Universidad Miguel Hernández de Elche

MASTER UNIVERSITARIO EN ROBÓTICA

Segmentación semántica mediante aprendizaje profundo usando modelos SAM aplicado a la detección de grietas

Trabajo de Fin de Máster

Curso académico 2024/2025

Autor: Borja López Soler

Tutor: Luis Miguel Jiménez García

Agradecimientos

Este trabajo marca el cierre de una etapa y el inicio de un nuevo capítulo, lleno de ilusión y determinación para afrontar los retos que el futuro depare.

No quiero dejar pasar la oportunidad de agradecer a quienes han sido mi mayor apoyo en este camino: mi familia. En especial a mis padres, cuya dedicación, confianza y sacrificios han sido los cimientos sobre los que he construido tanto este como otros logros en mi vida. Sin ellos, nada de esto habría sido posible. También quiero expresar mi más profundo agradecimiento a mi pareja, por su apoyo incondicional, paciencia y comprensión.

Finalmente, agradecer a ValgrAI (Valencian Graduate School and Research Network of Artificial Intelligence) por la ayuda recibida para la realización de este Máster.

ÍNDICE

Lista de figuras

Lista de tablas

1 Introducción

La **visión por computador** es una rama de la inteligencia artificial que busca dotar a las máquinas de la capacidad de "ver", procesar e interpretar imágenes y videos de forma similar a los humanos. Mediante algoritmos y técnicas avanzadas, este campo permite que los sistemas identifiquen elementos visuales, comprendan sus relaciones y actúen en consecuencia. Su desarrollo está estrechamente vinculado al procesamiento de imágenes, al análisis de patrones y, más recientemente, a los avances en aprendizaje profundo.

El objetivo central de la visión por computador es lograr que las máquinas interpreten imágenes a nivel semántico, es decir, que comprendan su contenido y las relaciones entre sus componentes. Esto abarca desde tareas simples como el reconocimiento de objetos hasta problemas más complejos como la segmentación de imágenes, la reconstrucción tridimensional a partir de imágenes bidimensionales y el análisis de movimiento. Su aplicación tiene un impacto significativo en sectores como la medicina (diagnóstico basado en imágenes médicas), la seguridad (monitorización mediante cámaras), el transporte (vehículos autónomos), entre otros.

El campo de la visión por computador ha evolucionado significativamente desde sus inicios, cuando dependía en gran medida de técnicas heurísticas de procesamiento de imágenes y algoritmos diseñados manualmente. En sus primeras etapas, los sistemas de visión por computador requerían que los desarrolladores obtuvieran explícitamente las características de interés como bordes, texturas y formas. Este enfoque era limitado, ya que los algoritmos no eran lo suficientemente robustos para manejar la gran variabilidad presente en las imágenes del mundo real.

Con la llegada del aprendizaje automático, en particular del aprendizaje profundo, la visión por computador experimentó un cambio paradigmático. Las redes neuronales profundas, como las redes convolucionales (CNNs), eliminaron la necesidad de diseñar manualmente las características, permitiendo que los modelos aprendieran automáticamente las representaciones relevantes a partir de grandes volúmenes de datos. Esto resultó en un desempeño superior en tareas como la clasificación de imágenes, el reconocimiento facial y la detección de objetos.

Más recientemente, los modelos basados en arquitecturas de Transformers, originalmente desarrollados para el procesamiento del lenguaje natural, han mostrado resultados prometedores en visión por computador. Su capacidad para modelar relaciones globales dentro de una imagen ha permitido avances en tareas que requieren una comprensión contextual más profunda, como la segmentación de imágenes y el análisis de escenas en tiempo real.

1.1 Segmentación de Imágenes

Una de las tareas clave dentro de la visión por computador es la **segmentación de imágenes**, la cual consiste en dividir una imagen en regiones coherentes, etiquetando cada píxel con su categoría correspondiente. Este proceso es esencial para que los sistemas no solo reconozcan objetos, sino que también comprendan su contexto dentro de una escena.

Gracias a los avances en aprendizaje profundo, la segmentación de imágenes ha alcanzado niveles de precisión sin precedentes, lo que ha revolucionado diversas áreas de la visión por computador. Estas técnicas han permitido analizar imágenes de forma más precisa, adaptándose a diferentes contextos y necesidades específicas.

En este campo se distinguen tres enfoques principales de segmentación:

1. **Segmentación Semántica:** Asigna una etiqueta a cada píxel de la imagen, de modo que todas las regiones pertenecientes a una misma categoría, como "cielo" o "agua", se etiqueten de manera uniforme. Es crucial para tareas de análisis de escenas y reconocimiento de patrones.
2. **Segmentación de Instancias:** Identifica instancias individuales de una categoría, diferenciando entre objetos similares dentro de una escena, como varios vehículos en una imagen de tráfico.
3. **Segmentación Panóptica:** Integra la segmentación semántica y de instancias, proporcionando una visión completa que incluye tanto las áreas de fondo como las instancias individuales. Este enfoque es ideal para entornos complejos donde se necesita una representación detallada y contextual de la escena, aunque a menudo requiere mayores recursos computacionales.

En la Fig. 1 se puede ver una comparativa de los distintos tipos de segmentación que pueden darse en visión por computador.

(a) (b) (c) (d)

Fig. 1. Tipos de segmentación: (a) Imagen original, (b) segmentación semántica, (c) segmentación de instancias y (d) segmentación panóptica. Tomada de [1]

En definitiva, la visión por computador, potenciada por el aprendizaje profundo, ha ampliado significativamente las posibilidades de segmentación de imágenes. La capacidad de los sistemas para interpretar imágenes de manera precisa y contextual no solo revoluciona el análisis visual automatizado, sino que también abre nuevas oportunidades para resolver problemas complejos en múltiples sectores.

1.2 Aplicaciones de la visión por computador y de la segmentación de imágenes

La visión por computador tiene un amplio abanico de aplicaciones que abarcan diferentes sectores, transformando industrias y ofreciendo soluciones innovadoras a problemas complejos. A continuación, se destacan algunas de las aplicaciones más relevantes y su impacto en diversas áreas:

- **Medicina y Diagnóstico por Imágenes**

En el ámbito médico, la visión por computador ha demostrado ser una herramienta clave en el diagnóstico y análisis de imágenes médicas. Los sistemas automatizados pueden analizar radiografías**,** tomografías computarizadas (CT)**,** resonancias magnéticas (MRI) **y** ecografías, entre otros, con el objetivo de detectar enfermedades y condiciones anómalas. Por ejemplo, los modelos de visión por computador pueden identificar tumores o lesiones en tejidos humanos, como se ejemplifica en la Fig. 2, ayudando a los radiólogos a realizar diagnósticos más rápidos y precisos.

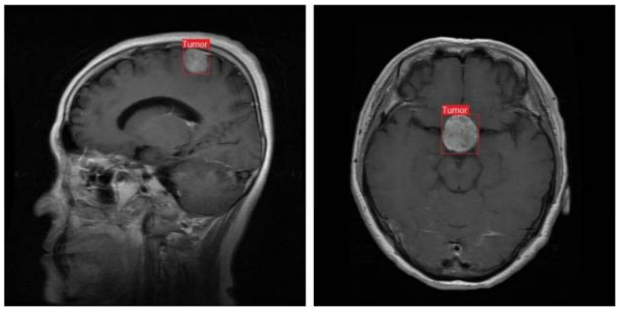


Fig. 2. Detección de un tumor mediante visión por computador. Tomada de [2]

Además, la visión por computador se utiliza en el análisis de imágenes histológicas y en la detección temprana de enfermedades degenerativas, como el Alzheimer o el Parkinson, a partir de imágenes cerebrales. Así mismo, las herramientas basadas en visión por computador también son fundamentales en la cirugía asistida por computador, donde los cirujanos pueden realizar procedimientos con más precisión mediante el uso de imágenes en tiempo real.

- **Vehículos Autónomos**

Uno de los mayores avances de la visión por computador ha sido su integración en los vehículos autónomos. Estos vehículos utilizan cámaras y sensores para capturar imágenes y datos de su entorno, que luego son procesados para realizar tareas cruciales como la detección de objetos, el reconocimiento de señales de tráfico, y la navegación en carreteras.

La visión por computador permite que los vehículos autónomos identifiquen y clasifiquen objetos como otros vehículos, peatones**,** señales de tráfico u obstáculos en el camino, lo que es fundamental para tomar decisiones en tiempo real. Este tipo de tecnología tiene el

potencial de reducir los accidentes de tráfico, mejorar la eficiencia de los sistemas de transporte y permitir la creación de *smart cities* más seguras y sostenibles. En la Fig. 3 se representa una posible aplicación de la visión por computador enfocado a la monitorización del tráfico.

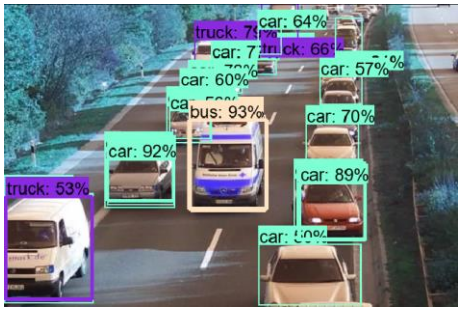


Fig. 3. Detección y reconocimiento de vehículos de una escena urbana. Tomada de [3]

- **Industria y Automatización**

Relacionado con la segmentación semántica, la visión por computador juega un papel crucial en la automatización de procesos de fabricación. Desde el control de calidad de productos en una línea de ensamblaje hasta la determinación del estado de las piezas de maquinaria, la visión por computador permite realizar tareas repetitivas con alta precisión y reduciendo la intervención humana en procesos de poco valor añadido.

Por ejemplo, los sistemas basados en visión pueden detectar defectos de fabricación **(**Fig. 4**)** como grietas, arrugas o imperfecciones en la superficie de productos, lo que permite realizar ajustes en tiempo real en el proceso de producción. Además, se puede utilizar en la gestión de inventarios, el seguimiento de productos y la optimización de la logística en almacenes y centros de distribución.

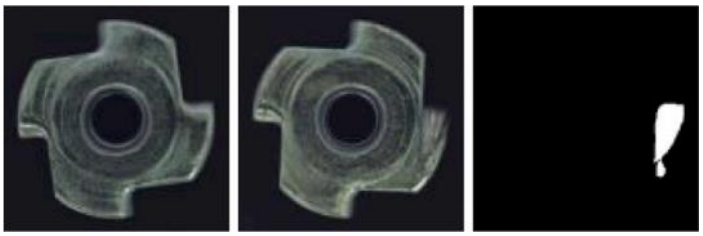


Fig. 4. Detección y segmentación de defectos en piezas industriales. Tomada de [4]

- **Agricultura Inteligente y robótica**

La visión por computador está teniendo también un gran impacto en el sector primario, donde se utiliza para monitorizar cultivos, detectar plagas y evaluar la salud de las plantas y/o ganado. Por ejemplo, a través de imágenes aéreas obtenidas con drones y satélites, los agricultores pueden obtener información precisa sobre el estado de sus cultivos, lo que les permite tomar decisiones informadas sobre riego, fertilización y cosecha.

Otro ejemplo de aplicación es el sistema **Agronav** (Fig. 5), un marco de navegación autónoma para robots agrícolas que utiliza segmentación semántica y detección semántica

de líneas. Este sistema permite a los robots autónomos navegar con precisión a través de campos agrícolas, identificar áreas cultivadas y realizar tareas como la siembra o cosecha de manera más eficiente, reduciendo la necesidad de intervención humana y mejorando la sostenibilidad de las operaciones agrícolas.

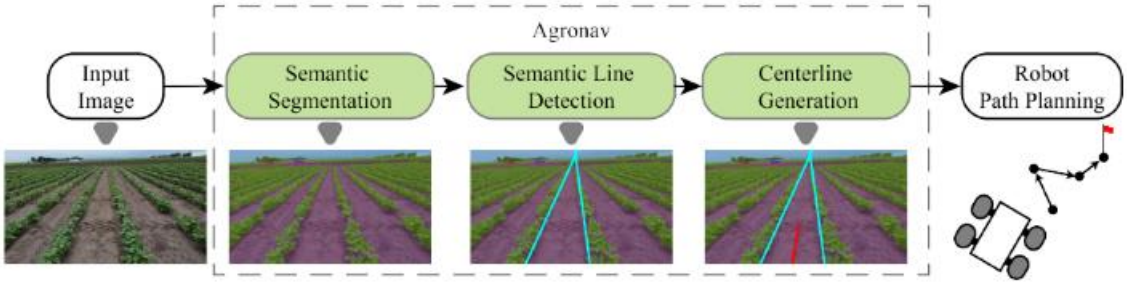


Fig. 5. Flujo de procesamiento de Agronav, consistente en segmentación semántica, detección semántica de líneas y generación de la línea central de la trayectoria. Tomada de [5]

1.3 Justificación, objetivos y estructura del TFM

El propósito principal de este trabajo es implementar técnicas avanzadas de segmentación semántica para la identificación y análisis de grietas en imágenes de superficies. Las grietas representan una de las formas más comunes de deterioro en infraestructuras y materiales de construcción y su detección temprana es fundamental para prevenir daños graves que podrían comprometer la seguridad estructural, incrementar los costes de mantenimiento y afectar a la vida útil de las estructuras. Sin embargo, la segmentación automática de grietas plantea desafíos importantes debido a factores como la diversidad de superficies, la calidad variable de las imágenes y las condiciones de iluminación.

Este estudio se justifica en la necesidad de desarrollar un sistema automatizado que no solo detecte grietas de forma precisa y eficiente, sino que también sea robusto frente a variaciones en los datos. Tal sistema permitiría mejorar significativamente la inspección y monitorización de infraestructuras, incrementando la precisión en la identificación de defectos y reduciendo los costos asociados con las inspecciones manuales, que suelen ser lentas, costosas y propensas a errores.

El **objetivo principal** de este trabajo es ajustar, mediante fine-tuning, y evaluar modelos basados en **Segment Anything Model (SAM)** para realizar la **segmentación semántica de grietas en imágenes de materiales de construcción**. Este enfoque busca aprovechar la capacidad de SAM para realizar segmentaciones generales y adaptarlo específicamente a la detección de grietas, evaluando tanto su eficacia como su capacidad de generalización.

Para alcanzar este fin se han definido los siguientes objetivos específicos:

- **Investigar la influencia de los componentes de SAM:** Analizar cómo cada componente del modelo contribuye a la precisión y calidad de la segmentación, identificando áreas clave para su ajuste.

- **Estudiar la capacidad de aprendizaje en datasets específicos:** Ajustar el modelo a un conjunto de datos específico y comparar su rendimiento con el modelo base sin ajustes para evaluar las mejoras obtenidas mediante el fine-tuning.
- **Evaluar la capacidad de generalización:** Analizar cómo el modelo ajustado se comporta al segmentar grietas en datasets con características similares y diferentes al conjunto de entrenamiento, lo que permitirá medir su robustez frente a la variabilidad en los datos.
- **Analizar el desempeño en un dataset combinado:** Evaluar la capacidad del modelo ajustado para realizar segmentaciones precisas en un dataset que incluya una diversidad de materiales y tipos de grietas, garantizando su aplicabilidad en escenarios reales.

Este enfoque permite cubrir tanto los aspectos teóricos como prácticos del ajuste de SAM para aplicaciones específicas en el análisis de grietas, sentando las bases para futuras investigaciones en el ámbito de la inspección automatizada de infraestructuras.

El resto del trabajo se estructura de la siguiente manera:

- **Capítulo 2:** Se presenta una revisión exhaustiva de la literatura previa relacionada con la segmentación de imágenes.
- **Capítulo 3:** Se describen en detalle las herramientas utilizadas, los métodos adoptados, la arquitectura del modelo de segmentación y los datasets empleados en el estudio. También se describen las métricas de evaluación empleadas en la valoración de los resultados obtenidos.
- **Capítulo 4:** Se detallan los experimentos realizados, incluyendo los procedimientos seguidos, los resultados cuantitativos obtenidos, así como imágenes de las segmentaciones realizadas.
- **Capítulo 5:** Finalmente, se resumen las conclusiones más relevantes del estudio, destacando los logros alcanzados y las limitaciones identificadas. Además, se proponen líneas de investigación futuras.

Esta estructura garantiza un análisis completo y riguroso, abordando tanto los aspectos técnicos como las implicaciones prácticas de los resultados obtenidos, con el fin de contribuir al avance en la automatización de procesos de inspección y monitorización de infraestructuras.

2 Estado del arte

La segmentación de imágenes es una disciplina clave en la visión por computador, cuyo propósito es dividir una imagen en regiones homogéneas para identificar objetos o áreas de interés. Este proceso resulta crucial en aplicaciones tan variadas como el análisis médico, la conducción autónoma, el control de calidad en procesos industriales, la agricultura de precisión, etc.

A lo largo del tiempo, los métodos de segmentación han experimentado una evolución notable. Los enfoques clásicos, como la umbralización y la detección de bordes, establecieron los fundamentos de la disciplina, pero enfrentaron importantes limitaciones al trabajar con imágenes complejas, ruidosas o con baja resolución. La incorporación del aprendizaje automático marcó un punto de inflexión al permitir una mayor precisión y adaptabilidad mediante el uso de algoritmos supervisados y no supervisados. Sin embargo, el verdadero avance se produjo con la llegada del aprendizaje profundo o *Deep learning*, especialmente a través de las Redes Neuronales Convolucionales (CNN), que revolucionaron el campo al automatizar la extracción de características directamente desde los datos, eliminando la necesidad de diseñarlas manualmente.

Más recientemente, los *Vision Transformers* (ViT) han emergido como una herramienta poderosa en visión por computador, permitiendo modelar relaciones globales en las imágenes con mayor eficacia. Esta tecnología ha llevado la segmentación a nuevos niveles de precisión y eficiencia. Un ejemplo destacado de esta tendencia es el modelo *Segment Anything Model* (SAM)**,** basado en ViT, que ha demostrado una capacidad notable para abordar tareas de segmentación semántica, adaptándose a diferentes dominios y necesidades.

En este capítulo, se explorará la evolución de las técnicas de segmentación de imágenes, analizando los métodos más relevantes y su transición hacia enfoques modernos basados en aprendizaje profundo.

2.1 Métodos clásicos de segmentación

Los métodos clásicos de segmentación han sido ampliamente utilizados debido a su simplicidad y bajo coste computacional. Estos enfoques se basan en principios matemáticos y estadísticos para dividir una imagen en regiones homogéneas o identificar objetos relevantes y pueden agruparse en tres categorías principales: **umbralización**, **técnicas basadas en frontera** y **técnicas basadas en regiones**.

La umbralización (Fig. 6) es un método ampliamente utilizado en segmentaciones sencillas de imágenes, el cual clasifica los píxeles en dos o más categorías según un valor umbral. Este valor puede determinarse mediante diversas estrategias. Una de las más comunes es la observación del histograma de la imagen (Fig. 7), donde los picos y valles permiten identificar posibles regiones o clases. Alternativamente, existen métodos

automáticos para seleccionar el umbral, como el algoritmo de **Otsu** [6], que optimiza la segmentación maximizando la separación entre clases al minimizar la varianza intraclase.

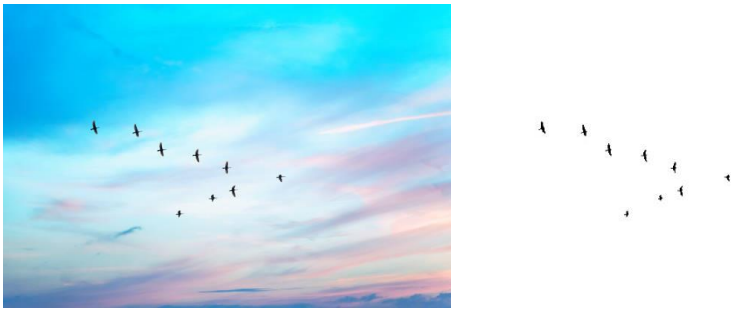


Fig. 6. Segmentación mediante umbralización

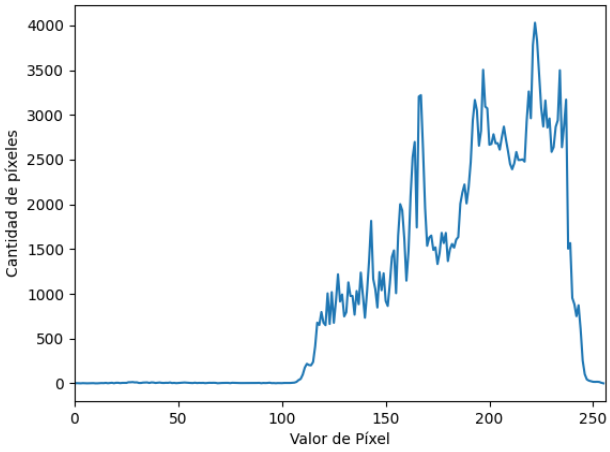


Fig. 7. Histograma de la Fig. 6

Por otro lado, las **técnicas basadas en frontera** se centran en identificar los contornos o límites que separan las distintas regiones dentro de una imagen, facilitando la segmentación precisa de las áreas de interés. Estos enfoques se basan en detectar discontinuidades significativas en las propiedades de la imagen, como cambios abruptos en la intensidad, el color o la textura de los píxeles.

Una de las aproximaciones más comunes para la segmentación basada en frontera es la **detección de bordes**, cuyo objetivo es localizar las transiciones más marcadas entre regiones que difieren notablemente. Para ello, se utilizan varios **operadores de detección de bordes**, cada uno con características y aplicaciones particulares. Entre los más conocidos se encuentran los operadores de **Sobel** y **Laplace**, así como el algoritmo de **Canny**.

- **Operador Sobel** [7]: Este operador se utiliza principalmente para detectar bordes en imágenes al calcular el gradiente de intensidad en las direcciones horizontal y vertical. Se basa en la convolución de la imagen con un filtro específico que resalta las áreas donde ocurren cambios significativos en la intensidad de los píxeles. Es muy eficaz para detectar bordes gruesos y es ampliamente utilizado debido a su simplicidad y rapidez de cálculo. Sin embargo, puede ser sensible al ruido en la imagen, lo que podría generar bordes falsos si no se aplica un preprocesamiento adecuado, como un suavizado previo.

- **Operador Laplaciano**: Se utiliza para detectar bordes a través de la segunda derivada de la imagen, resaltando las áreas con cambios rápidos en la intensidad de los píxeles. A diferencia del operador Sobel, que calcula gradientes en una dirección específica, el Laplaciano detecta bordes en todas las direcciones. Sin embargo, debido a su naturaleza, tiende a ser más sensible al ruido. A menudo se combina con un operador de suavizado previo para mejorar los resultados.
- **Canny** [8]: Es uno de los métodos más avanzados y populares para la detección de bordes. Se basa en una serie de etapas: primero, aplica un suavizado Gaussiano para reducir el ruido; luego calcula el gradiente de la imagen usando el operador Sobel para detectar las zonas de mayor cambio. Posteriormente, realiza una histéresis para eliminar bordes débiles que no estén conectados a bordes fuertes. El resultado es un conjunto de bordes bien definidos y menos sensibles al ruido.

Una vez identificados los bordes, es necesario aplicar algoritmos adicionales para conectar y refinar las líneas detectadas, con el objetivo de formar contornos cerrados que delimiten de manera precisa las regiones de interés. Este proceso de refinamiento es crucial, ya que los bordes detectados inicialmente suelen estar fragmentados o incompletos. Además, los gradientes y las diferencias de intensidades juegan un papel fundamental, permitiendo mejorar la definición de los bordes y hacer más preciso el delineado de las regiones. El uso de gradientes, por ejemplo, resalta las zonas con mayor variación en la intensidad de los píxeles, lo que ayuda a discernir los límites correctos de las regiones.

En la Fig. 8 se ilustra un ejemplo del proceso de segmentación mediante técnicas basadas en frontera, donde primero se identifican los bordes y luego se rellenan las regiones delimitadas por esos bordes.

Fig. 8. De izquierda a derecha, imagen original en escala de grises, imagen tras la detección de borde e imagen segmentada final

Otro enfoque clásico son las **técnicas basadas en regiones**, que plantean la segmentación de imágenes mediante la identificación de regiones homogéneas que comparten características similares como la intensidad, el color o la textura. A diferencia de las técnicas basadas en bordes, que dependen de las discontinuidades en la imagen, los

métodos basados en regiones buscan agrupar píxeles contiguos que cumplan ciertos criterios de homogeneidad.

Uno de los enfoques más comunes es el **crecimiento de regiones** [9]. En este método, se seleccionan uno o varios píxeles iniciales, llamados semillas, agrupando los píxeles vecinos que cumplen con un criterio de similitud, como una diferencia de intensidad o color dentro de un umbral determinado. Este proceso se repite de manera recursiva, expandiendo la región seleccionada a medida que se incorporan píxeles que cumplen con el criterio de homogeneidad. Como ilustración, en la Fig. 9 se muestra un ejemplo de segmentación basada en crecimiento de regiones, donde se seleccionan dos píxeles semilla (marcados con un cuadro) y se agrupan los píxeles que tienen una diferencia de intensidad menor a 10, resultando en dos regiones separadas del fondo.

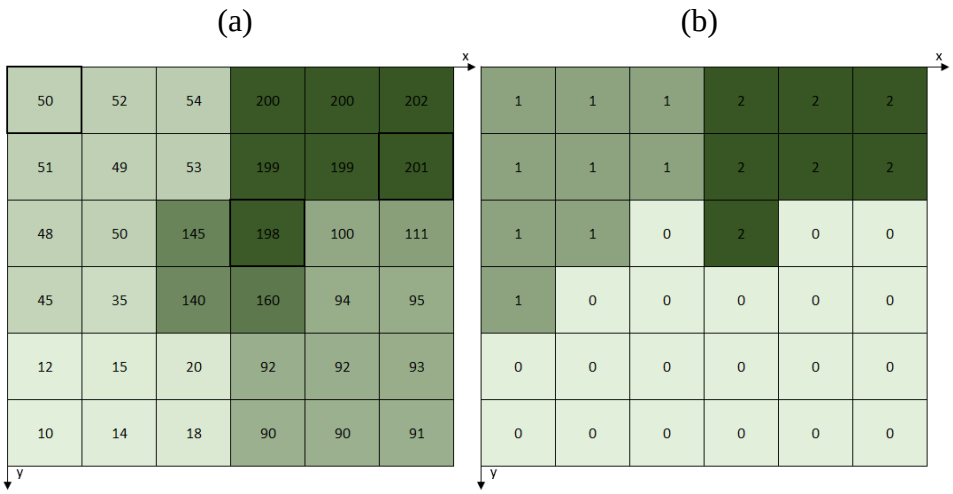


Fig. 9. Ejemplo de segmentación basado en crecimiento de regiones simple con dos semillas (píxeles con marco). (a) valores de intensidad originales de la imagen, (b) segmentación de la imagen con dos regiones más el fondo respectivamente

Otra variante del crecimiento de regiones es el **crecimiento e inauguración de regiones**, el cual no requiere una selección inicial de semillas. En este caso, la segmentación se lleva a cabo recorriendo fila por fila la imagen y, en función de un criterio de similitud, se van expandiendo las regiones o, si no se cumple el criterio, se inicia una nueva región. Este proceso recursivo permite segmentar toda la imagen, dividiendo el espacio en regiones homogéneas.

Además, surgieron otras técnicas como el **quadtree** y la **segmentación por división y fusión de regiones** [10]. El **quadtree** es un método que divide la imagen en cuatro subregiones de manera iterativa, evaluando si cada subregión cumple con un criterio de homogeneidad, como color o textura. Si la región es homogénea, se mantiene sin dividir; de lo contrario, se subdivide en cuatro subregiones más pequeñas. Este proceso continúa hasta alcanzar un nivel de homogeneidad satisfactorio. Un ejemplo del método quadtree se ilustra en la Fig. 10, donde la imagen se divide iterativamente en subregiones según las características de homogeneidad.

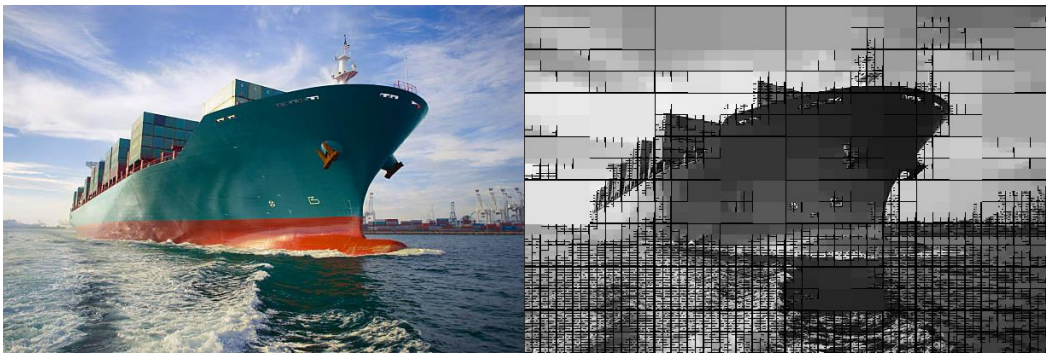


Fig. 10. Ejemplo de segmentación usando la técnica quadtree

Por otro lado, la **segmentación por división y fusión de regiones** comienza dividiendo la imagen en pequeñas regiones homogéneas según un criterio determinado. Luego, fusiona las regiones resultantes si son lo suficientemente similares, lo que permite obtener una segmentación de mayor nivel o más global. Este enfoque combina la eficiencia de dividir la imagen en segmentos más manejables con la flexibilidad de fusionar aquellos que son coherentes entre sí, logrando una segmentación más precisa y adaptada a las características de la imagen.

2.2 Métodos basados en aprendizaje automático

A medida que las imágenes digitales aumentaban en complejidad, los métodos clásicos de segmentación mostraban limitaciones para abordar casos en los que las regiones no presentan bordes claros, los histogramas eran ambiguos o las características de homogeneidad no eran evidentes. En este contexto, los enfoques basados en aprendizaje automático surgieron como herramientas potentes para la segmentación de imágenes, aprovechando su capacidad para modelar relaciones complejas y adaptarse a datos variados. Entre estos métodos, los algoritmos de agrupamiento como **K-means** y los métodos supervisados como **Random Forest** o las **Máquinas de Vectores Soporte (SVM)** destacaron por su eficacia y versatilidad.

El **Random Forest** [11] es un método de aprendizaje automático supervisado basado en el uso de múltiples árboles de decisión para realizar tareas de clasificación, regresión y segmentación. El funcionamiento del Random Forest en tareas de segmentación semántica se ejemplifica en la Fig. 11 y se resume en los siguientes pasos:

1. **Extracción de características**: Cada píxel de la imagen se describe mediante un vector de características que captura atributos relevantes como intensidad, color, textura o incluso descriptores más complejos como SIFT o HOG. Estas características constituyen la base para que el modelo pueda diferenciar entre clases.
2. **Entrenamiento del modelo**: Durante el entrenamiento, se generan múltiples árboles de decisión. Cada árbol se construye utilizando un subconjunto aleatorio de los datos de entrenamiento y de las características, lo que introduce diversidad en el conjunto de árboles. Esta aleatoriedad contribuye a la capacidad de generalización del modelo. Los nodos de cada árbol dividen los datos basándose

en reglas simples (como umbrales de características), de manera que los píxeles con características similares acaben en el mismo grupo o clase.

3. **Clasificación y segmentación**: Para clasificar un píxel, se pasa su vector de características por todos los árboles del bosque. Cada árbol realiza una predicción, y la clase final se decide por votación mayoritaria entre los árboles. Este proceso genera un mapa de segmentación en el que cada píxel recibe una etiqueta de clase.

