

UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS



TESIS

**Integración de métodos avanzados de visión
computacional para el desarrollo de un sistema de
segmentación de lesiones en pie diabético**

PARA OBTENER EL TÍTULO PROFESIONAL DE:
INGENIERO FÍSICO

ELABORADO POR:
LUCAS NICOLAS TAIPE RAMOS
 0009-0009-4945-2386

ASESOR:
DR. GERMÁN YURI COMINA BELLIDO
 0000-0003-2114-0486

LIMA - PERÚ

2025

Citar/How to cite	Taipe Ramos [1]
Referencia/Reference	[1] L. Taibe Ramos, <i>“Integración de métodos avanzados de visión computacional para el desarrollo de un sistema de segmentación de lesiones en pie diabético”</i> [Tesis de pregrado]. Lima (Perú): Universidad Nacional de Ingeniería, 2025.
Estilo/Style: IEEE (2020)	
Citar/How to cite	(Taibe, 2025)
Referencia/Reference	Taibe, L. (2025). <i>Integración de métodos avanzados de visión computacional para el desarrollo de un sistema de segmentación de lesiones en pie diabético</i> . [Tesis de pregrado, Universidad Nacional de Ingeniería]. Repositorio Institucional Cybertesis UNI.
Estilo/Style: APA (7ma ed.)	

Dedicatoria

A mis padres, Lucas y Dora, y hermanas, Katherin, Rosemary y Angie, cuyo apoyo incondicional fue vital en mi vida.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a mi asesor, Germán Comina, por su invaluable orientación y apoyo constante y a Karelia Peña, de quien aprendí enormemente durante mi pasantía en University of Delaware.

Resumen

Este estudio presenta un sistema de visión por computadora que integra modelos clásicos y fundacionales basados en Transformers para segmentar lesiones de pie diabético, complicación frecuente y severa de la diabetes. Modelos fundacionales como Segment Anything Model, entrenados con millones de imágenes, permiten obtener segmentaciones precisas mediante prompts en lugar de costosos procesos de fine tuning. El flujo de trabajo consiste en lo siguiente. Primero YOLOv5 localiza la lesión; segundo EDSR aumenta la resolución de la región de interés; tercero SLIC genera superpíxeles y VGG16 extrae descriptores; cuarto una regresión logística clasifica cada superpíxel como lesión o piel sana; quinto los centroides positivos, las cajas de YOLOv5 y la imagen original actúan como prompt para SAM, que produce la máscara final aprovechando su atención global. En pruebas con un conjunto diverso de imágenes, el sistema alcanzó un Índice de Jaccard de 0.941 y un coeficiente de Dice de 0.969, superando el ajuste directo de SAM y enfoques puramente convolucionales, a la vez que redujo tiempo y coste computacional. La solución brinda una herramienta avanzada para monitorizar úlceras, disminuye el riesgo de infecciones y optimiza el seguimiento de la cicatrización en pacientes con pie diabético, al aplicar los avances más recientes en visión por computadora a esta área clínica.

Palabras clave: pie diabético, segmentación, modelos fundacionales, visión por computadora.

Abstract

This study introduces a computer-vision system that melds classical techniques with foundation models based on Transformers to segment diabetic-foot ulcers, a frequent and serious complication of diabetes. Foundation models such as the Segment Anything Model, trained on millions of images, can generate accurate segmentations via prompts instead of costly fine-tuning.

The workflow proceeds as follows. YOLOv5 first locates the ulcer; EDSR then enhances the region's resolution; SLIC forms superpixels and VGG16 extracts their descriptors; a logistic-regression classifier labels each superpixel as ulcer or healthy skin; finally, the positive centroids, YOLOv5 bounding boxes, and the original image serve as a prompt for SAM, which outputs the final mask using its global-attention mechanism.

Tested on a diverse image set, the system achieved a Jaccard Index of 0.941 and a Dice coefficient of 0.969, outperforming direct SAM fine-tuning and purely convolutional approaches while cutting training time and computational cost.

The resulting tool advances ulcer monitoring, lowers infection risk, and streamlines healing assessment in diabetic-foot patients by bringing the latest computer-vision breakthroughs to this clinical domain.

Keywords: diabetic-foot, semantic segmentation, foundation models, computer vision.

