa la máscara correspondiente a la imagen original y el prompt para generar contenido inicial. La generación se realiza a la misma resolución que la imagen original.
4. **Generación de contenido con GLIDE:** Se procesa la misma máscara correspondiente a la imagen original y el mismo prompt, generando la imagen de resolución 256x256.
5. **Evaluación visual:** Las imágenes generadas por ambos modelos fueron inspeccionadas manualmente para comparar la capacidad de mantener la naturalidad en las áreas generadas, la precisión en la representación de los elementos especificados en el prompt y cómo las áreas generadas se combinan con el resto de la imagen.

4.2.3.2.2. Detalles técnicos

Un *pipeline* es una abstracción que organiza los pasos necesarios para procesar datos de entrada, realizar inferencias y generar resultados. En el contexto de modelos generativos como Stable Diffusion y GLIDE, un pipeline coordina la interacción entre los

componentes del modelo, como la entrada del usuario, procesamiento del modelo o la salida final.

Ambos modelos utilizan pipelines personalizados para realizar la tarea de in-painting. En Stable Diffusion, el pipeline se implementó utilizando la librería Diffusers de Hugging Face, una herramienta robusta que facilita la integración y personalización de modelos generativos [46]. El pipeline de Stable Diffusion incluye los siguientes componentes:

- **Carga del modelo preentrenado:** Desde el repositorio runwayml/stable-diffusion-inpainting.
- **Entrada del usuario:** Mediante una imagen base donde se realizarán las modificaciones, su máscara asociada que indica las áreas modificables y un prompt textual que describe el contenido deseado para las áreas a modificar.
- **Difusión latente:** Utilizada para transformar las áreas eliminadas en contenido coherente y realista. La generación inicial ocurre a resoluciones más bajas con el objetivo de generar contenido rápido y eficiente.
- **Salida generada:** La imagen procesada contiene áreas rellenadas de acuerdo con el prompt y el contexto visual de la imagen base.

Además, se utiliza float16 en GPU para optimizar memoria y velocidad, y se generó un token de acceso en Hugging Face para integrar el modelo de manera segura. Este token se almacenó en la sección “Secretos” de Google Colab, garantizando su privacidad.

En GLIDE, el pipeline se implementó utilizando el repositorio oficial de OpenAI [47], con configuraciones diseñadas para tareas de in-painting de alta resolución. El pipeline de GLIDE incluye los siguientes componentes:

- **Carga del modelo base:** Se inicializa en dos etapas, un modelo base para generación inicial y un modelo *upsampler* para mejorar la resolución de las imágenes. Ambos modelos se cargan desde puntos de control preentrenados (*base-inpaint* y *upsample-inpaint*).

- **Entrada del usuario:** Mediante una imagen base donde se realizarán las modificaciones, su máscara asociada que indica las áreas modificables redimensionada a resoluciones específicas (64x64 para el modelo base, 256x256 para el upsampler) y un prompt textual que describe el contenido deseado para las áreas a modificar.
- **Inferencia en dos etapas:** En la etapa 1, el modelo base genera una versión inicial de las áreas modificadas a una resolución de 64x64 píxeles. En la etapa 2, el upsampler toma la salida del modelo base y la refina, aumentando la resolución a 256x256 píxeles y mejorando los detalles visuales.
- **Salida generada:** La salida del upsampler se presenta como la imagen final procesada, con un equilibrio entre coherencia visual y detalles.

Además, también se utiliza float16 en GPU para optimizar memoria y velocidad.

4.3. Resultados obtenidos

4.3.1. Resultados del entrenamiento y validación

Una vez evaluados los resultados obtenidos con distintas configuraciones, se seleccionó la configuración que obtuvo la menor pérdida en validación. La configuración final elegida es:

- Tamaño de batch: 16
- Tasa de aprendizaje: 0.0005

A continuación, las Tablas 3 y 4 muestran los valores de la función de pérdida obtenida con los conjuntos de entrenamiento y validación, respectivamente, para cada configuración probada.

Tamaño de batch	Tasa de aprendizaje	Pérdida en entrenamiento
16	0.00005	0.15568
16	0.0001	0.19224
16	0.0005	0.17563
8	0.001	0.12935

Tabla 3: Comparación de la función de pérdida en el conjunto de entrenamiento

Tamaño de batch	Tasa de aprendizaje	Pérdida en validación
16	0.00005	0.37157
16	0.0001	0.31908
16	0.0005	0.26017
8	0.001	0.27650

Tabla 4: Comparación de la función de pérdida en el conjunto de validación

El modelo U-Net fue entrenado durante 35 épocas en el conjunto de datos Cityscapes utilizando la configuración mencionada previamente, produciéndose la activación de la parada temprana y no alcanzando el máximo de 100 épocas. Durante el proceso, se monitorizaron cinco indicadores clave para evaluar su rendimiento: la función de pérdida (cross-entropy loss), la métrica IoU, el coeficiente Dice, la precisión y la sensibilidad. A continuación, se describen los resultados basados en los gráficos obtenidos:

- **Función de pérdida (*loss function*):** La gráfica muestra una disminución constante de la pérdida tanto en el conjunto de entrenamiento como en el de validación, indicando que el modelo aprende de manera efectiva sin signos evidentes de sobreajuste. Al final del entrenamiento, los valores de la pérdida se estabilizan entre 0.14 y 0.15 para el entrenamiento, y entre 0.26 y 0.27 para la validación (ver *Figura 17*).

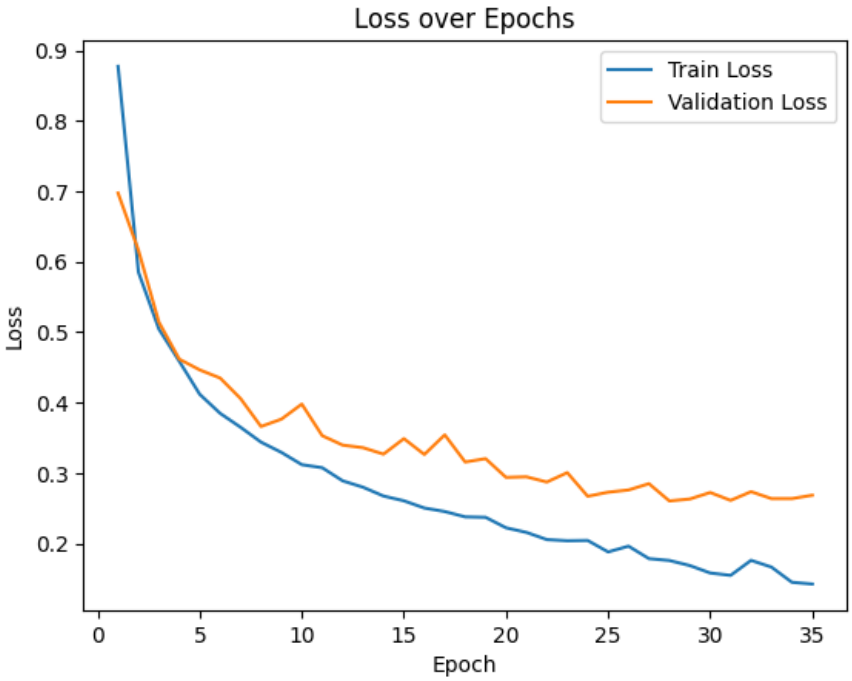

Figura 17: Evolución de la función de pérdida utilizando la configuración inicial de hiperparámetros (tamaño de batch de 16, una tasa de aprendizaje de 0.0005)

- **Intersección sobre unión (IoU):** La métrica confirma el buen rendimiento del modelo. El valor de IoU promedio se incrementa rápidamente durante las primeras épocas y luego se estabiliza en valores cercanos a 0.92 para el entrenamiento y 0.8 para la validación, lo que indica una adecuada generalización del modelo (ver *Figura 18*).

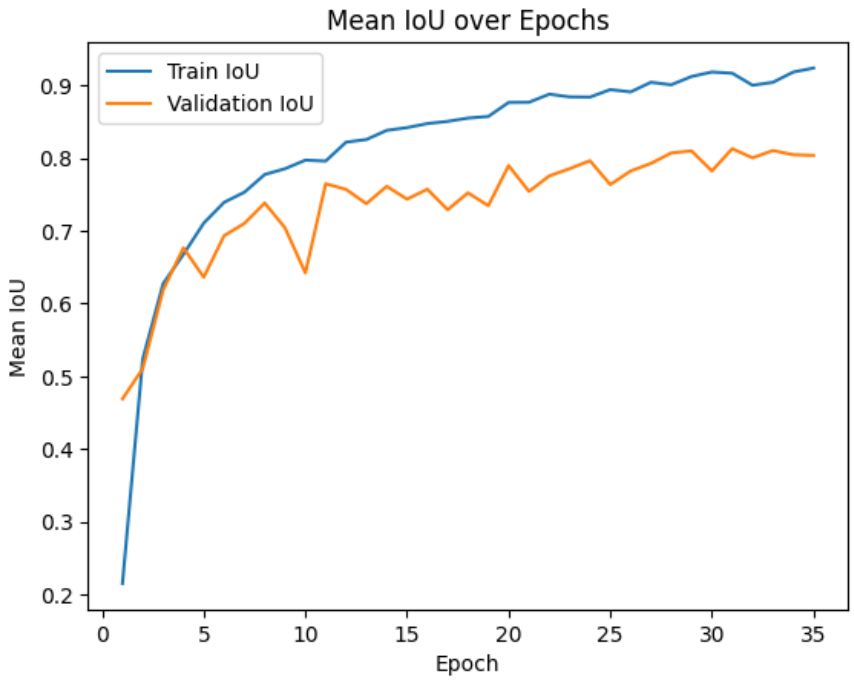

Figura 18: Evolución de la intersección sobre la unión (IoU) utilizando la configuración inicial de hiperparámetros (tamaño de batch de 16, una tasa de aprendizaje de 0.0005)

- **Coeficiente Dice:** Este coeficiente, que muestra la similitud entre las predicciones y las etiquetas reales, sigue una tendencia muy similar a la IoU. Al final del entrenamiento, alcanza valores estables alrededor de 0.96 en el conjunto de entrenamiento y 0.89 en validación, lo que sugiere una buena generalización (ver *Figura 19*).