Los Random Forest presentan diversas ventajas, como su robustez frente al ruido y datos incompletos gracias a la combinación de múltiples árboles, su flexibilidad para trabajar con características heterogéneas sin necesidad de normalización exhaustiva y su simplicidad en comparación con métodos más sofisticados como las redes neuronales. Además, son capaces de modelar relaciones no lineales complejas entre las características de los píxeles. Sin embargo, también tienen limitaciones como su dependencia de la calidad de las características extraídas, lo que exige un diseño manual significativo, su complejidad computacional al manejar grandes bosques en imágenes de alta resolución, y su limitada precisión en tareas que requieren altos niveles de detalle, especialmente en bordes.

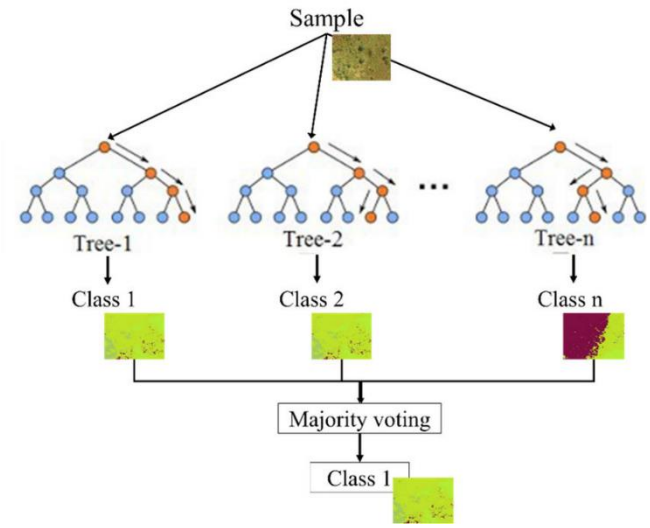


Fig. 11. Flujo de decisión de Random Forest con 3 árboles. Tomada de [12]

Por otro lado, el algoritmo **K-means** [13] es un método de agrupamiento no supervisado ampliamente utilizado en tareas de clasificación. Este método clasifica los píxeles de una imagen en k grupos o clústeres basándose en sus características, como intensidad, color o textura. El proceso de segmentación con K-means sigue los siguientes pasos:

1. **Inicialización**: Se seleccionan k centroides iniciales, que representan los centros de los clústeres.
2. **Asignación de clúster**: Cada píxel se asigna al clúster cuyo centroide esté más cercano, basándose en una métrica como la distancia euclidiana.
3. **Reajuste de centroides**: Los centroides se actualizan como el promedio de las características de los píxeles asignados a cada clúster.
4. **Iteración**: Se repiten los pasos 2 y 3 hasta que los centroides converjan o se alcance un criterio de parada.

Aunque es eficiente para imágenes sencillas, K-means presenta limitaciones en imágenes con ruido o regiones poco definidas, ya que no considera las relaciones espaciales entre píxeles. Así mismo, el resultado puede variar en función de la ubicación de inicio de los centroides. En la Fig. 12 se puede ver un ejemplo de k-means, donde se representa la iteración inicial y la final.

(a)

(b)

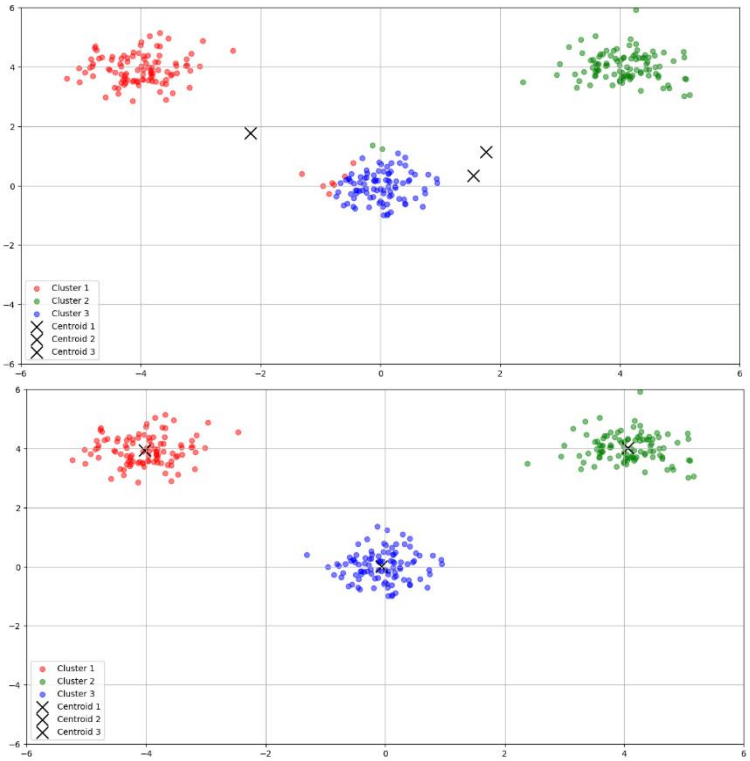


Fig. 12. Representación gráfica de los puntos a clasificar y de los centroides. (a) Iteración inicial. (b) Iteración final

Otro método de aprendizaje automático destacado es la **Máquina de Vectores Soporte (SVM)** [14]. Este es un método de aprendizaje automático supervisado, principalmente enfocado a problemas de clasificación y regresión, aunque puede ser aplicado a segmentación de imágenes. Son especialmente útiles en tareas donde se requiere separar datos en dos o más clases basándose en un conjunto de características. El objetivo de una SVM es encontrar un hiperplano óptimo que separe los datos de diferentes clases con el mayor margen posible. Este margen es la distancia entre el hiperplano y los puntos más cercanos de cada clase, conocidos como vectores soporte (Fig. 13). En el caso de segmentación de imágenes el procedimiento sería el siguiente:

1. **Extracción de Características**: Cada píxel de la imagen se convierte en un vector de características, que puede incluir información como intensidad, color, textura o cualquier otro atributo relevante para distinguir las clases de interés. Estas características forman un espacio de representación en el que se basará la clasificación.
2. **Entrenamiento del Modelo**: Utilizando un conjunto de datos etiquetados, el SVM aprende un modelo que construye un hiperplano óptimo para separar las clases presentes en el espacio de características. Este proceso maximiza el margen entre las clases, permitiendo una distinción más robusta.

3. **Clasificación de Píxeles**: Una vez entrenado, el modelo aplica el hiperplano aprendido para asignar una etiqueta de clase a cada píxel de la imagen, basándose en la posición de su vector de características respecto al hiperplano. Esto genera una segmentación en la que cada píxel pertenece a una de las clases definidas.

Los SVM son particularmente útiles para la segmentación de imágenes cuando las clases están claramente separadas, gracias a su capacidad para maximizar el margen entre ellas. Además, son efectivos para manejar datos en espacios de alta dimensionalidad sin requerir grandes volúmenes de datos etiquetados. Sin embargo, presentan limitaciones significativas en escenarios complejos, como imágenes con clases altamente solapadas, bordes ambiguos o distribuciones no lineales, donde no logran segmentaciones precisas de manera consistente.

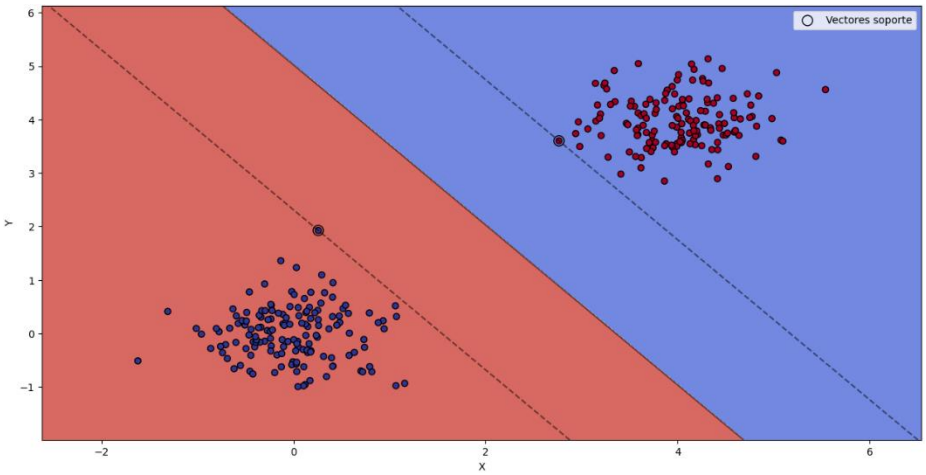


Fig. 13. Ejemplo de SVM con 2 conjuntos

2.3 Métodos basados en aprendizaje profundo

Los métodos tradicionales de segmentación de imágenes presentaban limitaciones en escenarios complejos debido a su dependencia de características manualmente diseñadas y su rendimiento limitado en condiciones como bordes difusos o regiones solapadas.

Sin embargo, los avances en el campo del aprendizaje profundo [15] han superado estas restricciones mediante el uso de redes neuronales, en particular las **Redes Neuronales Convolucionales (CNN).** Las CNN, inicialmente diseñadas para la clasificación de imágenes, son capaces de aprender automáticamente representaciones jerárquicas de los datos a través de sus capas convolucionales, eliminando la necesidad de diseñar características manualmente. Estas redes se han mostrado altamente efectivas en la extracción de características espaciales, lo que las convierte en una opción ideal para tareas de clasificación y segmentación de imágenes complejas.

Un avance fundamental en la segmentación de imágenes mediante CNN fue la introducción de las **Redes Completamente Convolucionales (FCN)** [16]. Estas redes adaptaron las CNN tradicionales para realizar segmentación semántica, generando predicciones a nivel de píxel en lugar de una única predicción para toda la imagen. La clave de las FCN es que eliminan las capas totalmente conectadas, presentes en las CNN

(Fig. 14) y las reemplazan por capas convolucionales, permitiendo que el modelo trabaje con entradas de cualquier tamaño y produzca salidas espaciales con resoluciones proporcionales a las de entrada, permitiendo así la tarea de segmentación. La estructura de una FCN se compone de los siguientes elementos:

1. **Bloques convolucionales iniciales:** Estas capas son similares a las de una CNN tradicional y se encargan de extraer características jerárquicas de la imagen. Aplican convoluciones para detectar bordes, texturas y patrones, además de usar funciones de activación como ReLU para introducir no linealidad. El tamaño espacial de la imagen disminuye debido a las convoluciones y el uso de operaciones de pooling.
2. **Capas deconvolucionales (o transpuestas):** Después de extraer las características, las FCN recuperan la resolución espacial original mediante deconvoluciones o upsampling. Estas operaciones amplían las dimensiones espaciales de los mapas de características. Aprenden filtros para interpolar las características, ajustando la predicción al tamaño original de la imagen.
3. **Skip Connections (conexiones de salto):** Una característica importante de las FCN es la introducción de conexiones de salto entre las capas profundas (tienen mayor abstracción) y las capas superficiales (tienen mayor resolución). Esto ayuda a refinar los detalles en los bordes y mejorar la segmentación en áreas complejas.
4. **Salida final (mapa de segmentación):** La última capa genera un mapa de probabilidad donde cada píxel está etiquetado según la clase a la que pertenece. Utiliza funciones de activación como softmax para problemas de múltiples clases. Esto da como resultado un mapa de etiquetas de igual resolución que la imagen de entrada.

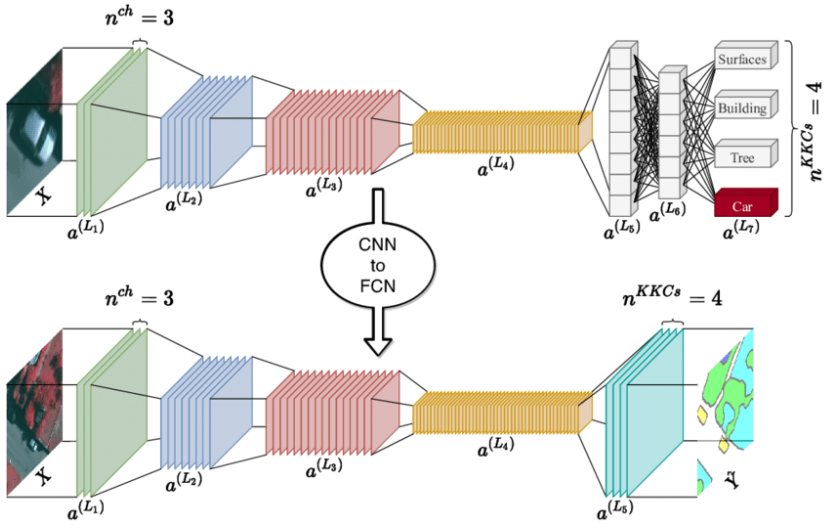


Fig. 14. Diferencia entre una CNN y una FCN. Tomada de [17]

Otra de las arquitecturas que surgieron fue **U-Net** [18], basada en las Redes Completamente Convolucionales (FCN). Su diseño se caracteriza por una estructura en forma de "U" (Fig. 15), que consta de dos fases principales: la **fase de codificación** (o

contracción) y la **fase de decodificación** (o expansión). A continuación, se describen los componentes estructurales de U-Net:

1. **Fase de Codificación (Contracción):** La fase de codificación está diseñada para extraer características jerárquicas de la imagen mediante capas convolucionales y de pooling. A medida que la imagen pasa a través de esta fase, la red va extrayendo características de más alto nivel, pero también pierde detalles espaciales debido al proceso de downsampling.

2. **Fase de Decodificación (Expansión):** Una vez que se han extraído las características relevantes, la fase de decodificación tiene como objetivo recuperar la resolución espacial original de la imagen para generar la segmentación a nivel de píxel. Esta etapa se realiza mediante capas deconvolucionales, las cuales aumentan las dimensiones espaciales de los mapas de características, invirtiendo el proceso de reducción realizado en la fase de codificación. Las deconvoluciones aprenden filtros que permiten recuperar detalles espaciales y mejorar la resolución de las características.

3. **Skip Connections (Conexiones de salto):** Una de las innovaciones clave de **U-Net** son las skip connections que permiten fusionar información de capas profundas (con características más abstractas) con capas más superficiales (con mayor resolución espacial). Estas conexiones permiten que los detalles finos que se pierden durante el proceso de downsampling en la fase de codificación se conserven y se integren nuevamente en el proceso de decodificación.

4. **Salida Final (Mapa de Segmentación):** La última capa de la red es responsable de generar un mapa de segmentación con la misma resolución que la imagen de entrada. En lugar de usar una capa totalmente conectada, U-Net utiliza una última convolución para producir un mapa de características donde cada píxel se clasifica en una de las clases de interés. Dependiendo del problema (segmentación binaria o multiclase), se aplica una función de activación como softmax para problemas multiclase o sigmoid para segmentación binaria, asegurando que cada píxel esté etiquetado correctamente según su clase.

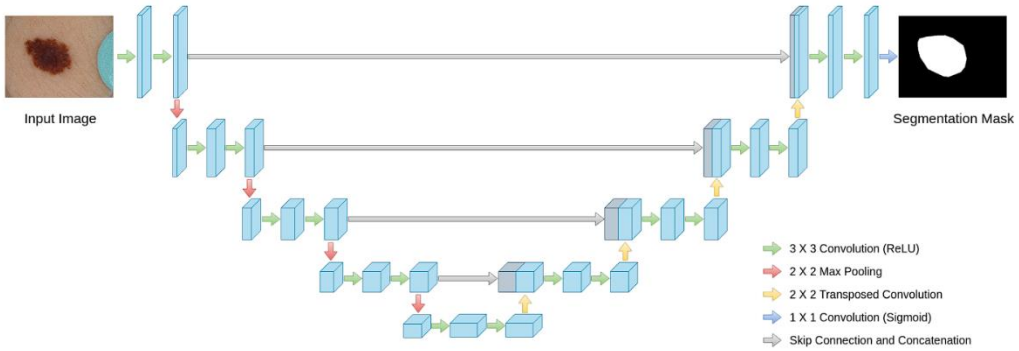


Fig. 15. Diagrama de la arquitectura de U-NET. Tomada de [19]

Otro modelo importante es **Mask R-CNN** [20], una extensión de la arquitectura Faster R-CNN**,** que fue diseñada para la detección de objetos. Mientras que Faster R-CNN es

capaz de predecir *bounding boxes* para objetos, Mask R-CNN lleva esta capacidad un paso más allá al permitir la segmentación de instancias de objetos con máscaras a nivel de píxel. A continuación, se describen los principales componentes de esta arquitectura:

1. **Backbone Network**: Es una red neuronal convolucional preentrenada que actúa como extractor de características. Procesa la imagen de entrada para generar un mapa de características que se utilizará en las etapas posteriores.
2. **Region Proposal Network (RPN)**: genera regiones de interés (RoI) propuestas en la imagen. La RPN utiliza un enfoque de deslizamiento sobre el mapa de características para proponer regiones que podrían contener objetos, utilizando anclas (*anchors*) de diferentes tamaños y aspectos.
3. **RoI Align**: Una mejora crucial sobre RoI Pooling de Faster R-CNN. RoI Align asegura una alineación precisa entre las características y la región propuesta, eliminando el redondeo que podría causar inexactitudes en la segmentación. Este paso es fundamental para la generación de máscaras precisas.
4. **Red de Clasificación y Regresión de Bounding Boxes**: Cada región propuesta pasa por una red que realiza las siguientes tareas:

    - **Clasificación**: Determina qué clase de objeto existe dentro de la región propuesta.
    - **Regresión de Bounding Boxes**: Ajusta la posición y tamaño del bounding box para mejorar la precisión de la localización del objeto.
    - **Máscara de Segmentación**: Una vez que se ha clasificado una región, se pasa a través de una tercera rama dedicada a la generación de máscaras. Esta rama es una FCN que predice una máscara binaria de alta resolución para cada clase. La salida es una máscara para cada clase, pero solo se toma la máscara de la clase predicha por la rama de clasificación.

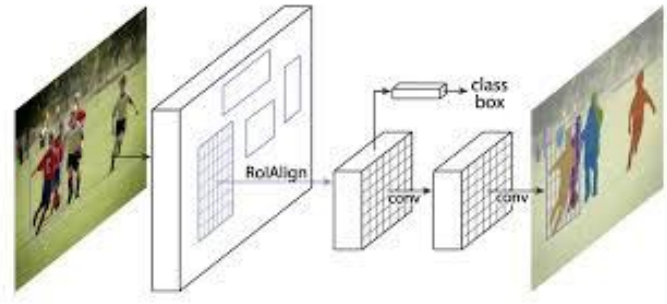


Fig. 16. Diagrama de la arquitectura MAsk R-CNN. Tomada de [20]

Otra contribución significativa ha sido **DeepLab** [21], una familia de arquitecturas desarrolladas para la segmentación semántica, que se destaca por su uso de convoluciones dilatadas o *atrous*, las cuales permiten aumentar el campo receptivo sin perder resolución espacial. Este enfoque es útil para capturar contextos más amplios sin perder detalles finos en las fronteras de las regiones segmentadas. Además, DeepLab utiliza atención espacial para mejorar la capacidad de la red para segmentar áreas con bordes difusos o regiones con texturas complejas. A continuación, se describe la estructura básica de DeepLab (Fig. 17) y sus componentes clave:

1. **Backbone Network**: DeepLab emplea una red neuronal convolucional preentrenada para extraer características de la imagen de entrada. Estas redes preentrenadas se modifican para integrar **convoluciones *atrous*** en sus últimas capas, lo que permite mantener la resolución espacial a la vez que amplía el campo receptivo.
2. **Convoluciones dilatadas o *atrous***: Las convoluciones dilatadas son una de las características distintivas de DeepLab. Este tipo de convolución aumenta el campo receptivo de la red sin incrementar el número de parámetros ni reducir la resolución espacial de las imágenes. En lugar de realizar la convolución de forma estándar, los filtros se aplican con una expansión de los píxeles, lo que permite que cada filtro cubra un área más amplia de la imagen. Esto es esencial para capturar relaciones de largo alcance y contextos globales sin perder la precisión en las características locales.
3. ***Atrous Spatial Pyramid Pooling* (ASPP)**: Una de las innovaciones clave en DeepLab es la introducción de **ASPP**, que aplica varias convoluciones dilatadas con diferentes tasas de dilatación en paralelo. Además, se incluye una operación de **pooling global**, lo que permite que el modelo capture información de contexto a diferentes escalas simultáneamente. ASPP no solo mejora la capacidad de la red para identificar patrones a múltiples resoluciones, sino que también ayuda a extraer información más relevante de áreas de la imagen con diferentes tamaños y características, lo que aumenta la robustez y precisión de la segmentación.
4. **Capa de Salida**: La última capa de DeepLab es una capa convolucional final que genera un mapa de características con tantas dimensiones como el número de clases de segmentación deseadas. Esta capa asigna a cada píxel de la imagen una probabilidad de pertenecer a una clase específica, lo que da como resultado un mapa de segmentación donde cada píxel tiene una etiqueta correspondiente a su clase. Dependiendo del problema de segmentación (binaria o multiclase), se puede utilizar una función de activación como softmax (para segmentación multiclase) o sigmoid (para segmentación binaria) para generar las probabilidades finales.

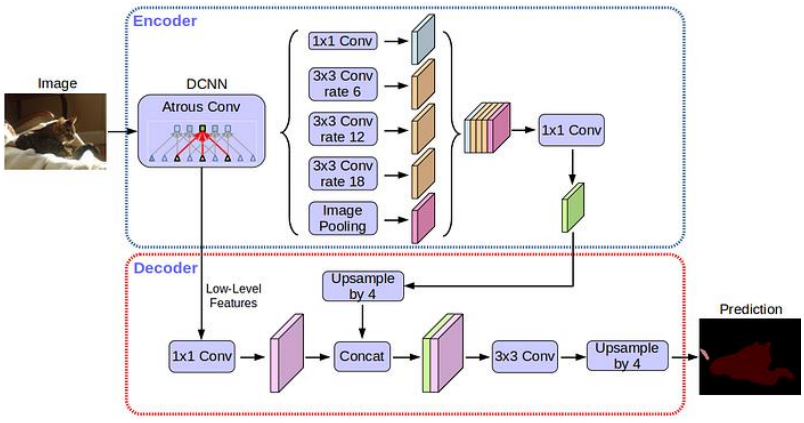


Fig. 17. Esquema de DeepLab. Tomada de [22]

Las versiones más recientes de DeepLab, como **DeepLabV3** y **DeepLabV3+** [23], mejoran la precisión y eficiencia en la segmentación semántica mediante el uso de redes preentrenadas y técnicas de normalización y regularización avanzadas, como Batch

Normalization y Dropout. Estas mejoras contribuyen a evitar el sobreajuste y a estabilizar el entrenamiento.

Recientemente, la introducción de **Vision Transformer (ViT)** [22] ha aportado un enfoque innovador a la clasificación de imágenes, así como modelos derivados para segmentación semántica. A diferencia de las CNN, que trabajan con píxeles locales mediante convoluciones, ViT divide la imagen en parches (*patches*) y los procesa como una secuencia utilizando Transformers [24], una arquitectura que originalmente fue desarrollada para procesamiento de lenguaje natural. A continuación, se describe su estructura básica:

1. **División en Parches (*patch embedding*)**: La imagen de entrada se divide en parches de tamaño fijo, típicamente 16x16 píxeles. Cada parche se aplana y se transforma mediante una capa lineal, generando un vector de características de dimensión fija. Esto es similar a los *word embeddings* en NLP, donde cada palabra se representa como un vector.
2. ***Positional Embedding***: Dado que los Transformers no tienen una noción inherente de orden espacial (a diferencia de las CNN), se añaden *positional embeddings* a los vectores de los parches para que el modelo pueda capturar la posición relativa de cada parche en la imagen.
3. **Transformer (Encoder)**: ViT utiliza un encoder basado en Transformers, compuesto por múltiples bloques de atención y redes feed-forward:

    - **Self-Attention**: Cada parche puede "atender" a todos los demás parches de la imagen, lo que permite al modelo capturar dependencias globales y relaciones entre parches.
    - **Multi-Head Attention**: Se utilizan múltiples cabezas de atención para capturar diferentes aspectos de las interacciones entre los parches.
    - **Feed-Forward Networks (FFN)**: Cada capa de atención es seguida por una red feed-forward que aplica transformaciones no lineales a las salidas de la atención.
    - **Layer Normalization y Residual Connections**: Estas técnicas se emplean para mejorar la estabilidad del entrenamiento y permitir el paso fluido de los gradientes a través de la red.
4. **Clase Token (CLS Token)**: Un token especial, conocido como el **CLS Token**, se añade al inicio de la secuencia de parches. Este token se utiliza para la clasificación. Después de pasar por las capas del encoder, la representación final de este token se considera la representación global de la imagen para la tarea de clasificación.
5. **Capa de Clasificación**: Finalmente, la representación del CLS Token se pasa a través de una capa lineal para producir las probabilidades de las clases, lo que permite clasificar la imagen en una categoría.

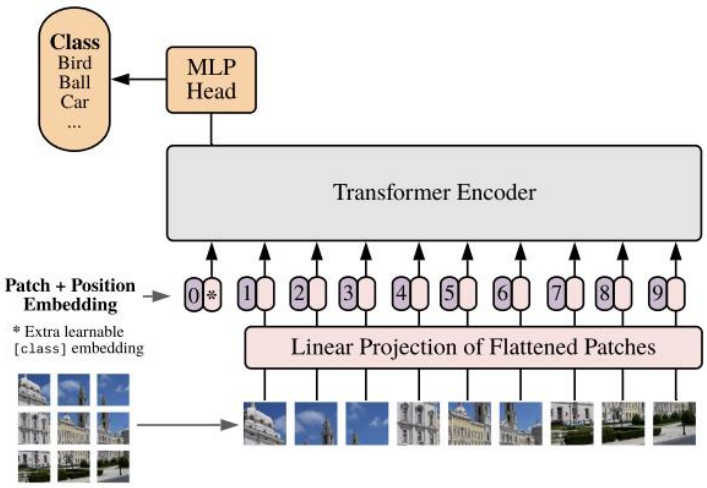


Fig. 18. Esquema de los Vision Transformers. Tomada de [22]

El modelo **Vision Transformer (ViT)** ha sido una de las arquitecturas más innovadoras para tareas de clasificación de imágenes y su éxito en este campo ha servido de base para su adaptación a otras tareas, incluida la segmentación de imágenes. Su capacidad para aprender relaciones globales entre los parches de una imagen a través de atención ha demostrado ser particularmente poderosa en la comprensión de contextos más amplios, lo cual es crucial para tareas de segmentación a nivel de píxel.

En este contexto, un avance significativo en la segmentación semántica ha sido **Segment Anything Model (SAM)**, un modelo propuesto por Meta AI que busca facilitar la segmentación de cualquier objeto en imágenes, independientemente de su contexto. SAM emplea una arquitectura basada en ViT que permite la segmentación rápida y precisa, sin necesidad de un entrenamiento específico para cada tipo de objeto o clase, lo cual lo hace muy versátil. Este modelo se describirá en detalle en el capítulo 3.

A lo largo de este capítulo se han revisado las principales arquitecturas y enfoques en segmentación de imágenes, desde los métodos tradicionales hasta los avances más recientes basados en redes neuronales profundas, como U-Net, Mask R-CNN y DeepLab. Estas técnicas han sido fundamentales para mejorar la precisión y eficiencia en tareas complejas de segmentación. Sin embargo, la constante evolución del campo ha dado lugar a modelos más avanzados que buscan simplificar y generalizar aún más el proceso de segmentación. En este sentido, Segment Anything Model (SAM) emerge como un modelo innovador que promete revolucionar el enfoque tradicional, permitiendo la segmentación de cualquier objeto en cualquier imagen de manera más accesible y adaptable. En los siguientes capítulos se profundizará en las características y capacidades de SAM, explorando las fortalezas y limitaciones de esta arquitectura.

3 Materiales y métodos

En este capítulo se describen las herramientas, métricas y recursos empleados a lo largo del desarrollo de este trabajo, enfocado en el fine-tuning de Segment Anything Model (SAM) para la segmentación de grietas en materiales de construcción.

El modelo SAM es una arquitectura avanzada basada en aprendizaje profundo, diseñada para tareas de segmentación de objetos en imágenes con un alto grado de generalización. En este trabajo, se adapta y personaliza este modelo mediante técnicas de fine-tuning, para mejorar su desempeño en un dominio específico: la identificación precisa de grietas en superficies de hormigón y mármol.

Se emplean tres conjuntos de datos principales: dos de ellos contienen imágenes de grietas en hormigón, mientras que el tercero está compuesto por imágenes de grietas en mármol. Estos datasets fueron seleccionados para abarcar una variedad de texturas, patrones y condiciones de iluminación, garantizando que el modelo fuera entrenado con datos representativos de los escenarios reales en los que se aplicará.

Además, se detallan las herramientas seleccionadas para este trabajo, que incluyen frameworks de aprendizaje automático, bibliotecas especializadas en manipulación y visualización de datos e imágenes, así como los entornos de desarrollo en los que se llevaron a cabo los experimentos. Estas herramientas no solo facilitaron la implementación y evaluación del modelo, sino que también garantizaron la reproducibilidad y escalabilidad de los resultados.

Finalmente, se introducen las métricas utilizadas para evaluar el desempeño del modelo en las tareas de segmentación. Estas métricas son esenciales para cuantificar la precisión y la efectividad del modelo en la identificación de grietas, permitiendo realizar comparaciones objetivas entre diferentes configuraciones y optimizaciones.

3.1 Frameworks, librerías y entornos de desarrollo

El desarrollo de este trabajo se basó en el uso de frameworks robustos y ampliamente reconocidos y utilizados en el ámbito del aprendizaje profundo y la visión por computador, proporcionando las bases necesarias para implementar y personalizar el modelo SAM. Estas herramientas facilitaron las tareas de entrenamiento, evaluación y análisis de resultados, asegurando un enfoque sistemático y reproducible en todas las etapas del proyecto.

Un aspecto clave del proceso fue la manipulación de imágenes, fundamental para el procesamiento de datos en modelos de visión por computador. Este paso incluyó tareas como la carga, transformación y adaptación de las imágenes a las especificaciones necesarias para la segmentación, garantizando que los datos fueran interpretados y procesados de manera eficiente por el modelo. Además, el manejo adecuado de los datos

fue esencial para estructurar y presentar la información de manera óptima durante el entrenamiento y la evaluación, lo que permitió maximizar el rendimiento del modelo.

Asimismo, los entornos de desarrollo utilizados desempeñaron un papel crucial en la implementación y prueba de las soluciones. Estos entornos ofrecieron herramientas integradas para la gestión de código, la ejecución de experimentos y la visualización de resultados, facilitando un flujo de trabajo eficiente y organizado.

Para alcanzar estos objetivos, se emplearon una serie de frameworks, bibliotecas y entornos de desarrollo especializados que se detallan a continuación, destacando sus principales características y el papel que desempeñaron en el desarrollo de este trabajo:

- **PyTorch**: Desarrollado inicialmente por Meta AI, se utilizó como el framework principal para la implementación, entrenamiento y ajuste fino del modelo SAM. Su flexibilidad y soporte para arquitecturas modernas, como los Vision Transformers (ViT), fueron fundamentales para abordar las complejas tareas de segmentación. Además, PyTorch facilita la gestión de datos mediante tensores y ofrece una integración eficiente con la aceleración GPU, lo que optimiza el rendimiento del modelo durante el entrenamiento y la inferencia.
- **MONAI**: Es una biblioteca especializada en el desarrollo de soluciones de inteligencia artificial para el ámbito de la salud, diseñada para facilitar la creación de modelos de aprendizaje automático en tareas de segmentación médica. En este proyecto, MONAI se utilizó para la definición de la función de pérdida, específicamente la función DiceCELoss (1), que combina el coeficiente de Dice (2) y la pérdida de entropía cruzada (3), proporcionando una métrica eficaz para segmentación en contextos con clases desequilibradas. Esta función de pérdida fue crucial para optimizar el rendimiento del modelo SAM en la tarea de segmentación de grietas.

$$DiceCELoss = \alpha \cdot DiceLoss + (1 - \alpha) \cdot CrossEntropyLoss \quad (1)$$

$$DiceLoss = 1 - \frac{2\sum_i p_i g_i}{\sum_i {p_i}^2 + \sum_i {g_i}^2} \quad (2)$$

$$CrossEntropyLoss = -\frac{1}{N}\sum_i [g_i \log(p_i) + (1 - g_i)\log(1 - p_i)] \quad (3)$$

Donde,

- α: parámetro de ponderación que equilibra la contribución de cada pérdida
- p_i: valor predicho para el píxel i
- g_i: valor real del píxel i
- N: número total de píxeles

- **Transformers:** Se utilizó la librería Transformers de la empresa Hugging Face para integrar el modelo preentrenado Segment Anything Model (SAM), basado en la arquitectura Vision Transformer (ViT), con el fin de realizar tareas de

segmentación de imágenes. Esta librería facilitó tanto la carga del modelo SAM y el procesamiento de entradas como los puntos de referencia y bounding boxes, optimizando el flujo de trabajo del modelo en tareas de segmentación.