Tabla de Contenido

Resumen	V
Abstract	VI
I Introducción	1
A Planteamiento del problema	2
B Justificación	2
C Objetivos	4
1 Objetivo general	4
2 Objetivos específicos	4
II Marco teórico	5
A Imágenes digitales	5
1 Espacios de colores	5
B Superpíxeles	8
1 Método simple linear iterative clustering (SLIC)	9
C Machine learning	10
1 Modelo de regresion logistica	11
D Fundamentos de redes neuronales	12
1 Redes neuronales artificiales	12
2 Redes neuronales convolucionales	13
3 Tipos de entrenamiento en redes neuronales	15
4 Aprendizaje supervisado en redes neuronales	18
5 Aprendizaje por transferencia	32
6 Fine tuning	33
E Deep learning	34
1 Red neuronal VGG16	34
	VII

F	Detectores de objetos	36
1	Detector de objeto YOLOv5	37
G	Segmentación de imágenes	40
1	Segmentación basada en regiones	40
2	Superpíxeles en segmentación de imágenes	41
H	Limpieza y preparación de datos (data cleaning)	41
I	Modelo de superresolución EDSR	42
1	Modificaciones en la arquitectura residual	42
2	Bloques residuales	43
3	Upsampling y expansión de la imagen	44
4	Normalización y preservación de los colores	44
5	Función de pérdida	45
6	Desafíos computacionales	45
J	Transformers en visión computacional	45
1	Modelo SAM (segment anything model)	46
2	Proceso de entrenamiento de SAM	48
3	Prompting del modelo SAM	49
4	Limitaciones	50
K	Métricas de desempeño	50
1	Especificidad	51
2	Precisión	51
3	Exactitud	52
4	Sensibilidad o Recall	52
5	Medida-F	52
6	Curvas ROC	52
7	Intersection over union (IoU)	53
8	Coeficiente de Dice	53

9	Mean average precision (mAP)	54
L	Gasto computacional en modelos de aprendizaje automático	54
III	Desarrollo del sistema de segmentación	56
A	Recursos de desarrollo	57
B	Dataset	58
C	Entrenamiento del modelo de detección YOLOv5	60
1	Preprocesamiento	60
2	Entrenamiento	62
D	Sistema de segmentación de lesiones	65
1	Superresolución con el método EDSR	67
2	Generación de superpíxeles:	68
3	Data cleaning:	69
4	Etiquetado de superpíxeles	73
5	Extracción de características	75
6	Clasificación binaria	77
7	Prompting SAM	78
IV	Evaluación del sistema de segmentación	82
A	Evaluación del sistema con referencias bibliográficas	82
B	Evaluación con enfoques similares	83
C	Evaluación de la eficiencia	86
1	Gasto computacional	89
V	Discusión de Resultados y Conclusiones	91
A	Discusión de resultados	91
1	Entrenamiento del modelo YOLOv5	91
2	Data cleaning	91

3	Superresolución con el método EDSR	92
4	Etiquetado de superpíxeles	92
5	Extracción de características	93
6	Prompting SAM	93
7	Análisis del sistema de segmentación	94
VI	Conclusiones	99
VII	Referencias bibliográficas	102
	Anexos	111

Lista de Tablas

Tabla I	Funciones de activación [29].	24
Tabla II	Matriz de confusión.	51
Tabla III	Resumen de bases de datos de lesiones, incluyendo tipos, tamaños y si están etiquetadas [6].	59
Tabla IV	Distribución de imágenes para entrenamiento de YOLOv5.	62
Tabla V	Métricas de desempeño del algoritmo YOLOv5.	65
Tabla VI	Métricas de desempeño del modelo de regresión logística para la clasificación de superpíxeles.	79
Tabla VII	Resumen de modelos de aprendizaje profundo para segmentación de úlceras del pie diabético, tamaño de los datasets, métricas de desempeño, acceso y fuentes de datos.	83
Tabla VIII	Comparativa de métricas de desempeño. La evaluación involucró un conjunto de prueba de 83 imágenes.	83
Tabla IX	Comparativa de métricas de desempeño en la generación de máscaras de segmentación aplicando cuadros delimitadores, puntos, SAM con fine-tuning y MedSAM. La evaluación involucró un conjunto de prueba de 83 imágenes.	84
Tabla X	Desglose detallado de los tiempos de procesamiento en las diferentes fases del sistema de segmentación.	89

Lista de Figuras

Figura I	Matriz de intensidad de escala de grises.	5
Figura II	Composición de canales de color para crear una imagen RGB.	6
Figura III	Modelo de color HSV [17].	7
Figura IV	Espacio de color CIE L*a*b* [18].	8
Figura V	Implementación del método SLIC en una imagen de 512x512 píxeles, generando 190 segmentos.	10
Figura VI	Diagrama de un perceptrón simple [25].	13
Figura VII	Operación convolución con un filtro de 3x3 [29].	14
Figura VIII	Operación de agrupamiento promedio y máximo en una CNN.	15
Figura IX	Flujo del modelo de aprendizaje supervisado [31].	16
Figura X	Flujo del modelo de aprendizaje no supervisado [31].	17
Figura XI	Flujo del modelo de aprendizaje por refuerzo [31].	18
Figura XII	Procesamiento secuencial de entradas en una red neuronal [34].	19
Figura XIII	Retropropagación del error para ajuste de pesos en la red [34].	20
Figura XIV	Optimización de pesos en una red neuronal multicapa [34].	21
Figura XV	Flujo de aprendizaje en redes neuronales multicapa [35].	22
Figura XVI	Visualización de características en la red neuronal para clasificación de dígitos [35].	23
Figura XVII	Gráfica comparativo de la efectividad de métodos de optimización [44].	29
Figura XVIII	Diferencias al variar el learning rate [45].	30
Figura XIX	Comparativa entre el entrenamiento desde cero y el entrenamiento mediante transfer learning [50].	33
Figura XX	Arquitectura del modelo de red neuronal VGG 16 [54].	36