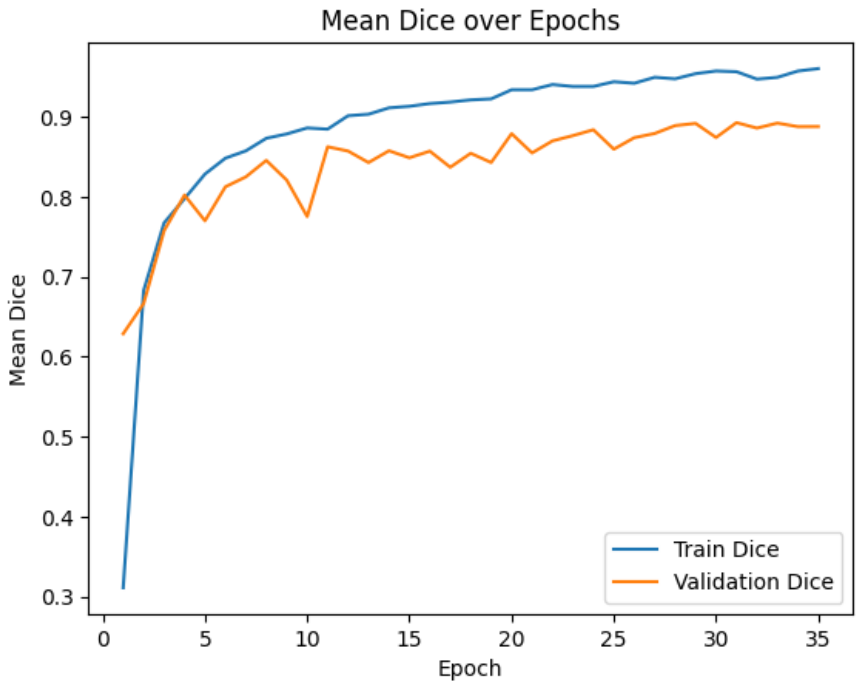

Figura 19: Evolución del coeficiente Dice utilizando la configuración inicial de hiperparámetros (tamaño de batch de 16, una tasa de aprendizaje de 0.0005)

- **Precisión:** Este coeficiente evalúa la proporción de predicciones correctas entre todas las predicciones positivas realizadas por el modelo. Durante el entrenamiento, se comporta de manera consistente, incrementándose gradualmente y estabilizándose en valores cercanos a 0.95. Sin embargo, en el conjunto de validación, la precisión muestra oscilaciones significativas, sin llegar a estabilizarse del todo. Esto podría indicar una variabilidad en cómo el modelo generaliza para diferentes clases o escenarios del conjunto de validación (ver *Figura 20*).

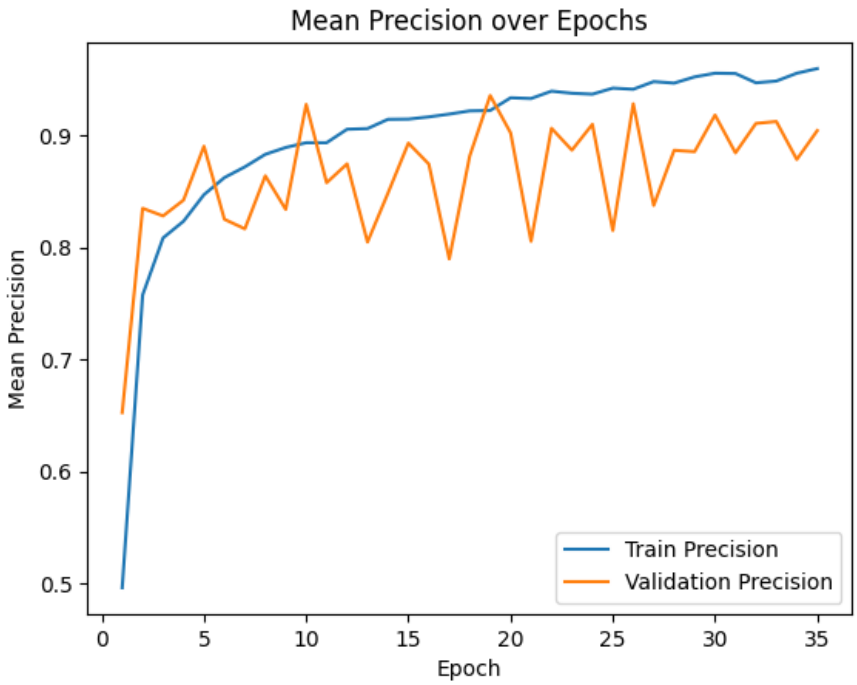

Figura 20: Evolución de la precisión utilizando la configuración inicial de hiperparámetros (tamaño de batch de 16, una tasa de aprendizaje de 0.0005)

- **Sensibilidad (*recall*):** Este coeficiente mide qué proporción de los elementos positivos reales son correctamente identificados por el modelo. Al igual que la precisión, la sensibilidad es consistente durante el entrenamiento, alcanzando valores cercanos a 0.95 hacia el final del entrenamiento. Sin embargo, en la validación, la sensibilidad también oscila significativamente, alcanzando valores promedio entre 0.88 y 0.90 al final del entrenamiento. Estas oscilaciones sugieren que el modelo podría estar enfrentándose a desafíos al identificar correctamente los elementos positivos en datos más variados o complejos del conjunto de validación (ver *Figura 21*).

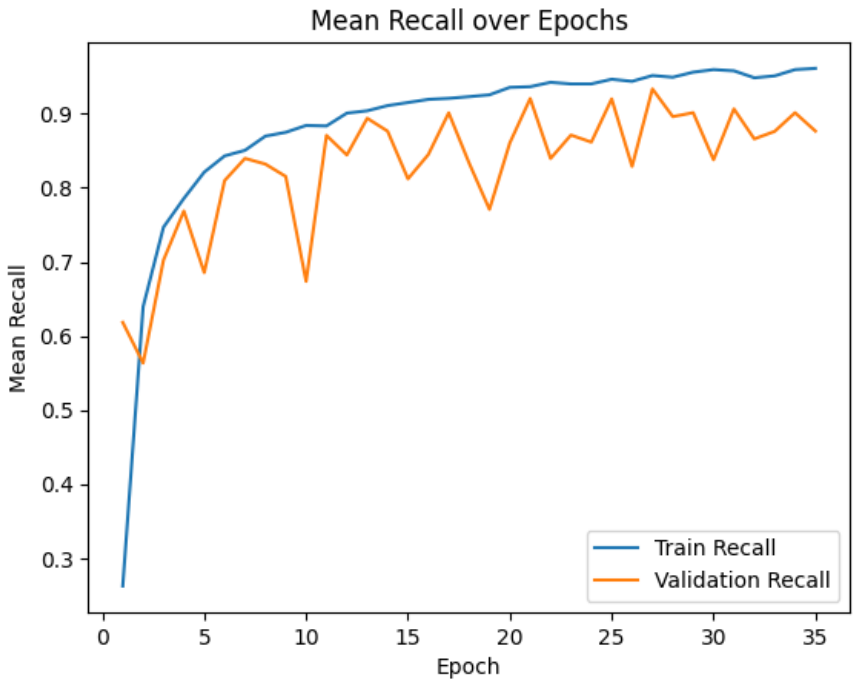

Figura 21: Evolución de la de sensibilidad (recall) utilizando la configuración inicial de hiperparámetros (tamaño de batch de 16, una tasa de aprendizaje de 0.0005)

Para complementar el análisis gráfico de las métricas durante el entrenamiento y validación, las Tablas 5 y 6 muestran los valores finales (promedio de la última época) de las métricas IoU, coeficiente Dice, precisión y sensibilidad. Estas tablas también comparan diferentes configuraciones del modelo evaluadas en el conjunto de entrenamiento y de validación, respectivamente, dado que no se dispone de etiquetas reales en el conjunto de prueba.

Tamaño de batch	Tasa de aprendizaje	IoU	Dice	Precisión	Sensibilidad
16	0.00005	0.92209	0.95934	0.95742	0.96178
16	0.0001	0.88752	0.94002	0.93878	0.94269
16	0.0005	0.90056	0.94737	0.94642	0.94941
8	0.001	0.92749	0.96214	0.96316	0.96171

Tabla 5: Comparación de métricas clave al final del entrenamiento para diferentes configuraciones del modelo en el conjunto de entrenamiento

Tamaño de batch	Tasa de aprendizaje	IoU	Dice	Precisión	Sensibilidad
16	0.00005	0.76365	0.86189	0.88078	0.84623
16	0.0001	0.76180	0.86013	0.90006	0.83002
16	0.0005	0.80685	0.88889	0.88649	0.89620
8	0.001	0.80883	0.88932	0.88161	0.90492

Tabla 6: Comparación de métricas clave al final del entrenamiento para diferentes configuraciones del modelo en el conjunto de validación

4.3.2. Resultados cualitativos en los conjuntos de validación y test

Para evaluar la capacidad del modelo de segmentación semántica, también se analizaron los resultados cualitativos generados en los conjuntos de validación y test del conjunto de datos Cityscapes. Para facilitar la interpretación de los resultados generados por el modelo, las máscaras de predicción (que están en formato de índices trainId) se transforman en imágenes coloreadas mediante la función *decode_segmap*. Esta función asigna un color único a cada clase según el diccionario *trainid_to_color*, permitiendo visualizar fácilmente las regiones segmentadas.

4.3.2.1. Conjunto de validación

A continuación, se presenta un análisis cualitativo del desempeño del modelo en el conjunto de validación, basado en la comparación entre las etiquetas reales y los resultados obtenidos a través de las predicciones.

El modelo muestra una alta precisión en la segmentación de clases predominantes, como carretera, edificio, cielo, vegetación y coche, así como en objetos grandes (ver *Figura 22*). Sin embargo, incluso en estos casos, se observan imprecisiones en los bordes de las clases, por ejemplo, el límite entre carretera y acera (ver *Figura 22, subfigura 6;* y *Figura 23, subfiguras 3* y *4*).

Por otro lado, el modelo presenta una mayor confusión entre objetos pequeños, como la clase poste, que suelen ser omitida o mal segmentada (ver *Figura 22*, *subfigura 3,* y *Figura 23, subfiguras 3* y *5*). También da lugar a predicciones incorrectas al intentar diferenciar entre clases menos comunes, como muro o valla (ver *Figura 22*, *subfiguras 2* y *4*). En el caso de la clase autobús, está frecuentemente mal segmentada (ver *Figura 2*3, *subfiguras 1* y *2*), al igual que la clase moto (ver *Figura 23, subfigura 2*). Un problema adicional se observa en la segmentación de la clase conductor (ciclistas o motoristas), que en muchas ocasiones es confundida con la clase persona debido a las similitudes visuales entre ambas clases en las imágenes (ver *Figura* 23, *subfigura 2*), al igual que ocurre con las clases vegetación y terreno (ver *Figura* 23*, subfigura 6*).