- **Scikit-learn:** Es una biblioteca de Python utilizada para tareas de aprendizaje automático y procesamiento de datos. En este proyecto, scikit-learn se empleó para dividir el conjunto de imágenes en subconjuntos de entrenamiento, validación y test.
- **Pillow**: E**s** una biblioteca de Python para el procesamiento y manipulación de imágenes, la cual proporciona herramientas para realizar operaciones como redimensionado, recorte, conversión de formato, y otras tareas esenciales para preparar las imágenes para la segmentación. En este proyecto, se utilizó Pillow principalmente para cargar y preprocesar las imágenes para adaptarlas al formato de entrada requerido por el modelo SAM.
- **NumPy**: Es una biblioteca fundamental para la computación científica en Python, la cual proporciona soporte para matrices y matrices multidimensionales, así como una colección de funciones matemáticas y lógicas para realizar operaciones eficientes en estos arreglos. En este proyecto, se utilizó NumPy principalmente para la manipulación y el procesamiento de datos numéricos relacionados con las imágenes, como el manejo de tensores de píxeles, operaciones matemáticas sobre las imágenes preprocesadas y la conversión de datos entre diferentes formatos, facilitando la integración con otros frameworks y el modelo SAM.
- **Pandas**: Es una biblioteca de Python utilizada principalmente para el manejo y análisis de datos estructurados. En este proyecto, se utilizó Pandas para la organización, limpieza y preprocesamiento de los datasets antes de alimentarlos al modelo SAM.
- **Datasets:** Es una librería de Python que proporciona un conjunto de herramientas para cargar, preprocesar y gestionar grandes volúmenes de datos. En este proyecto, se utilizó Hugging Face Datasets para importar y manejar los datos a través de la clase Dataset y DatasetDict. Estas clases facilitaron la carga y organización de los conjuntos de datos, permitiendo estructurar los datos de manera eficiente para su uso en el entrenamiento y evaluación del modelo SAM.

Los entornos de desarrollo juegan un papel fundamental en la implementación y ejecución de modelos complejos como SAM. En este caso, se utilizaron varias plataformas que facilitaron el desarrollo y la ejecución de los experimentos:

- **Google Colab**: Fue la plataforma principal utilizada para llevar a cabo los experimentos de entrenamiento e inferencia de SAM. Su integración con los recursos de Google Cloud permitió acceder a potentes GPUs, como las NVIDIA Tesla T4 y A100, lo que facilitó el entrenamiento de modelos de gran tamaño y la ejecución de tareas computacionales intensivas de manera eficiente. Además, Google Colab ofrece un entorno de trabajo colaborativo y remoto, lo que posibilita compartir y ejecutar código en la nube sin necesidad de configurar hardware local.
- **Google Drive**: Plataforma de almacenamiento y gestión de archivos en la nube. Dado que los datasets, resultados de experimentos y modelos entrenados pueden ocupar grandes cantidades de espacio, Google Drive ofreció una solución

accesible y segura para almacenar estos recursos. Además, su integración con Google Colab permitió gestionar y cargar los archivos de forma eficiente, facilitando el acceso a datos y modelos sin la necesidad de descargarlos constantemente al entorno local. Esto resultó fundamental para realizar experimentos que requerían acceder a grandes volúmenes de información de manera fluida.

- **PC (local)**: El ordenador personal (PC) se utilizó como medio principal para interactuar con el entorno de desarrollo. A través de este dispositivo, se gestionaron los archivos, se escribieron los scripts y se supervisaron los experimentos. El PC facilitó la interacción directa con Google Colab y Google Drive, permitiendo la carga de archivos y la gestión de los datos en el entorno de trabajo remoto. Dado que la ejecución de los experimentos se realizó aprovechando los recursos computacionales disponibles en Google Colab, no se describirán las características técnicas del PC.

3.2 Segment Anything Model

SAM (Segment Anything Model) es un modelo fundacional de segmentación desarrollado por Meta con el objetivo de abordar una amplia variedad de tareas de segmentación en imágenes de manera universal. Este modelo representa un avance significativo en el campo de la visión por computador, ya que ofrece una solución versátil y de alto rendimiento para la segmentación de objetos en imágenes, independientemente del contexto, dominio o nivel de complejidad visual.

Lo que hace que SAM sea especialmente valioso es su capacidad para generar máscaras de segmentación de alta precisión a partir de entradas mínimas proporcionadas por el usuario. Estas entradas, conocidas como *prompts*, pueden incluir puntos de interés, bounding boxes o incluso realizar una segmentación total de la imagen. Por ejemplo, un usuario podría seleccionar un punto o realizar un recuadro en el centro de un objeto y el modelo generaría automáticamente una máscara que segmenta el objeto completo, como se ejemplifica en la Fig. 19. Esta flexibilidad interactiva permite al modelo adaptarse a las necesidades del usuario, segmentando con precisión tanto objetos claramente delimitados como aquellos más difusos o ambiguos.

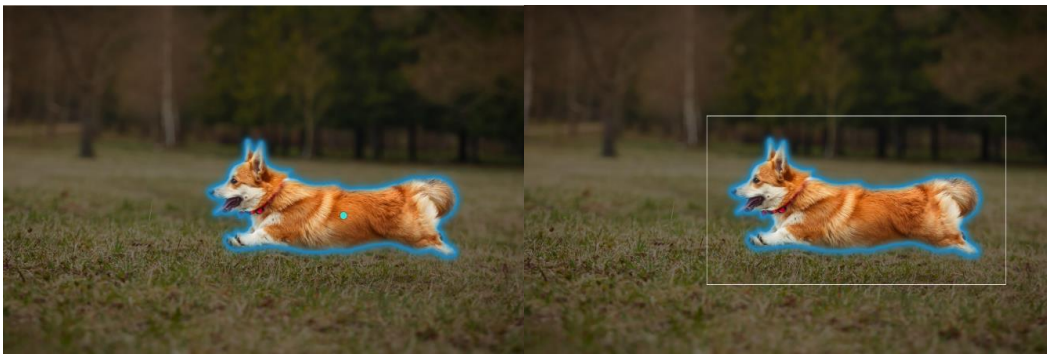


Fig. 19. De izquierda a derecha, segmentación mediante un punto y mediante un bounding box de una imagen del repositorio oficial de la demo de SAM [25]

Además, SAM ha sido diseñado para ser aplicable de manera generalizada, lo que significa que puede trabajar con una amplia variedad de dominios y escenarios sin

requerir un entrenamiento adicional. Su entrenamiento masivo, llevado a cabo con un conjunto de datos extremadamente amplio y diverso le otorga una capacidad de generalización excepcional. Esto lo posiciona como una herramienta avanzada y versátil, adecuada tanto para aplicaciones académicas como industriales. En el contexto de la visión por computador, SAM marca un hito en la creación de modelos generalistas que eliminan la necesidad de desarrollar soluciones específicas para cada tarea de segmentación, abriendo nuevas posibilidades para aplicaciones rápidas y flexibles.

Sin embargo, a pesar de su versatilidad, el rendimiento de SAM puede verse limitado en contextos altamente especializados. En dominios técnicos donde las características del objeto a segmentar son sutiles o difíciles de identificar, como en la detección de grietas en materiales donde incluso los operadores humanos experimentan dificultades para realizar segmentaciones consistentes. En estos casos, es necesario realizar un ajuste del modelo mediante técnicas de *fine-tuning* para adaptarlo a las características específicas del dominio y optimizar su desempeño.

A lo largo de los próximos apartados, se describirán en detalle el dataset de entrenamiento, la arquitectura y los componentes clave de SAM, destacando las características técnicas que lo convierten en una herramienta innovadora para tareas de segmentación.

3.2.1 Dataset de entrenamiento

El rendimiento de **SAM (Segment Anything Model)** se ve profundamente influenciado por el conjunto de datos con el que ha sido entrenado. Para garantizar que SAM sea robusto y versátil en una amplia gama de tareas de segmentación, el modelo fue entrenado utilizando el conjunto de datos **SA-1B**. Este dataset fue diseñado para proporcionar imágenes diversas y anotaciones de segmentación de alta calidad, lo que le permite abordar tareas complejas y variadas sin la necesidad de un entrenamiento específico para cada dominio.

El conjunto de datos **SA-1B** es uno de los más grandes y completos en cuanto a imágenes y anotaciones de segmentación. Algunas de sus características clave son las siguientes:

- **Tamaño:** El conjunto de datos SA-1B contiene 11 millones de imágenes de alta resolución, acompañadas de 1.1 billones de máscaras de segmentación.
- **Alta resolución**: Las imágenes de SA-1B son de alta resolución, con un tamaño promedio de 3300 x 4950 píxeles, aunque hay disponibles versiones con resoluciones menores debido a las limitaciones de almacenamiento y accesibilidad.
- **Diversidad de imágenes:** Las imágenes del conjunto cubren una amplia variedad de escenas, desde objetos cotidianos hasta escenas más concretas como aquellas de ámbito industrial, científico y médico. Esta diversidad ayuda a SAM a generalizar mejor a diferentes tipos de imágenes y contextos.
- **Máscaras de alta calidad**: Se estima que el 94% de las máscaras tienen una IoU superior al 90% con las máscaras corregidas por anotadores profesionales.

- **Protección de la privacidad**: Las imágenes utilizadas en SA-1B fueron obtenidas bajo licencia y con derechos de privacidad, como la difuminación de caras y matrículas de vehículos en las imágenes publicadas.

3.2.2 Arquitectura de SAM

El modelo SAM se basa en una arquitectura modular, flexible y eficiente que le permite abordar tareas de segmentación con alta precisión. Sus componentes clave son: el Image Encoder, el Prompt Encoder y el Mask Decoder. A continuación, se describen estos módulos en detalle.

El **Image Encoder** es responsable de extraer las características visuales relevantes de la imagen de entrada. En lugar de usar redes convolucionales tradicionales, SAM emplea un Vision Transformer (ViT-B en este trabajo) como backbone, lo que le permite capturar relaciones espaciales globales mediante mecanismos de atención, a diferencia de las CNNs que tienden a centrarse en características locales.

El Vision Transformer divide la imagen en subimágenes y procesa cada una de ellas de forma independiente mientras aprende las dependencias globales entre las partes utilizando mecanismos de atención. Este enfoque es más eficiente que las CNNs tradicionales para tareas que requieren una comprensión global de la imagen. Cada subimagen se convierte en un vector que alimenta al Transformer, lo que permite aprender las relaciones entre ellas. Como resultado, el Image Encoder crea un mapa de características que describe la imagen de manera abstracta, identificando patrones complejos y estructuras espaciales.

Por ejemplo, si SAM recibe una imagen de una grieta en un bloque de hormigón, el image encoder dividirá la imagen en partes. A través del mecanismo de atención del Transformer, el modelo puede identificar cómo la grieta se extiende a lo largo de la pared, reconociéndola incluso si tiene bordes irregulares o está parcialmente oculta.

El P**rompt Encoder** procesa las entradas proporcionadas por el usuario, como puntos o bounding boxes y las convierte en representaciones vectoriales que guiarán el proceso de segmentación. Estos prompts hacen que SAM sea un modelo interactivo y flexible, permitiendo que el usuario especifique áreas de interés en la imagen mediante diferentes tipos de entradas.

El **Mask Decoder** es el módulo encargado de generar la máscara de segmentación final. Este componente toma las características visuales extraídas por el image encoder y las representaciones de los prompts procesados por el prompt encoder, combinándolas mediante mecanismos de atención para producir la máscara de segmentación.

En general, SAM genera tres posibles máscaras para cada tarea, junto con sus respectivas puntuaciones de confianza. Esto permite al modelo ofrecer una segmentación robusta, mejorando su capacidad para manejar variaciones complejas en las imágenes y optimizando la precisión según el contexto.

El flujo de datos dentro de SAM sigue una secuencia estructurada tal y como se ilustra en la Fig. 20. Primero, la imagen se procesa a través del image encoder, generando un mapa de características visuales que describe la imagen en un nivel abstracto. Posteriormente, los prompts proporcionados por el usuario se procesan mediante el prompt encoder,

generando representaciones vectoriales que indican la región de interés. Finalmente, el mask decoder toma las características visuales y las representaciones de los prompts para generar una máscara de segmentación precisa.

Este diseño modular no solo facilita la adaptabilidad del modelo, sino que también lo hace escalable permitiendo futuras mejoras o personalizaciones específicas según las necesidades del dominio.

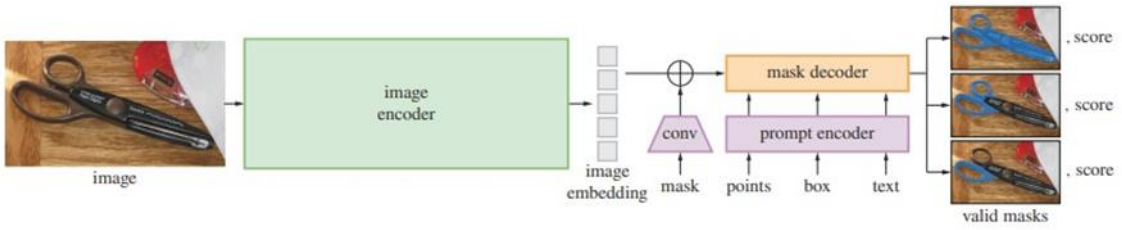


Fig. 20. Estructura y componentes principales de Segment Anything Model (original)

3.2.3 SAM2

Durante el desarrollo de este Trabajo Fin de Máster, META AI introdujo una actualización denominada SAM2. Una versión ampliada que introduce una nueva funcionalidad de segmentación en videos. Esta actualización expande las capacidades de SAM más allá de las imágenes estáticas, permitiendo que el modelo realice segmentaciones precisas y coherentes en secuencias de video. Para alcanzar este objetivo, la arquitectura de SAM2 incorpora algunos cambios significativos, destacando la introducción de componentes relacionados con la memoria, como **memory attention** y **memory encoder (**ver Fig. 21), los cuales permiten al modelo mantener y aprovechar información contextual a lo largo del tiempo en los videos.

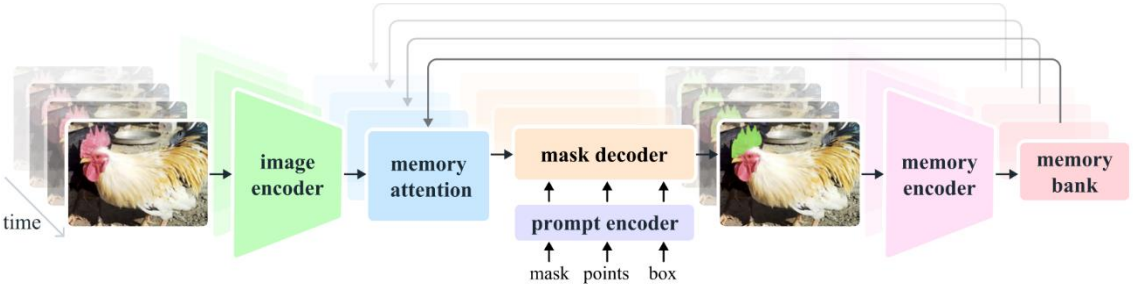


Fig. 21. Estructura y componentes principales de Segment Anything Model 2

3.2.4 Fortalezas y limitaciones de SAM

Una vez introducido el modelo y su arquitectura, es oportuno destacar las principales virtudes e inconvenientes que definen el rendimiento y la aplicabilidad de SAM en tareas de segmentación de imágenes. Respecto a sus fortalezas se puede destacar:

- **Generalización.** SAM destaca por su capacidad de realizar segmentaciones en una gran variedad de objetos y dominios sin requerir entrenamiento específico para cada tarea. Este modelo de segmentación universal puede identificar y segmentar una amplia gama de objetos en las imágenes, desde los más comunes hasta los más complejos. Su entrenamiento en un amplio conjunto de datos le permite abordar tareas de segmentación en diversos contextos de complejidad media o baja.

- **Interactividad**. Una de las principales ventajas de SAM es su interactividad. Los usuarios pueden guiar el proceso de segmentación mediante distintos tipos de prompts, como puntos o bounding boxes. Esto no solo permite que SAM sea útil para tareas de segmentación automatizadas, sino también para aplicaciones que requieren intervención humana para mejorar la precisión. El modelo puede adaptarse a los cambios en tiempo real durante el proceso de segmentación, lo que facilita su uso en entornos dinámicos.

Aunque SAM es un modelo completo y versátil, tiene algunas limitaciones que deben tenerse en cuenta, especialmente cuando se aplica a tareas muy especializadas o contextos que no se ajustan a los datos con los que fue entrenado inicialmente.

- **Rendimiento en tareas especializadas o con datos ruidosos.** Una de las principales limitaciones de SAM es su rendimiento cuando se enfrenta a tareas extremadamente especializada. Aunque SAM es capaz de realizar segmentación de una amplia variedad de objetos, su desempeño puede disminuir en contextos técnicos o en condiciones con características visuales complejas. Estas situaciones presentan un desafío ya que el modelo podría no tener datos representativos de tales características en su conjunto de entrenamiento original.
- **Sensibilidad a los prompts**. SAM depende de las entradas proporcionadas por el usuario, como puntos de referencia o cuadros delimitadores (bounding boxes), para guiar el proceso de segmentación. La colocación precisa de estos prompts es crucial para obtener resultados precisos. Si los prompts no se colocan correctamente o existe variabilidad en cómo el usuario interactúa con el modelo, la precisión de la segmentación puede verse afectada. Esta dependencia puede complicar su implementación en aplicaciones donde se requiera una segmentación totalmente automatizada sin intervención humana.

Estas limitaciones resaltan la importancia de realizar ajustes específicos como el *fine-tuning*, especialmente en contextos complejos o levemente representados en el conjunto de datos original.

3.3 Datasets empleados durante el fine-tuning y la inferencia

Se plantea la realización de varios experimentos para evaluar la capacidad de SAM de ser reentrenado en la tarea específica de segmentación de grietas en materiales de construcción. Para ello se hará uso de 3 datasets diferentes, los cuales se definen a continuación.

3.3.1 Concrete Crack Dataset

El **Concrete Crack Segmentation Dataset** consta de 458 pares de imágenes-anotaciones diseñados para tareas de segmentación semántica binaria a nivel de píxel, clasificando entre dos clases: grieta y fondo. Las imágenes fueron tomadas en diversos edificios de la **Middle East Technical University** (METU) ubicada en Ankara, Turquía [26]. Este conjunto de datos es relevante para la industria de la construcción, ya que permite abordar

problemas críticos relacionados con la inspección estructural. Las principales características del dataset son las siguientes:

- **Número de imágenes:** 458 imágenes de 4032x3024 píxeles
- **Número de objetos etiquetados:** 1.010, pertenecientes a la clase “grieta”
- **Etiquetas:** Anotaciones realizadas manualmente con dos clases: grieta y fondo
- **Ausencia de divisiones predefinidas:** No se incluyen particiones específicas para entrenamiento, validación o prueba
- **Lanzamiento:** Publicado en 2019

En la Fig. 22 se presentan ejemplos de los datos contenidos en el Concrete Crack Segmentation Dataset.

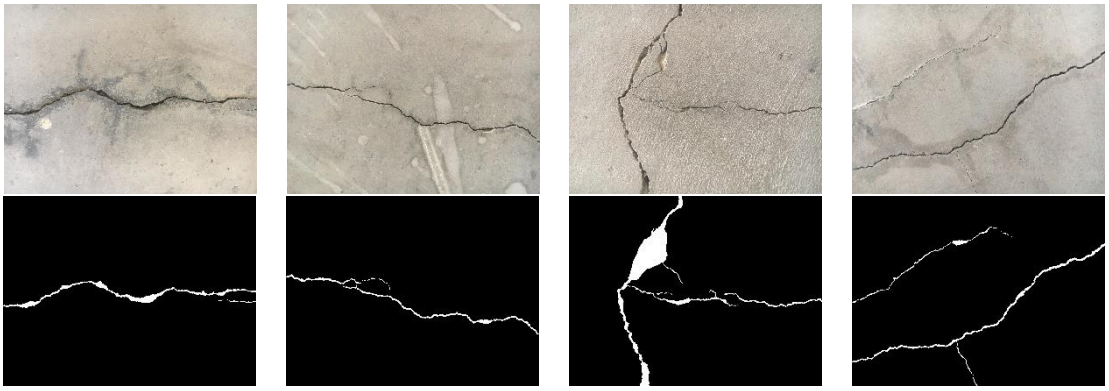


Fig. 22. Ejemplo del Concrete Crack Segmentation dataset. Arriba imágenes RGB y abajo su respectiva máscara verdadera

3.3.2 DeepCrack Dataset

El dataset **DeepCrack** [27], compuesto por imágenes RGB y sus correspondientes máscaras que indican la presencia de grietas en hormigón. Este conjunto de datos está diseñado específicamente para tareas de segmentación semántica e inspección estructural en el ámbito de la construcción, con un enfoque particular en la identificación de grietas a nivel de píxel. Las características más relevantes del dataset se resumen a continuación:

- **Número de imágenes:** 537 imágenes de 544x384 píxeles
- **Etiquetas:** Las anotaciones se realizan a nivel de píxel, con dos clases (grieta-fondo), obtenidas mediante un proceso de anotación manual
- **División predefinida:** El conjunto de datos se divide en dos subconjuntos: uno para entrenamiento (300 imágenes) y otro para prueba (237 imágenes).
- **Lanzamiento:** Publicado en 2019

Es importante recalcar la variedad de texturas, iluminación, tamaño y forma de grieta que se presentan en este dataset (ver Fig. 23) en comparación con Concrete Crack Dataset, donde había una mayor uniformidad de los datos.

Fig. 23. Ejemplo de DeepCrack dataset. Arriba imágenes RGB y abajo su respectiva máscara verdadera

3.3.3 Marble Crack Dataset

El **Marble Crack Segmentation (MCS) dataset** está diseñado para abordar el desafío de la segmentación semántica de grietas en superficies de mármol. Según la Real Academia Española (RAE), el mármol es una piedra caliza metamórfica de textura compacta y cristalina, que puede ser pulida de manera eficiente y es frecuentemente mezclada con otras sustancias que le otorgan colores diversos o forman manchas y vetas.

Las **fisuras** son rupturas superficiales y estrechas que generalmente no comprometen la estabilidad estructural del material, siendo causadas habitualmente por factores como cambios de temperatura o el envejecimiento. Por su parte, las **grietas** son fisuras más profundas que pueden afectar la seguridad y la estabilidad estructural de la superficie, originándose principalmente por problemas como asentamientos del terreno, sobrecargas o un mal procesamiento del material durante su fabricación. Mientras que las fisuras se consideran defectos estéticos y superficiales, las grietas poseen el potencial de comprometer la integridad estructural del material.

Debido a la complejidad estructural del mármol y la presencia de fisuras que, aunque visibles, no afectan la estabilidad del material, la tarea de distinguir entre estas fisuras y las grietas reales se convierte en un desafío significativo en la inspección automatizada. Recalcar que distinguir entre una grieta y una fisura no se encuentra en el alcance de este trabajo, por lo que puede haber anotaciones donde además de grietas se encuentren fisuras, puesto que el dataset no confirma que se componga 100% de las primeras. En la Fig. 24 se muestra una comparativa entre fisura y grieta en un bloque de mármol.

(a) (b)

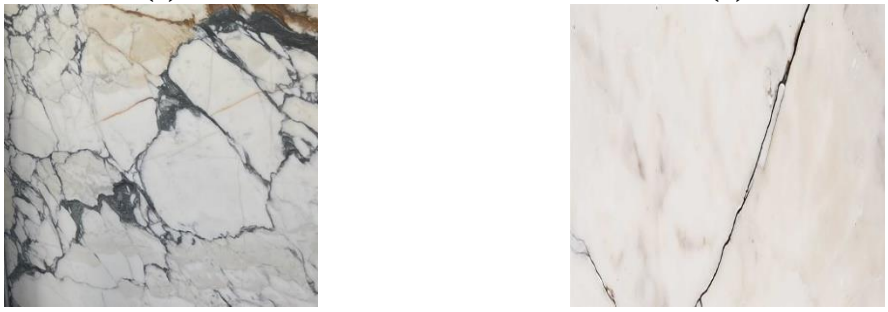


Fig. 24. (a) fisuras en superficie de mármol, (b) grieta en superficie de mármol. Imágenes tomadas de [28] y [29] respectivamente.

Las principales características del **Marble Crack Segmentation (MCS) dataset** son las siguientes:

- **Número de imágenes:** 246 imágenes de 256x256 píxeles
- **Etiquetas:** Anotaciones a nivel de píxel, con dos etiquetas (grieta-fondo), obtenidas mediante anotación manual
- **División predefinida:** Se divide el dataset en 2 conjuntos, uno de entrenamiento y uno de test. En el experimento 3 se usaron todas las imágenes para la inferencia.
- **Lanzamiento:** Este dataset [30] fue generado a partir del original llamado Marble Surface Anomaly Detection-2, publicado en Kaggle por Aman Rastogi, al añadir anotaciones para segmentación semántica de grietas

En definitiva, el MCS dataset puede resultar de gran utilidad en la tarea de segmentación de grietas en mármol, reduciendo la necesidad de dedicar recursos económicos, temporales y humanos en dicha tarea. En la Fig. 25.se muestra un ejemplo de las imágenes y sus respectivas anotaciones.

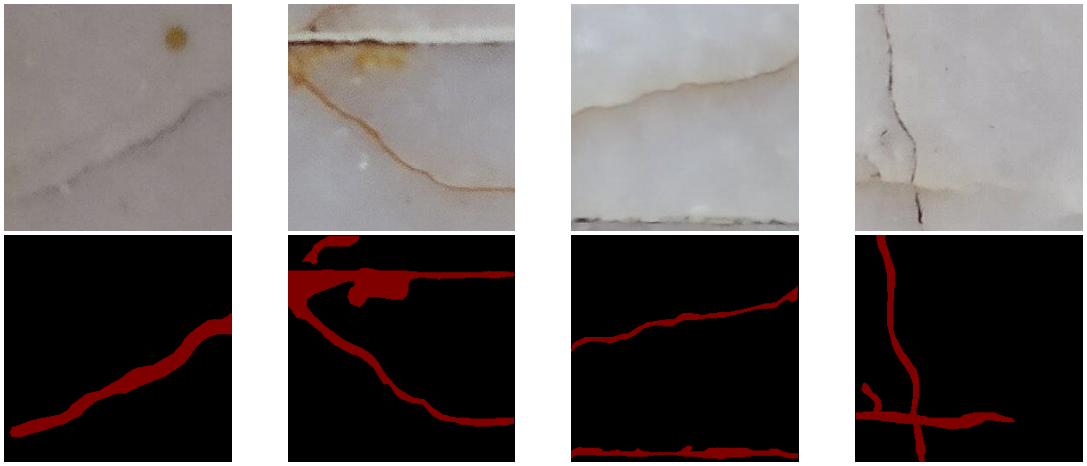


Fig. 25. Ejemplo de Marble Crack Segmentation dataset. Arriba imágenes RGB y abajo su respectiva máscara verdadera

3.4 Métricas de evaluación

Las métricas de evaluación son fundamentales para medir la eficacia del modelo de segmentación, ya que nos permiten comparar las predicciones del modelo con las máscaras verdaderas, también conocidas como *ground truth*. Para calcular estas métricas se utiliza la **matriz de confusión** (Tabla 1), que ofrece cuatro valores clave:

- **Verdadero Positivo (VP)**. Se refiere a los píxeles que han sido correctamente clasificados como pertenecientes a la clase de interés. Es decir, clasificó un píxel perteneciente a una grieta como tal.
- **Falso Positivo (FP)**. Se refiere a los píxeles que han sido incorrectamente clasificados como pertenecientes a la clase de interés, aunque en realidad no deberían. En este caso clasificaría un píxel perteneciente al fondo como grieta.

- **Verdadero Negativo (VN).** Son los píxeles que han sido correctamente identificados como no pertenecientes a la clase de interés, es decir, el modelo correctamente identificó el fondo en el caso de estudio.
- **Falso Negativo (FN).** Son los píxeles que han sido incorrectamente clasificados como no pertenecientes a la clase de interés, aunque en realidad deberían haber sido identificados como parte de ella. Por ejemplo, el modelo clasifica un píxel perteneciente a la grieta como fondo.

Tabla 1. Matriz de confusión de una segmentación binaria

		MÁSCARA VERDADERA	
		OBJETO (1)	FONDO (2)
MÁSCARA PREDICHA	OBJETO (1)	VP	FP
	FONDO (2)	FN	VN

A partir de estos valores, se calculan las principales métricas que permiten evaluar la calidad de la segmentación. A continuación, se describen las métricas utilizadas y Las ecuaciones empleados:

- **Accuracy (exactitud)**. Mide la proporción de predicciones correctas (tanto positivas como negativas) respecto al total de predicciones realizadas por el modelo (4). Aunque es una métrica útil en situaciones balanceadas, su efectividad se ve limitada en datasets desbalanceados, ya que puede generar valores artificialmente altos si la clase de interés es minoritaria frente al fondo (como se verá posteriormente). En tales casos, no se recomienda su uso como métrica principal (como es el caso de estudio).

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN} \quad (4)$$

- **Precision (precisión)**. Mide la proporción de predicciones positivas correctas en relación con todas las predicciones positivas realizadas por el modelo (5). Indica cuántos de los píxeles que el modelo clasificó como parte de la grieta realmente pertenecen a ella.

$$Precision = \frac{VP}{VP + FP} \quad (5)$$

- **Recall (Sensibilidad)**. Mide la proporción de predicciones positivas correctas en relación con todas las muestras que deberían haber sido clasificadas como positivas (6). Es decir, cuántos de los píxeles de grieta que el modelo debería haber detectado realmente ha detectado.

$$Recall = \frac{VP}{VP + FN} \quad (6)$$

- **F1-score**. Es la media armónica entre la precisión y la sensibilidad (7). Esta métrica es útil para conjuntos de datos desequilibrados, ya que penaliza tanto los falsos positivos como los falsos negativos. Un valor alto indica una buena superposición entre la segmentación predicha y la de referencia.

$$F1score = \frac{2 * VP}{2 * VP + FP + FN} \quad (7)$$

- **IoU (Intersection over Union)**. Mide la intersección sobre la unión de la segmentación predicha y la segmentación de referencia (8). Es decir, evalúa el área que se solapa entre la predicción y la verdadera, dividido por el área total cubierta por ambas. Esta métrica es la más restrictiva de las definidas.

$$IoU = \frac{VP}{VP + FP + FN} \quad (8)$$

4 Experimentos y resultados

El presente capítulo detalla la metodología utilizada y los resultados obtenidos al adaptar y evaluar el modelo SAM (Segment Anything Model) en la tarea de segmentación de grietas. Este enfoque explora cómo personalizar modelos fundacionales para aplicaciones específicas, como el análisis de materiales de construcción con el fin de identificar potenciales fallos estructurales. Para ello se formularon preguntas clave que guiaron los experimentos realizados:

1. **¿Qué tan preciso es SAM, ajustado mediante fine-tuning, en segmentar grietas en superficies de hormigón?**
2. **¿Cómo afecta la variabilidad de texturas, iluminación y entorno de aplicación del hormigón procedentes de otro dataset en el rendimiento del modelo ajustado?**
3. **¿Es posible generalizar el aprendizaje a otro tipo de materiales como el mármol?**
4. **¿Puede SAM segmentar grietas en materiales variados si se entrena con un dataset combinado que abarque diferentes morfologías y texturas?**

Para responder estas preguntas, se desarrollaron varios experimentos con ajustes en distintos componentes del modelo: **Mask Decoder**, **Prompt Encoder**, **Image Encoder**, la combinación **Mask Decoder+Prompt Encoder** y el **ajuste completo** (full fine-tuning). Los resultados de cada experimento se compararon con el modelo base sin ajuste, utilizando métricas cuantitativas y evaluaciones cualitativas. Se emplearon varios datasets, tal y como se describió en el apartado 3.3, diseñados para evaluar el rendimiento y la capacidad de generalización del modelo.