Figura XXI	Diferencia entre clasificación y detección de objetos [56].	37
Figura XXII	Grilla de clasificación para detección de objetos del modelo YOLOv5 [58].	38
Figura XXIII	Arquitectura de YOLOv5 [59].	39
Figura XXIV	Arquitectura de la red de SR a escala única propuesta (EDSR) [71]. .	42
Figura XXV	Descripción del modelo basado en transformer para clasificación de imágenes [9].	46
Figura XXVI	Arquitectura general del modelo SAM [10].	48
Figura XXVII	Metodología de entrenamiento y etiquetado automático de SAM [10].	49
Figura XXVIII	Aplicación del modelo SAM en la segmentación de imágenes mediante prompts [10].	50
Figura XXIX	Flujo de trabajo para la detección y segmentación de heridas combinando YOLOv5, SLIC, Regresión Logística y el Modelo SAM.	57
Figura XXX	(a) Ejemplo de imagen de entrenamiento con resolución de 512x512 píxeles. (b) Máscara de segmentación correspondiente, mostrando las áreas de interés etiquetadas.	60
Figura XXXI	Creación de la etiqueta en formato YOLOv5.	62
Figura XXXII	Evolución de la precisión, <i>recall</i> y pérdidas de las cajas delimitadoras durante el entrenamiento.	64
Figura XXXIII	Superpíxeles generados utilizando el método SLIC para la segmentación de imágenes.	66
Figura XXXIV	(a) Imagen inicial y extracción de características de baja calidad, (b) Mejora con EDSR y extracción avanzada.	68
Figura XXXV	Histograma del canal verde de las imágenes analizadas. La variabilidad observada justifica la elección del rango 8-200 para el cálculo del exponente en el filtro especular.	70

Figura XXXVI	Ejemplo del proceso de eliminación de reflexión especular utilizando un filtro mejorado y completación de imagen mediante el método de inpainting de OpenCV.	72
Figura XXXVII	Comparación de la aplicación del filtro de reflexión especular en imágenes con diferentes medias del canal verde.	73
Figura XXXVIII	(a) Recorte de la lesión aplicando el método SLIC, (b) Máscara asociada al recorte, (c) Imagen con superpíxeles etiquetados de amarillo (no pertenecientes a la lesión) y rojo (pertenecientes a la lesión).	75
Figura XXXIX	(a) Aplicación de superresolución EDSR, (b) método SLIC para la generación de superpíxeles, (c) extracción de características con mapas de características VGG16 y canales RGB, (d) obtención de un vector de características de 264 elementos para cada superpíxel.	76
Figura XL	Diagrama del cálculo de factores de escala y ajuste de coordenadas en recortes con superresolución.	80
Figura XLI	Segmentación del modelo SAM utilizando cuadros delimitadores y puntos de lesión/no lesión.	81
Figura XLII	Gráficos comparativos del error por modelo en la data de test utilizando el complemento del índice IoU frente al área de la etiqueta en píxeles.	85
Figura XLIII	Gráficos comparativos del desempeño de los enfoques propuestos utilizando el complemento del IoU por imagen, ordenados de forma ascendente según el área, destacando la img_83 como la de mayor área.	86
Figura XLIV	Capturas de pantalla enfocadas en la segmentación de lesiones para diversos tipos de casos.	88
Figura XLV	(a) Clasificación de superpíxeles en lesión (rojo) y no lesión (amarillo), (b) identificación de verdaderos positivos (verde), falsos negativos (azul), falsos positivos (rojo), verdaderos negativos (gris) con un índice de 0.87 en la unión sobre intersección (IoU).	93

Figura XLVI	Comparativa de segmentación con diferentes enfoques de SAM, donde las imágenes en colores representan verdaderos positivos en verde, falsos negativos en azul, falsos positivos en rojo, verdaderos negativos en gris, junto con métricas IoU y DSC.	98
-------------	--	----

I. Introducción

En esta sección haremos una introducción global al tema de la presente investigación. Asimismo, haremos el planteamiento del problema, la justificación del trabajo y los objetivos.

La diabetes mellitus (DM), una enfermedad crónica de gran impacto global, ha sido identificada como la quinta causa principal de muerte a nivel mundial. Se estima que para 2025, alrededor de 380 millones de personas sufrirán de DM, lo que representa un aumento significativo desde los 246 millones reportados en 2006 [1].

Una de las complicaciones más serias de la DM es el desarrollo de úlceras en el pie, que afectan aproximadamente al 15% de los pacientes a lo largo de sus vidas y pueden llevar a amputaciones no traumáticas, siendo estas últimas 15 veces más frecuentes en diabéticos que en la población general [2].