Por último, las variaciones en iluminación dentro de las imágenes, como las causadas por sombras, afectan negativamente el rendimiento del modelo. Estas condiciones provocan errores en la identificación de clases, y en algunos casos, una misma clase es segmentada de manera inconsistente dependiendo de si se encuentra en áreas iluminadas o sombreadas (ver *Figura* 23, *subfigura 5*).

En general, el modelo demuestra un buen rendimiento en la segmentación de clases principales y objetos grandes, pero destaca la necesidad de mejoras en la identificación de objetos pequeños, clases similares y en escenarios con condiciones de iluminación desiguales. Estas limitaciones subrayan áreas clave de mejora para alcanzar un nivel

de precisión más robusto en situaciones complejas y en la generalización a entornos más diversos.

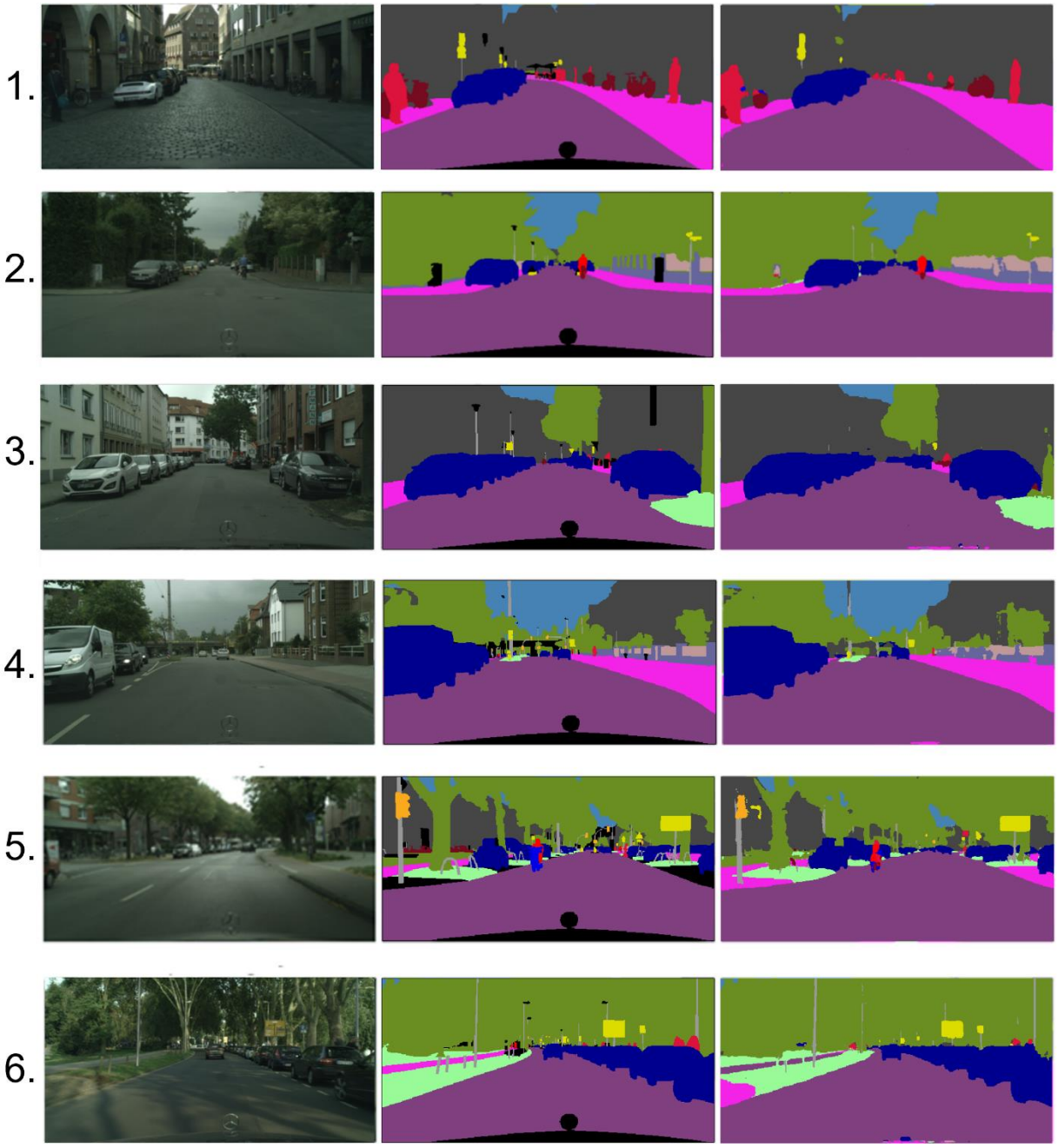

Figura 22: Resultados en el conjunto de validación con una buena identificación de clases. Cada fila está compuesta por tres imágenes, de izquierda a derecha: imagen original, etiqueta real, y etiqueta predicha, cada una con su correspondiente subfigura. Configuración de hiperparámetros: tamaño de batch 16, tasa de aprendizaje 0.0005

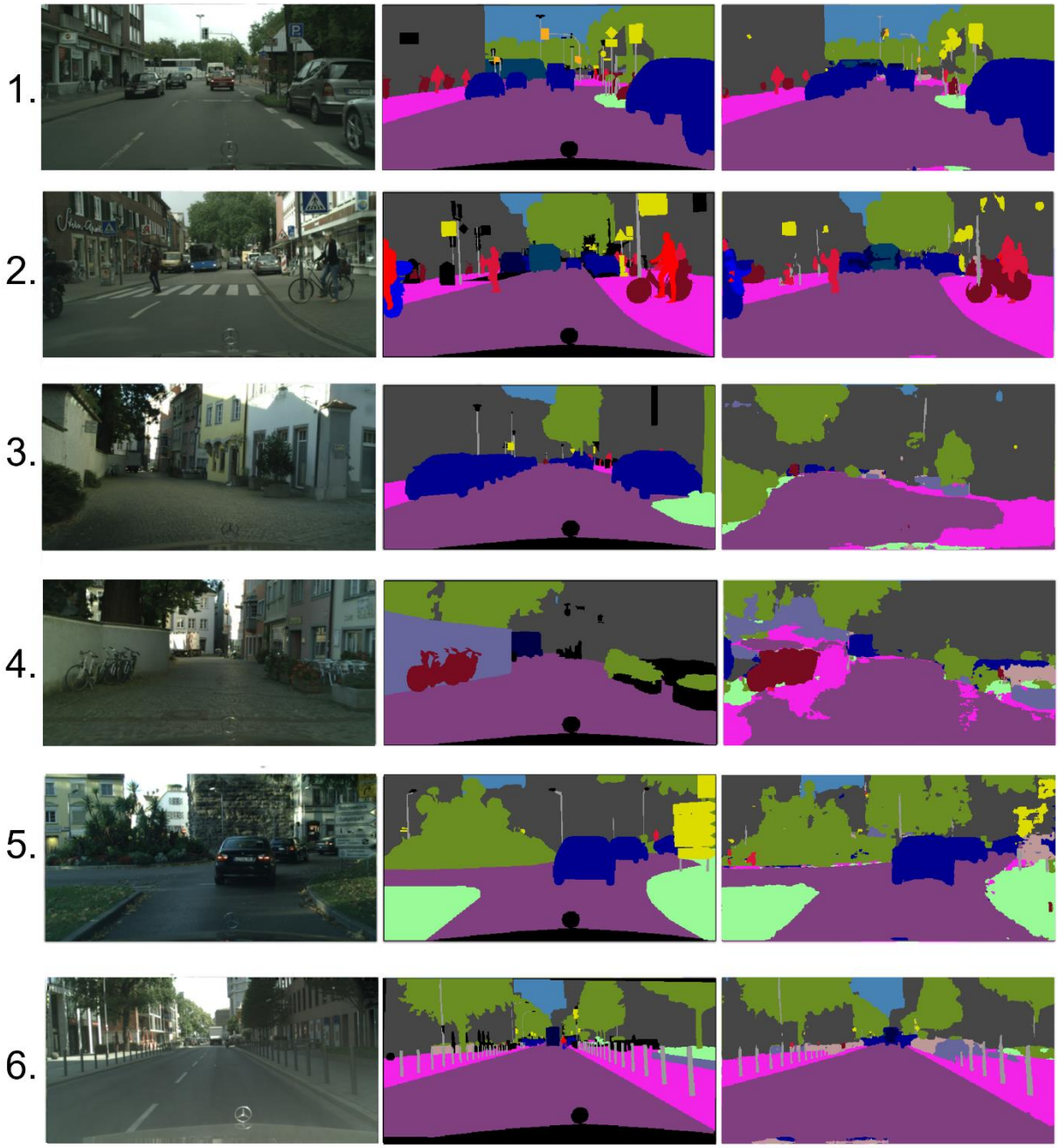

Figura 23: Resultados en el conjunto de validación con una mala identificación de clases. Cada fila está compuesta por tres imágenes, de izquierda a derecha: imagen original, etiqueta real, y etiqueta predicha, cada una con su correspondiente subfigura. Configuración de hiperparámetros: tamaño de batch 16, tasa de aprendizaje 0.0005

4.3.2.2. Conjunto de prueba

A continuación, se presenta un análisis cualitativo del desempeño del modelo en el conjunto de prueba, evaluando su capacidad de predicción en imágenes completamente nuevas y no etiquetadas. Este análisis se basa en la calidad aparente de las segmentaciones realizadas, dado que no se dispone de etiquetas reales.

El análisis visual de los resultados de las predicciones generadas destaca por su coherencia visual, especialmente en áreas con estructuras claras como edificios, coches, cielos y carreteras. Aunque no se dispone de etiquetas reales para comparación directa, los resultados predichos muestran una segmentación semántica consistente con los patrones observados durante el entrenamiento (ver *Figura 24*). Incluso parece que se identifican bien clases que eran identificadas erróneamente, por ejemplo, la clase autobús se predice sin tantas confusiones con otras clases (ver *Figura 24*, *subfigura 6*) o la clase moto (ver *Figura 24, subfigura 3*). De manera similar, la clase muro, que presentó dificultades en validación, se identifica correctamente (ver *Figura* 24, *subfigura 5*).