Cada experimento siguió una estructura clara y reproducible:

1. **Definición breve del dataset empleado**
2. **Parámetros de entrenamiento y/o inferencia:** Componente entrenado o evaluado, hiperparámetros…
3. **Evaluación cuantitativa:** Se utilizaron métricas como F1-score o IoU para medir la calidad de las segmentaciones generadas.
4. **Evaluación cualitativa:** Se generaron visualizaciones comparativas entre las máscaras generadas por el modelo y el ground-truth, permitiendo interpretar visualmente el rendimiento.

A través de este enfoque, el capítulo busca demostrar las capacidades de SAM para adaptarse a tareas complejas de segmentación en dominios técnicos, así como identificar áreas de mejora y limitaciones. En los apartados siguientes se describen detalladamente los cuatro experimentos realizados, resaltando tanto los logros obtenidos como los desafíos encontrados en cada caso.

4.1 Experimento 1. Fine-tuning sobre Concrete Crack Dataset

Para evaluar el desempeño del modelo SAM en la tarea de segmentación de grietas en hormigón, se empleó inicialmente el **Concrete Crack Dataset**, un conjunto de datos que contiene imágenes de alta resolución de superficies de hormigón con grietas de diferentes tamaños y formas. Este dataset se utilizó tanto para el entrenamiento como para la evaluación del modelo, proporcionando un conjunto diverso de ejemplos representativos de las variaciones que pueden presentarse en escenarios del mundo real.

El objetivo principal del experimento fue estudiar el ajuste de los componentes del modelo de manera que pudiera segmentar con precisión las grietas. Además, el proceso incluyó una etapa de test en la que se evaluó la capacidad del modelo para segmentar nuevas imágenes de grietas en hormigón del mismo dataset. Los resultados obtenidos en esta fase se compararon con las métricas estándar de evaluación de segmentación, lo que permitió obtener una visión detallada del rendimiento del modelo en este contexto específico.

A continuación, se detallan los principales parámetros empleados durante el fine-tuning, así como las configuraciones específicas de entrenamiento utilizadas en el experimento:

- **Tasa de aprendizaje**. Se adoptó una tasa de aprendizaje fija de 10^{-5} para garantizar un ajuste gradual del modelo sin incurrir en oscilaciones o divergencias durante el entrenamiento.
- **Optimizador**. El modelo fue entrenado utilizando el optimizador Adam. Este optimizador se eligió por su capacidad para manejar tasas de aprendizaje pequeñas, adaptándose bien a la estructura del modelo.
- **Función de pérdida**. Para la evaluación de los errores, se utilizó DiceCELoss, una función que combina Dice Loss y Cross-Entropy Loss. Esta combinación es particularmente útil en tareas de segmentación binaria, como en este caso, al abordar el desbalance de clases, donde las grietas representan un porcentaje reducido de los píxeles totales.
- **Número de épocas**. El entrenamiento se llevó a cabo durante 30 épocas, suficiente para garantizar la convergencia del modelo y evaluarlo en un amplio rango.
- **División del dataset**. El dataset se dividió en proporciones estándar: 70% para entrenamiento, 15% para validación, y 15% para test. Esta división permite realizar una evaluación representativa y equilibrada del rendimiento del modelo.
- **Tamaño de lote (batch)**. Dadas las restricciones computacionales del entorno, se estableció un tamaño de lote igual a 2, lo que permitió un entrenamiento eficiente sin exceder la capacidad de memoria disponible.
- **Recursos computacionales**. El proceso de fine-tuning fue implementado en la plataforma Google Colab, aprovechando los recursos de GPU ofrecidos. Se detallará en el apartado 4.5.

4.1.1 Resultados del entrenamiento

En este apartado se discutirán los resultados obtenidos durante el proceso de fine-tuning de SAM con el Concrete Crack Dataset. El análisis se centra en la evolución de la pérdida durante las etapas de entrenamiento y validación, así como en el comportamiento de las métricas de evaluación a lo largo de las épocas.

Los resultados se presentan de forma estructurada en función de los componentes ajustados permitiendo un análisis exhaustivo y comparativo de su rendimiento, siendo estos los siguientes:

- **Mask Decoder**
- **Prompt Encoder**
- **Prompt Encoder + Mask Decoder**
- **Image Encoder**
- **Full Fine-Tuning** (ajuste simultáneo de los tres componentes).

Cada módulo fue evaluado de manera independiente, asegurando que las conclusiones derivadas sean representativas y contribuyan a identificar fortalezas y limitaciones específicas en cada caso. A continuación, se presentan los resultados específicos para cada uno de los componentes ajustados, donde se analizarán en detalle su rendimiento y se discutirá sobre la efectividad de la segmentación.

4.1.1.1 Mask decoder

El análisis del ajuste del Mask Decoder revela una evolución notable en las etapas iniciales del entrenamiento. Como se observa en la Fig. 26, la pérdida de entrenamiento disminuye significativamente durante las primeras 9 épocas, con especial intensidad durante las 5 primeras, disminuyendo posteriormente de forma más leve hasta estabilizarse alrededor de la época 25. Este comportamiento sugiere que la mayor parte del aprendizaje ocurre en las primeras iteraciones, con una velocidad de aprendizaje más reducido en las épocas posteriores. Por otro lado, la pérdida en la validación sigue un patrón distinto. Se aprecia un descenso progresivo hasta la época 5, seguido de una estabilización, lo cual indica que el modelo no mejora a partir de esta época.

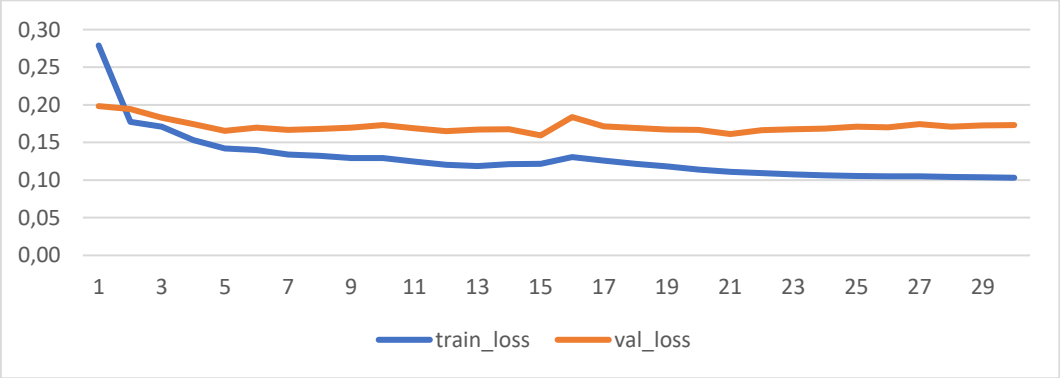


Fig. 26. Evolución de la pérdida durante el ajuste del Mask Decoder (Concrete Crack Dataset)

En cuanto a la evolución de las métricas de evaluación (ver Fig. 27), durante la etapa de entrenamiento se observa un incremento significativo en las primeras 5 épocas. Posteriormente, las métricas muestran una desaceleración en la velocidad de aprendizaje, alcanzando los valores más altos al final del entrenamiento. En contraste, las métricas en la etapa de validación presentan un comportamiento más irregular. Aunque se observa un progreso general, los mejores resultados se alcanzan en un rango amplio, entre las épocas 14 y 22 dependiendo de la métrica evaluada.

Los valores obtenidos en la validación, 69.5% IoU y 81.63% F1-score, evidencian la efectividad del ajuste del Mask Decoder para capturar patrones en las primeras fases del entrenamiento, aun evidenciando áreas de mejora en la estabilidad y consistencia del aprendizaje durante la validación.

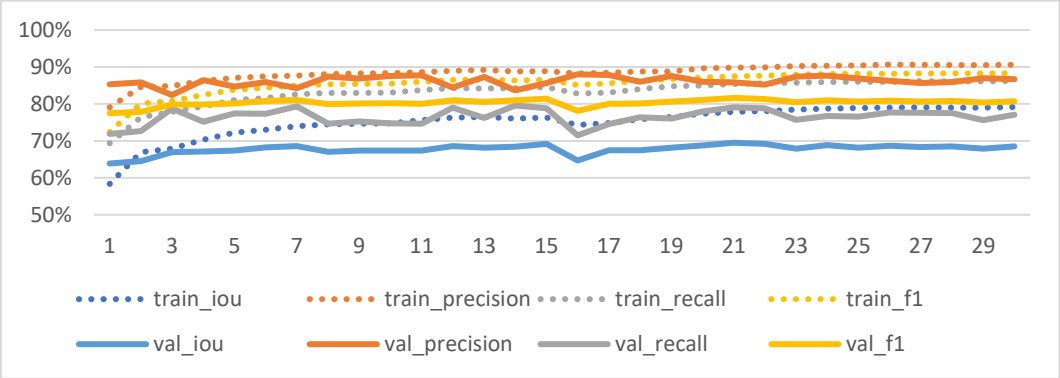


Fig. 27. Evolución de las Métricas durante el entrenamiento-validación del Mask Decoder (Concrete Crack Dataset)

4.1.1.2 Prompt Encoder

El ajuste del Prompt Encoder presenta un comportamiento notablemente diferente al observado en el Mask Decoder. Como se muestra en la Fig. 28, tanto la pérdida durante el entrenamiento como en la validación siguen una tendencia lineal descendente. Sin embargo, a pesar de esta mejora gradual, los valores absolutos de la pérdida permanecen significativamente elevados, siendo estos de un orden de magnitud superiores a los obtenidos durante el ajuste del Mask Decoder. Este hecho sugiere que el modelo enfrenta dificultades para aprender patrones efectivos relacionados con los prompts en el contexto de la segmentación de grietas.

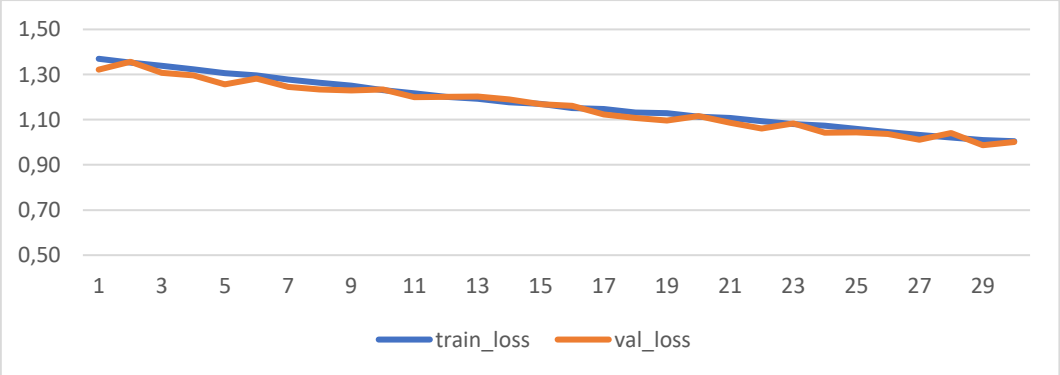


Fig. 28. Evolución de pérdida durante el ajuste del Prompt Encoder (Concrete Crack Dataset)

En relación con las métricas de evaluación (ver Fig. 29), se aprecia una mejora ligera pero consistente a lo largo de las épocas tanto en la etapa de entrenamiento como en la de validación. No obstante, los valores obtenidos son considerablemente bajos en comparación con los alcanzados por el Mask Decoder (20.53% IoU y 27.84% F1-score), reflejando un pésimo desempeño del Prompt Encoder en esta tarea específica. Esto podría atribuirse a la menor relevancia del codificador de prompts en la generación de máscaras precisas en el contexto del experimento. El análisis combinado de la pérdida y las métricas pone de manifiesto que, aunque el modelo mejora gradualmente con el entrenamiento, la capacidad del Prompt Encoder para contribuir de manera significativa al ajuste específico de segmentación de grietas es limitada.

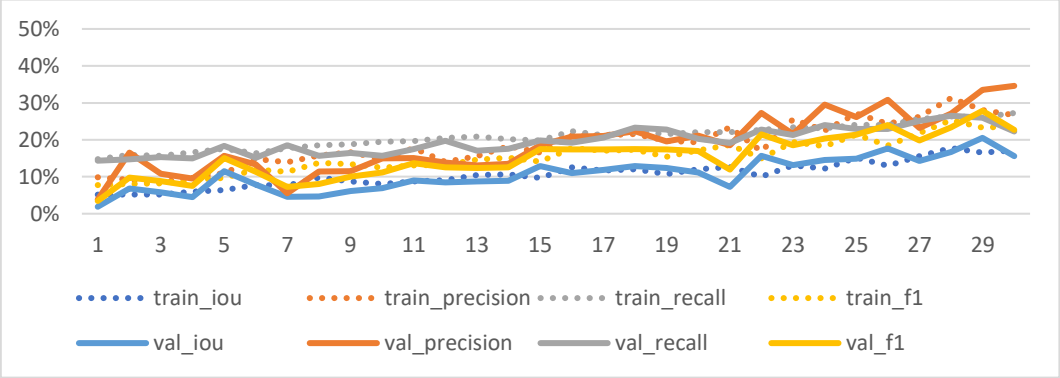


Fig. 29. Evolución de las Métricas durante el entrenamiento-validación del Prompt Encoder (Concrete Crack Dataset)

4.1.1.3 Mask Decoder + Prompt Encoder

En esta ocasión se llevó a cabo el ajuste simultáneo del Prompt Encoder junto con el Mask Decoder, buscando evaluar si la combinación de ambos componentes mejora el desempeño general de los modelos ajustando dichos componentes por separado. Como se observa en la Fig. 30, la pérdida de entrenamiento experimenta un descenso pronunciado durante las primeras 3 épocas, indicando una rápida adaptación inicial del modelo. Posteriormente, la pérdida se reduce de forma sostenida a un ritmo más lento, lo que sugiere un ajuste fino progresivo en etapas posteriores. Sin embargo, la pérdida de validación alcanza un punto de estancamiento a partir de la época 10, posiblemente indicando una saturación en la capacidad del modelo para generalizar a los datos de validación.

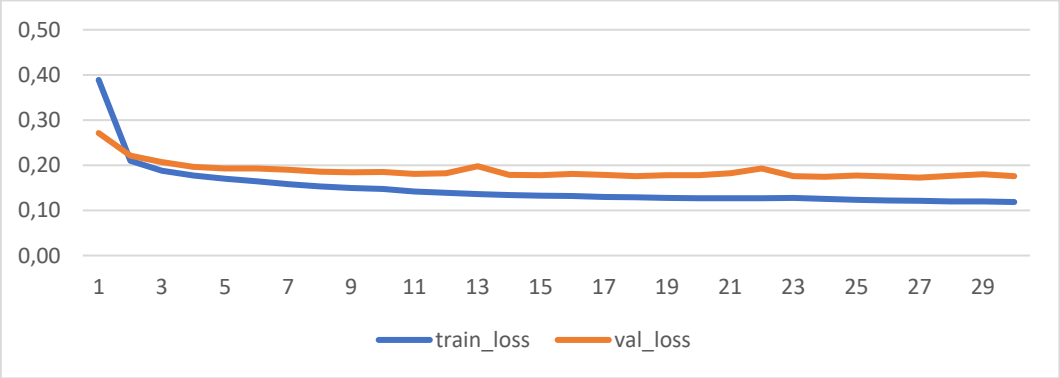


Fig. 30. Evolución de pérdida durante el ajuste del Mask Decoder junto al Prompt Encoder (Concrete Crack Dataset)

Respecto a las métricas de evaluación (Fig. 31), se refleja una tendencia similar a la observada en los experimentos previos: las mejoras más significativas ocurren durante las primeras 2 épocas. Posteriormente, las métricas continúan mejorando ligeramente tanto en la etapa de entrenamiento como en la de validación, alcanzando valores similares a los obtenidos en el ajuste del Mask Decoder (69.13% IoU, 81.28% F1-score). Este resultado muestra que la incorporación del Prompt Encoder no añade un valor adicional relevante al proceso de segmentación de grietas respecto al ajuste individual del Mask Decoder.

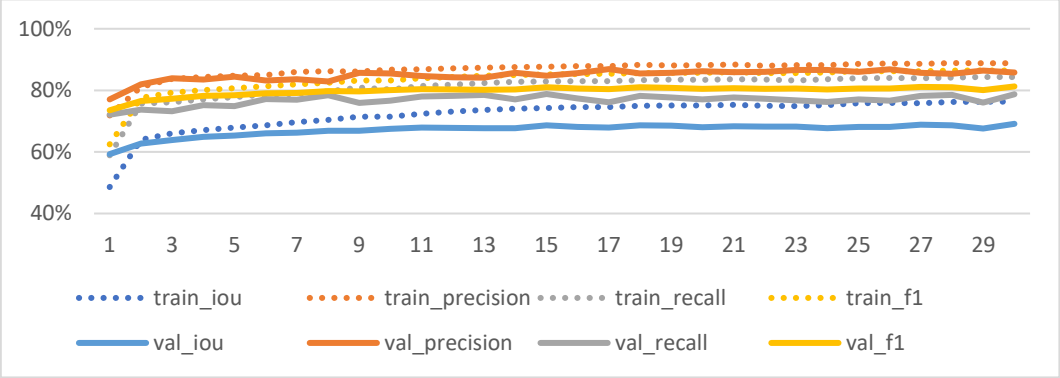


Fig. 31. Evolución de las Métricas durante el entrenamiento-validación del Mask Decoder + Prompt Encoder (Concrete Crack Dataset

4.1.1.4 Image Encoder

En este experimento se ajustó el Image Encoder, responsable de codificar las características visuales de las imágenes de entrada. Como se ilustra en la Fig. 32, la pérdida de entrenamiento muestra una reducción pronunciada durante las primeras 7 épocas, con una caída especialmente destacada en las 2 primeras épocas. Este comportamiento hace indicar que el modelo adapta rápidamente sus pesos para capturar características relevantes del dataset. Sin embargo, la pérdida de validación presenta un descenso más moderado durante las primeras 4 épocas y se estabiliza en las épocas siguientes, indicando una posible limitación en la capacidad de aprendizaje del modelo en este componente específico.

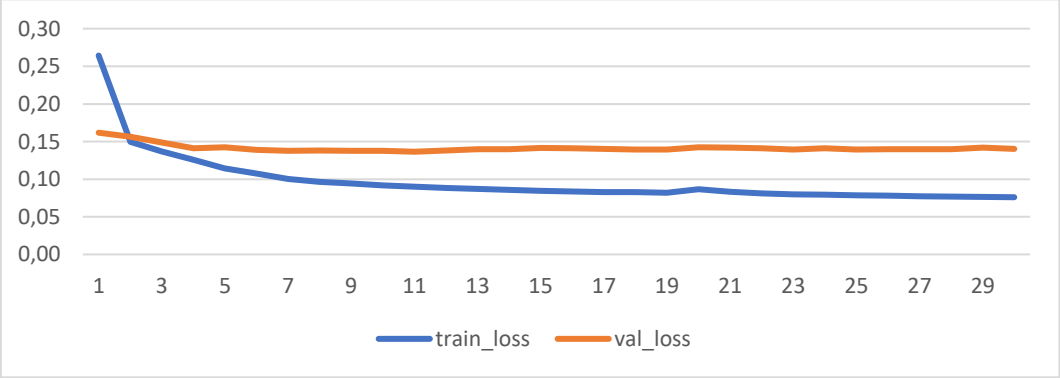


Fig. 32. Evolución de pérdida durante el ajuste del Image Encoder (Concrete Crack Dataset)

Respecto a las métricas de evaluación, Fig. 33, se observa un comportamiento similar al reportado en los ajustes del Mask Decoder y del Mask Decoder + Prompt Encoder, pero con resultados ligeramente superiores en algunas métricas clave (74.35% IoU, 85.59%

F1-score). Este incremento puede atribuirse a la capacidad del Image Encoder para procesar directamente las características visuales de las imágenes, optimizando su representación y facilitando la segmentación precisa de las grietas. El desempeño observado en este ajuste pone de manifiesto el impacto significativo que tiene la mejora de la representación inicial de las imágenes en el rendimiento global del modelo.

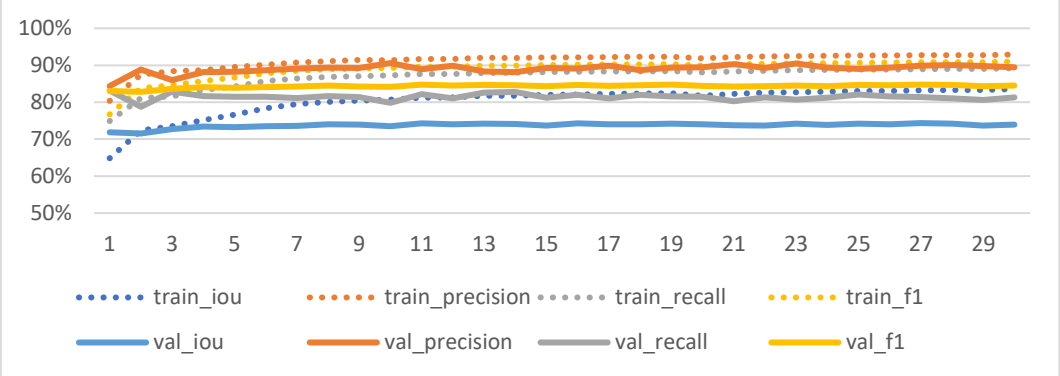


Fig. 33. Evolución de las Métricas durante el entrenamiento-validación del Image Encoder (Concrete Crack Dataset)

4.1.1.5 Fine-Tuning Completo

El ajuste completo del modelo, que incluye el Mask Decoder, el Prompt Encoder y el Image Encoder, se presenta como la etapa final de la fase de entrenamiento del primer experimento. Según se ilustra en la Fig. 34, la evolución de las pérdidas de entrenamiento y validación sigue un patrón similar al observado en los ajustes de componentes individuales, salvo por las discrepancias observadas en el ajuste del Prompt Encoder. En las primeras épocas, la pérdida disminuye rápidamente, estabilizándose en fases posteriores. Este comportamiento refleja una convergencia efectiva del modelo, aunque sugiere una posible saturación en el aprendizaje después de un número determinado de épocas.

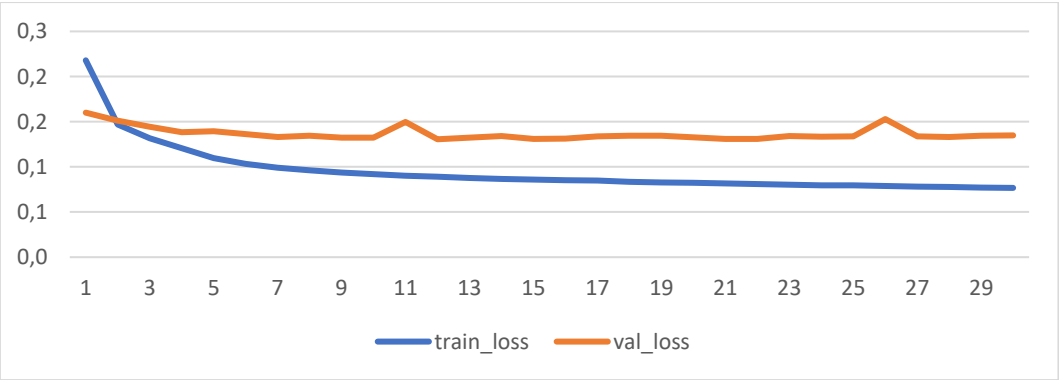


Fig. 34. Evolución de pérdida durante el fin-tuning de los 3 componentes (Concrete Crack Dataset)

Por otro lado, la evolución de las métricas (Fig. 35) muestra tendencias similares respecto al de los ajustes previos, destacando una mejora significativa durante las primeras épocas de entrenamiento. Las métricas finales alcanzan tasas de éxito comparables a las logradas en el ajuste del Image Encoder (75.24% IoU, 85.41 % F1-score), consolidando la importancia de este componente en la segmentación precisa de grietas. Sin embargo, la contribución conjunta de los tres componentes parece no superar significativamente las

capacidades del ajuste individual del Image Encoder, lo que indica que este último desempeña un rol crucial en la calidad de la segmentación.

Estos resultados resaltan que, aunque el ajuste completo permite integrar y optimizar de manera conjunta todos los componentes del modelo, no siempre garantiza una mejora sustancial respecto a los ajustes específicos.

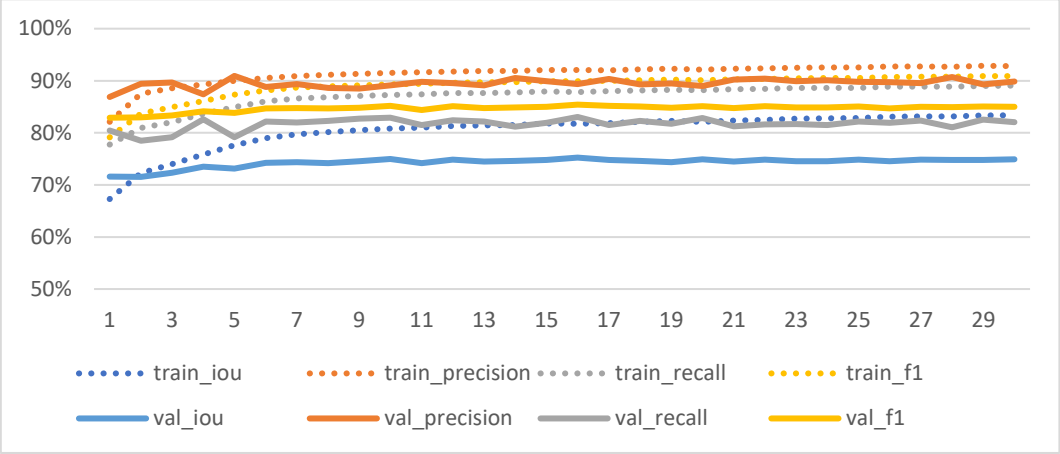


Fig. 35. Evolución de las Métricas durante el entrenamiento del modelo completo (Concrete Crack Dataset)

4.1.2 Resultados de la etapa de test

En este último apartado del primer experimento se expondrán los resultados obtenidos en la etapa de test sobre el mismo dataset empleado en el fine-tuning. Debido a ello, tal y como se ilustra en la Fig. 36, los resultados obtenidos se asemejan a los alcanzados durante el entrenamiento. Es decir, las métricas de evaluación alcanzan altas tasas de éxito en todos los componentes ajustados con la salvedad del ajuste del codificador de prompts, que muestra un resultado bastante deficiente superando solamente y de forma muy ligera al modelo base, representado como SAM_VIT_b en la gráfica.

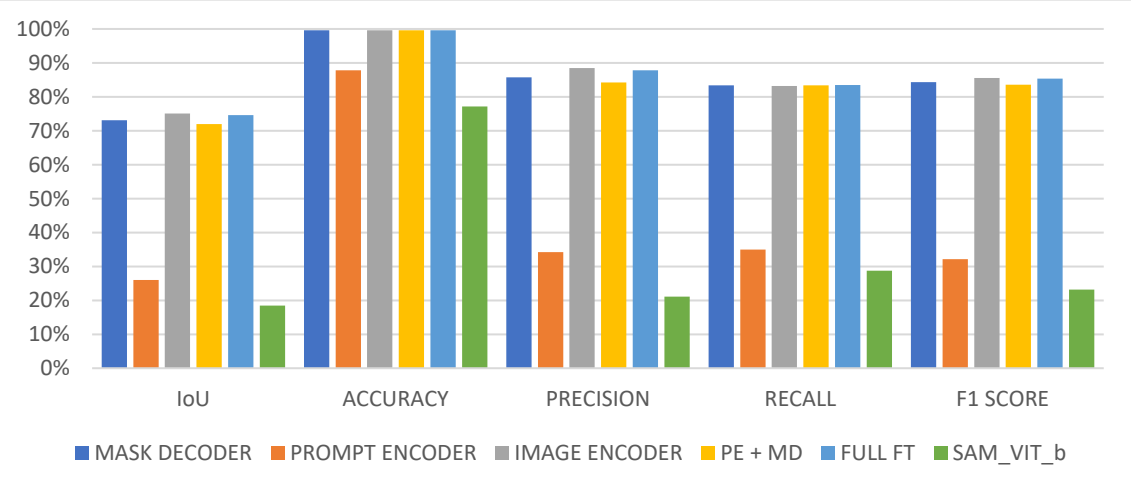


Fig. 36. Comparativa de las métricas obtenidas en la etapa de test sobre el Concrete Crack Segmentation dataset

Al hilo de lo mencionado anteriormente, en la Tabla 2 se puede ver que los resultados obtenidos con el ajuste del Mask Decoder, del Mask Decoder + Prompt Encoder, del Image Encoder y del todos los componentes ajustados simultáneamente obtienen resultados similares, resultando en una tasa de acierto ligeramente superior en el modelo

con el Image Encoder ajustado y del fine-tuning completo. Por el contrario, el modelo base obtiene los resultados más bajos.

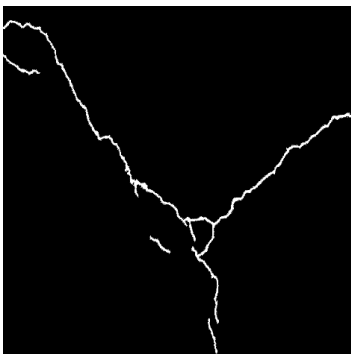
Tabla 2. Resultados de las métricas de la inferencia sobre el Concrete Crack Segmentation dataset

	IoU	ACCURACY	PRECISION	RECALL	F1-SCORE
MASK DECODER	0,7312	0,9959	0,8574	0,8337	0,8430
PROMPT ENCODER	0,2603	0,8784	0,3419	0,3501	0,3219
IMAGE ENCODER	0,7503	0,9964	0,8849	0,8316	0,8559
PE + MD	0,7196	0,9957	0,8419	0,8336	0,8354
FULL FT	0,7462	0,9963	0,8777	0,8343	0,8533
SAM_VIT_b	0,1845	0,7719	0,2110	0,2871	0,2316

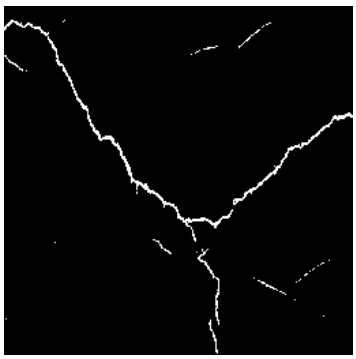
Finalmente, en la Tabla 3 se ilustra un ejemplo de las máscaras obtenidas durante la etapa de test con el mejor modelo de cada componente ajustado. Se puede ver cómo, siguiendo la tendencia observada en la Tabla 2 y en la Fig. 36, las máscaras obtenidas en el modelo ajustando el Mask Decoder, el Mask Decoder + Prompt Encoder, el Image Encoder y el fine-tuning completo se ajustan con bastante similitud a la máscara verdadera, fallando en la zona donde la grieta se estrecha o está más difuminada y tendiendo a generar falsos positivos en zonas del hormigón ligeramente arañadas. En cambio, tanto el modelo ajustando el Prompt Encoder como el modelo base obtiene máscaras muy alejadas de la verdadera, estando en consonancia con los resultados cuantitativos analizados previamente.