Los métodos actuales para el monitoreo de estas lesiones se base principalmente en la inspección visual o medidas manuales, los cuales presentan limitaciones en términos de precisión y pueden aumentar el riesgo de infección [3]. En este contexto, surge la necesidad de herramientas más avanzadas y confiables.

Esta investigación se enfoca en el desarrollo de un sistema de segmentación de lesiones no invasivo, empleando métodos novedosos surgidos en el campo de la visión computacional. Nuestro enfoque combina técnicas tradicionales de machine learning con un avanzado modelo de segmentación basado en la tecnología de Transformers. Esta estrategia dual no solo mejora notablemente la precisión en la identificación de lesiones, sino que también optimiza el seguimiento de las áreas afectadas. Mediante este enfoque, potenciamos la capacidad de monitorear y reducir el riesgo de complicaciones graves en pacientes con diabetes mellitus.

A. Planteamiento del problema

El diagnóstico y seguimiento de las lesiones de pie diabético representan un desafío considerable en la medicina moderna. Los métodos de diagnóstico tradicional, a pesar de su amplio uso, se enfrentan a complejidades y son propensos a errores, impactando negativamente tanto en la salud física como en la mental de los pacientes [4]. Esta situación resalta la urgente necesidad de innovar y adoptar técnicas de monitoreo más avanzadas y precisas, especialmente para evaluar cambios en el área, la profundidad y los tipos de tejidos de las heridas, siendo la reducción de la superficie de la úlcera un indicador crítico en el proceso de cicatrización [5].

En el campo emergente de la visión computacional, las técnicas de Machine Learning y Deep Learning han sido ampliamente exploradas y propuestas para afrontar desafíos como la detección y segmentación de lesiones, como se refleja en la investigación de [6]. A pesar de los avances significativos, estos métodos aún enfrentan desafíos clave, especialmente en la segmentación de lesiones en contextos donde el fondo no está estandarizado o las imágenes no se toman en ambientes controlados. La variabilidad en estas condiciones puede complicar la segmentación precisa. En este escenario, la emergente tecnología de los transformers, inicialmente revolucionaria en el campo del procesamiento del lenguaje natural, se está perfilando como una solución prometedora [7]. Estas avanzadas tecnologías tienen el potencial de comprender y procesar las imágenes de manera más completa y contextual, superando algunas de las limitaciones inherentes a los enfoques tradicionales.

B. Justificación

La necesidad de desarrollar un sistema avanzado para la segmentación de lesiones en pie diabético se fundamenta en superar las deficiencias de los métodos actuales. Estos métodos frecuentemente recurren a enfoques manuales como la colocación de stickers o la medición con reglas. Según [3], tales prácticas no solo son inexactas, sino que también

incrementan el riesgo de complicaciones serias, incluyendo infecciones. Paralelamente, ha habido esfuerzos significativos para aplicar técnicas de aprendizaje automático en esta área. Un ejemplo destacado es el de [8], quienes desarrollaron DFUNet, una arquitectura innovadora de red neuronal convolucional para la clasificación de úlceras de pie diabético. En una línea similar, [3] desarrolló un método de segmentación de úlceras de pie diabético basado en Support Vector Machine y el método GrabCut. A pesar de estos avances, tales métodos todavía enfrentan desafíos significativos en cuanto a generalización y adaptabilidad en distintos entornos clínicos, lo que subraya la importancia de explorar soluciones más avanzadas y flexibles.

El campo de la visión por computadora ha experimentado un avance significativo con la introducción de los transformers, una innovación que ha permitido el entrenamiento de modelos con enormes volúmenes de datos [9]. Un ejemplo notable es el modelo SAM (Segment Anything Model), que ha sido entrenado con más de 1.1 mil millones de imágenes [10]. Este modelo ofrece un potencial considerable en el ámbito de las imágenes médicas, tal como se evidencia en el trabajo de [11], en ese trabajo se realizó un fine-tuning específico del modelo SAM con una extensa colección de imágenes médicas, resultando en la variante MedSAM, este modelo fue entrenado con más de un millón de imágenes abarcando diversas modalidades y patologías. Cabe señalar que, dentro de esta amplia gama de imágenes, no se incluyeron imágenes relacionadas con lesiones de pie diabético. A pesar de la amplia capacidad de SAM para procesar y comprender diferentes tipos de imágenes y contextos, se podría considerar un enfoque alternativo que consista en mejorar la calidad de los datos de entrada que recibe el modelo. Este enfoque podría potenciar aún más la habilidad de SAM para reconocer y segmentar de manera efectiva las imágenes médicas en una variedad de escenarios clínicos.