Sin embargo, se identifican ciertas limitaciones recurrentes, como los errores en la segmentación de bordes finos y objetos pequeños, incluyendo postes y señales de tráfico. Estas imprecisiones son más evidentes en escenarios con alta densidad de objetos menores, donde el modelo tiende a fusionar o confundir clases similares. Además, se observa que los bordes entre clases, como entre carretera y acera, sigue presentando inconsistencias (ver *Figura 25, subfigura 5)*, lo que puede deberse a la dificultad de segmentar áreas con transiciones suaves. También persisten errores en la distinción entre la clase conductor y persona (ver *Figura 25, subfiguras 2* y *6*), lo que sugiere que el modelo aún enfrenta dificultades para diferenciar estos elementos en ciertos contextos.

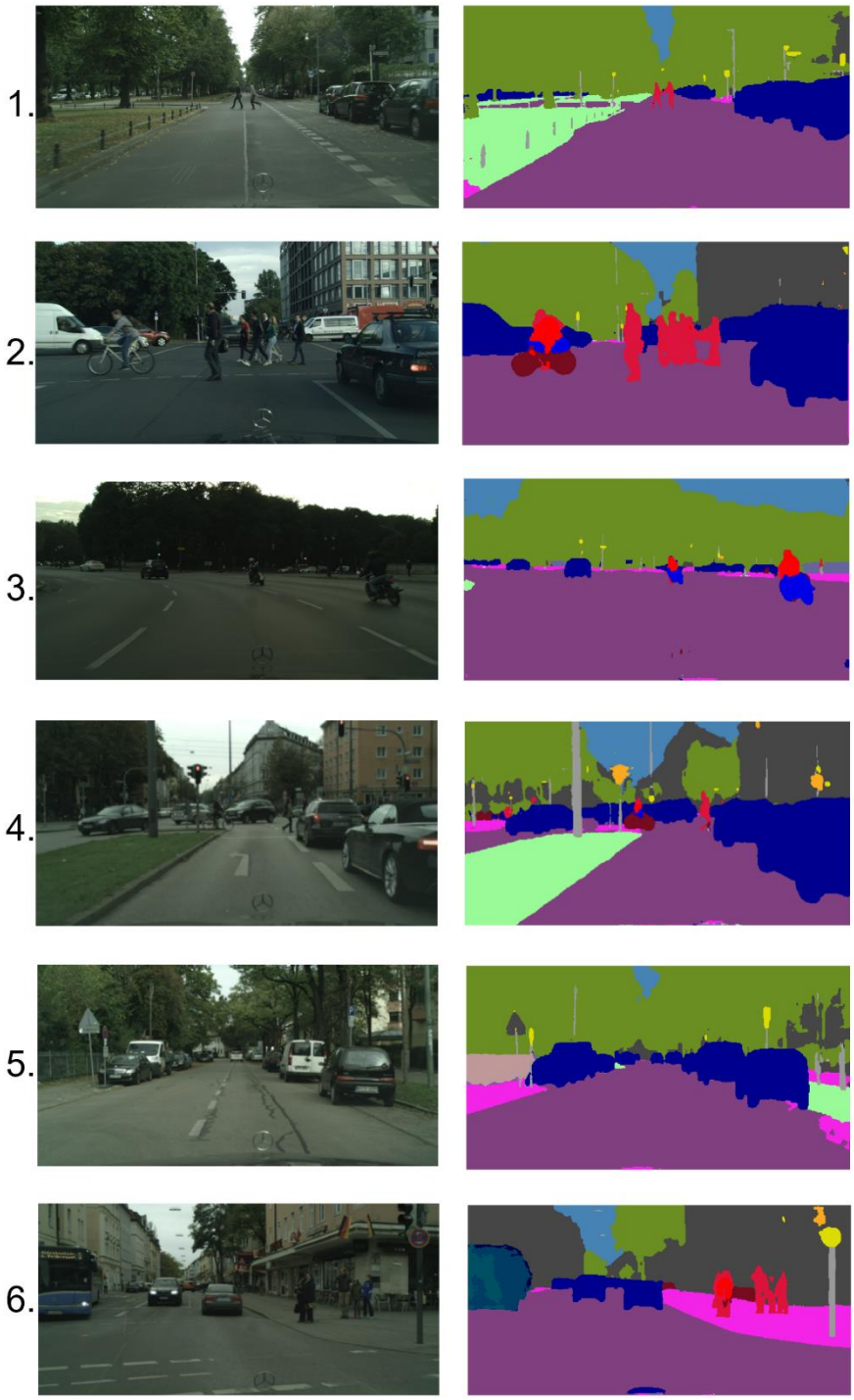

Figura 24: Resultados en el conjunto de prueba. De izquierda a derecha: imagen original, y etiqueta predicha. Cada par de imagen original y etiqueta predicha se presenta con su correspondiente subfigura. Configuración de hiperparámetros: tamaño de batch 16, tasa de aprendizaje 0.0005

4.3.3. Visualización y análisis de los modelos generativos

En el análisis de los resultados obtenidos con los modelos generativos utilizados para tareas de in-painting se observa que Stable Diffusion destaca en términos de rendimiento visual, superando a GLIDE en la integración de las áreas modificadas con el contexto global de la imagen. Esto se refleja en su capacidad para generar transiciones más suaves y coherentes, logrando un mayor realismo en las imágenes procesadas.

Por ejemplo, al utilizar los textos descriptivos “Campo lleno de girasoles”, “Una piscina” y “Una mesa de picnic sobre césped” en ambos modelos para reemplazar la clase carretera de la imagen original, se puede apreciar que las áreas completadas por Stable Diffusion no solo son más detalladas, sino que se adaptan de forma más natural al entorno, otorgando una apariencia fluida y cohesiva al resultado final (ver *Figura 25, subfiguras 1, 3 y 5*). Por el contrario, GLIDE muestra un desempeño inferior en aspectos clave como la nitidez y la cohesión. En las imágenes generadas con este modelo (ver *Figura 25, subfiguras 2, 4 y 6*), se pueden apreciar una menor resolución y zonas parcialmente completadas, dejando áreas visibles de la imagen original sin modificar. Además, la borrosidad general en las imágenes de GLIDE afecta negativamente la percepción del realismo, lo que limita su aplicabilidad en tareas que requieren precisión visual o una alta calidad en los detalles generados. Incluso en el caso del texto "Campo de girasoles", GLIDE no generó girasoles, sino que simplemente reconstruyó una carretera, sin reflejar la intención del texto descriptivo.

Otro aspecto que destacar es la comprensión del lenguaje. Stable Diffusion fue capaz de interpretar directamente los textos descriptivos en español, generando imágenes acordes a la descripción. Por el contrario, GLIDE requirió que los textos fueran traducidos al inglés, ya que, al ingresarlos en español, los resultados eran idénticos a la subfigura 2 de la Figura 40, donde en ambos idiomas se produjo la misma salida.

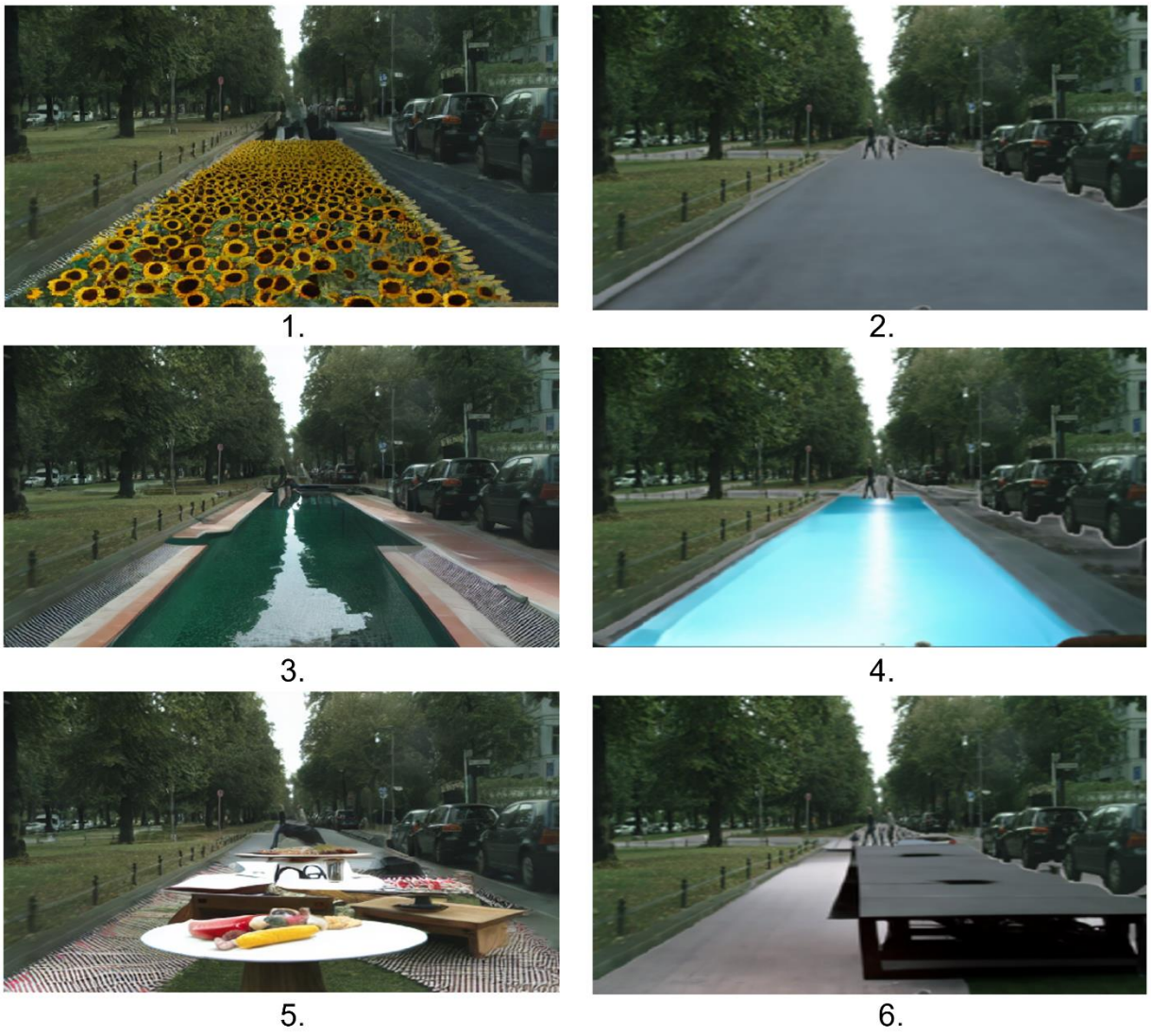

Figura 25: Resultado de in-painting utilizando Stable Diffusion (izquierda) y GLIDE (derecha). Subfiguras 1. y 2.: Transformando la carretera en un campo de girasoles, prompt = Campo lleno de girasoles. Subfiguras 3. y 4.: Transformando la carretera en una piscina, prompt = Una piscina. Subfiguras 5. y 6.: Transformando la carretera en un picnic, prompt = Una mesa de picnic sobre césped.