Tabla 3. Máscaras obtenidas durante la etapa de test de Concrete Crack Segmentation dataset

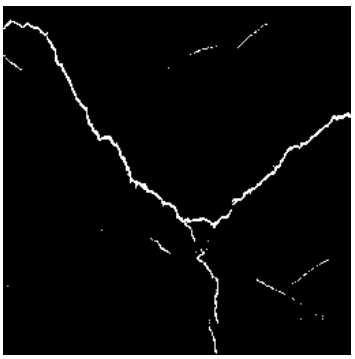
IMAGEN ORIGINAL



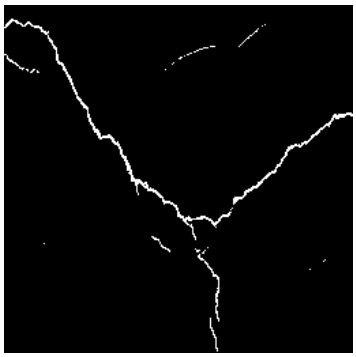
GROUND-TRUTH



MASK DECODER



MASK DECODER + PROMPT ENCODER



FULL FINE-TUNE

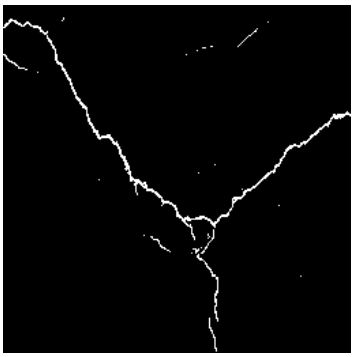
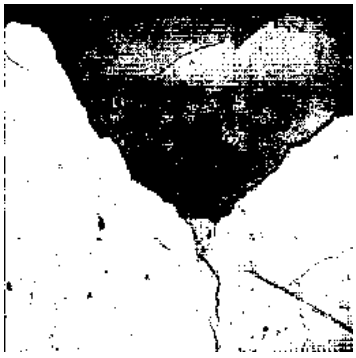
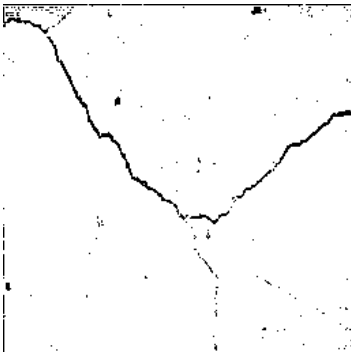


IMAGE ENCODER



PROMPT ENCODER



SAM_VITB

4.2 Experimento 2. Inferencia sobre DeepCrack Dataset

En el experimento 2 se llevó a cabo la inferencia utilizando los modelos que fueron previamente ajustados con el Concrete Crack Dataset. En este caso, el objetivo principal fue evaluar la capacidad de generalización de los modelos ajustados aplicándolos a un conjunto de datos diferente, **Deep Crack Dataset**.

El Deep Crack Dataset presenta un conjunto de imágenes más diverso en términos de resolución, iluminación y variabilidad en las grietas, lo que representa un reto adicional para el modelo. Este experimento permitió comprobar la capacidad del modelo para realizar inferencias precisas en un conjunto de datos que, aunque relacionado, no formaba parte de su entrenamiento original.

4.2.1 Resultados de la inferencia

En la Fig. 37 se representa un gráfico del desempeño de los modelos ajustados con el dataset del primer experimento al aplicarlos al dataset de este segundo experimento. De manera similar a lo observado en el primer experimento, el modelo base es el que muestra el peor rendimiento en la inferencia, siendo superado, aunque de forma marginal, por el modelo con el Prompt Encoder ajustado, cuyos valores en las métricas obtenidas fueron notablemente bajos.

Por otro lado, en las métricas IoU, Recall y F1-score la tendencia es similar, siendo el modelo ajustando el codificador de imágenes el que mejor se comporta, seguido del modelo fine-tune completo, mientras que un escalón por debajo se encuentran los

modelos ajustando el Mask Decoder y el modelo ajustando simultáneamente el Mask Decoder y el Prompt Encoder.

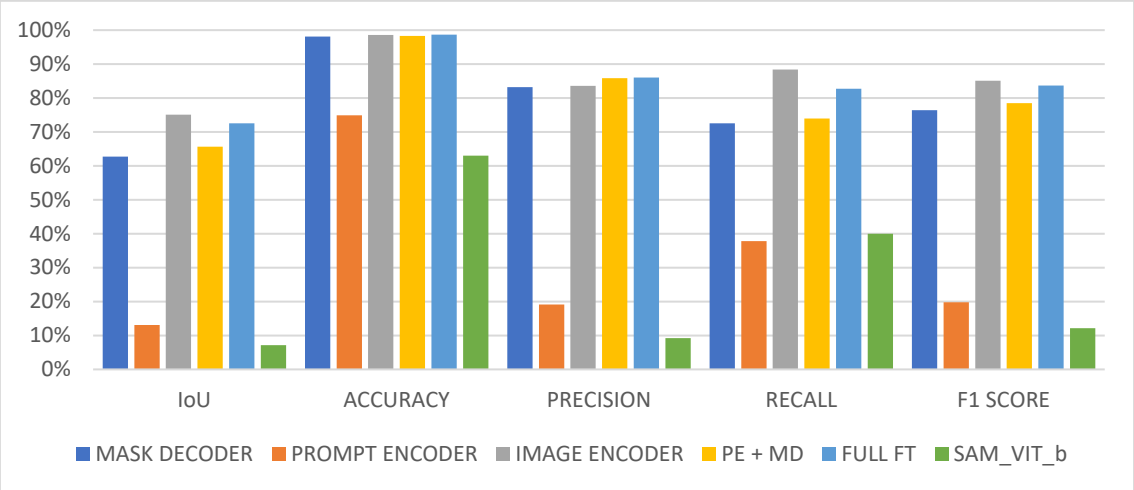


Fig. 37. Comparativa de las métricas obtenidas en la inferencia de DeepCrack dataset, utilizando los modelos de segmentación ajustados con el Concrete Crack Dataset

En relación al análisis cuantitativo del comportamiento de la inferencia, en la Tabla 4 se dispone de una comparativa de los resultados que se obtuvieron en el experimento 1 con el Concrete Crack Dataset (fila E1) con los obtenidos en el experimento 2 con el DeepCrack Dataset (fila E2).

Al analizar los modelos con peor rendimiento en el primer experimento, es decir, el modelo ajustando el Prompt Encoder y el modelo base, se observa que continúan siendo los menos eficaces, con una reducción aún mayor en su tasa de éxito en la segmentación en todas las métricas, excepto en recall, donde presentan un desempeño ligeramente superior.

Tabla 4. Comparativa de los resultados obtenidos en la inferencia con el dataset de entrenamiento (filas E1) y con el dataset del experimento 2 (filas E2)

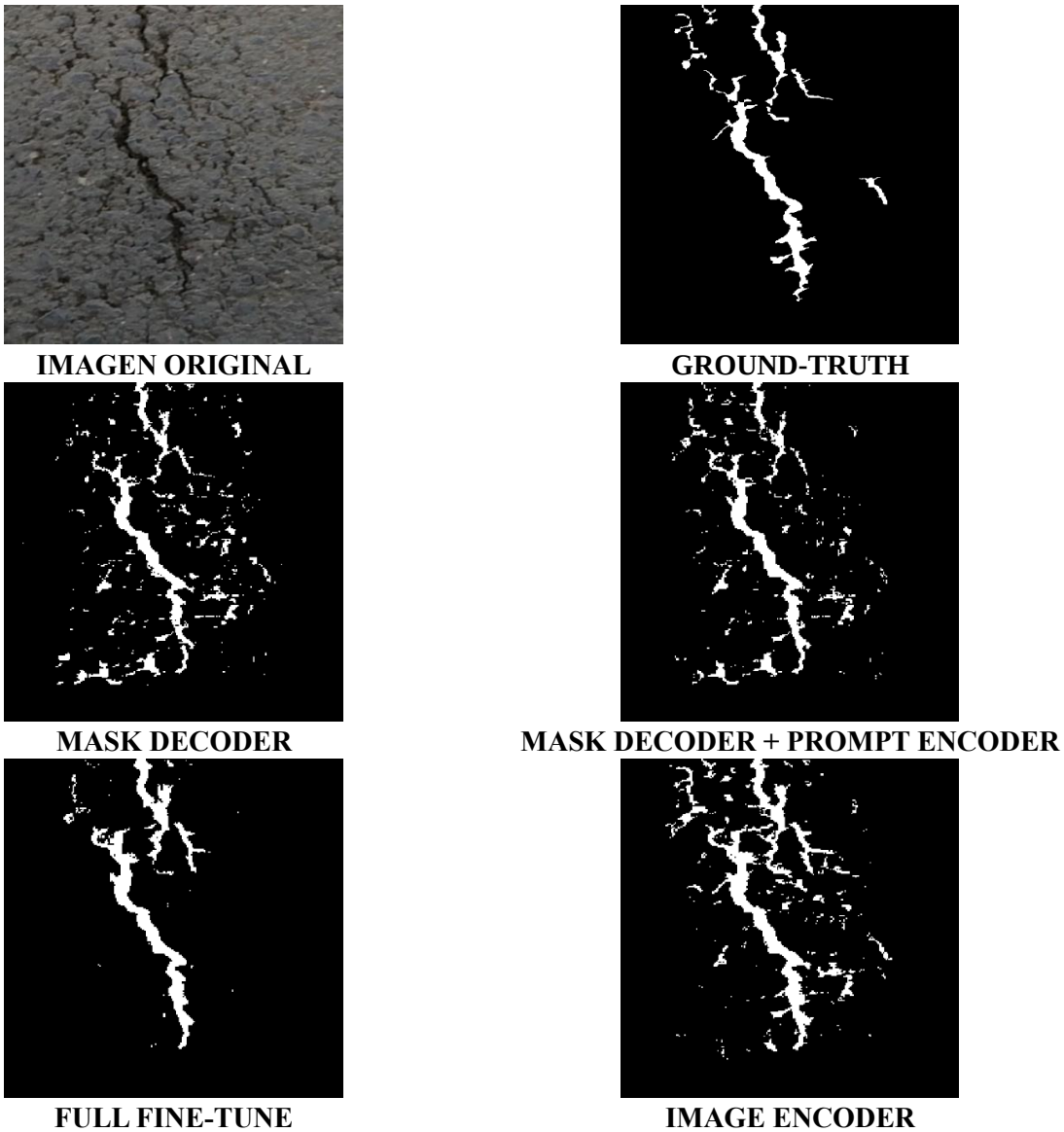
	IoU	ACCURACY	PRECISION	RECALL	F1-SCORE	
MASK DECODER	0,7312	0,9959	0,8574	0,8337	0,8430	E1
	0,6275	0,9807	0,8320	0,7255	0,7644	E2
PROMPT ENCODER	0,2603	0,8784	0,3419	0,3501	0,3219	E1
	0,1308	0,7485	0,1909	0,3778	0,1980	E2
IMAGE ENCODER	0,7503	0,9964	0,8849	0,8316	0,8559	E1
	0,7510	0,9855	0,8360	0,8836	0,8510	E2
PE + MD	0,7196	0,9957	0,8419	0,8336	0,8354	E1
	0,6562	0,9830	0,8585	0,7399	0,7852	E2
FULL FT	0,7462	0,9963	0,8777	0,8343	0,8533	E1
	0,7254	0,9865	0,8606	0,8276	0,8366	E2
SAM_VIT_b	0,1845	0,7719	0,2110	0,2871	0,2316	E1
	0,0713	0,6300	0,0923	0,4001	0,1211	E2

Los mejores modelos (el modelo fine-tune completo y el modelo ajustando el Image Encoder) presentan valores muy similares a los obtenidos en el primer experimento,

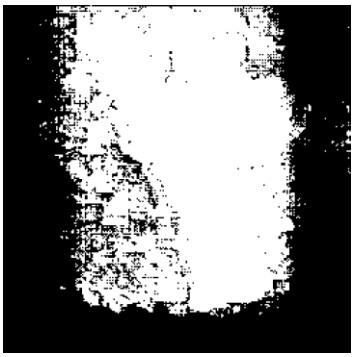
incluso logrando valores ligeramente superiores en IoU y Recall en el modelo que ajusta el Image Encoder.

Finalmente, en la Tabla 5 se presentan las máscaras obtenidas en la inferencia sobre una de las imágenes del dataset. Tal y como se anticipaba a partir de los resultados cuantitativos previamente discutidos, el modelo ajustando únicamente el Prompt Encoder y el modelo base de SAM no lograron realizar una segmentación que se asemejara mínimamente a la anotación. En contraste, el resto de los modelos fueron capaces de segmentar, en mayor o menor grado, la grieta de mayor magnitud, aunque presentaron dificultades para identificar las grietas de menor tamaño. Cabe destacar que la segmentación de esta imagen no resulta sencilla, debido a la complejidad de texturas presentes y la heterogeneidad de la superficie del hormigón, lo cual incluso convierte la anotación manual en una tarea desafiante.

Tabla 5. Máscaras obtenidas durante la inferencia usando el dataset DeepCrack



PROMPT ENCODER



SAM_VITB

4.3 Experimento 3. Inferencia sobre Marble Crack Dataset

En el tercer experimento se procedió a realizar la inferencia utilizando los modelos previamente ajustados con el dataset del primer experimento para estudiar la capacidad del modelo de generalizar con un dataset de un material distinto, el **Marble Crack Dataset**. Aunque ambos datasets se centran en la segmentación de grietas, es importante destacar que las características visuales tanto de las grietas como de las superficies varían considerablemente: el mármol presenta una textura, color y patrones de grietas distintos a los del hormigón. Este experimento permitió evaluar cómo el modelo, entrenado en un conjunto de datos con características diferentes, maneja las variaciones en las imágenes del Marble Crack Dataset**.**

4.3.1 Resultados de la inferencia

El primer análisis que se puede realizar es la comparativa de las métricas que obtienen cada uno de los modelos SAM ajustados, los cuales se ilustran en la Fig. 38. En términos generales se observa que, con excepción de la métrica de exactitud (accuracy)**,** los modelos ajustando el Prompt Encoder y el modelo base obtienen valores muy bajos, lo que coincide con los resultados obtenidos en experimentos anteriores.

Sin embargo, se presenta una discrepancia notable respecto a los resultados obtenidos con los dataset de hormigón, donde los modelos que mejores resultados obtenían era el modelo ajustando los tres componentes (fine-tuning completo) y el modelo ajustando el Image Encoder. En cambio, en esta ocasión el modelo ajustado el Mask Decoder y Prompt Encoder simultáneamente es el que muestra el mejor desempeño, seguido por el modelo ajustado únicamente en el Mask Decoder. En este caso, el ajuste del Image Encoder en el dataset de hormigón parece haber reducido la capacidad de generalización del modelo, lo que resulta en segmentaciones menos precisas en comparación con los modelos que no ajustaron dicho componente.

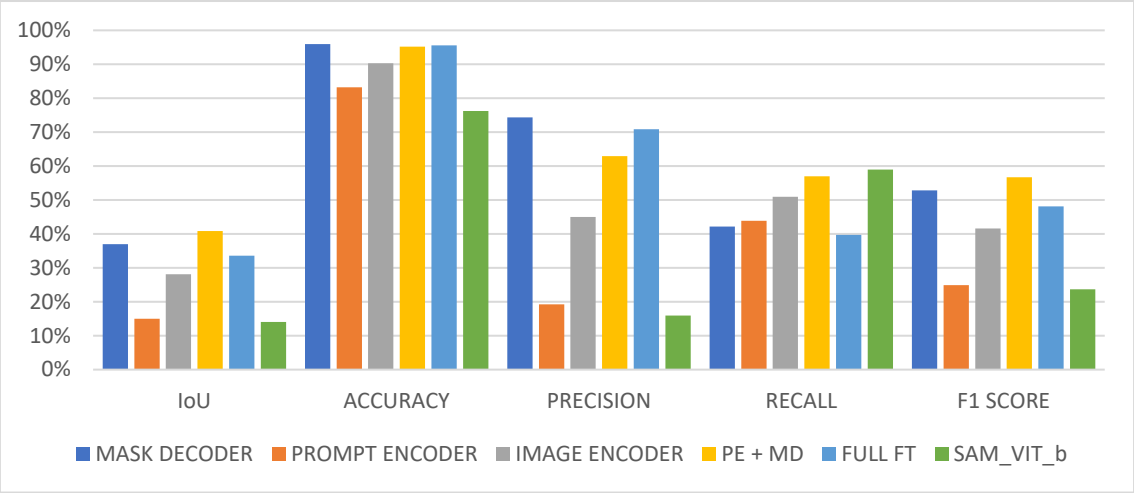


Fig. 38. Comparativa de las métricas obtenidas en la inferencia de Marble Crack dataset, utilizando los modelos de segmentación ajustados con el Concrete Crack Dataset

En otro orden, resulta pertinente realizar una comparación cuantitativa entre los valores de las métricas obtenidas en este experimento (filas E3) y aquellos obtenidos durante el test del primer experimento (filas E1), dispuestos en la Tabla 6. En ella se observa un empeoramiento considerable de los valores de las métricas obtenidos en el experimento 3 respecto al experimento 1. En la métrica IoU se produce una reducción de entre un 24% en el modelo base hasta cerca de un 63% en el modelo ajustando el codificador de imagen. Los valores de F1-score se ven reducido también en una proporción de entre aproximadamente un 23% hasta cerca de un 52% en función de la métrica de evaluación, a excepción del modelo base que obtiene un resultado levemente superior. Los valores de precisión siguen la tendencia de disminución de sus valores al igual que recall, que solamente mejora en el modelo ajustando el prompt encoder y el modelo base.

Tabla 6. Comparativa de los resultados obtenidos en la inferencia con el dataset de entrenamiento (filas E1) y con el dataset del experimento 3 (filas E3)

	IoU	ACCURACY	PRECISION	RECALL	F1-SCORE	
MASK DECODER	0,7312	0,9959	0,8574	0,8337	0,843	E1
	0,3693	0,9595	0,7431	0,4221	0,5281	E3
PROMPT ENCODER	0,2603	0,8784	0,3419	0,3501	0,3219	E1
	0,1499	0,8319	0,1919	0,4386	0,2494	E3
IMAGE ENCODER	0,7503	0,9964	0,8849	0,8316	0,8559	E1
	0,2810	0,9032	0,4496	0,5091	0,4161	E3
PE + MD	0,7196	0,9957	0,8419	0,8336	0,8354	E1
	0,4081	0,9521	0,6295	0,5699	0,5666	E3
FULL FT	0,7462	0,9963	0,8777	0,8343	0,8533	E1
	0,3361	0,9556	0,7081	0,3974	0,4809	E3
SAM_VIT_b	0,1845	0,7719	0,211	0,2871	0,2316	E1
	0,1408	0,7627	0,1595	0,5895	0,2365	E3

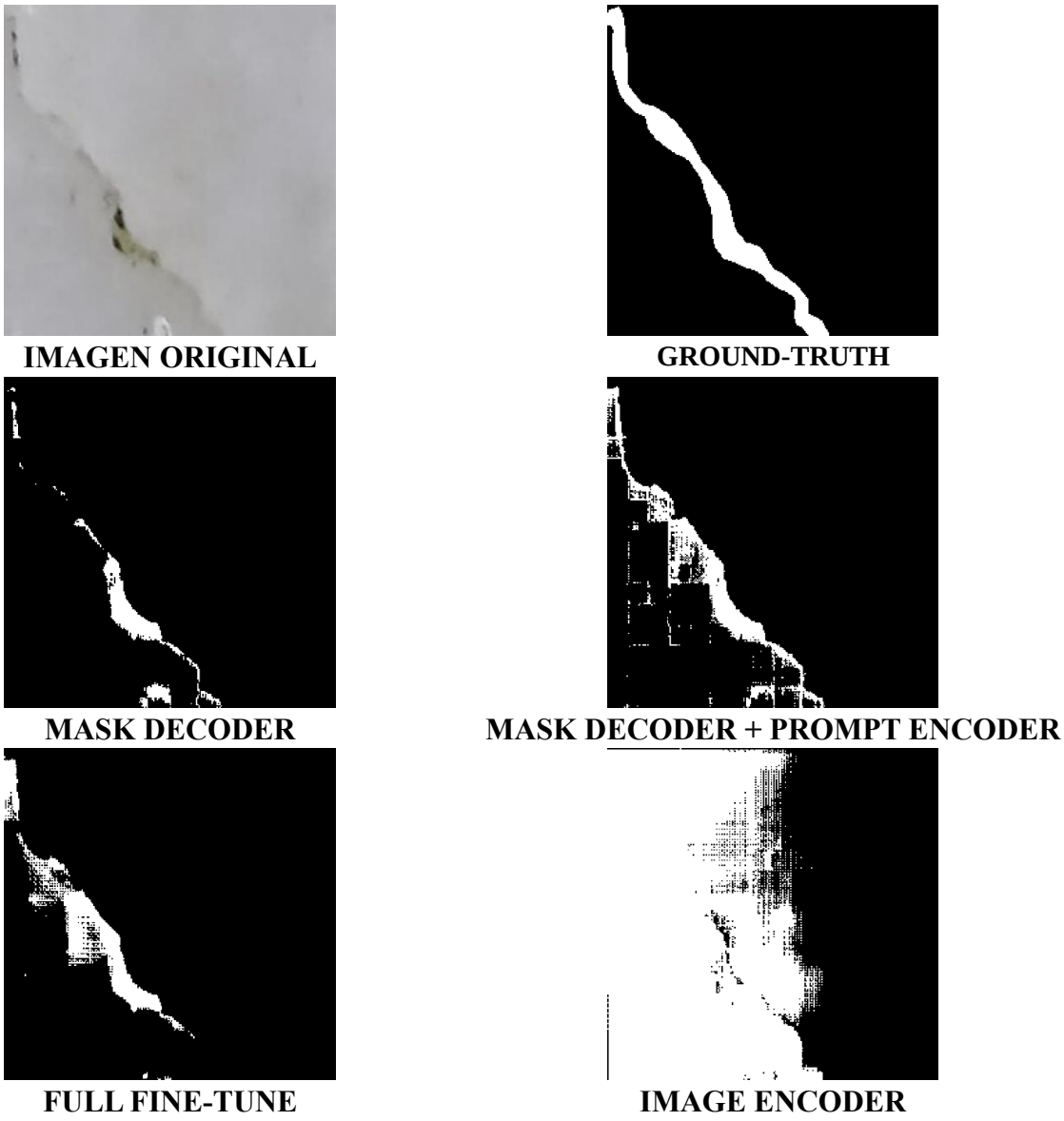
Para enriquecer el análisis del experimento se extrajeron máscaras con cada uno de los modelos ajustados, así como del modelo base, de varias imágenes con el fin de poder visualizar el comportamiento de cada modelo y comparar sus resultados. En la Tabla 7 se

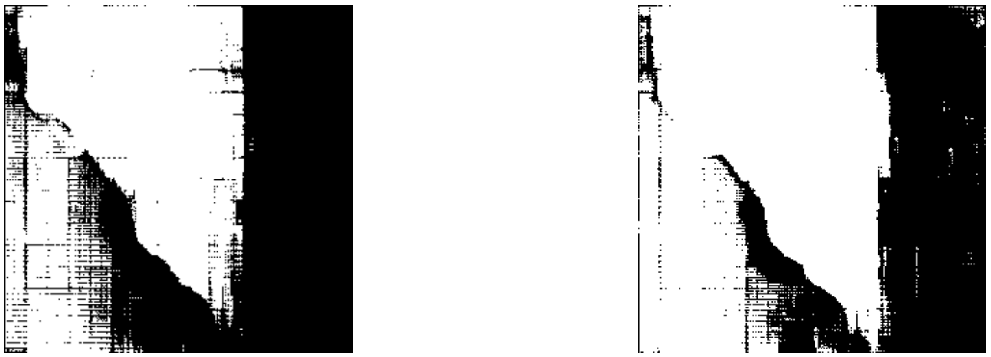
puede ver un ejemplo de las máscaras obtenidas con cada uno de los modelos dada una imagen de entrada.

En este ejemplo se puede ratificar los resultados vistos en la Tabla 6 e ilustrados en la Fig. 38, donde se observa que tanto el modelo base como el modelo ajustando el codificador de prompts no consiguen realizar una segmentación mínimamente aproximada a la anotación, al igual que ocurría en el experimento 2. Sin embargo, en este experimento se añade la novedad de que el modelo ajustando el codificador de imágenes muestra un resultado similar a estos modelos.

Por otro lado, los modelos ajustando el Mask Decoder, el modelo ajustando el Mask Decoder junto al Prompt Encoder y el modelo fine-tune completo obtienen máscaras que, si bien no pueden categorizarse como exitosas, sí que se aproximan mínimamente al ground-truth.

Tabla 7. Máscaras obtenidas durante la inferencia usando el Marble Crack Dataset





PROMPT ENCODER **SAM_VITB**

4.4 Experimento 4. Fine-tuning sobre dataset combinado

En este último experimento, se llevó a cabo el entrenamiento utilizando un dataset compuesto por los tres datasets empleados en los experimentos anteriores, integrando imágenes de grietas en hormigón (provenientes tanto del Concrete Crack Dataset como del DeepCrack Dataset) e imágenes de grietas en mármol (del MCS Dataset). El objetivo principal de este experimento fue reentrenar los componentes principales de SAM empleando este dataset conjunto, con el propósito de evaluar la eficacia de la segmentación de grietas independientemente de si el material corresponde a mármol u hormigón. Las características principales del dataset combinado se resumen en:

- **Número de imágenes:** 511 imágenes de entrenamiento, 111 imágenes de validación y 111 imágenes de test
- **Composición del dataset:** Concrete crack dataset (48%), Deep crack dataset (26%) y Marble crack dataset (26%)

En la Fig. 39 se puede ver una muestra representativa con pares imagen-anotación de cada uno de los tres dataset que forman el conjunto de datos del experimento para ilustrar la heterogeneidad del conjunto de datos.

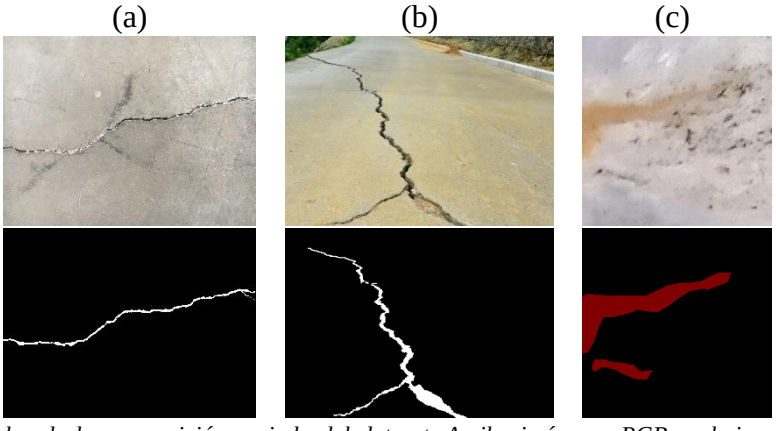


Fig. 39. Ejemplos de la composición variada del dataset. Arriba imágenes RGB y abajo su respectiva máscara verdadera. (a): Concrete Crack Dataset, (b): DeepCrack Dataset, (c): Marble Crack Dataset

El procedimiento de fine-tuning se diseñó siguiendo una configuración similar a la del primer experimento, incorporando los siguientes ajustes específicos:

- **Tasa de aprendizaje**. Se utilizó un valor fijo de 1e-5, seleccionado para permitir un ajuste gradual del modelo sin perder estabilidad en el proceso de entrenamiento.
- **Optimizador**. Se empleó el optimizador Adam.
- **Función de pérdida**. Se optó por la función DiceCELoss, una combinación de Dice Loss y Cross-Entropy Loss.
- **Número de épocas**. El modelo se entrenó durante 30 épocas, suficientes para garantizar la convergencia.
- **División del dataset**. El conjunto de datos se segmentó en proporciones estándar, asignando un 70% para entrenamiento, 15% para validación y 15% para prueba, proporcionando una evaluación representativa y confiable.
- **Tamaño de lote (batch)**. Se utilizó un tamaño de lote de 4.
- **Recursos computacionales**. El entrenamiento se ejecutó en la plataforma Google Colab, aprovechando los recursos de aceleración disponibles. Se ofrecerán más detalles en el apartado 4.5.

4.4.1 Resultados del entrenamiento

Inicialmente se llevó a cabo un análisis detallado de la evolución de la función de pérdida durante las etapas de entrenamiento y validación, así como del comportamiento de las principales métricas de evaluación a lo largo de las épocas. Este análisis permitió evaluar la capacidad del modelo para generalizar en diferentes tipos de superficies y bajo condiciones heterogéneas.

Los resultados obtenidos se organizan, al igual que en el primer experimento, en función de los componentes ajustados durante el proceso de fine-tuning, los cuales son:

- **Mask Decoder**
- **Prompt Encoder**
- **Prompt Encoder + Mask Decoder**
- **Image Encoder**
- **Full Fine-Tuning (ajuste completo)**

A continuación, se describirán los resultados obtenidos en el fine-tuning de cada uno de los componentes del modelo definidos anteriormente.

4.4.1.1 Mask Decoder

Durante el entrenamiento del Mask Decoder, la evolución de la pérdida de entrenamiento mostró una disminución significativa, iniciando en 0.45 y reduciéndose rápidamente a 0.24 en las tres primeras épocas. Esto indica que el modelo aprendió de forma rápida los patrones en las etapas iniciales. Posteriormente, el descenso fue más gradual alcanzando un valor de 0.15 en la época 30, reflejando una convergencia progresiva y estable.

En cuanto a la pérdida de validación se observó una reducción de la pérdida pero con una menor tasa, comenzando en 0.3, bajando a 0.22 hacia la época 10 y estabilizándose alrededor de 0.21 desde la época 20. Esta evolución estable en ambas pérdidas sugiere un modelo bien ajustado, con una capacidad sólida de generalización y sin señales de sobreajuste. La evolución de las pérdidas de entrenamiento y validación para el Mask Decoder se representa en la Fig. 40.

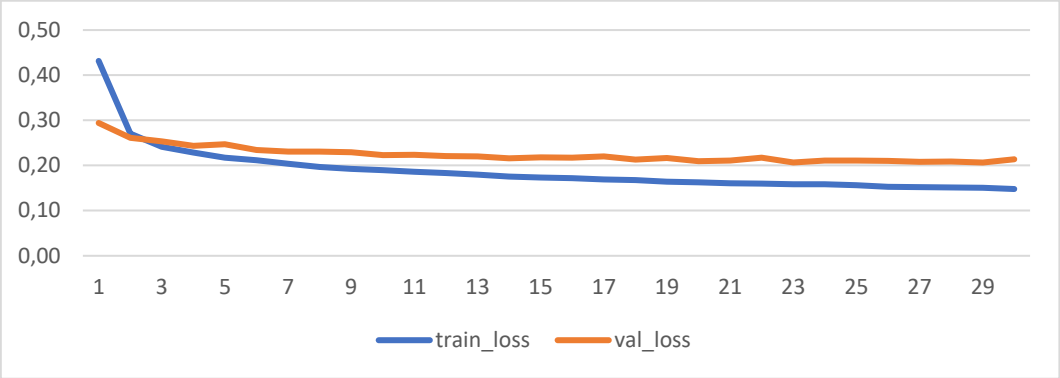


Fig. 40. Evolución de pérdida durante el ajuste del Mask Decoder (dataset combinado)

Además de la evolución de la pérdida, el análisis de las métricas de evaluación proporciona una perspectiva complementaria del rendimiento del modelo. En la Fig. 41, se observa que todas las métricas en la etapa de entrenamiento incrementaron rápidamente durante las dos primeras épocas, estabilizándose posteriormente con un crecimiento más gradual en las etapas finales.