El enfoque propuesto se centra en maximizar las capacidades del modelo SAM, enfatizando en la mejora de la calidad de sus entradas. A través de una cuidadosa selección y combinación de puntos específicos con cuadros delimitadores. Se considera que esta

metodología híbrida permitirá una segmentación más precisa y detallada de lesiones cutáneas en pie diabético, un área que MedSAM no había abordado específicamente. Esta aproximación no solo amplía la funcionalidad de SAM, sino que también satisface una necesidad clínica específica, destacando la versatilidad y el potencial de los modelos basados en transformers en el ámbito de las aplicaciones médicas especializadas.

C. Objetivos

1. *Objetivo general*

Desarrollar un sistema automatizado para la segmentación precisa de lesiones cutáneas en el pie diabético, combinando el modelo SAM con técnicas de Deep Learning y Machine Learning.

2. *Objetivos específicos*

- Realizar una búsqueda exhaustiva de bases de datos con código de ética adecuado para el uso de imágenes de lesiones de pie diabético, asegurando el cumplimiento de los estándares éticos y de calidad en la investigación.
- Entrenar el modelo YOLOv5 para la detección eficaz de lesiones en imágenes de pie diabético, logrando precisión y eficiencia en la identificación de áreas lesionadas.
- Desarrollar un método de extracción de características para imágenes de pie diabético, utilizando técnicas de procesamiento de imágenes y aprendizaje automático.
- Implementar técnicas Machine Learning para identificar y extraer puntos representativos de lesiones y áreas no lesionadas, utilizando estos datos críticos como inputs para la segmentación en el modelo SAM.
- Integrar y adaptar el modelo SAM (Segment Anything Model) para la segmentación final de las lesiones, utilizando una combinación de cuadros delimitadores y puntos específicos para generar máscaras de segmentación.

II. Marco teórico

A. Imágenes digitales

Una imagen digital es una representación bidimensional de información visual, codificada como una matriz de datos donde cada elemento, conocido como píxel, representa un punto específico de la imagen. Los píxeles contienen información sobre la luz e intensidad, contribuyendo a formar la imagen completa. Las imágenes digitales pueden variar en su composición, abarcando desde escalas de grises, donde cada píxel refleja únicamente la intensidad de luz, hasta representaciones a color que combinan distintos valores para crear un espectro visual más amplio [12]. En la Figura I, se ilustra un ejemplo de una imagen en escala de grises. Es importante destacar que el concepto de imagen digital no se limita a un único esquema de colores, sino que puede adaptarse a diferentes modelos de color, cada uno con sus características y aplicaciones específicas.

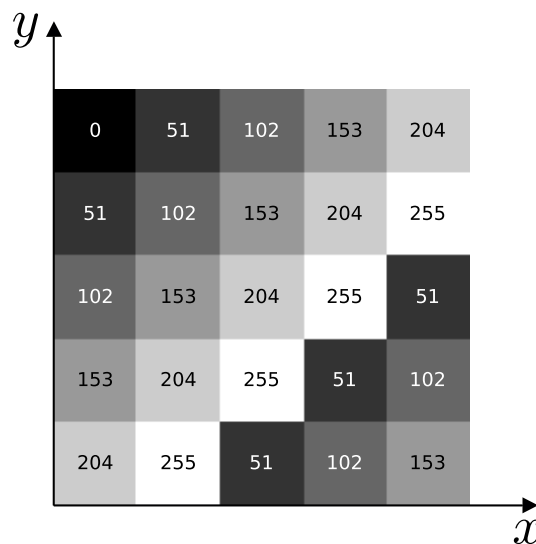


Figura I: Matriz de intensidad de escala de grises.

1. Espacios de colores

La representación de colores en imágenes digitales se fundamenta en el uso de espacios de color, sistemas matemáticos que definen la combinación de canales de color

para producir el espectro visual que percibimos. La imagen en escala de grises emplea un único canal para las intensidades de luz, variando del negro al blanco, lo que simplifica el procesamiento de la imagen al trabajar solo con la luminancia [13].

Espacio de color RGB : En el modelo RGB, los colores se forman mediante una combinación lineal de rojo (R), verde (G) y azul (B). Matemáticamente, cada color C se representa como $C = aR \cdot \hat{r} + bG \cdot \hat{g} + cB \cdot \hat{b}$, donde aR , bG , y cB son los coeficientes de escalamiento correspondientes a las intensidades de los componentes rojo, verde y azul, respectivamente, y $\hat{r}$, $\hat{g}$, $\hat{b}$ son los vectores unitarios en las direcciones de los ejes R, G y B en el espacio cúbico tridimensional. Cada color se representa por un trío de valores (R, G, B) (como se muestra en la Figura II), con cada componente variando generalmente entre 0 y 255, que corresponde a los coeficientes aR , bG , y cB . Este modelo se puede visualizar como un punto $P = (aR, bG, cB)$ en un espacio cúbico tridimensional, donde cada eje representa uno de los colores primarios. La distancia euclidiana en este espacio, dada por $d = \sqrt{(aR_1 - aR_2)^2 + (bG_1 - bG_2)^2 + (cB_1 - cB_2)^2}$, puede ser utilizada para cuantificar la diferencia percibida entre dos colores C_1 y C_2 [14, 15].