Por otro lado, ambos modelos presentan dificultades al tratar de generar detalles muy específicos o representaciones precisas de objetos en escenarios con descripciones textuales ambiguas o complejas, como personas y animales (ver *Figura 26, subfiguras 1* y *2*). En estas subfiguras, al caballo generado por Stable Diffusion le falta una pierna, mientras que el generado por GLIDE es irreconocible como un animal. También

presentan dificultades en el tamaño y proporción de elementos como vehículos (ver *Figura 26, subfiguras 3* y *4*). En este caso, los coches del fondo no han sido correctamente implementados por ninguno de los modelos, mientras que el coche en primer plano generado por Stable Diffusion muestra una fusión incorrecta entre el parabrisas y el capó, y los generados por GLIDE presentan ruedas desproporcionadas y casi transparentes.

Estas limitaciones se atribuyen, en parte, a las restricciones inherentes de los modelos gratuitos, los cuales poseen capacidades reducidas en términos de resolución y precisión en comparación con sus versiones completas o comerciales.

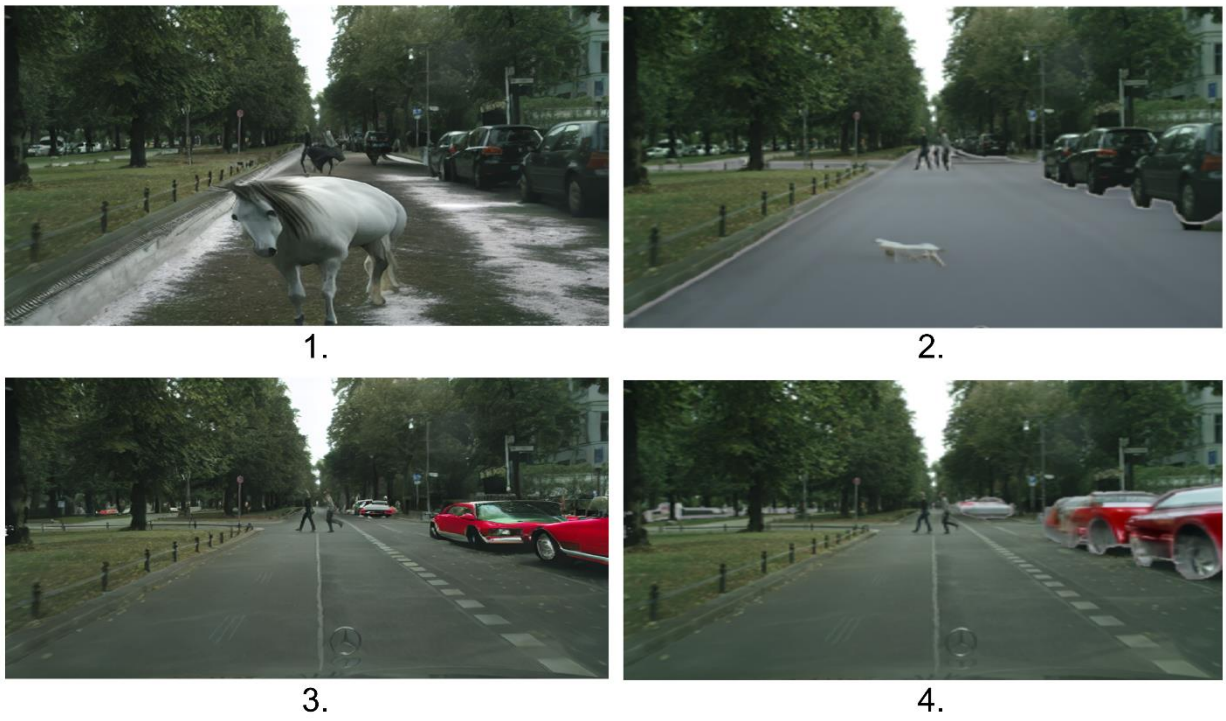

Figura 26: Resultado de in-painting utilizando Stable Diffusion (izquierda) y GLIDE (derecha). Subfiguras 1. y 2.: Transformando la carretera con un caballo blanco, prompt = Un caballo blanco sobre hierba. Subfiguras 3. y 4.: Transformando los coches en coches rojos, prompt = Coches rojos.

En general, aunque Stable Diffusion es la opción más adecuada para este tipo de tareas debido a su realismo y coherencia, es importante tener en cuenta las limitaciones

técnicas de los modelos gratuitos en la calidad de los resultados. Estas restricciones abren la puerta a futuras investigaciones y mejoras, como el uso de versiones más avanzadas de los modelos o el ajuste de los parámetros para optimizar la generación en escenarios específicos. No obstante, los resultados actuales demuestran el potencial significativo de estos modelos generativos en aplicaciones de edición y creación de contenido visual.

5. Conclusiones y trabajos futuros

5.1. Conclusiones

El objetivo principal de este trabajo era desarrollar un modelo de segmentación semántica capaz de identificar y clasificar las clases presentes en una imagen destinada a impresión, permitiendo al usuario eliminar ciertas clases no deseadas o incluso modificarlas mediante la generación de contenido basado en descripciones textuales. Tras finalizar este proyecto, se puede concluir que dicho objetivo ha sido alcanzado con éxito.

Para lograr este propósito, se han cumplido los siguientes objetivos específicos:

- Se llevó a cabo un estudio de distintas arquitecturas de redes neuronales y se seleccionó la arquitectura U-Net como base para el modelo, adaptando una implementación disponible en GitHub para convertirla en funcional y lograr una segmentación semántica precisa.
- Se eligió el conjunto de datos Cityscapes, que cumple con los requisitos necesarios para entrenar un modelo de segmentación semántica, proporcionando etiquetas precisas y una amplia variedad de escenarios urbanos.
- Se mejoró el rendimiento del modelo experimentando con diferentes configuraciones de hiperparámetros y técnicas de aumento de datos, alcanzando un coeficiente Dice de 0.8.
- Se desarrolló una interfaz de usuario que permite seleccionar las clases que se desean eliminar, así como la opción de modificar esas áreas eliminadas mediante contenido generado a partir de descripciones textuales proporcionadas por el usuario.

A lo largo del desarrollo de este trabajado he enfrentado ciertas limitaciones significativas debido a la falta de acceso a una GPU en mi ordenador personal que hizo inviable entrenar el modelo localmente, ya que el procesamiento de modelos de segmentación semántica requiere una alta capacidad computacional. Por ello, opté por

un enfoque de modelo cliente/servidor, utilizando Google Colab para la ejecución y aprovechando máquinas con GPUs más potentes que las disponibles localmente. Sin embargo, la GPU gratuita de Google Colab está limitada a sesiones de aproximadamente 2-3 horas, lo cual no era suficiente para entrenar el modelo completamente, especialmente al necesitar realizar ajustes tras visualizar los resultados parciales.

Como solución, tuve que crear múltiples cuentas de Google para continuar con los entrenamientos, lo que complicó significativamente la organización, ya que resultaba confuso recordar en qué cuenta estaba almacenada la última versión del cuaderno. A pesar de estas restricciones, el objetivo del trabajo se logró satisfactoriamente, pero esta experiencia resalta la importancia de contar con recursos computacionales adecuados para este tipo de proyectos.

Además, quiero destacar el impacto personal y académico que este trabajo ha tenido en mi desarrollo. La realización de este proyecto no solo ha significado un desafío técnico, sino también un crecimiento significativo en mi conocimiento y experiencia en el área de la inteligencia artificial, particularmente en la segmentación semántica. Aunque inicialmente tenía conocimientos limitados sobre el tema y no tenía mucho interés en esta disciplina, asumí este proyecto como un reto para expandir mis habilidades y explorar nuevas áreas de la informática. Estoy muy satisfecha con los resultados obtenidos y agradecida por todo lo aprendido durante este proceso.

Quiero también agradecer a mi tutora, cuya guía y experiencia en segmentación semántica han sido fundamentales para conseguir realizar este trabajo. Su apoyo y orientación me han permitido superar obstáculos técnicos y conceptuales, alcanzando resultados que superan mis expectativas iniciales.

5.2. Trabajos futuros

Como líneas futuras, sería recomendable trabajar con un conjunto de datos más amplio y diverso. Esto permitiría mejorar la precisión del modelo, especialmente en la detección

de clases minoritarias, bordes complejos y detalles más sutiles. Asimismo, la incorporación de más clases podría aumentar la versatilidad del modelo, mejorando los resultados obtenidos en las imágenes que los usuarios deseen modificar o imprimir.

También sería deseable disponer de acceso a recursos computacionales más avanzados. Por ejemplo, comprar una suscripción a Google Colab Pro o Colab Pro+ proporcionaría acceso a GPUs de mayor capacidad y sesiones de mayor duración, evitando las limitaciones actuales y permitiendo entrenar el modelo de manera continua y eficiente. Alternativamente, contar con un ordenador local equipado con una GPU potente sería otra solución para realizar entrenamientos completos sin interrupciones.

Además, sería adecuado invertir en el acceso a modelos generativos de in-painting más avanzados, que ofrezcan mayor realismo y precisión en la generación de contenido. Modelos comerciales o premium podrían permitir la creación de imágenes más detalladas y coherentes, ampliando las posibilidades de modificación visual según las necesidades del usuario.

Finalmente, considero que esta línea de trabajo tiene un alto potencial para ser implementada en el futuro. La inteligencia artificial está evolucionando rápidamente hacia la creación de contenido visual que es indistinguible de la realidad, y su integración en aplicaciones prácticas, como la edición y personalización de imágenes para impresión, podría facilitar significativamente la vida de muchos usuarios. Esto abre oportunidades para seguir explorando y mejorando soluciones que combinen segmentación semántica y generación de contenido en beneficio de tareas creativas y funcionales.