En la etapa de validación, las métricas mostraron un comportamiento más irregular. Aunque se observó un incremento general durante el proceso, se identificaron picos significativos, especialmente en las métricas de precisión y recall. Por otro lado, las métricas de IoU y F1-score presentaron un aumento más gradual, con menor presencia de picos, lo que sugiere una evolución más estable.

Respecto a los valores máximos alcanzado, la IoU y F1-score alcanzan porcentajes de acierto aproximados de 67% y 80% respectivamente en la etapa 20. Por otro lado, se alcanza un 84% para la precisión en la etapa 30 y de 79% de recall en la etapa 17.

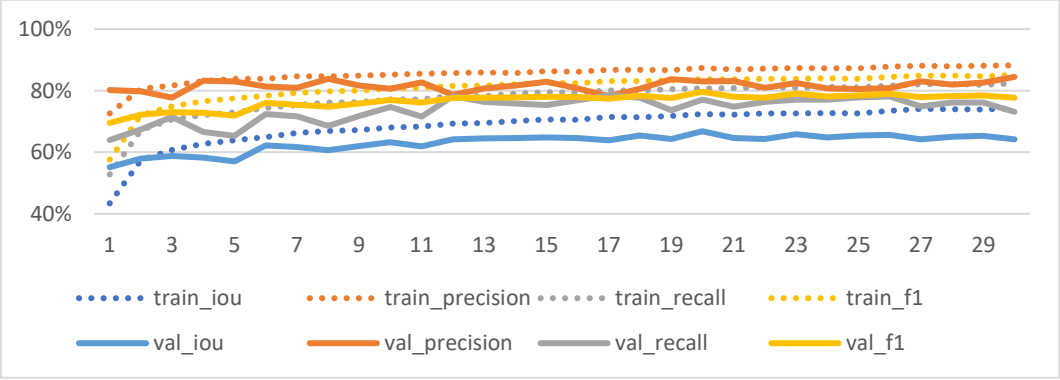


Fig. 41. Evolución de las Métricas durante el entrenamiento-validación del Mask Decoder (dataset combinado)

4.4.1.2 Prompt Encoder

El segundo ajuste realizado fue el del Prompt Encoder, cuyo entrenamiento se inició con valores de pérdida considerablemente altos, reflejando las discrepancias entre las predicciones y las anotaciones en las primeras épocas. En la Fig. 42 se observa que la pérdida de entrenamiento comenzó en 1.38, mientras que la pérdida de validación inició en 1.28. Estas cifras reflejan predicciones iniciales muy alejadas de las anotaciones reales.

Conforme avanzó el entrenamiento, ambas pérdidas mostraron una reducción progresiva y lineal, alcanzando al final de la época 30 valores de 0.87 en la pérdida de entrenamiento y 0.74 en la de validación. Este comportamiento indica una mejora constante, pero con valores finales elevados en comparación con los resultados obtenidos durante el ajuste del Mask Decoder. A pesar de que las pérdidas de entrenamiento y validación fueron coherentes entre sí, el desempeño global del modelo quedó limitado como ya se observó en el primer experimento

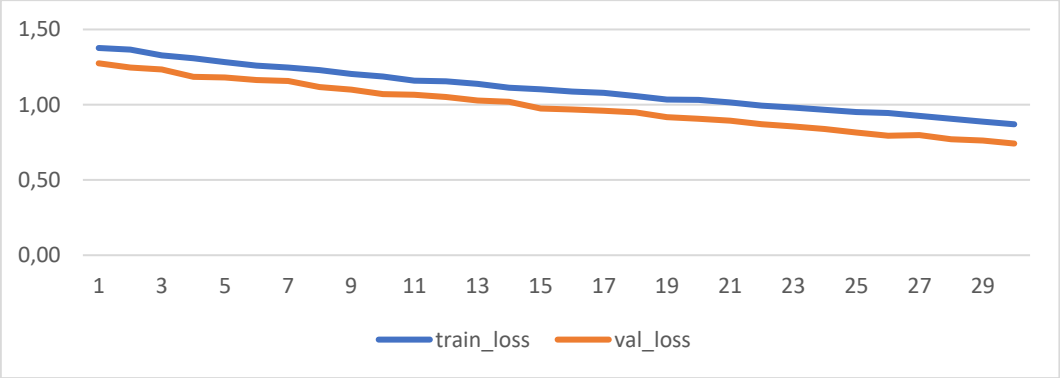


Fig. 42. Evolución de pérdida durante el ajuste del Prompt Encoder (dataset combinado)

En cuanto a las métricas de evaluación (Fig. 43), a excepción del recall, el resto de métricas siguen una tendencia alcista partiendo de valores por debajo del 20% hasta alcanzar en las últimas épocas los mejores resultados: 26.61% IoU, 43.69% precisión y 37.52% F1-score. Los resultados obtenidos están muy por debajo del mask decoder, al igual que ya sucedió en el primer experimento.

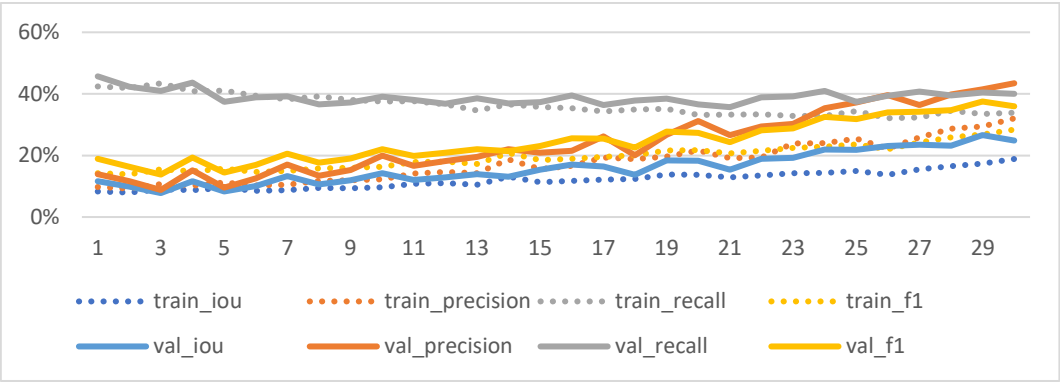


Fig. 43. Evolución de las Métricas durante el entrenamiento-validación del Prompt Encoder (dataset combinado)

4.4.1.3 Mask Decoder + Prompt Encoder

El ajuste combinado del Mask Decoder y el Prompt Encoder permite evaluar si la combinación de ambos componentes mejora el rendimiento general en comparación con su ajuste individual. En la Fig. 44, se presenta la evolución de la pérdida durante el entrenamiento y la validación.

El comportamiento de las pérdidas muestra una tendencia inicial similar al ajuste individual del Mask Decoder. En las primeras épocas, se observa una caída significativa con la pérdida de entrenamiento reduciéndose de 0.45 a 0.20 y la pérdida de validación de 0.31 a 0.24 hacia la época 7. Posteriormente, la reducción es más gradual, alcanzando valores estables de 0.15 para el entrenamiento y 0.21 para la validación a partir de la época 20.

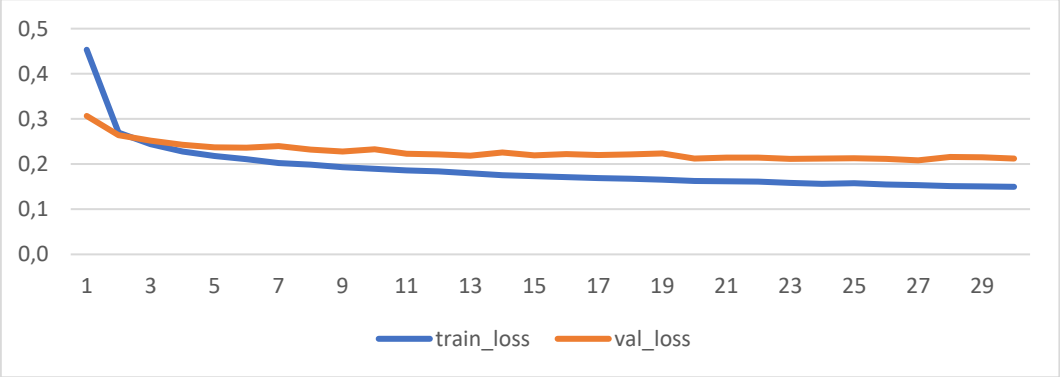


Fig. 44. Evolución de pérdida durante el ajuste del Mask Decoder + Prompt Encoder (dataset combinado

Una vez observada que la pérdida se mantiene en valores relativamente bajos, es de esperar que las métricas obtengan buenos resultados. Tal y como se puede ver en la Fig. 45, la mayor parte del aprendizaje se produce en las primeras etapas, especialmente en la primera donde ya se obtienen valores de validación de 55 % IoU, 77 % precisión, 66 % recall y 70 % F1-score. En las siguientes épocas aumentan gradual y muy levemente hasta alcanzar entorno a la época 22 los mejores resultados de IoU (66,27%), recall (79,54%) y F1-score (79,13%) y en la época 26 se alcanza el mejor resultado de precisión, siendo este de 84,15%.

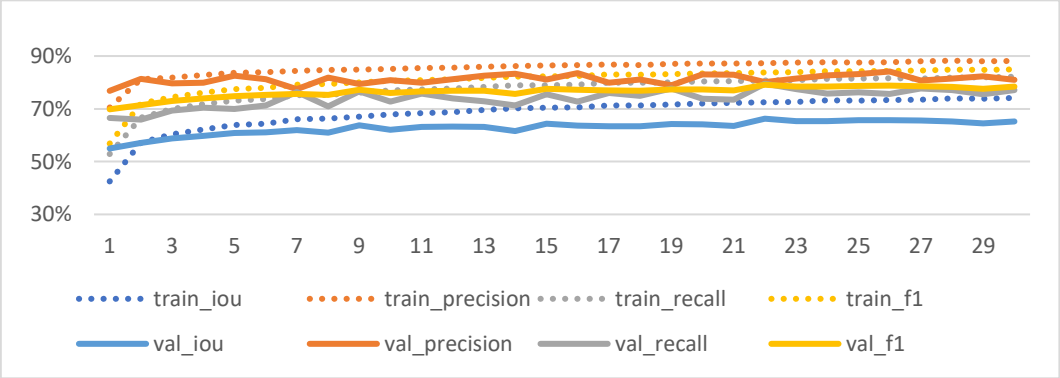


Fig. 45. Evolución de las Métricas durante el entrenamiento-validación del Mask Decoder + Prompt Encoder (dataset combinado

4.4.1.4 Image Encoder

En la Fig. 46 se observa que la pérdida de entrenamiento empieza con valores relativamente bajos, inferiores a 0.33 en la primera época. Este descenso inicial indica que el modelo logra identificar rápidamente patrones significativos. A lo largo del entrenamiento la pérdida de entrenamiento disminuye de forma muy progresiva alcanzando valores mínimos de aproximadamente 0.07 al finalizar la época 30, lo que sugiere una convergencia clara. Por otro lado, la pérdida de validación sigue una tendencia de descenso menos pronunciada, comenzando en 0.18 y estabilizándose cerca de 0.14 en la última época.

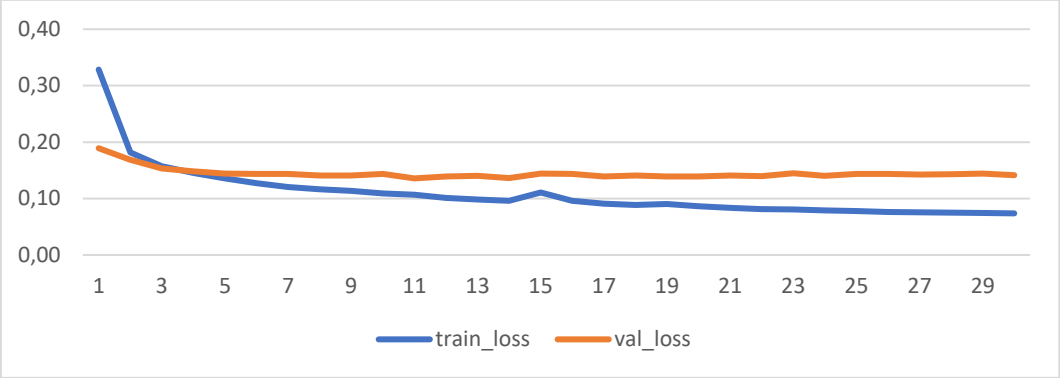


Fig. 46. Evolución de pérdida durante el ajuste del Image Encoder (dataset combinado)

En cuanto a las métricas, tal y como se muestra en la Fig. 47, el mayor crecimiento se da en las primeras épocas, obteniendo valores altos ya en la validación de la primera época: 69,32% IoU, 86,67 % precisión, 77,44% recall y 81,26% F1-score. Las mejores métricas obtenidas son: 76,64 % IoU, 89.91 % precisión, 84,89 % recall y 86,50 % F1-score, obtenidas la mejor IoU, recall y F1-score en la época 14, mientras que el mejor resultado de precisión fue obtenido en la época 24.

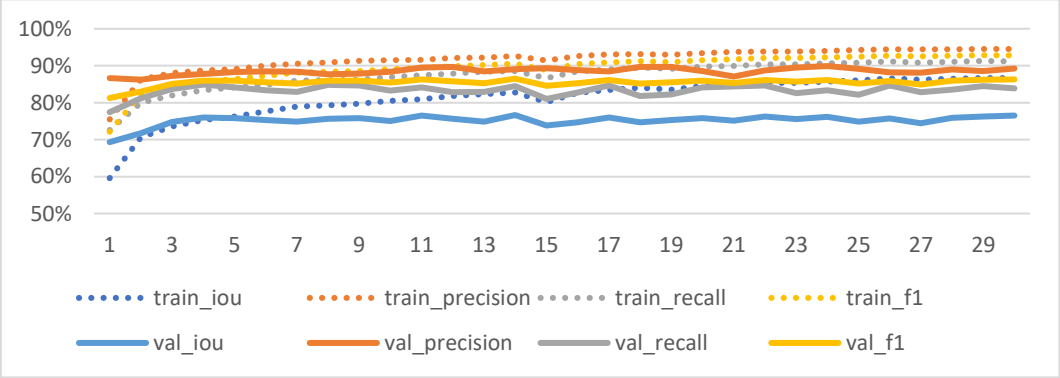


Fig. 47. Evolución de las Métricas durante el entrenamiento-validación del Image Encoder (dataset combinado)

4.4.1.5 Fine-tuning completo

El último ajuste realizado con el conjunto de datos combinado fue el fine-tuning completo. Este enfoque, que implica la actualización de los parámetros de los tres componentes del modelo, resultó en un rendimiento destacado consolidándose como el método con los mejores resultados globales de entrenamiento y validación junto al Image Encoder.

Como se muestra en la Fig. 48, la evolución de la pérdida durante el entrenamiento y la validación siguió una tendencia similar a la observada en el ajuste del Image Encoder. La pérdida en la etapa de entrenamiento comenzó a disminuir rápidamente, estabilizándose en valores bajos de 0.07 hacia las últimas épocas. De manera similar, la pérdida de validación presentó una tendencia estable, convergiendo en un valor aproximado de 0.14 al final del entrenamiento. Esta consistencia entre las pérdidas de entrenamiento y validación refuerza la idea de un modelo bien ajustado.

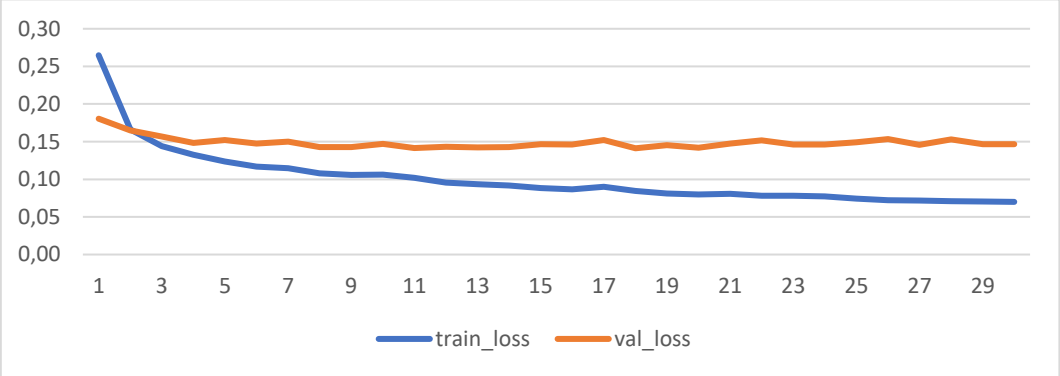


Fig. 48. Evolución de la pérdida durante el ajuste completo (dataset combinado)

En cuanto a las métricas de evaluación (Fig. 49), el crecimiento es similar al visto en el ajuste de los otros componentes, excepto el del Prompt Encoder, donde se produce un gran aumento en las primeras épocas estabilizándose a partir de la quinta época. Los mejores valores de validación obtenidos fueron: 76,41 % IoU, 90,16% precisión, 86,15 % recall y 86,35 % F1-score.

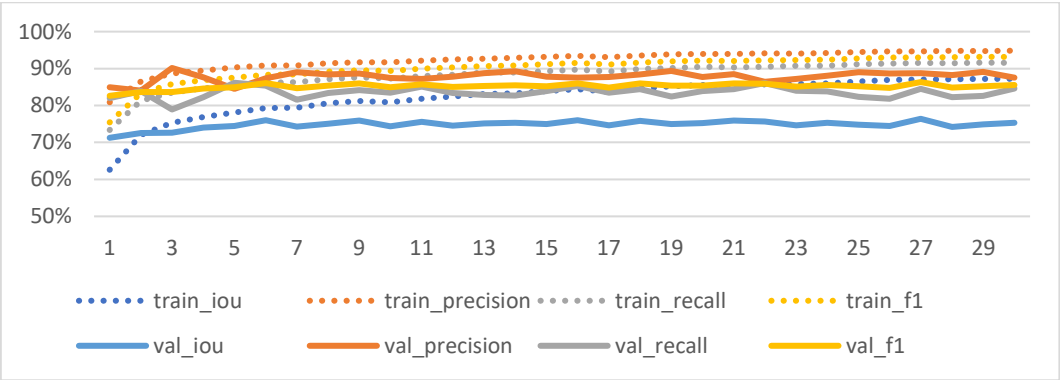


Fig. 49. Evolución de las Métricas durante el entrenamiento-validación del modelo completo (dataset combinado)

4.4.2 Resultados de la inferencia

Una vez estudiados los resultados obtenidos durante las etapas de entrenamiento y validación, se procedió a evaluar el desempeño de los modelos en imágenes del mismo conjunto de datos, pero no utilizadas previamente en dichas etapas. Esta evaluación permite analizar la capacidad de los modelos para generalizar y realizar segmentaciones precisas en datos no vistos durante el entrenamiento

De manera coherente con los resultados observados en las etapas previas, las métricas de inferencia (Fig. 50) indican que el modelo base de SAM y el modelo ajustado únicamente en el Prompt Encoder obtienen los peores resultados en todas las métricas, lo cual

confirma sus limitaciones para capturar las características necesarias para la segmentación efectiva de grietas. En contraste, los modelos ajustando el Image Encoder y el fine-tuning completo destacan como los mejores, con resultados significativamente más altos en todas las métricas evaluadas. Así mismo, también se puede ver que el ajuste del Mask Decoder, aun obteniendo una tasa de acierto ligeramente inferior a los mencionados anteriormente, alcanza altas tasas de éxito. En cambio, al ajuste combinado de Mask Decoder y Prompt Encoder no produce una mejora significativa respecto al ajuste únicamente del Mask Decoder.

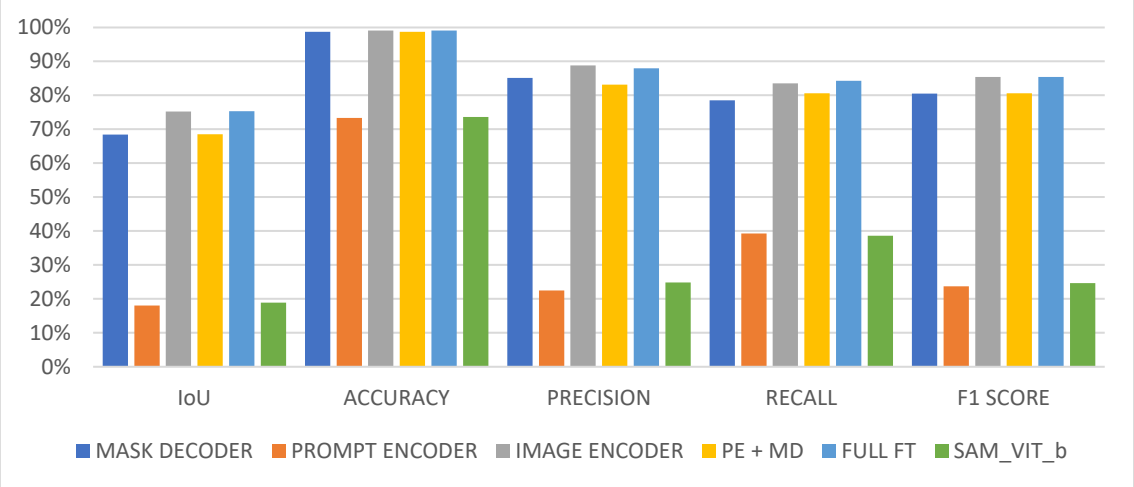


Fig. 50. Comparativa de las métricas obtenidas en la etapa de test sobre el dataset conjunto

Los valores obtenidos para estos modelos en las métricas clave se resumen en la Tabla 8. Los modelos fine-tuning completo y el ajuste del Image Encoder muestran un rendimiento sobresaliente, obteniendo métricas similares y las más altas en esta evaluación. Ambos alcanzaron aproximadamente un 75% IoU, 88% precisión, 84% recall, y 85% F1-score, lo cual indica un rendimiento notable y consistente en la segmentación de grietas.

Tabla 8. Resultados de las métricas de la inferencia sobre el dataset completo

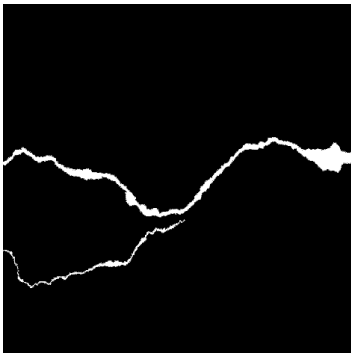
	IoU	ACCURACY	PRECISION	RECALL	F1-SCORE
MASK DECODER	0,6844	0,9875	0,8518	0,7852	0,8053
PROMPT ENCODER	0,1806	0,7338	0,2251	0,3927	0,2369
IMAGE ENCODER	0,7524	0,9908	0,8883	0,8349	0,8539
PE + MD	0,6851	0,9872	0,8314	0,8057	0,8058
FULL FT	0,7531	0,9909	0,8797	0,8426	0,8545
SAM_VIT_b	0,1894	0,7358	0,2481	0,3864	0,2469

Para complementar el análisis, en la Tabla 9, Tabla 10 y Tabla 11 se presentan ejemplos visuales de las máscaras predichas por los diferentes modelos con un ejemplo de cada dataset original. Estas ilustraciones permiten comparar visualmente las predicciones respecto al ground-truth. En particular, se observa que los modelos fine-tuning completo, Image Encoder, Mask Decoder y Mask Decoder + Prompt Encoder generan máscaras que reflejan de manera más precisa las grietas presentes en las imágenes. En comparación, los resultados del modelo base y el modelo ajustando el Prompt Encoder no consiguen

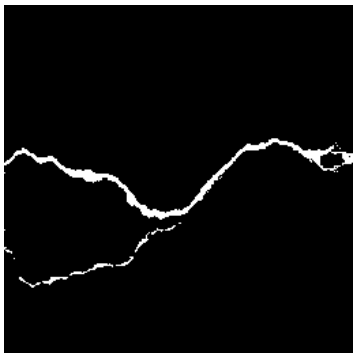
generar máscaras cercanas a la anotación (el modelo ajustando el prompt encoder solo consigue una segmentación similar a la anotación en la imagen de la Tabla 10).

Tabla 9. Máscaras obtenidas durante la etapa de test del dataset combinado, perteneciente a Concrete Crack Dataset

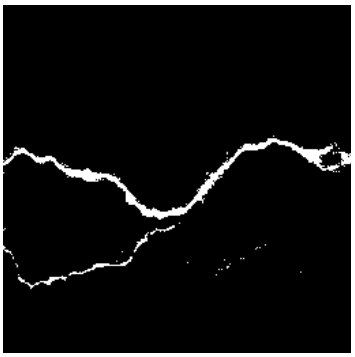
IMAGEN ORIGINAL



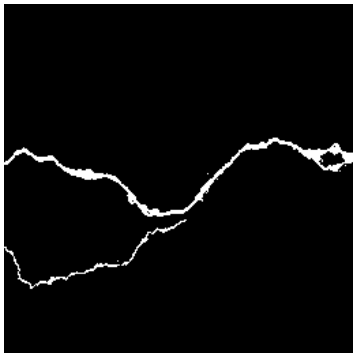
GROUND-TRUTH



MASK DECODER



MASK DECODER + PROMPT ENCODER



FULL FINE-TUNE

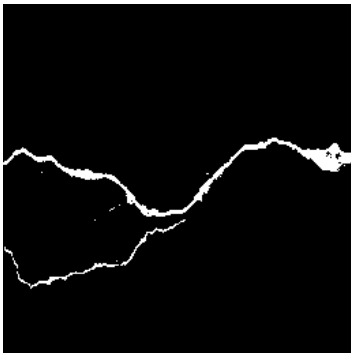
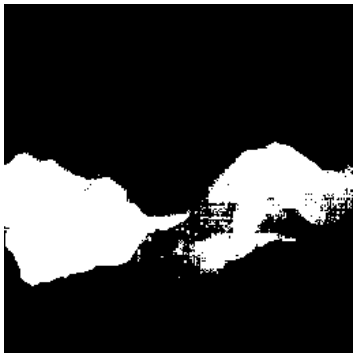


IMAGE ENCODER



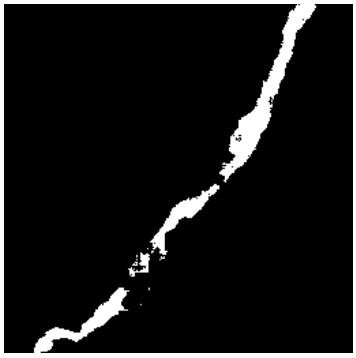
PROMPT ENCODER

SAM_VITB

Tabla 10. Máscaras obtenidas durante la etapa de test del dataset combinado, perteneciente a DeepCrack Dataset

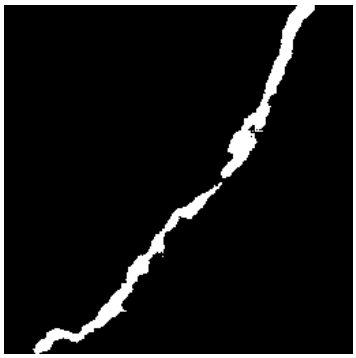
IMAGEN ORIGINAL

GROUND-TRUTH



MASK DECODER

MASK DECODER + PROMPT ENCODER



FULL FINE-TUNE

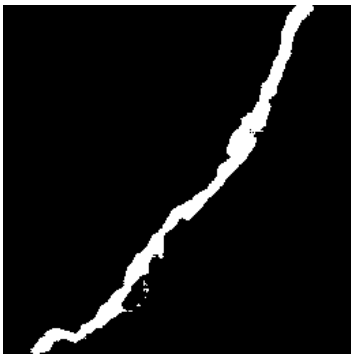
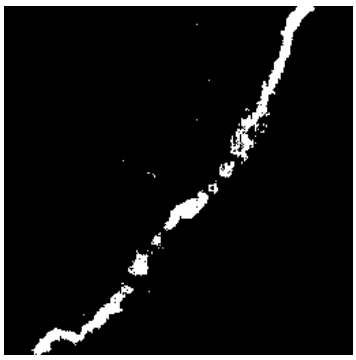
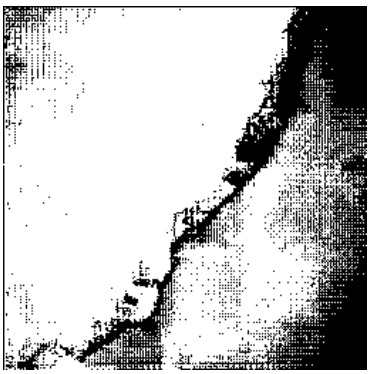


IMAGE ENCODER



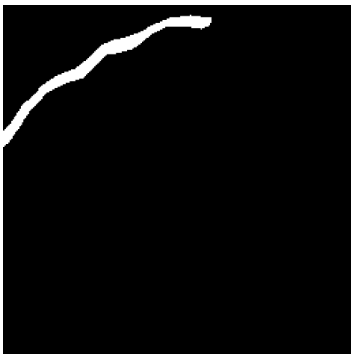
PROMPT ENCODER



SAM_VITB

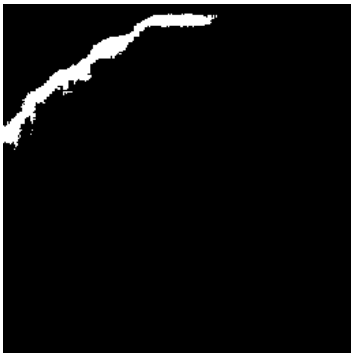
Tabla 11. Máscaras obtenidas durante la etapa de test del dataset combinado, perteneciente a Marble Crack Dataset

IMAGEN ORIGINAL



GROUND-TRUTH

MASK DECODER

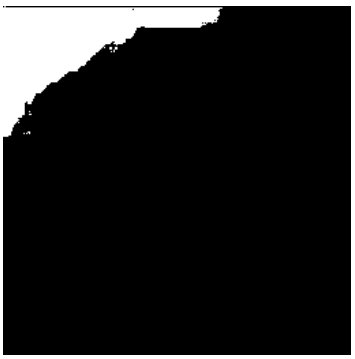


MASK DECODER + PROMPT ENCODER

FULL FINE-TUNE

IMAGE ENCODER

PROMPT ENCODER



SAM_VITB

4.5 Recursos computacionales

En esta sección se detallan los recursos computacionales empleados durante el entrenamiento de los componentes del modelo, considerando las configuraciones de hardware utilizadas, los requerimientos de memoria y el tiempo de ejecución por época. Los experimentos se llevaron a cabo utilizando la plataforma Google Colab, empleando dos tipos de GPU: NVIDIA Tesla T4 y NVIDIA A100, cuyas unidades informáticas por hora (UI/h) oscilan entre 0 UI/h y 1,44 UI/h para la NVIDIA Tesla T4 y entre 8,47 UI/h y 10,87 UI/h para la NVIDIA A100.

A continuación, se presentan los recursos computacionales registrados en el fine-tuning del experimento 1 y 4.

Durante el fine-tuning de SAM con el Concrete Crack Dataset, **experimento 1**, se registraron los valores de los recursos computacionales consumidos durante su ejecución, los cuales se ilustran en la Tabla 12.