Figura II: Composición de canales de color para crear una imagen RGB.

Espacio de Color HSV : El espacio de color HSV clasifica los colores basándose en tonalidad (H), saturación (S) y valor (V). En una representación RGB, donde un color se indica por la longitud de tres barras correspondientes a R, G y B, el valor V en HSV corresponde a la altura de la barra más alta, representando la intensidad o brillo del color. La saturación S se puede entender como la cantidad de gris en el color, definida por la altura de la barra más baja (X) en RGB. Esta altura de gris (X, X, X) representa el nivel de

desaturación del color. Al sustraer este nivel de gris del color original en RGB, obtenemos la información de la tonalidad (H) como una mezcla proporcional de dos colores primarios. De esta forma, un color puede considerarse una mezcla de hasta tres colores primarios, una tonalidad de hasta dos, y un color primario de uno solo. La representación de HSV se muestra en la Figura III y es útil para manipular colores en términos perceptuales, separando el color de su luminosidad [16].

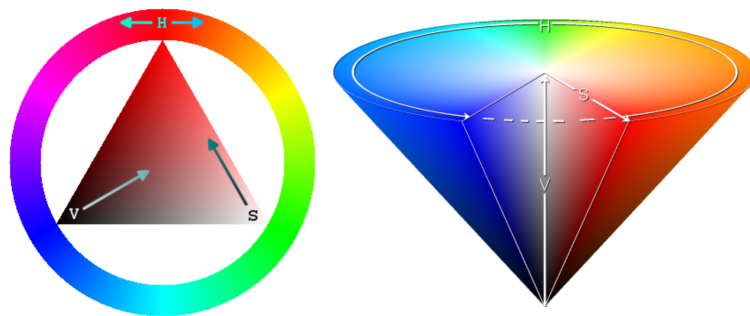


Figura III: Modelo de color HSV [17].

Espacio de color CIELAB : El modelo CIELAB se define en un espacio tridimensional con componentes L^* , a^* , y b^* . El componente L^* mide la luminosidad, oscilando de 0 (negro absoluto) a 100 (blanco puro). Los ejes a^* y b^* representan los colores opuestos en dos dimensiones perpendiculares: a^* varía entre verde y rojo, y b^* entre azul y amarillo. Estos ejes tienen un rango teóricamente ilimitado, pero en la práctica suelen restringirse a valores que reflejen de manera efectiva la percepción humana del color. La relación entre L^* , a^* , y b^* es no lineal, la Figura IV muestra una representación de este espacio de color, lo que permite una mejor correspondencia con la percepción humana del color en comparación con modelos lineales como RGB.

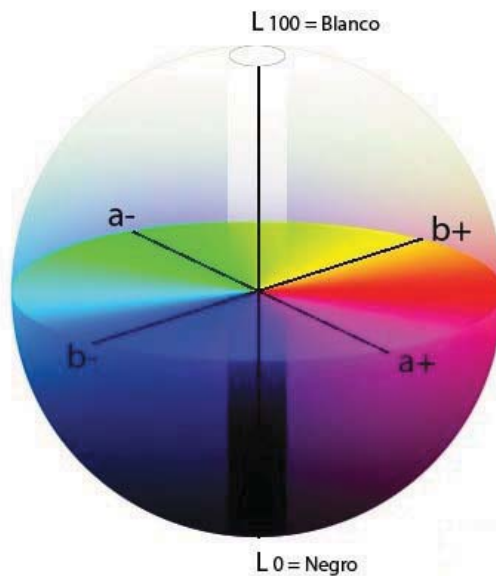


Figura IV: Espacio de color CIE $L^*a^*b^*$ [18].

B. Superpíxeles

En el ámbito del procesamiento de imágenes y la visión por computadora, el concepto de superpíxeles se ha establecido como una herramienta clave. Los superpíxeles se forman mediante la agrupación de píxeles adyacentes que comparten atributos como color, intensidad o textura, sirviendo como entidades visuales fundamentales. Esta metodología simplifica la representación de la imagen, manteniendo al mismo tiempo sus características y estructura cruciales. Esto resulta en una reducción significativa de la complejidad computacional y una mejora en la eficiencia de diversas aplicaciones, como la segmentación de imágenes y el reconocimiento de patrones. Diversos estudios han explorado diferentes modelos y enfoques para la creación de superpíxeles, demostrando la versatilidad y la importancia de esta técnica en el campo de la inteligencia artificial visual. Entre los trabajos notables en esta área se incluyen el de Achanta et al. [19], que compara los superpíxeles SLIC con otros métodos de vanguardia, el estudio de Ren y Malik [20], que se enfoca en el aprendizaje de un modelo de clasificación para la segmentación, y la investigación de Levinshtein et al. [21], que presenta TurboPixels, una técnica de superpíxeles basada en flujos geométricos. Estas referencias subrayan la diversidad de métodos y aplicaciones en el estudio de los

superpíxeles.