6. Bibliografía

[1] Plan de Recuperación, Transformación y Resiliencia, «Qué es la Inteligencia Artificial» Gobierno de España, 19 abril 2023. [En línea]. Available: https://planderecuperacion.gob.es/noticias/que-es-inteligencia-artificial-ia-prtr. [Último acceso: 02 enero 2025].

[2] EITCA, «¿Existe algún tipo de entrenamiento de un modelo de IA en el que los enfoques de aprendizaje supervisado y no supervisado se implementan al mismo tiempo?» 07 noviembre 2024. [En línea]. Available: https://es.eitca.org/inteligencia-artificial/eitc-ai-gcml-google-nube-aprendizaje-autom%C3%A1tico/introducci%C3%B3n/que-es-el-aprendizaje-automatico/%C2%BFExiste-alg%C3%BAn-tipo-de-entrenamiento-de-un-modelo-de-IA-en-el-que-se-implementen-al-mismo-tiempo-. [Último acceso: 29 enero 2025].

[3] E. G. Rosicart, «El aprendizaje no supervisado y cómo descubrir patrones en datos» 15 enero 2025. [En línea]. Available: https://www.obsbusiness.school/blog/el-aprendizaje-no-supervisado-y-como-descubrir-patrones-en-datos. [Último acceso: 29 enero 2025].

[4] J. Holdsworth y M. Scapicchio, «What is deep learning?» IBM, [En línea]. Available: https://www.ibm.com/topics/deep-learning. [Último acceso: 02 enero 2025].

[5] O. Bruce-Gardyne, T. Koo y R. Zatsarenko, «Autograding in Education Using Artificial Intelligence» OxJournal, 10 noviembre 2021. [En línea]. Available: https://www.oxjournal.org/autograding-in-education-using-artificial-intelligence/. [Último acceso: 02 enero 2025].

[6] A. G. García, S. O. Escolano, S. Oprea, V. V. Martínez, P. M. González y J. G. Rodríguez, «A survey on deep learning techniques for image and video semantic segmentation» *Applied Soft Computing,* vol. 70, pp. 41-65, 2018.

[7] IBM, «¿Qué es la segmentación de imágenes?» [En línea]. Available: https://www.ibm.com/es-es/topics/image-segmentation. [Último acceso: 29 enero 2025].

[8] IBM, «¿Qué es la segmentación semántica?» [En línea]. Available: https://www.ibm.com/es-es/topics/semantic-segmentation. [Último acceso: 02 enero 2025].

[9] P. Alzamora, «Segmentación semántica de imágenes con Deep Learning» Data Machine Learning Visualization, 10 marzo 2023. [En línea]. Available: https://blog.damavis.com/segmentacion-semantica-de-imagenes-con-deep-learning/. [Último acceso: 02 enero 2025].

[10] S. Fazekas, B. Budai, R. Stollmayer, P. Kaposi y V. Bérczi, «Artificial intelligence and neural networks in radiology – Basics that all radiology residents should know» *Imaging,* vol. 14, pp. 73-81, 23 diciembre 2022.

[11] Encord, «Guide to Panoptic Segmentation» 13 septiembre 2023. [En línea]. Available: https://encord.com/blog/panoptic-segmentation-guide/. [Último acceso: 02 enero 2025].

[12] J. A. Camacho, «Introducción a las Redes Neuronales» JacobSoft, 15 junio 2021. [En línea]. Available: https://www.jacobsoft.com.mx/es_mx/introduccion-a-las-redes-neuronales/. [Último acceso: 02 enero 2025].

[13] A. Mishra, «Understanding Machine Learning for Friction Stir WeldingTechnology» *International Journal of Machine Learning and Networked Collaborative Engineering,* vol. 3, pp. 143-158, 10 noviembre 2019.

[14] Na8, «¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador» 29 noviembre 2018. [En línea]. Available: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/. [Último acceso: 02 enero 2025].

[15] J. Long, E. Shelhamer y T. Darrell, «Fully convolutional networks for semantic segmentation» *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 3431-3440. 08 marzo 2015.

[16] codificandobits, «Padding, strides, max-pooling y stacking en las Redes Convolucionales» [En línea]. Available: https://codificandobits.com/blog/padding-strides-maxpooling-stacking-redes-convolucionales/. [Último acceso: 02 enero 2025].

[17] D. Calvo, «Función de activación – Redes neuronales» 07 diciembre 2018. [En línea]. Available: https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/. [Último acceso: 29 enero 2025].

[18] H. Noh, S. Hong y B. Han, «Learning deconvolution network for semantic segmentation» pp. 1520-1528, 2015.

[19] V. Badrinarayanan, A. Kendall y R. Cipolla, «SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, nº 12, pp. 2481-2495, 01 diciembre 2017.

[20] O. Ronneberger, P. Fischer y T. Brox, «U-net: Convolutional networks for biomedical image segmentation» pp. 234-241, 2015.

[21] M. Tran, «Understanding U-Net» 15 noviembre 2022. [En línea]. Available: https://towardsdatascience.com/understanding-u-net-61276b10f360. [Último acceso: 02 enero 2025].

[22] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy y A. L. Yuille, «DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 40. pp. 834-848, 01 abril 2018.

[23] content79qw, «What is Stable Diffusion? Importance and Working» GeeksforGeeks, 01 agosto 2024. [En línea]. Available: https://www.geeksforgeeks.org/stable-diffusion/. [Último acceso: 02 enero 2025].

[24] paperswithcode, «Guided Language to Image Diffusion for Generation and Editing» [En línea]. Available: https://paperswithcode.com/method/glide#:~:text=GLIDE%20is%20a%20generative%20model%20based%20on%20text-guided,the%20model%20is%20able%20to%20handle%20free-form%20prompts.. [Último acceso: 02 enero 2025].

[25] OpenAI, «DALL-E 3» [En línea]. Available: https://openai.com/index/dall-e-3/. [Último acceso: 02 enero 2025].

[26] Git, «Git» Software Freedom Conservancy, [En línea]. Available: https://git-scm.com/. [Último acceso: 02 enero 2025].

[27] J. Torrado, «Introducción a GitLab» 12 octubre 2017. [En línea]. Available: https://desarrolloweb.com/articulos/introduccion-gitlab.html. [Último acceso: 02 enero 2025].

[28] sadasivareddy, «What Is Gitlab? Complete Guide» GeeksforGeeks, 22 abril 2024. [En línea]. Available: https://www.geeksforgeeks.org/gitlab/. [Último acceso: 02 enero 2025].

[29] Google Colaboratory, «Te damos la bienvenida a Colab» [En línea]. Available: https://colab.research.google.com/#scrollTo=OwuxHmxllTwN. [Último acceso: 02 enero 2025].

[30] d2anubis, «What is PyTorch?» GeeksforGeeks, 26 diciembre 2023. [En línea]. Available: https://www.geeksforgeeks.org/getting-started-with-pytorch/. [Último acceso: 02 enero 2025].

[31] NumPy, «What is NumPy?» [En línea]. Available: https://numpy.org/doc/stable/user/whatisnumpy.html. [Último acceso: 02 enero 2025].

[32] PyPI, «Pillow 11.0.0» [En línea]. Available: https://pypi.org/project/pillow/. [Último acceso: 02 enero 2025].

[33] PyTorch, «torchvision» [En línea]. Available: https://pytorch.org/vision/stable/index.html. [Último acceso: 02 enero 2025].

[34] Matplotlib, «Matplotlib: Visualization with Python» [En línea]. Available: https://matplotlib.org/. [Último acceso: 01 enero 2025].

[35] A. Yadav, «Mastering Python Progress Bars with tqdm: A Comprehensive Guide» 31 mayo 2023. [En línea]. Available: https://www.askpython.com/python-modules/python-progress-bars-with-tqdm. [Último acceso: 02 enero 2025].

[36] J. Jin Won y S. Yoan, «Extended U-Net for Satellite Image Semantic Segmentation» *2023 Fourteenth International Conference on Ubiquitous and Future Networks (ICUFN),* pp. 875-877, 2023.

[37] F. Zhu, J. Cui, B. Zhu, H. Li y Y. Liu, «Semantic segmentation of urban street scene images based on improved U-Net network» *Optoelectronics Letters,* vol. 19, pp. 179-185, 2023.

[38] J. Chen, J. Mei, X. Li, Y. Lu, Q. Yu, Q. Wei, X. Luo, Y. Xie, E. Adeli, Y. Wang, M. P. Lungren, S. Zhang, L. Xing, L. Lu, A. Yuille y Y. Zhou, «TransUNet: Rethinking the U-Net architecture design for medical image segmentation through the lens of transformers» *Medical Image Analysis,* vol. 97, pp. 1361-8415, 01 octubre 2024.

[39] CityScapes, «Dataset Overview» [En línea]. Available: https://www.cityscapes-dataset.com/dataset-overview/. [Último acceso: 10 enero 2025].

[40] PyTorch, «Dropout» 2024. [En línea]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html. [Último acceso: 29 enero 2025].

[41] K. Pykes, «Cross-Entropy Loss Function in Machine Learning: Enhancing Model Accuracy» 10 agosto 2024. [En línea]. Available: https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning. [Último acceso: 03 enero 2025].

[42] D. Shah, «Intersection over Union (IoU): Definition, Calculation, Code» 30 mayo 2023. [En línea]. Available: https://www.v7labs.com/blog/intersection-over-union-guide. [Último acceso: 03 enero 2025].

[43] Kukil, «Intersection over Union (IoU) in Object Detection & Segmentation» 28 junio 2022. [En línea]. Available: https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/. [Último acceso: 03 enero 2025].

[44] A. A. Taha y A. Hanbury , «Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool» *BMC Med Imaging,* vol. 15, nº 1, p. 29, 2015.

[45] N. Huynh, «Understanding Evaluation Metrics in Medical Image Segmentation» Medium, 01 marzo 2023. [En línea]. Available: https://medium.com/@nghihuynh_37300/understanding-evaluation-metrics-in-medical-image-segmentation-d289a373a3f#:~:text=The%20difference%20between%20the%20two%20metrics%20is%20that,the%20total%20number%20of%20pixel%20in%20both%20images.. [Último acceso: 29 enero 2025].

[46] Hugging Face, «Inpainting» [En línea]. Available: https://huggingface.co/docs/diffusers/api/pipelines/stable_diffusion/inpaint. [Último acceso: 30 enero 2025].

[47] OpenAI, «glide-text2im» 21 marzo 2022. [En línea]. Available: https://github.com/openai/glide-text2im. [Último acceso: 30 enero 2025].