Tabla 12. Recursos computacionales requeridos en el fine-tuning de SAM con el Concrete Crack Dataset

	RAM (GB)	VRAM (GB)	t/época (s)	GPU	UI/h
MASK DECODER	3,7	5,9	174	NVIDIA Tesla T4	0-1,44
PROMPT ENCODER	3,8	5,9	174		
PE + MD	3,4	5,9	181		
IMAGE ENCODER	4,3	19,1	134	NVIDIA A100	8,47-10,87
FULL FT	4,3	19,1	130		

Como se puede observar, los ajustes del Mask Decoder, Prompt Encoder y Prompt Encoder + Mask Decoder fueron ejecutados utilizando la NVIDIA Tesla T4, debido a sus menores requerimientos de memoria de GPU (VRAM) y un consumo eficiente de unidades informáticas por hora. En contraste, el ajuste del Image Encoder y el Fine-tuning completo demandaron un mayor uso de VRAM, por lo que fue necesaria utilizar la NVIDIA A100, reduciendo ligeramente los tiempos de ejecución a costa de un mayor consumo de unidades de cómputo.

Respecto al entrenamiento utilizando el conjunto de datos combinado, **experimento 4**, mostró un incremento notable en los requerimientos de memoria y tiempo por época, como se detalla a continuación en la Tabla 13:

Al trabajar con el dataset combinado, los ajustes iniciales (Mask Decoder, Prompt Encoder y Prompt Encoder + Mask Decoder) siguieron aprovechando la NVIDIA Tesla T4. Sin embargo, se evidenció un incremento en el uso de VRAM y tiempo por época debido a la mayor complejidad y tamaño del dataset. Los ajustes del Image Encoder y del fine-tuning completo mantuvieron requerimientos elevados de VRAM, empleando nuevamente la NVIDIA A100 para optimizar el proceso.

Tabla 13. Recursos computacionales requeridos en el fine-tuning de SAM con el dataset combinado

	RAM (GB)	VRAM (GB)	t/época (s)	GPU	UI/h
MASK DECODER	4,5	11,1	380	NVIDIA Tesla T4	0-1,44
PROMPT ENCODER	4,1	11,1	376		
PE + MD	3,9	14,1	401		
IMAGE ENCODER	4,6	37,1	250	NVIDIA A100	8,47-10,87
FULL FT	4,6	37,1	263		

4.6 Segmentaciones de imágenes tomadas de la vía pública

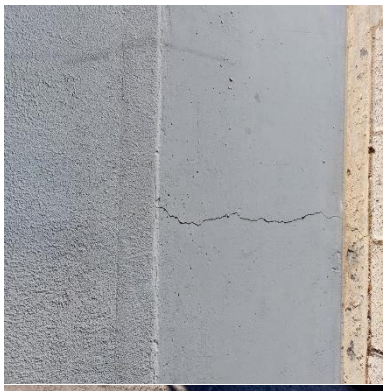
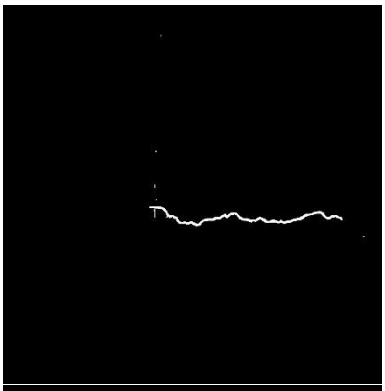
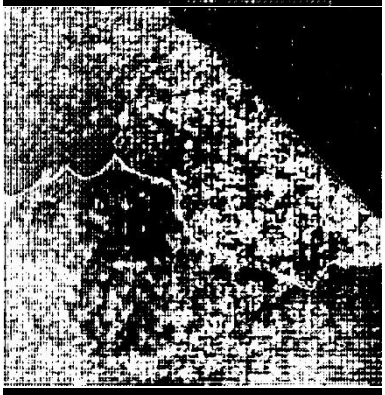
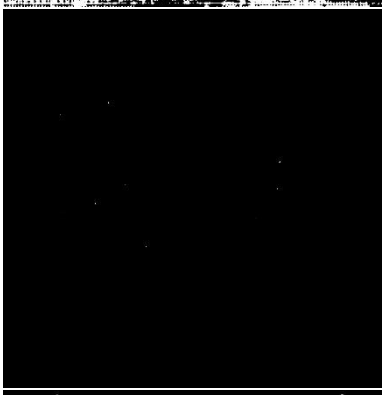
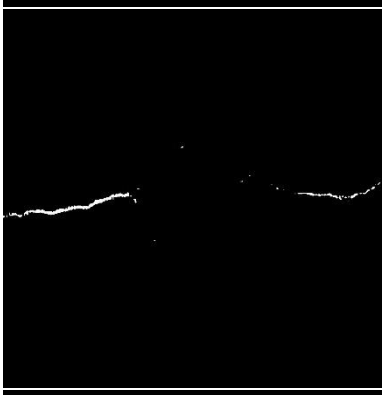
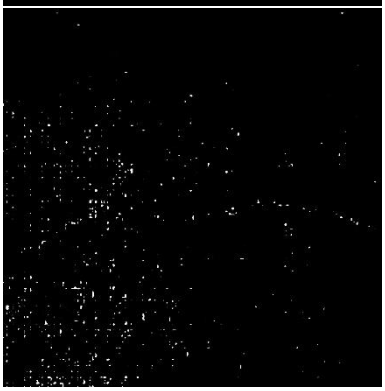
Por último, se llevó a cabo un experimento adicional con el propósito de evaluar en mayor profundidad la eficacia de los modelos ajustados en comparación con el modelo base. Para ello, se tomaron imágenes de la vía pública de la ciudad de Albacete, centrándose principalmente en la segmentación semántica de grietas presentes en distintos tipos de superficies, como muros, edificios y carreteras. Este análisis permitió probar la capacidad de los modelos para identificar y segmentar correctamente estas imperfecciones estructurales en un entorno real y no controlado.

Para la fase de inferencia se utilizó tanto el modelo base de SAM (vit-B) como el modelo completamente ajustado con el dataset conjunto del cuarto experimento. La finalidad era observar las diferencias en el desempeño de ambos modelos al enfrentarse a imágenes reales sin intervención manual previa. En la Tabla 14 se presentan los resultados obtenidos, los cuales evidencian que el modelo base mostró una clara incapacidad para realizar segmentaciones precisas de las grietas, fallando en la delimitación de los contornos y generando máscaras que no se corresponden con la forma real de las fisuras.

Por otro lado, las segmentaciones obtenidas con el modelo ajustado arrojaron resultados más prometedores, aunque con cierta variabilidad. En la mayoría de los casos, las máscaras generadas por este modelo se aproximaron mejor a la región real de la grieta, lo que indica una mejora significativa respecto al modelo base. No obstante, en algunos escenarios específicos se evidenciaron segmentaciones menos precisas, posiblemente debido a la complejidad de las texturas o las condiciones lumínicas de las imágenes utilizadas.

Es importante destacar que en este experimento no se realizó la anotación manual de las imágenes. La razón principal es que el objetivo de este experimento no era calcular métricas exactas de desempeño, sino más bien realizar una comparación visual y cualitativa entre los resultados obtenidos por el modelo base y el modelo ajustado. De este modo, se pudo evaluar de manera general la efectividad del ajuste realizado y su impacto en la segmentación de grietas en entornos reales.

Tabla 14. Máscaras de imágenes obtenidas en la vía pública

MODELO BASE	IMAGEN ORIGINAL	MODELO AJUSTADO
		
		
		
		

5 Conclusiones y trabajos futuros

En este trabajo se ha propuesto el uso del modelo fundacional Segment Anything Model (SAM) para abordar el problema de segmentación de grietas en imágenes de materiales de construcción como el hormigón y el mármol. Para ello, se ha empleado la versión base preentrenada de SAM, realizando un proceso de fine-tuning para adaptar el modelo a las características específicas de las grietas en diferentes superficies.

Los experimentos realizados han permitido identificar las limitaciones del modelo base de SAM en la segmentación de grietas. En cambio, tras el proceso de fine-tuning se logró una segmentación eficaz de grietas en imágenes de hormigón. Los resultados fueron igualmente satisfactorios al enfrentarse a datos de materiales con texturas y características distintas como el mármol. Mediante el uso de métricas como F1-score e IoU, así como análisis visuales comparativos entre las máscaras obtenidas y las anotaciones, se validó la capacidad de los modelos ajustados para generalizar más allá de los datos utilizados en el entrenamiento.

Respecto al aprendizaje, durante las primeras fases del entrenamiento el modelo mostró una rápida mejora en la segmentación, lo que permitió obtener buenos resultados con relativamente pocas épocas. Sin embargo, a medida que avanzaba el entrenamiento, las mejoras se volvieron marginales, lo que sugiere que la mayor parte del aprendizaje se produjo en las primeras etapas, por lo que se podría optimizar el entrenamiento con un menor número de épocas.

Se observó también la influencia significativa del Image Encoder y en menor medida del Mask Decoder en el rendimiento del modelo ajustado. Sin embargo, el ajuste del Prompt Encoder individualmente no mejoró sustancialmente el desempeño en comparación con el modelo base. Además, se comprobó que el ajuste del Image Encoder puede impactar en la capacidad de generalización del modelo, particularmente cuando se infieren imágenes de materiales con texturas y características diferentes a las del dataset de entrenamiento.

Además, como se observó en el experimento adicional, el modelo ajustado demostró una mejora significativa en la segmentación de grietas en entornos reales, aunque con cierta variabilidad en los resultados. Este hallazgo refuerza la importancia de continuar explorando estrategias de fine-tuning y la posible combinación con técnicas de postprocesamiento para refinar aún más la segmentación en escenarios no controlados.

Para futuras líneas de investigación se sugiere explorar el uso de versiones avanzadas del modelo SAM, como SAM2 que permite la segmentación en vídeos. Este avance permitiría su uso en la inspección del estado del asfaltado de las carreteras, donde la segmentación en tiempo real podría detectar y clasificar deterioros de forma automatizada.

Bibliografía

[1] A. D. Costea, A. Petrovai, y S. Nedevschi, «Fusion Scheme for Semantic and Instance-level Segmentation», *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-November, pp. 3469-3475, dic. 2018, doi: 10.1109/ITSC.2018.8570006.

[2] A. B. Abdusalomov, M. Mukhiddinov, y T. K. Whangbo, «Brain Tumor Detection Based on Deep Learning Approaches and Magnetic Resonance Imaging», *Cancers (Basel)*, vol. 15, n.o 16, p. 4172, ago. 2023, doi: 10.3390/CANCERS15164172.

[3] S. Roy y M. S. Rahman, «Emergency Vehicle Detection on Heavy Traffic Road from CCTV Footage Using Deep Convolutional Neural Network», *2nd International Conference on Electrical, Computer and Communication Engineering, ECCE 2019*, abr. 2019, doi: 10.1109/ECACE.2019.8679295.

[4] X. Shi, S. Zhang, M. Cheng, L. He, X. Tang, y Z. Cui, «Few-shot semantic segmentation for industrial defect recognition», *Comput Ind*, vol. 148, p. 103901, jun. 2023, doi: 10.1016/J.COMPIND.2023.103901.

[5] S. K. Panda, Y. Lee, y M. Khalid Jawed, «Agronav: Autonomous Navigation Framework for Agricultural Robots and Vehicles using Semantic Segmentation and Semantic Line Detection», en *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, IEEE, jun. 2023, pp. 6272-6281. doi: 10.1109/CVPRW59228.2023.00667.

[6] N. Otsu, «THRESHOLD SELECTION METHOD FROM GRAY-LEVEL HISTOGRAMS.», *IEEE Trans Syst Man Cybern*, vol. SMC-9, n.o 1, pp. 62-66, 1979, doi: 10.1109/TSMC.1979.4310076.

[7] I. Sobel y G. Feldman, «An Isotropic 3×3 image gradient operator», 1990, doi: 10.13140/RG.2.1.1912.4965.

[8] J. Canny, «A Computational Approach to Edge Detection», *IEEE Trans Pattern Anal Mach Intell*, vol. PAMI-8, n.o 6, pp. 679-698, 1986, doi: 10.1109/TPAMI.1986.4767851.

[9] R. Adams y L. Bischof, «Seeded region growing», *IEEE Trans Pattern Anal Mach Intell*, vol. 16, n.o 6, pp. 641-647, jun. 1994, doi: 10.1109/34.295913.

[10] N. Ikonomakis, K. N. Plataniotis, M. Zervakis, y A. N. Venetsanopoulos, «Region growing and region merging image segmentation», *International Conference on Digital Signal Processing, DSP*, vol. 1, pp. 299-301, 1997, doi: 10.1109/ICDSP.1997.628077.

[11] L. Breiman, «Random forests», *Mach Learn*, vol. 45, n.o 1, pp. 5-32, oct. 2001, doi: 10.1023/A:1010933404324/METRICS.

[12] S. Bhatnagar, L. Gill, y B. Ghosh, «Drone Image Segmentation Using Machine and Deep Learning for Mapping Raised Bog Vegetation Communities», *Remote Sensing 2020, Vol. 12, Page 2602*, vol. 12, n.o 16, p. 2602, ago. 2020, doi: 10.3390/RS12162602.

[13] J. MacQueen, «Some methods for classification and analysis of multivariate observations», 1967.

[14] C. Cortes y V. Vapnik, «Support-Vector Networks», *Mach Learn*, vol. 20, n.o 3, pp. 273-297, 1995, doi: 10.1023/A:1022627411411.

[15] Y. LeCun, Y. Bengio, y G. Hinton, «Deep learning», *Nature*, vol. 521, n.o 7553, pp. 436-444, may 2015, doi: 10.1038/nature14539.

[16] J. Long, E. Shelhamer, y T. Darrell, «Fully convolutional networks for semantic segmentation», *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June-2015, pp. 431-440, oct. 2015, doi: 10.1109/CVPR.2015.7298965.

[17] H. Oliveira, C. Silva, G. L. S. Machado, K. Nogueira, y J. A. dos Santos, «Fully Convolutional Open Set Segmentation», *Mach Learn*, vol. 112, n.o 5, pp. 1733-1784, jun. 2020, doi: 10.1007/s10994-021-06027-1.

[18] W. Weng y X. Zhu, «U-Net: Convolutional Networks for Biomedical Image Segmentation», *IEEE Access*, vol. 9, pp. 16591-16603, may 2015, doi: 10.1109/ACCESS.2021.3053408.

[19] N. Ibtehaz y M. S. Rahman, «MultiResUNet : Rethinking the U-Net Architecture for Multimodal Biomedical Image Segmentation», 2019.

[20] K. He, G. Gkioxari, P. Dollár, y R. Girshick, «Mask R-CNN», mar. 2017.

[21] 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 Trans Pattern Anal Mach Intell*, vol. 40, n.o 4, pp. 834-848, jun. 2016, doi: 10.1109/TPAMI.2017.2699184.

[22] A. Dosovitskiy *et al.*, «AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE», Accedido: 22 de enero de 2025. [En línea]. Disponible en: https://github.com/

[23] W. Wang, H. He, y C. Ma, «An Improved Deeplabv3+ Model for Semantic Segmentation of Urban Environments Targeting Autonomous Driving», *INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL*, vol. 18, n.o 6, p. 5879, oct. 2023, doi: 10.15837/IJCCC.2023.6.5879.

[24] A. Vaswani *et al.*, «Attention Is All You Need», *Adv Neural Inf Process Syst*, vol. 2017-December, pp. 5999-6009, jun. 2017, Accedido: 22 de enero de 2025. [En línea]. Disponible en: https://arxiv.org/abs/1706.03762v7

[25] «Segment Anything | Meta AI». Accedido: 2 de enero de 2025. [En línea]. Disponible en: https://segment-anything.com/demo

[26] Ç. F. Özgenel, «Concrete Crack Segmentation Dataset», vol. 1, 2019, doi: 10.17632/JWSN7TFBRP.1.

[27] Y. Liu, J. Yao, X. Lu, R. Xie, y L. Li, «DeepCrack: A deep hierarchical feature learning architecture for crack segmentation», *Neurocomputing*, vol. 338, pp. 139-153, abr. 2019, doi: 10.1016/J.NEUCOM.2019.01.036.

[28] «Mármol Arabescato Corchia - Mármol Spain». Accedido: 28 de diciembre de 2024. [En línea]. Disponible en: https://www.marmolspain.es/producto/arescato-corchia/

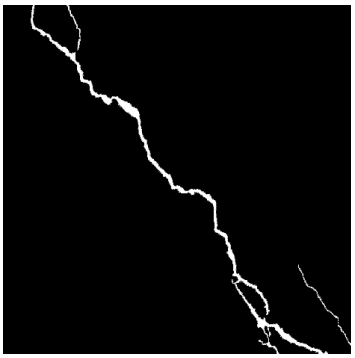
[29] «Cómo reparar las grietas en el mármol - Químicas Novelda». Accedido: 28 de diciembre de 2024. [En línea]. Disponible en: https://www.quimicasnovelda.com/masillas-reparadoras/como-reparar-las-grietas-en-el-marmol/

[30] «MachineLearningVisionRG/mcs-dataset: Marble Crack Segmentation (MCS) Dataset for semantic segmentation on marble images.» Accedido: 28 de diciembre de 2024. [En línea]. Disponible en: https://github.com/MachineLearningVisionRG/mcs-dataset

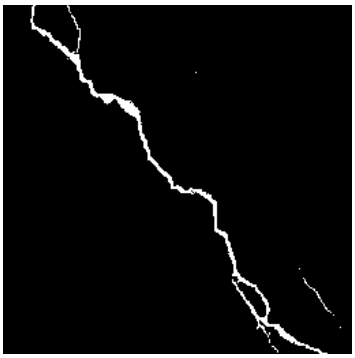
Anexo A. Resultados adicionales del experimento 1

Tabla 15. Máscaras obtenidas de la imagen 060.jpg

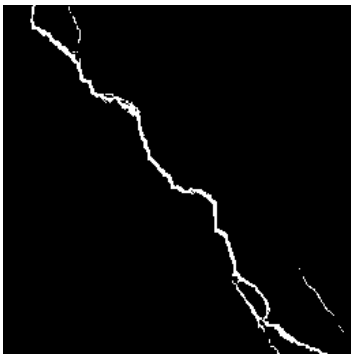
IMAGEN ORIGINAL



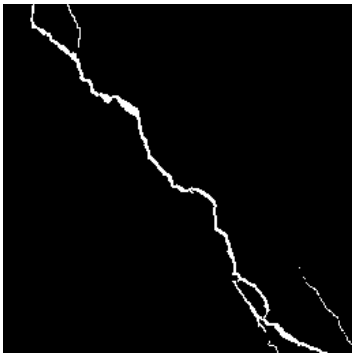
GROUND-TRUTH



MASK DECODER



MASK DECODER + PROMPT ENCODER



FULL FINE-TUNE

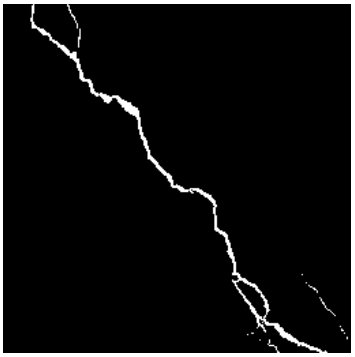
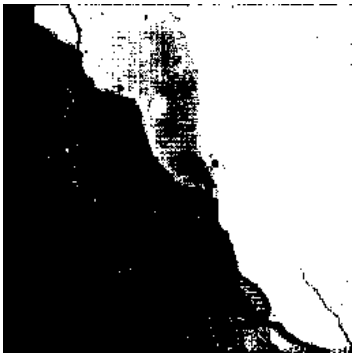
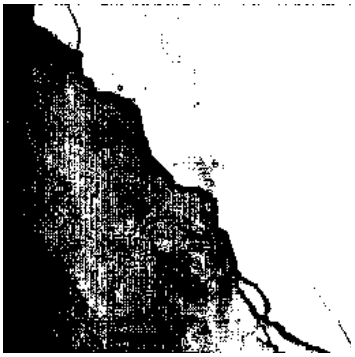


IMAGE ENCODER



PROMPT ENCODER

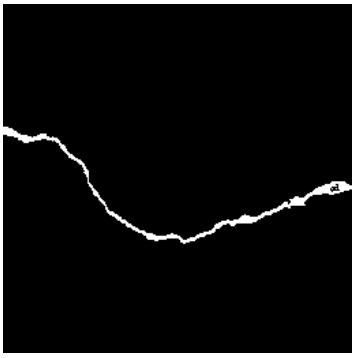


SAM_VITB

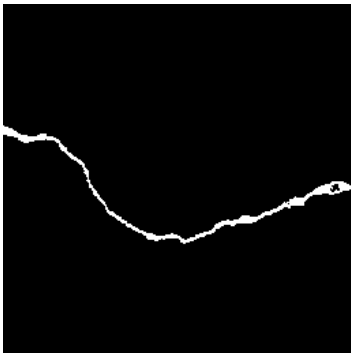
Tabla 16. Máscaras obtenidas de la imagen 179.jpg

IMAGEN ORIGINAL

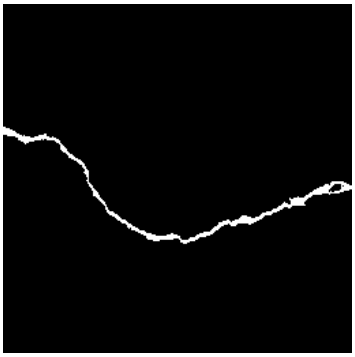
GROUND-TRUTH



MASK DECODER



MASK DECODER + PROMPT ENCODER



FULL FINE-TUNE

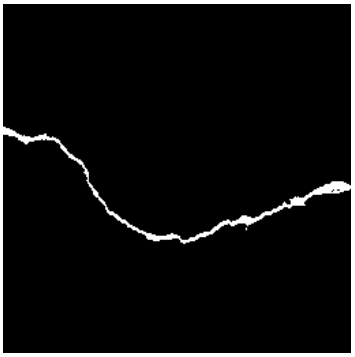
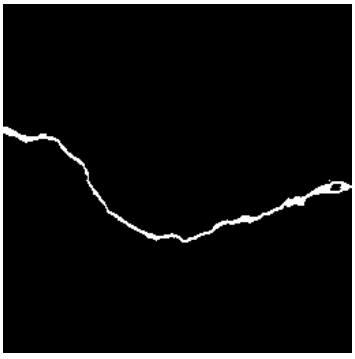
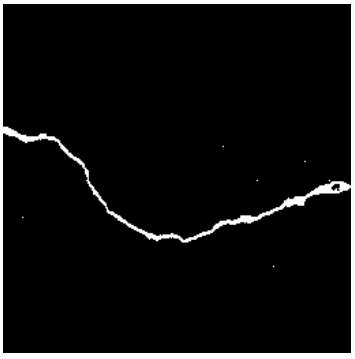


IMAGE ENCODER



PROMPT ENCODER

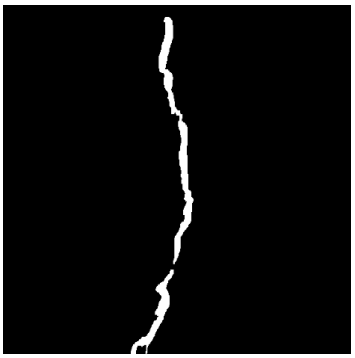


SAM_VITB

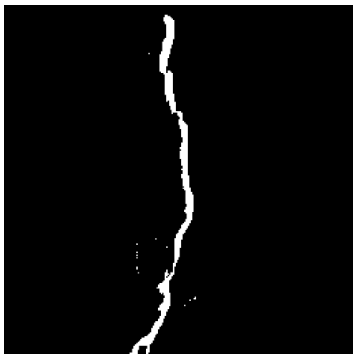
Anexo B. Resultados adicionales del experimento 2

Tabla 17. Máscaras obtenidas de la imagen IMG_6536-3.jpg

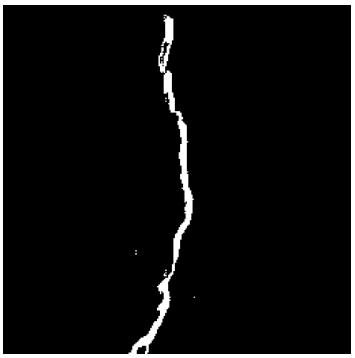
IMAGEN ORIGINAL



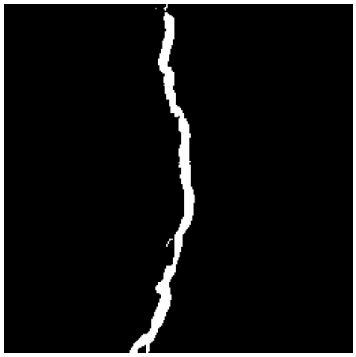
GROUND-TRUTH



MASK DECODER



MASK DECODER + PROMPT ENCODER



FULL FINE-TUNE

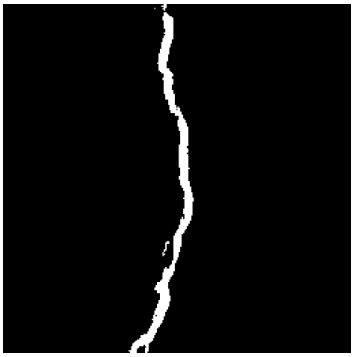
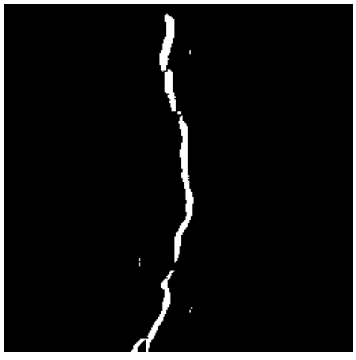
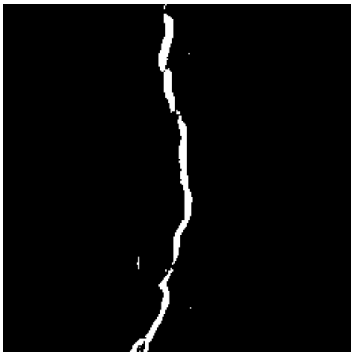


IMAGE ENCODER



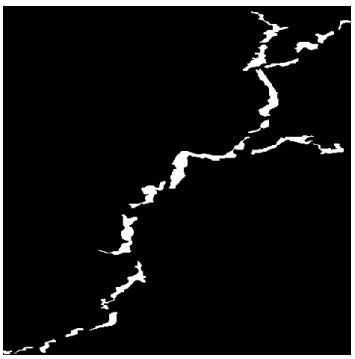
PROMPT ENCODER



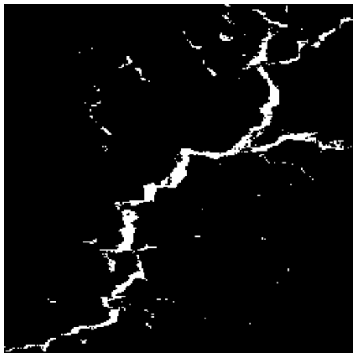
SAM_VITB

Tabla 18. Máscaras obtenidas de la imagen IMG_6522-1.jpg

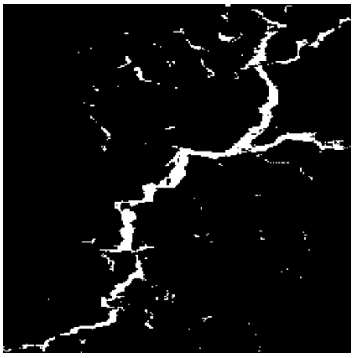
IMAGEN ORIGINAL



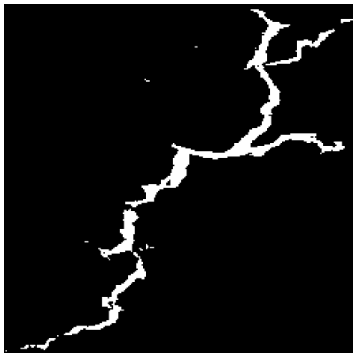
GROUND-TRUTH



MASK DECODER



MASK DECODER + PROMPT ENCODER



FULL FINE-TUNE

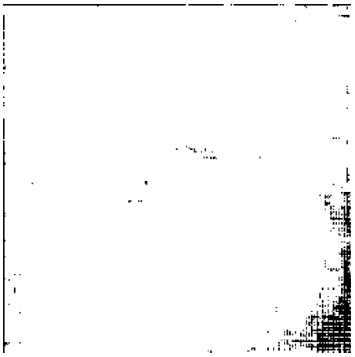
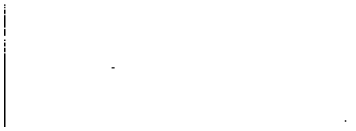


IMAGE ENCODER



PROMPT ENCODER

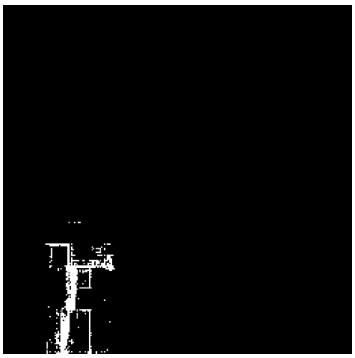
SAM_VITB

Anexo C. Resultados adicionales del experimento 3

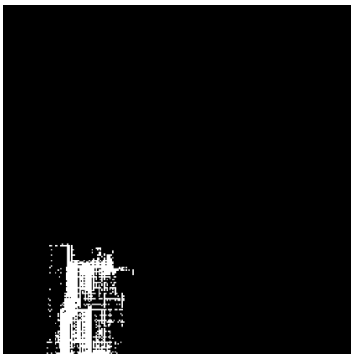
Tabla 19. Máscaras obtenidas de la imagen 1280_512_20210525_15150.jpg

IMAGEN ORIGINAL

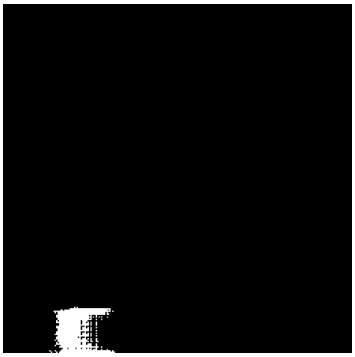
GROUND-TRUTH



MASK DECODER

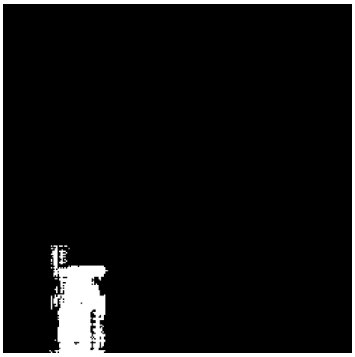


MASK DECODER + PROMPT ENCODER

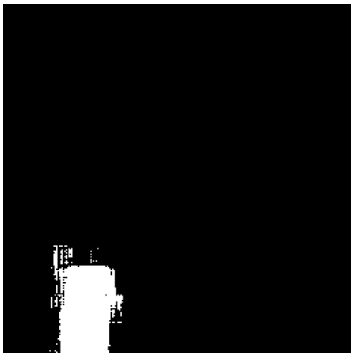


FULL FINE-TUNE

IMAGE ENCODER



PROMPT ENCODER

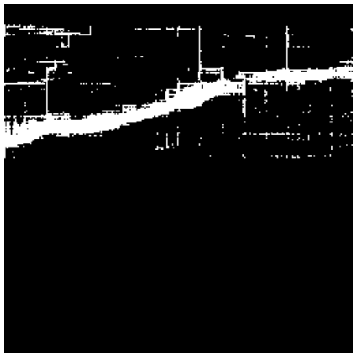


SAM_VITB

Tabla 20. Máscaras obtenidas de la imagen 768_2816_20210525_14443.jpg

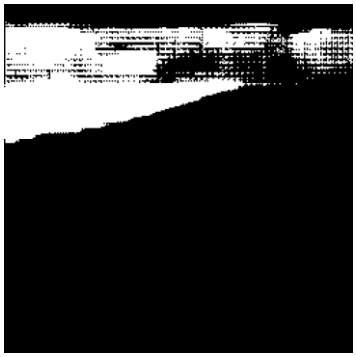
IMAGEN ORIGINAL

GROUND-TRUTH



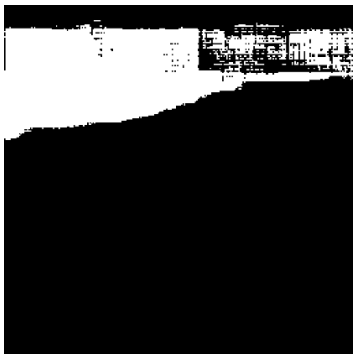
MASK DECODER

MASK DECODER + PROMPT ENCODER



FULL FINE-TUNE

IMAGE ENCODER



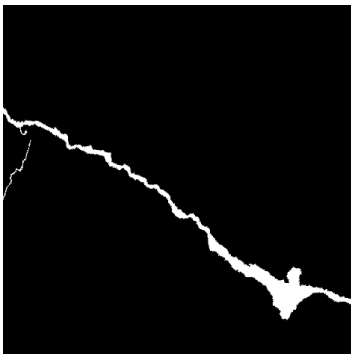
PROMPT ENCODER

SAM_VITB

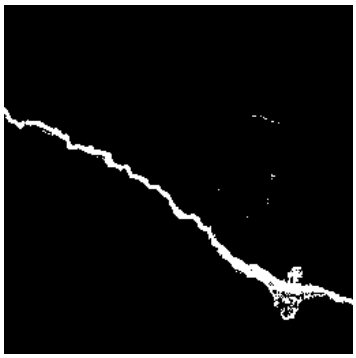
Anexo D. Resultados adicionales del experimento 4