1. *Método simple linear iterative clustering (SLIC)*

El método SLIC se basa en una adaptación del algoritmo k-means para segmentar una imagen en componentes que son más significativos y manejables que los píxeles individuales. La superioridad de SLIC se manifiesta en su capacidad para conformarse a los contornos de los objetos dentro de una imagen y en su eficiencia computacional, especialmente en lo que respecta al uso de memoria y al tiempo de procesamiento.

La innovación de SLIC reside en su enfoque de búsqueda localizada para la asignación de píxeles a superpíxeles. Tradicionalmente, k-means evalúa la distancia entre cada píxel y todos los centroides de los clústeres, un proceso que tiene una complejidad computacional de $O(kNI)$, donde k es el número de clústeres, N es el número de píxeles y I es el número de iteraciones necesarias para la convergencia.

Por el contrario, SLIC reduce esta complejidad a $O(N)$, independientemente del número de superpíxeles, mediante la restricción del espacio de búsqueda para la asignación de clústeres. En lugar de considerar toda la imagen, SLIC solo calcula las distancias en una región localizada $2S \times 2S$ alrededor de cada centroide de clúster, donde $S = \sqrt{N/k}$ es el tamaño esperado de un superpíxel. Esta aproximación no solo minimiza las operaciones de distancia sino que también asegura que la complejidad del algoritmo permanezca lineal respecto al número de píxeles.

La función de distancia utilizada por SLIC para asignar píxeles a superpíxeles es una combinación de diferencias de color en el espacio de color CIELAB y diferencias espaciales, dada por.

$$D = \sqrt{d_{lab}^2 + \left(\frac{d_{xy}}{S}\right)^2} \quad (1)$$

donde d_{lab} representa la distancia euclidiana en el espacio de color LAB y d_{xy} es la distancia euclidiana espacial entre el píxel y el centroide del superpíxel. El parámetro S actúa

como un regulador que equilibra la importancia relativa del color y la proximidad espacial.

Este proceso iterativo continúa hasta que los cambios en la asignación de los superpíxeles son mínimos, lo que generalmente se logra en menos de 10 iteraciones. El resultado es una colección de superpíxeles que respetan los bordes de los objetos y proporcionan una representación compacta y eficiente de la imagen [19]. La Figura V ilustra los superpíxeles generados mediante el método SLIC.



Figura V: Implementación del método SLIC en una imagen de 512x512 píxeles, generando 190 segmentos.

C. Machine learning

El Machine Learning es una disciplina dentro de la inteligencia artificial y la ciencia de la computación que se enfoca en el desarrollo de algoritmos capaces de aprender de los datos y tomar decisiones autónomas. Su esencia radica en la habilidad para procesar grandes cantidades de información, identificando patrones y realizando predicciones basadas en ellos. Esta capacidad se logra a través del uso de modelos matemáticos y estadísticos, como la regresión lineal y la regresión logística, que se entrenan con datos. Durante el entrenamiento, estos modelos ajustan sus parámetros de manera iterativa para mejorar su capacidad de predicción y generalización, lo cual les permite actuar eficazmente ante nuevos conjuntos de datos, haciendo del Machine Learning una herramienta poderosa y

versátil en el análisis de datos [22].

1. *Modelo de regresión logística*

Los modelos de clasificación probabilística como la Regresión Logística destaca por su eficacia en modelar la incertidumbre y manejar casos donde las categorías no son mutuamente excluyentes. La Regresión Logística, utilizada en clasificación binaria, predice la probabilidad de que una entrada pertenezca a una de dos clases. Este modelo emplea una transformación logística sobre una combinación lineal de las características de entrada. Matemáticamente, se define como:

$$p(C_1|\phi) = \sigma(\mathbf{w}^T \phi) \quad (2)$$

donde σ es la función sigmoide logística, $\mathbf{w}$ el vector de pesos, y ϕ las características transformadas de la entrada. La función sigmoide, dada por.

$$\sigma(a) = \frac{1}{1 + e^{-a}} \quad (3)$$

convierte la entrada lineal en una probabilidad, manteniendo el resultado entre 0 y 1. Además, la Regresión Logística utiliza el logaritmo de la razón de probabilidades (logit) para modelar la relación entre las características y la probabilidad de pertenecer a una clase. Esta relación logit se expresa como:

$$\log \left(\frac{p(C_1|\phi)}{1 - p(C_1|\phi)} \right) = \mathbf{w}^T \phi \quad (4)$$

Esto demuestra cómo la Regresión Logística no solo modela probabilidades directamente, sino que también conecta linealmente el logaritmo de las probabilidades con las características de entrada. Esta característica hace que la Regresión Logística sea particularmente útil en situaciones de decisión binaria, como sí/no o verdadero/falso, proporcionando una base sólida para la toma de decisiones basada en probabilidades [23].