Anexo

En este apartado se presentan las tablas con los resultados obtenidos para las distintas configuraciones evaluadas durante el entrenamiento del modelo. Estos datos incluyen métricas clave como la pérdida, la métrica IoU, el coeficiente Dice, la precisión y la sensibilidad, tanto en el conjunto de entrenamiento como en el de validación, permitiendo una comparación detallada del rendimiento de cada configuración.

Las filas resaltadas en naranja en la tabla de entrenamiento corresponden a la época en la que se alcanzó la menor pérdida en el conjunto de validación. En la tabla de validación, las filas resaltadas en azul representan la menor pérdida en validación en cada configuración.

Esto permite identificar fácilmente en qué punto específico del entrenamiento el modelo logró su mejor rendimiento en validación, así como los valores alcanzados en el entrenamiento en esa misma época.

Tabla de entrenamiento de la 1ª configuración (batch 16, tasa de aprendizaje 0.00005):

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
1	1.524543	0.000151	0.000299	0.016737	0.000160
2	0.764122	0.149942	0.207416	0.531213	0.176301
3	0.588581	0.608972	0.753315	0.782499	0.740137
4	0.516691	0.668564	0.798910	0.821579	0.788099
5	0.480453	0.700343	0.821434	0.840111	0.812352
6	0.441558	0.728856	0.841498	0.854866	0.835712

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
7	0.415880	0.748083	0.853881	0.866474	0.847839
8	0.391582	0.765793	0.866082	0.875564	0.862781
9	0.370566	0.779717	0.874979	0.883650	0.872514
10	0.348843	0.800275	0.888111	0.894025	0.886302
11	0.330418	0.811791	0.895173	0.899345	0.895147
12	0.315360	0.818932	0.899567	0.902693	0.900176
13	0.302078	0.827953	0.905018	0.907602	0.906322
14	0.287590	0.837669	0.911096	0.911196	0.913999
15	0.270793	0.849883	0.918238	0.920440	0.918701
16	0.259523	0.858134	0.923088	0.923888	0.924703
17	0.244195	0.866087	0.927784	0.926542	0.930608
18	0.234067	0.878745	0.935139	0.934265	0.937274
19	0.222965	0.884226	0.938212	0.937414	0.940081
20	0.212878	0.890827	0.942052	0.940900	0.944098
21	0.213574	0.883121	0.937555	0.937480	0.939143
22	0.196358	0.900883	0.947674	0.946930	0.949084
23	0.185446	0.905829	0.950411	0.949477	0.952139
24	0.176476	0.913789	0.954839	0.953350	0.956790

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
25	0.180731	0.901940	0.948217	0.947730	0.949705
26	0.166761	0.918663	0.957499	0.956147	0.959307
27	0.166077	0.907846	0.951500	0.949707	0.954226
28	0.155678	0.922091	0.959338	0.957419	0.961776
29	0.147008	0.929695	0.963488	0.962429	0.964795
30	0.142666	0.930783	0.964059	0.963158	0.965294
31	0.138791	0.928872	0.963005	0.961959	0.964509
32	0.139224	0.923525	0.960025	0.958411	0.962391
33	0.142092	0.923463	0.960059	0.959645	0.960980
34	0.133815	0.930644	0.963982	0.963452	0.964904
35	0.132115	0.931972	0.964691	0.964314	0.965398

Tabla de validación de la 1ª configuración (batch 16, tasa de aprendizaje 0.00005):

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
1	1.082146	0.000000	0.000000	0.000000	0.000000
2	0.839644	0.501448	0.661888	0.653513	0.677744
3	0.696013	0.537499	0.692030	0.784600	0.625623
4	0.775528	0.559678	0.712100	0.643609	0.804859

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
5	0.635372	0.650725	0.784119	0.771616	0.800799
6	0.579729	0.614629	0.755510	0.801268	0.718269
7	0.604931	0.656104	0.788277	0.744380	0.842656
8	0.511512	0.681406	0.805733	0.823348	0.792304
9	0.505609	0.661702	0.790667	0.869499	0.728638
10	0.471163	0.705870	0.822810	0.856383	0.794679
11	0.458285	0.705965	0.824067	0.794706	0.860758
12	0.450893	0.714096	0.829172	0.861283	0.803447
13	0.417690	0.736040	0.843766	0.847324	0.843416
14	0.447420	0.729053	0.839546	0.877937	0.809258
15	0.430352	0.735539	0.843925	0.856523	0.835802
16	0.436347	0.744125	0.849040	0.832267	0.870334
17	0.405098	0.742679	0.847583	0.863011	0.837062
18	0.397207	0.739891	0.846568	0.889270	0.811145
19	0.396141	0.750917	0.853864	0.821434	0.894516
20	0.389310	0.754705	0.856341	0.845369	0.872370
21	0.448317	0.747675	0.851222	0.869402	0.836862
22	0.380740	0.756779	0.856756	0.861532	0.859007

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
23	0.367077	0.769855	0.865040	0.853347	0.880735
24	0.380709	0.765867	0.863598	0.863201	0.869101
25	0.395219	0.765377	0.862841	0.866108	0.862888
26	0.421847	0.766322	0.863201	0.854005	0.876420
27	0.385632	0.764126	0.861028	0.852364	0.875092
28	0.371568	0.763650	0.861889	0.880777	0.846225
29	0.402621	0.756968	0.857293	0.857104	0.861192
30	0.393869	0.780288	0.873407	0.879598	0.870952
31	0.398405	0.764827	0.862820	0.839479	0.891328
32	0.423802	0.741758	0.846068	0.830882	0.869752
33	0.391301	0.768643	0.864998	0.878005	0.855739
34	0.401814	0.775321	0.869252	0.884261	0.857917
35	0.402932	0.761646	0.860544	0.851229	0.873181

Tabla de entrenamiento de la 2ª configuración (batch 16, tasa de aprendizaje 0.0001):

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
1	1.221194	0.011743	0.020981	0.284569	0.015744

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
2	0.626553	0.501613	0.655756	0.735238	0.628064
3	0.524881	0.647045	0.783072	0.804576	0.774218
4	0.473241	0.691676	0.815232	0.835696	0.807512
5	0.436593	0.725809	0.838918	0.854050	0.832881
6	0.406681	0.745357	0.852366	0.867651	0.845435
7	0.377130	0.763225	0.864153	0.875851	0.861527
8	0.352022	0.786767	0.879092	0.888234	0.875208
9	0.335210	0.797596	0.886078	0.891734	0.885426
10	0.307973	0.820247	0.900400	0.903790	0.900294
11	0.296798	0.814562	0.896858	0.900518	0.897164
12	0.277101	0.836015	0.910024	0.911023	0.912608
13	0.268055	0.840191	0.912570	0.913136	0.915241
14	0.249888	0.848919	0.917474	0.917910	0.919987
15	0.241787	0.855395	0.921256	0.920840	0.924386
16	0.233058	0.864243	0.926450	0.925701	0.929687
17	0.220869	0.879078	0.935283	0.936077	0.935850
18	0.214609	0.877813	0.934457	0.932969	0.937552
19	0.197431	0.890133	0.941521	0.941060	0.943331

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
20	0.192239	0.887519	0.940023	0.938775	0.942690
21	0.179935	0.901618	0.948058	0.947562	0.949521
22	0.174851	0.907571	0.951393	0.950877	0.952600
23	0.168227	0.908865	0.952078	0.951270	0.953670
24	0.159828	0.915484	0.955744	0.954603	0.957404
25	0.160095	0.908790	0.951969	0.951322	0.953447
26	0.152597	0.918089	0.957144	0.955578	0.959195
27	0.152569	0.920655	0.958575	0.958214	0.959440

Tabla de validación de la 2ª configuración (batch 16, tasa de aprendizaje 0.0001):

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
1	0.844860	0.051516	0.097519	0.758317	0.052420
2	0.663506	0.581379	0.729199	0.698909	0.769258
3	0.589865	0.614262	0.754854	0.752421	0.762640
4	0.613425	0.536549	0.693782	0.564388	0.913142
5	0.502181	0.676011	0.801763	0.829231	0.781930
6	0.500875	0.642406	0.776214	0.895858	0.693005
7	0.474821	0.667518	0.795152	0.898546	0.718593

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
8	0.410762	0.734335	0.842280	0.826817	0.862386
9	0.428164	0.735551	0.843482	0.882092	0.812432
10	0.413651	0.724390	0.835308	0.879999	0.800008
11	0.385812	0.749194	0.852258	0.896465	0.815237
12	0.366019	0.763790	0.861999	0.872545	0.855074
13	0.354091	0.748322	0.851278	0.880009	0.830495
14	0.363429	0.750025	0.852686	0.895926	0.816140
15	0.348804	0.758117	0.857074	0.854154	0.868214
16	0.338235	0.765404	0.862238	0.895425	0.837033
17	0.339451	0.760618	0.858539	0.897742	0.825838
18	0.329408	0.774058	0.868415	0.902412	0.841101
19	0.336917	0.761856	0.859559	0.848749	0.874864
20	0.319076	0.761802	0.860131	0.900064	0.830018
21	0.320997	0.784625	0.874997	0.899480	0.857413
22	0.326766	0.785856	0.876068	0.878203	0.878019
23	0.359528	0.778255	0.870117	0.870682	0.874593
24	0.323922	0.771621	0.867481	0.922128	0.823424
25	0.350599	0.767742	0.863575	0.894610	0.840543

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
26	0.323278	0.772662	0.867604	0.899700	0.844563
27	0.359418	0.768339	0.864397	0.902066	0.834152

Tabla de entrenamiento 3ª configuración (batch 16, tasa de aprendizaje 0.0005):

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
1	0.877264	0.215167	0.311015	0.496352	0.262346
2	0.584972	0.523911	0.682654	0.757465	0.639543
3	0.504198	0.627211	0.767244	0.808343	0.746936
4	0.458459	0.667579	0.797530	0.823232	0.785726
5	0.411616	0.710475	0.828269	0.846908	0.821256
6	0.384573	0.738999	0.848156	0.862131	0.843169
7	0.364819	0.752892	0.857270	0.871717	0.850723
8	0.343706	0.777310	0.873009	0.882880	0.869869
9	0.329120	0.785239	0.878428	0.889005	0.875029
10	0.311706	0.796975	0.885919	0.893254	0.884300
11	0.307351	0.795759	0.884436	0.893241	0.883637
12	0.288733	0.821739	0.901208	0.905267	0.900938
13	0.279655	0.825298	0.903089	0.905848	0.904104