Tabla 21. Máscaras obtenidas de la imagen 237.jpg

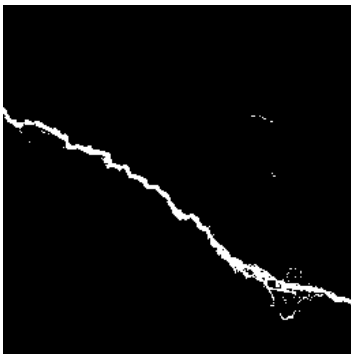
IMAGEN ORIGINAL



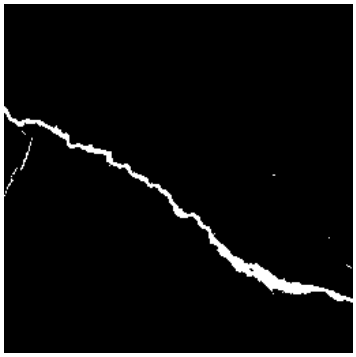
GROUND-TRUTH



MASK DECODER

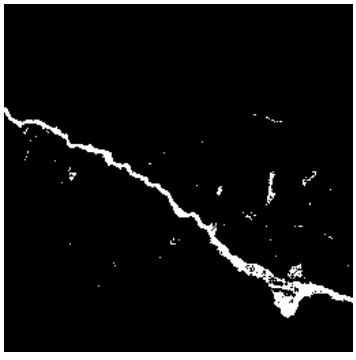


MASK DECODER + PROMPT ENCODER

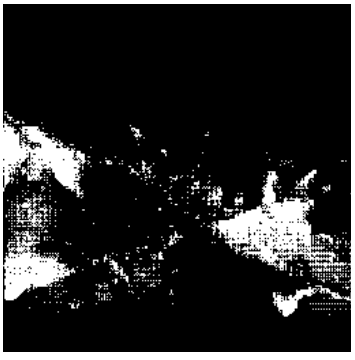


FULL FINE-TUNE

IMAGE ENCODER



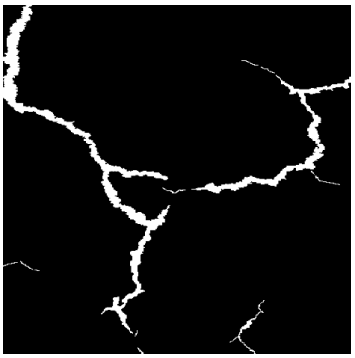
PROMPT ENCODER



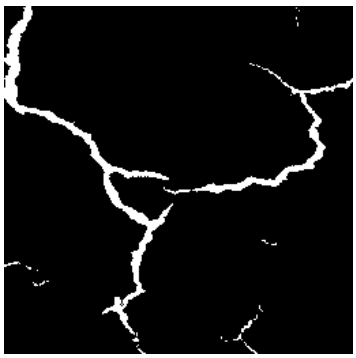
SAM_VITB

Tabla 22. Máscaras obtenidas de la imagen 11125-2.jpg

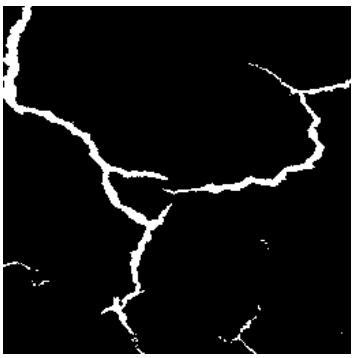
IMAGEN ORIGINAL



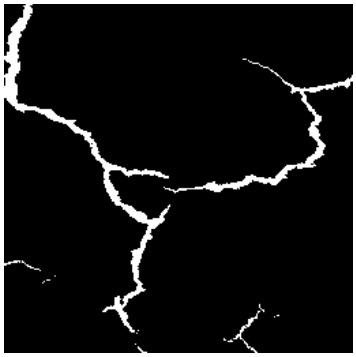
GROUND-TRUTH



MASK DECODER



MASK DECODER + PROMPT ENCODER



FULL FINE-TUNE

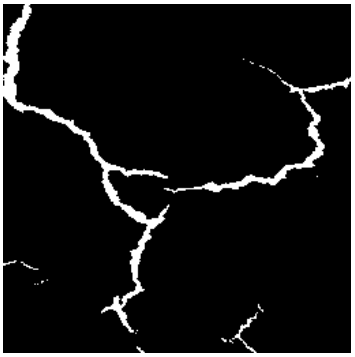
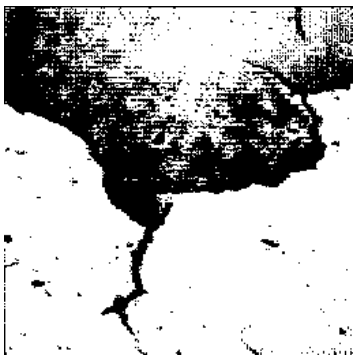
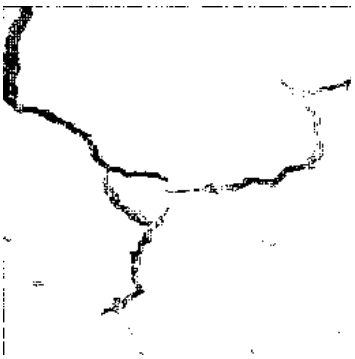


IMAGE ENCODER



PROMPT ENCODER



SAM_VITB

Tabla 23. Máscaras obtenidas de la imagen _1536_256_20210531_10561.jpg

IMAGEN ORIGINAL

GROUND-TRUTH

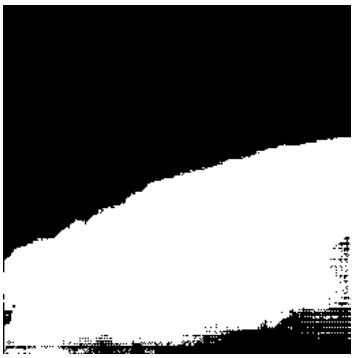
MASK DECODER

MASK DECODER + PROMPT ENCODER

FULL FINE-TUNE

IMAGE ENCODER

PROMPT ENCODER



SAM_VITB

Anexo E. Código del fine-tuning

1. Configuración inicial

```
from google.colab import drive
drive.mount('/content/drive')

!pip install -q git+https://github.com/huggingface/transformers.git
!pip install -q datasets
!pip install -q monai
```

2. Carga de datos y configuración del dataset

```
import os
from PIL import Image
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from torchvision import transforms
import numpy as np

# Definir la ruta a las imágenes y a las anotaciones
dataset_path = '/content/drive/MyDrive/Colab
Notebooks/concreteCrackSegmentationDataset/'

image_folder = os.path.join(dataset_path, 'image')
annotation_folder = os.path.join(dataset_path, 'annotation')

# Asegurarse de que las imágenes y anotaciones estén bien emparejadas
image_files = [f for f in os.listdir(image_folder) if
f.endswith('.jpg') or f.endswith('.png')]
annotation_files = [f.replace('.jpg', '.jpg').replace('.jpeg', '.jpg')
for f in image_files]  # Cambiar de .png a .jpg

# Verificar que el número de imágenes y anotaciones coincidan
assert len(image_files) == len(annotation_files), "El número de
imágenes no coincide con el número de anotaciones."

# Dividir en entrenamiento, validación y prueba
train_images, test_val_images = train_test_split(image_files,
test_size=0.3, random_state=42)
train_annotations = [f.replace('.jpg', '.jpg').replace('.jpeg',
'.jpg') for f in train_images]  # Cambiar de .png a .jpg
test_val_annotations = [f.replace('.jpg', '.jpg').replace('.jpeg',
'.jpg') for f in test_val_images]  # Cambiar de .png a .jpg

# Dividir el 30% restante en 50% para validación y 50% para prueba
(15% cada uno)
val_images, test_images = train_test_split(test_val_images,
test_size=0.5, random_state=42)
val_annotations = [f.replace('.jpg', '.jpg').replace('.jpeg', '.jpg')
for f in val_images]  # Cambiar de .png a .jpg
test_annotations = [f.replace('.jpg', '.jpg').replace('.jpeg', '.jpg')
for f in test_images]  # Cambiar de .png a .jpg

# Crear las rutas finales de las imágenes y anotaciones
train_dataset = [(os.path.join(image_folder, img),
os.path.join(annotation_folder, ann)) for img, ann in
zip(train_images, train_annotations)]
```

```
val_dataset = [(os.path.join(image_folder, img),
os.path.join(annotation_folder, ann)) for img, ann in zip(val_images,
val_annotations)]
test_dataset = [(os.path.join(image_folder, img),
os.path.join(annotation_folder, ann)) for img, ann in zip(test_images,
test_annotations)]

print(f"Conjunto de entrenamiento tiene {len(train_dataset)}
ejemplos.")
print(f"Conjunto de validación tiene {len(val_dataset)} ejemplos.")
print(f"Conjunto de prueba tiene {len(test_dataset)} ejemplos.")

from datasets import Dataset, DatasetDict

# Crear una lista de diccionarios con las rutas a las imágenes y las
anotaciones
train_data = []
for img, ann in zip(train_images, train_annotations):
    train_data.append({
        'image': os.path.join(image_folder, img),
        'annotation': os.path.join(annotation_folder, ann)
    })

# Crear el Dataset de Hugging Face
hf_train_dataset = Dataset.from_dict({
    'image': [item['image'] for item in train_data],
    'annotation': [item['annotation'] for item in train_data]
})

val_data = []
for img, ann in zip(val_images, val_annotations):
    val_data.append({
        'image': os.path.join(image_folder, img),
        'annotation': os.path.join(annotation_folder, ann)
    })

hf_val_dataset = Dataset.from_dict({
    'image': [item['image'] for item in val_data],
    'annotation': [item['annotation'] for item in val_data]
})

test_data = []
for img, ann in zip(test_images, test_annotations):
    test_data.append({
        'image': os.path.join(image_folder, img),
        'annotation': os.path.join(annotation_folder, ann)
    })

hf_test_dataset = Dataset.from_dict({
    'image': [item['image'] for item in test_data],
    'annotation': [item['annotation'] for item in test_data]
})


# Verificar la estructura final del Dataset
print(hf_train_dataset)
print(hf_val_dataset)
print(hf_test_dataset)
```

3. Función para definir los prompts (bounding boxes)

```
from PIL import Image
import numpy as np

def get_bounding_box(ground_truth_map):
    if isinstance(ground_truth_map, str):
        ground_truth_map =
np.array(Image.open(ground_truth_map).convert("L"))

    y_indices, x_indices = np.where(ground_truth_map > 0)
    x_min, x_max = np.min(x_indices), np.max(x_indices)
    y_min, y_max = np.min(y_indices), np.max(y_indices)


    H, W = ground_truth_map.shape
    x_min = max(0, x_min - np.random.randint(0, 20))
    x_max = min(W, x_max + np.random.randint(0, 20))
    y_min = max(0, y_min - np.random.randint(0, 20))
    y_max = min(H, y_max + np.random.randint(0, 20))

    bbox = [x_min, y_min, x_max, y_max]

    return bbox

"""Código para verificar si se generan correctamente los bounding
boxes"""

import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

# Obtener un ejemplo del dataset
sample = hf_train_dataset[5]

# Cargar la imagen
image_path = sample["image"]
image = Image.open(image_path).convert("RGB")
image_np = np.array(image)

# Cargar la máscara
annotation_path = sample["annotation"]
mask = Image.open(annotation_path).convert("L")
mask_np = np.array(mask)

# Obtener el bounding box para la máscara
bbox = get_bounding_box(mask_np)

# Visualizar la imagen y la máscara
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

# Mostrar la imagen
ax[0].imshow(image_np)
ax[0].set_title("Imagen con Grieta")
ax[0].add_patch(plt.Rectangle(
    (bbox[0], bbox[1]), bbox[2] - bbox[0], bbox[3] - bbox[1],
    fill=False, color='red', linewidth=2
))
ax[0].axis("off")

# Mostrar la máscara verdadera
ax[1].imshow(mask_np, cmap="gray")
```

```
ax[1].set_title("Máscara de Grieta")
ax[1].axis("off")

plt.tight_layout()
plt.show()
```

4. Preparar el dataset para trabajar con PyTorch

```
from PIL import Image
import numpy as np
from torch.utils.data import Dataset

class SAMDataset(Dataset):
    def __init__(self, dataset, processor):
        self.dataset = dataset
        self.processor = processor

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        item = self.dataset[idx]

        # Cargar la imagen y redimensionarla a 1024x1024
        image = Image.open(item["image"]).convert("RGB")
        original_width, original_height = image.size
        image_resized = image.resize((1024, 1024))
        ground_truth_mask = item["annotation"]
        ground_truth_mask =
np.array(Image.open(ground_truth_mask).convert("L"))

        # Redimensionar la máscara a 1024x1024
        ground_truth_mask_resized =
np.array(Image.fromarray(ground_truth_mask).resize((1024, 1024),
Image.NEAREST))

        # Convertir la máscara a binaria (blanco = 1, negro = 0)
        ground_truth_mask_resized = (ground_truth_mask_resized >
0).astype(np.uint8)

        # Calcula el bounding box para la máscara en la imagen
original
        prompt = get_bounding_box(ground_truth_mask)

        # Ajustar las coordenadas del bounding box de acuerdo con el
tamaño de la imagen redimensionada
        scale_x = 1024 / original_width  # Factor de escala para la
dimensión horizontal
        scale_y = 1024 / original_height  # Factor de escala para la
dimensión vertical
        x_min, y_min, x_max, y_max = prompt  # Coordenadas de la caja
original

        # Escalar las coordenadas de la caja
        x_min = int(x_min * scale_x)
        x_max = int(x_max * scale_x)
        y_min = int(y_min * scale_y)
        y_max = int(y_max * scale_y)

        # Actualizar el prompt con las coordenadas ajustadas
```

```
        prompt = [x_min, y_min, x_max, y_max]

        # Prepara la imagen y el prompt para el modelo
        inputs = self.processor(image_resized, input_boxes=[[prompt]],
return_tensors="pt")

        # Elimina la dimensión adicional que añade el procesador
        inputs = {k: v.squeeze(0) for k, v in inputs.items()}

        # Agrega la anotación
        inputs["ground_truth_mask"] = ground_truth_mask_resized

        return inputs


from transformers import SamProcessor

processor = SamProcessor.from_pretrained("facebook/sam-vit-base")

train_dataset = SAMDataset(dataset=hf_train_dataset,
processor=processor)
val_dataset = SAMDataset(dataset=hf_val_dataset, processor=processor)
test_dataset = SAMDataset(dataset=hf_test_dataset,
processor=processor)

example = train_dataset[0]
for k,v in example.items():
  print(k,v.shape)
```

5. Definición del DataLoader

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(train_dataset, batch_size=2,
shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=2, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=1, shuffle=True)

batch = next(iter(train_dataloader))
for k,v in batch.items():
  print(k,v.shape)
```

6. Configuración del modelo y del entrenamiento

```
En esta sección se cargará y se configurará el modelo SAM
"""

from transformers import SamModel
from torch.optim import Adam
import monai

model = SamModel.from_pretrained("facebook/sam-vit-base")

# Componentes a reentrenar(True) o congelar (False)
# Congelar todos los parámetros por defecto
for param in model.parameters():
    param.requires_grad = False
```

```
# Descongelar los parámetros a ajustar
for param in model.mask_decoder.parameters():
    param.requires_grad = True
#for param in model.prompt_encoder.parameters():
    #param.requires_grad = True
#for param in model.vision_encoder.parameters():
    #param.requires_grad = True

"""Hiperparámetros y configuración del entrenamiento"""

optimizer = Adam(model.parameters(), lr=1e-5, weight_decay=0)

seg_loss = monai.losses.DiceCELoss(sigmoid=True, squared_pred=True,
reduction='mean')

num_epochs = 30

device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
```

7. Bucle de entrenamiento y validación

```
import pandas as pd
from tqdm import tqdm
from statistics import mean
import torch
import torch.nn.functional as F
from torch.optim import Adam
import monai
from transformers import SamModel

# Inicializamos el DataFrame donde vamos a almacenar las métricas
metrics_columns = ['epoch', 'train_loss', 'train_iou',
'train_accuracy', 'train_precision', 'train_recall', 'train_f1',
                   'val_loss', 'val_iou', 'val_accuracy',
'val_precision', 'val_recall', 'val_f1']
metrics_df = pd.DataFrame(columns=metrics_columns)

# Definir las métricas
def iou(pred, target, threshold=0.5):
    pred = pred > threshold
    intersection = (pred * target).sum()
    union = pred.sum() + target.sum() - intersection
    return (intersection + 1e-6) / (union + 1e-6)

def accuracy(pred, target, threshold=0.5):
    pred = pred > threshold
    correct = (pred == target).sum()
    total = target.numel()
    return correct / total

def precision(pred, target, threshold=0.5):
    pred = pred > threshold
    tp = (pred * target).sum()
    fp = (pred * (1 - target)).sum()
    return tp / (tp + fp + 1e-6)

def recall(pred, target, threshold=0.5):
    pred = pred > threshold
```

```
    tp = (pred * target).sum()
    fn = (~pred * target).sum()
    return tp / (tp + fn + 1e-6)

def f1_score(pred, target, threshold=0.5):
    p = precision(pred, target, threshold)
    r = recall(pred, target, threshold)
    return 2 * (p * r) / (p + r + 1e-6)


# Ruta de almacenamiento en Google Drive
save_dir = '/content/drive/MyDrive/Colab
Notebooks/concreteCrackSegmentationDataset/'

import torch
torch.cuda.empty_cache()

# Variables para almacenar las mejores métricas
best_val_loss = float('inf')

# Lista para almacenar las métricas de entrenamiento
train_metrics = []

# Bucle de entrenamiento y validación
for epoch in range(num_epochs):
    # Fase de entrenamiento
    model.train()  # Establecer el modelo en modo de entrenamiento
    epoch_losses = []  # Lista para almacenar las pérdidas de cada
batch
    epoch_iou, epoch_accuracy, epoch_precision, epoch_recall, epoch_f1
= [], [], [], [], []  # Métricas de entrenamiento

    for batch in tqdm(train_dataloader):
        # Forward pass
        outputs = model(pixel_values=batch["pixel_values"].to(device),
                        input_boxes=batch["input_boxes"].to(device),
                        multimask_output=False)

        # Calcular la pérdida
        predicted_masks = outputs.pred_masks.squeeze(1)
        ground_truth_masks =
batch["ground_truth_mask"].float().to(device)

        predicted_masks_resized = F.interpolate(predicted_masks,
size=(1024, 1024), mode='bilinear', align_corners=False)

        loss = seg_loss(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))

        # Cálculo de las métricas
        train_iou = iou(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
        train_accuracy = accuracy(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
        train_precision = precision(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
        train_recall = recall(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
        train_f1 = f1_score(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
```

```
        # Guardar las métricas del batch
        epoch_iou.append(train_iou)
        epoch_accuracy.append(train_accuracy)
        epoch_precision.append(train_precision)
        epoch_recall.append(train_recall)
        epoch_f1.append(train_f1)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()

        optimizer.step()
        epoch_losses.append(loss.item())

    # Promedio de métricas de entrenamiento
    mean_train_loss = torch.mean(torch.tensor(epoch_losses)).item()  #
    mean_train_iou = torch.mean(torch.tensor(epoch_iou)).item()
    mean_train_accuracy =
torch.mean(torch.tensor(epoch_accuracy)).item()
    mean_train_precision =
torch.mean(torch.tensor(epoch_precision)).item()
    mean_train_recall = torch.mean(torch.tensor(epoch_recall)).item()
    mean_train_f1 = torch.mean(torch.tensor(epoch_f1)).item()

    # Imprimir la pérdida promedio de entrenamiento
    print(f'Epoch {epoch+1}/{num_epochs} - Training Loss:
{mean_train_loss:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Training IoU:
{mean_train_iou:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Training Accuracy:
{mean_train_accuracy:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Training Precision:
{mean_train_precision:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Training Recall:
{mean_train_recall:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Training F1:
{mean_train_f1:.4f}')

    # Guardar las métricas de entrenamiento
    train_metrics.append({
        'epoch':epoch+1,
        'train_loss':mean_train_loss,
        'train_iou':mean_train_iou,
        'train_accuracy':mean_train_accuracy,
        'train_precision':mean_train_precision,
        'train_recall':mean_train_recall,
        'train_f1':mean_train_f1
    })

    # Fase de validación
    model.eval()  # Establecer el modelo en modo de evaluación
    val_losses = []  # Lista para almacenar las pérdidas de validación
    val_iou, val_accuracy, val_precision, val_recall, val_f1 = [], [],
[], [], []  # Métricas de validación

    with torch.no_grad():  # Deshabilitar el cálculo de gradientes
durante la validación
        for batch in tqdm(val_dataloader):
            # Forward pass
            outputs =
model(pixel_values=batch["pixel_values"].to(device),
```

```
input_boxes=batch["input_boxes"].to(device),
                            multimask_output=False)
            # Calcular la pérdida
            predicted_masks = outputs.pred_masks.squeeze(1)
            ground_truth_masks =
batch["ground_truth_mask"].float().to(device)
            predicted_masks_resized = F.interpolate(predicted_masks,
size=(1024, 1024), mode='bilinear', align_corners=False)
            val_loss = seg_loss(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))

            # Cálculo de las métricas
            val_iou_batch = iou(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
            val_accuracy_batch = accuracy(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
            val_precision_batch = precision(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
            val_recall_batch = recall(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
            val_f1_batch = f1_score(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))

            # Guardar las métricas del batch
            val_iou.append(val_iou_batch)
            val_accuracy.append(val_accuracy_batch)
            val_precision.append(val_precision_batch)
            val_recall.append(val_recall_batch)
            val_f1.append(val_f1_batch)

            val_losses.append(val_loss.item())

    # Promedio de métricas de validación
    avg_val_loss = torch.mean(torch.tensor(val_losses)).item()
    mean_val_iou = torch.mean(torch.tensor(val_iou)).item()
    mean_val_accuracy = torch.mean(torch.tensor(val_accuracy)).item()
    mean_val_precision =
torch.mean(torch.tensor(val_precision)).item()
    mean_val_recall = torch.mean(torch.tensor(val_recall)).item()
    mean_val_f1 = torch.mean(torch.tensor(val_f1)).item()

    # Imprimir las métricas de validación
    print(f'Epoch {epoch+1}/{num_epochs} - Validation Loss:
{avg_val_loss:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Validation IoU:
{mean_val_iou:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Validation Accuracy:
{mean_val_accuracy:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Validation Precision:
{mean_val_precision:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Validation Recall:
{mean_val_recall:.4f}')
    print(f'Epoch {epoch+1}/{num_epochs} - Validation F1:
{mean_val_f1:.4f}')

    # Guardar el modelo si la validación ha mejorado
    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        torch.save(model.state_dict(),
f'{save_dir}best_model_epoch_{epoch+1}.pth')
```

```
        print(f"Model saved at epoch {epoch+1} with validation loss:
{avg_val_loss:.4f}")

    # Guardar las métricas de validación
    train_metrics[-1].update({
        'val_loss': avg_val_loss,
        'val_iou': mean_val_iou,
        'val_accuracy': mean_val_accuracy,
        'val_precision': mean_val_precision,
        'val_recall': mean_val_recall,
        'val_f1': mean_val_f1
    })

    # Guardar las métricas de entrenamiento y validación
    metrics_df = pd.DataFrame(train_metrics)
    metrics_df.to_csv(f'{save_dir}training_metrics.csv', index=False)

import pandas as pd

# Cargar el archivo CSV
metrics_df = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/concreteCrackSegmentationDataset/training_metrics.csv')

# Crear gráfica para vicualizar algunas métricas
import matplotlib.pyplot as plt

plt.plot(metrics_df['epoch'], metrics_df['train_loss'],
label='Training Loss')
plt.plot(metrics_df['epoch'], metrics_df['val_loss'],
label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

8. Etapa de test

```
# Variables para almacenar las métricas de test
test_metrics = []  # Lista para almacenar las métricas de test

model.eval()  # Establecer el modelo en modo de evaluación (sin
gradientes)
test_losses = []
test_iou, test_accuracy, test_precision, test_recall, test_f1 = [],
[], [], [], []  # Métricas de test

# Configuración para no calcular gradientes durante la evaluación
with torch.no_grad():  # Deshabilitar el cálculo de gradientes
    for batch in tqdm(test_dataloader, desc="Evaluando Test..."):
        # Forward pass
        outputs = model(pixel_values=batch["pixel_values"].to(device),
                        input_boxes=batch["input_boxes"].to(device),
                        multimask_output=False)

        predicted_masks = outputs.pred_masks.squeeze(1)  # Máscaras
predichas
        ground_truth_masks =
batch["ground_truth_mask"].float().to(device)  # Máscaras verdaderas
        predicted_masks_resized = F.interpolate(predicted_masks,
size=(1024, 1024), mode='bilinear', align_corners=False)
```

```
        # Pérdida
        test_loss = seg_loss(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))

        # Calcular las métricas de test (IoU, precisión, etc.)
        test_iou_batch = iou(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
        test_accuracy_batch = accuracy(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
        test_precision_batch = precision(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
        test_recall_batch = recall(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))
        test_f1_batch = f1_score(predicted_masks_resized,
ground_truth_masks.unsqueeze(1))

        # Guardar las métricas del batch de test
        test_iou.append(test_iou_batch)
        test_accuracy.append(test_accuracy_batch)
        test_precision.append(test_precision_batch)
        test_recall.append(test_recall_batch)
        test_f1.append(test_f1_batch)

        # Guardar la pérdida de test
        test_losses.append(test_loss.item())

# Promediar las métricas de test
avg_test_loss = torch.mean(torch.tensor(test_losses)).item()
mean_test_iou = torch.mean(torch.tensor(test_iou)).item()
mean_test_accuracy = torch.mean(torch.tensor(test_accuracy)).item()
mean_test_precision = torch.mean(torch.tensor(test_precision)).item()
mean_test_recall = torch.mean(torch.tensor(test_recall)).item()
mean_test_f1 = torch.mean(torch.tensor(test_f1)).item()

# Imprimir las métricas de test
print(f"Test Loss: {avg_test_loss:.4f}")
print(f"Test IoU: {mean_test_iou:.4f}")
print(f"Test Accuracy: {mean_test_accuracy:.4f}")
print(f"Test Precision: {mean_test_precision:.4f}")
print(f"Test Recall: {mean_test_recall:.4f}")
print(f"Test F1: {mean_test_f1:.4f}")

# Guardar las métricas de test en la lista
test_metrics.append({
    'test_loss': avg_test_loss,
    'test_iou': mean_test_iou,
    'test_accuracy': mean_test_accuracy,
    'test_precision': mean_test_precision,
    'test_recall': mean_test_recall,
    'test_f1': mean_test_f1
})

# Guardar las métricas en un archivo CSV para procesar los datos
posteriormente
import pandas as pd
metrics_df = pd.DataFrame(test_metrics)
metrics_df.to_csv(f'{save_dir}test_metrics.csv', index=False)
```

Anexo F. Código de inferencia. Obtención de máscaras de segmentación

```
import torch
from transformers import SamProcessor, SamModel
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import torch.nn.functional as F
import os

# Definir de si se ejecuta en GPU o CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Cargar el procesador para el modelo base
processor = SamProcessor.from_pretrained("facebook/sam-vit-base")

# Modelo base
model_base = SamModel.from_pretrained("facebook/sam-vit-
base").to(device)

# Carga del modelo ajustado
model_finetuned = SamModel.from_pretrained("facebook/sam-vit-base")
model_finetuned.load_state_dict(torch.load("/content/drive/MyDrive/Col
ab Notebooks/concreteCrackSegmentationDataset/FULL
FT/best_model_epoch_FULLFT.pth"))
model_finetuned.to(device)

# Cargar la imagen
image_path = "/content/drive/MyDrive/Colab Notebooks/imagen.jpg"
image_resized = Image.open(image_path).convert("RGB").resize((1024,
1024))

# Preparar la entrada para el modelo
inputs = processor(image_resized, return_tensors="pt").to(device)

# Modo evaluación para ambos modelos
model_base.eval()
model_finetuned.eval()

# Realizar la inferencia
with torch.no_grad():
    outputs_base = model_base(**inputs, multimask_output=False)
    outputs_finetune = model_finetuned(**inputs,
multimask_output=False)

# Aplicar la función de activación y obtener las máscaras predichas
base_seg_prob = torch.sigmoid(outputs_base.pred_masks.squeeze(1))
finetune_seg_prob =
torch.sigmoid(outputs_finetune.pred_masks.squeeze(1))

# Redimensionar las segmentaciones al tamaño de la imagen
base_seg_resized = F.interpolate(base_seg_prob, size=(1024, 1024),
mode='bilinear')
finetune_seg_resized = F.interpolate(finetune_seg_prob, size=(1024,
1024), mode='bilinear')

# Binarizar las máscaras
```

```
x = 0.5
base_seg_bin = (base_seg_resized >
x).squeeze(1).cpu().numpy().astype(np.uint8) * 255
finetune_seg_bin = (finetune_seg_resized >
x).squeeze(1).cpu().numpy().astype(np.uint8) * 255

# Conversiones a PIL para guardar
original_image = image_resized
base_mask_image = Image.fromarray(base_seg_bin[0])
finetune_mask_image = Image.fromarray(finetune_seg_bin[0])

# Definir el directorio
output_dir = "/content/drive/MyDrive/Colab Notebooks/output_masks"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Guardado de las imágenes
original_image.save(os.path.join(output_dir, "original_image.jpg"))
base_mask_image.save(os.path.join(output_dir, "base_model_mask.jpg"))
finetune_mask_image.save(os.path.join(output_dir,
"finetuned_model_mask.jpg"))

# Visualizado
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Imagen original
axes[0].imshow(original_image)
axes[0].set_title("Original Image")
axes[0].axis("off")
axes[0].set_aspect('auto')

# Máscara con el modelo base
axes[1].imshow(base_seg_bin[0], cmap="gray", vmin=0, vmax=255)
axes[1].set_title("Predicted Mask (Base)")
axes[1].axis("off")
axes[1].set_aspect('auto')

# Máscara con el modelo ajustado
axes[2].imshow(finetune_seg_bin[0], cmap="gray", vmin=0, vmax=255)
axes[2].set_title("Predicted Mask (Fine-Tuned)")
axes[2].axis("off")
axes[2].set_aspect('auto')

plt.tight_layout()
plt.show()
```

Anexo G. Segmentaciones de imágenes tomadas de la vía pública

Tabla 24. Segmentaciones adicionales de imágenes obtenidas en la vía pública

MODELO BASE	IMAGEN ORIGINAL	MODELO AJUSTADO