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
14	0.267163	0.837903	0.911056	0.914118	0.911111
15	0.260372	0.841693	0.913022	0.914310	0.915263
16	0.250147	0.847233	0.916403	0.916293	0.919499
17	0.245152	0.850275	0.918282	0.918787	0.920909
18	0.237578	0.854785	0.921014	0.921649	0.923171
19	0.236939	0.857005	0.922275	0.921986	0.925661
20	0.222044	0.876309	0.933682	0.933315	0.935729
21	0.215521	0.876519	0.933694	0.932665	0.936624
22	0.205315	0.887669	0.940218	0.939241	0.942541
23	0.203670	0.883943	0.937806	0.937379	0.940399
24	0.204055	0.883657	0.937835	0.936552	0.940409
25	0.187801	0.893919	0.943718	0.941948	0.946673
26	0.195959	0.890753	0.941853	0.940957	0.943882
27	0.178263	0.903984	0.949253	0.947762	0.951678
28	0.175628	0.900560	0.947374	0.946423	0.949410
29	0.168467	0.911989	0.953813	0.951966	0.956246
30	0.157881	0.918107	0.957158	0.955235	0.959703
31	0.154527	0.916459	0.956269	0.954967	0.958088

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
32	0.175815	0.899947	0.946995	0.946703	0.948685
33	0.166252	0.903946	0.949195	0.948246	0.951228
34	0.144704	0.918021	0.957066	0.955225	0.959697
35	0.142207	0.923573	0.960166	0.959373	0.961385

Tabla de validación de la 3ª configuración (batch 16, tasa de aprendizaje 0.0005):

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
1	0.697409	0.468885	0.628588	0.652541	0.618383
2	0.615942	0.510612	0.666337	0.834744	0.563445
3	0.514046	0.618350	0.757391	0.827963	0.702462
4	0.461485	0.676485	0.801984	0.842061	0.768782
5	0.446269	0.636023	0.769823	0.890120	0.685567
6	0.434703	0.693120	0.812359	0.824960	0.809872
7	0.405564	0.710008	0.824762	0.816453	0.839821
8	0.365920	0.738105	0.845345	0.863716	0.832147
9	0.376310	0.703838	0.820695	0.833753	0.815427
10	0.397709	0.641916	0.775181	0.927458	0.673732
11	0.352822	0.764613	0.862384	0.857574	0.870857

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
12	0.339455	0.756755	0.857057	0.874371	0.844385
13	0.335799	0.737176	0.842649	0.804643	0.893990
14	0.326652	0.761019	0.857396	0.848270	0.876589
15	0.348662	0.743443	0.848653	0.892915	0.812192
16	0.326165	0.756981	0.856901	0.874184	0.845053
17	0.354008	0.728804	0.836816	0.789606	0.901317
18	0.315474	0.752094	0.854250	0.881426	0.833593
19	0.320229	0.734380	0.842705	0.935377	0.770982
20	0.293631	0.789493	0.878786	0.902012	0.860788
21	0.294432	0.754004	0.854701	0.805302	0.920644
22	0.287034	0.775327	0.869752	0.906120	0.839700
23	0.300373	0.785241	0.876347	0.886624	0.871304
24	0.266816	0.796215	0.883657	0.909660	0.861781
25	0.272586	0.763402	0.859393	0.814925	0.920166
26	0.275738	0.782075	0.873755	0.928003	0.828972
27	0.284700	0.792481	0.879016	0.837317	0.933615
28	0.260168	0.806845	0.888889	0.886487	0.896204
29	0.262909	0.809688	0.891610	0.885281	0.901537

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
30	0.272150	0.782102	0.873993	0.917917	0.837980
31	0.260980	0.812715	0.892567	0.884206	0.906635
32	0.273343	0.800148	0.885774	0.910479	0.866107
33	0.263508	0.810167	0.891952	0.912159	0.876295
34	0.263489	0.804435	0.887600	0.878313	0.901471
35	0.268295	0.803574	0.887675	0.904053	0.876712

Tabla de entrenamiento 4ª configuración (batch 16, tasa de aprendizaje 0.001):

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
1	0.861447	0.228161	0.339719	0.556738	0.285560
2	0.630832	0.468969	0.629512	0.730737	0.583310
3	0.558534	0.567832	0.716851	0.771580	0.698192
4	0.513058	0.639077	0.774588	0.809551	0.762146
5	0.466946	0.681245	0.804980	0.829454	0.800060
6	0.434106	0.708020	0.824910	0.841308	0.823497
7	0.409534	0.725632	0.837335	0.851657	0.836924
8	0.387633	0.743783	0.848643	0.861653	0.851263
9	0.373478	0.754500	0.856103	0.867332	0.857191

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
10	0.354185	0.768980	0.866352	0.874679	0.868121
11	0.337118	0.786124	0.877215	0.883063	0.879659
12	0.323588	0.793043	0.881345	0.888351	0.883852
13	0.312177	0.794584	0.882252	0.887607	0.887253
14	0.301135	0.809537	0.892516	0.896441	0.895815
15	0.285557	0.822845	0.900724	0.903482	0.904156
16	0.276858	0.823219	0.900330	0.902076	0.905561
17	0.266549	0.829625	0.905052	0.908193	0.907583
18	0.259245	0.837412	0.909754	0.912376	0.912612
19	0.245902	0.850068	0.917541	0.917854	0.921221
20	0.246723	0.850230	0.917490	0.919701	0.919236
21	0.234111	0.858302	0.922318	0.923346	0.925223
22	0.220463	0.869515	0.929165	0.928407	0.932686
23	0.221391	0.873338	0.931448	0.931511	0.934019
24	0.215499	0.869183	0.928749	0.929119	0.931728
25	0.199750	0.884649	0.938072	0.937563	0.940498
26	0.198619	0.885874	0.938694	0.937869	0.941418
27	0.188120	0.893890	0.943337	0.943246	0.944926

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
28	0.190459	0.891948	0.942246	0.942946	0.943448
29	0.179949	0.900444	0.947142	0.947263	0.948248
30	0.170544	0.906139	0.950266	0.949529	0.952142
31	0.169131	0.904622	0.949366	0.949088	0.951005
32	0.161029	0.912176	0.953744	0.953980	0.954297
33	0.157205	0.911500	0.953298	0.954106	0.953650
34	0.148829	0.922481	0.959468	0.960003	0.959522
35	0.156149	0.912400	0.953806	0.955135	0.953509
36	0.155036	0.913521	0.954357	0.954874	0.955025
37	0.141341	0.918513	0.957204	0.958655	0.956521
38	0.130902	0.931135	0.964154	0.964148	0.964482
39	0.129354	0.927485	0.962138	0.963159	0.961708
40	0.128973	0.933488	0.965454	0.965795	0.965424
41	0.143516	0.912847	0.953425	0.954027	0.954968
42	0.125717	0.929885	0.963442	0.964199	0.963163
43	0.114173	0.939780	0.968817	0.969278	0.968592
44	0.108575	0.943599	0.970884	0.971125	0.970817
45	0.109358	0.941324	0.969650	0.969839	0.969768

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
46	0.1209767265545745	0.9308095390265969	0.9637396792825234	0.9655431545158474	0.9630117752335983

Tabla de validación de la 4ª configuración (batch 16, tasa de aprendizaje 0.001):

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
1	0.840313	0.265392	0.408198	0.716797	0.298606
2	0.621146	0.517651	0.668189	0.624421	0.752896
3	0.671319	0.486165	0.640554	0.614063	0.708657
4	0.629504	0.616959	0.751861	0.831461	0.699362
5	0.557704	0.656318	0.782249	0.799725	0.777417
6	0.631373	0.558152	0.704570	0.608069	0.877700
7	0.577488	0.570651	0.714442	0.617576	0.880370
8	0.432340	0.713236	0.823733	0.776623	0.893659
9	0.456584	0.589847	0.731777	0.913525	0.622781
10	0.406379	0.701458	0.814928	0.797606	0.848506
11	0.419027	0.687505	0.804751	0.790072	0.834917
12	0.367822	0.723382	0.830194	0.790822	0.891200

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
13	0.387086	0.747978	0.849937	0.886415	0.824869
14	0.359884	0.739123	0.840097	0.835732	0.855821
15	0.324318	0.759224	0.855429	0.850289	0.874845
16	0.316506	0.749125	0.848837	0.814052	0.901863
17	0.361197	0.708069	0.818304	0.778169	0.880323
18	0.318163	0.759098	0.853620	0.839094	0.882983
19	0.331847	0.763737	0.856589	0.862662	0.860888
20	0.322322	0.749302	0.847319	0.897414	0.812143
21	0.289466	0.783428	0.871272	0.896127	0.857109
22	0.293138	0.794073	0.877641	0.869002	0.896553
23	0.315579	0.769017	0.863020	0.881891	0.855311
24	0.282442	0.781500	0.870513	0.909577	0.842058
25	0.292497	0.783387	0.872542	0.890849	0.865273
26	0.315014	0.772097	0.863217	0.867784	0.868685
27	0.274892	0.786761	0.874279	0.902422	0.857604
28	0.318246	0.787472	0.873904	0.887123	0.872333
29	0.284997	0.792848	0.875707	0.893402	0.865637
30	0.284689	0.778757	0.868699	0.883090	0.866054

Época	Pérdida	IoU	Dice	Precisión	Sensibilidad
31	0.282283	0.796950	0.880139	0.904909	0.866137
32	0.279430	0.797262	0.880523	0.885971	0.884954
33	0.285972	0.792572	0.877372	0.891523	0.871766
34	0.315574	0.780475	0.869339	0.888220	0.864028
35	0.282522	0.800810	0.882684	0.883368	0.893078
36	0.280069	0.797534	0.880968	0.860956	0.912165
37	0.286290	0.784693	0.871128	0.896891	0.854952
38	0.287378	0.801615	0.883237	0.885208	0.890894
39	0.276503	0.808830	0.889319	0.881609	0.904916
40	0.277586	0.808165	0.888197	0.909111	0.877233
41	0.288323	0.805226	0.886727	0.886338	0.894803
42	0.286299	0.809416	0.887655	0.874650	0.909861
43	0.293097	0.810412	0.888398	0.897156	0.887856
44	0.279698	0.813030	0.890854	0.907719	0.882642
45	0.291425	0.816829	0.893406	0.894452	0.899489
46	0.357724	0.750468	0.848090	0.857376	0.848555