D. Fundamentos de redes neuronales

El aprendizaje automático es un campo que permite a las computadoras adquirir conocimiento a través de la inferencia y la generalización de patrones, prescindiendo de la programación directa. Esta capacidad de adaptación y mejora continua es importante para el desarrollo de sistemas inteligentes y autónomos.

En el ámbito del aprendizaje automático, las redes neuronales representan una aproximación computacional basada en la arquitectura del cerebro humano, donde las unidades de procesamiento interconectadas emulan de forma abstracta las dinámicas neuronales. Aunque inspiradas en los principios de funcionamiento cerebral, estas redes no replican la complejidad completa del procesamiento cognitivo humano, sino que constituyen una versión simplificada y funcional adaptada a los sistemas computacionales [24].

1. *Redes neuronales artificiales*

El perceptrón es la unidad básica de procesamiento en redes neuronales, actuando como una neurona artificial para simular el procesamiento de información. Se denota un perceptrón con entradas $\mathbf{X} = (X_1, X_2, \dots, X_n)$, donde cada X_i representa un atributo cuantitativo de los datos.

Los pesos $\mathbf{W} = (W_1, W_2, \dots, W_n)$ se asignan a estas entradas, análogos a la fuerza de las conexiones sinápticas en neuronas biológicas. Durante el entrenamiento, estos pesos se ajustan para optimizar la contribución de cada entrada a la salida.

El perceptrón calcula una suma ponderada de sus entradas, dada por la expresión:

$$z = \sum_{i=1}^n W_i X_i + b,$$

donde b es el término de sesgo, que actúa como umbral para la activación del perceptrón.

La función de activación f , transforma esta suma ponderada en una salida y :

$$y = f(z).$$

Esta función puede ser una función escalón, sigmoide, o cualquier otra función no lineal, transformando la entrada lineal en una salida que puede ser binaria o un valor continuo.

La salida y del perceptrón se utiliza como entrada para otros perceptrones en la red, permitiendo la construcción de redes neuronales complejas para tareas avanzadas de procesamiento de datos. La efectividad del perceptrón en la resolución de problemas depende en gran medida de la elección correcta de los pesos y del sesgo, ajustados a través de algoritmos de aprendizaje [24]. La Figura VI ilustra gráficamente el funcionamiento del perceptrón.

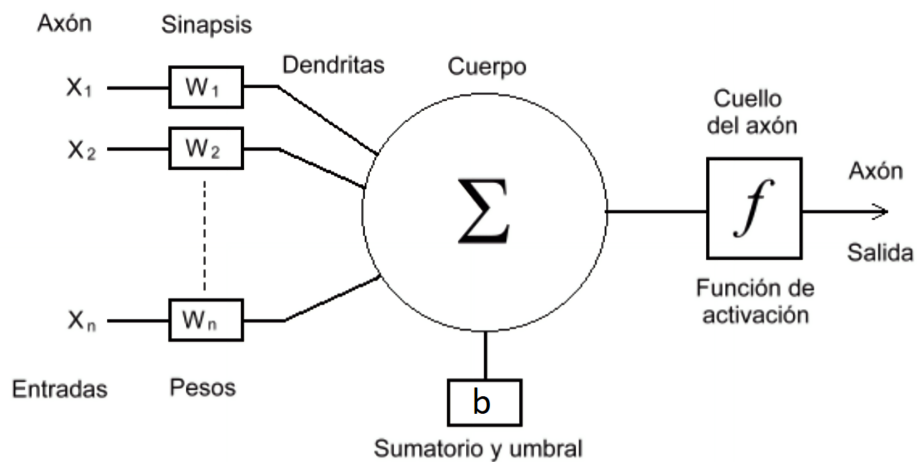


Figura VI: Diagrama de un perceptrón simple [25].

2. Redes neuronales convolucionales

Las Redes Neuronales Convolucionales (CNN) son una clase especializada de redes neuronales, diseñadas principalmente para el procesamiento de datos visuales como imágenes. Su estructura única, compuesta por capas convolucionales y de agrupamiento, les permite destacar en tareas de visión por computadora, como el reconocimiento facial, la detección de objetos y el análisis de imágenes médicas [26]. A continuación, se detalla la

función y la importancia de cada componente en una CNN.

Capas convolucionales: Estas capas emplean filtros o *kernels* que se desplazan a través de la imagen de entrada para producir mapas de características. La operación de convolución se ilustra para imágenes en escala de grises en la Figura VII y se define matemáticamente como:

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) \cdot K(m, n) \quad (5)$$

Esta operación implica la multiplicación elemento a elemento del kernel K con una subregión de la imagen I , seguida de la suma de estos productos, y se repite para cada posición en la imagen. En imágenes multicanal, como RGB, la convolución se realiza por separado en cada canal y los resultados se combinan para formar un único mapa de características:

$$(I * K)(i, j) = \sum_c \sum_m \sum_n I_c(i - m, j - n) \cdot K_c(m, n) \quad (6)$$

donde c representa el canal de color [27, 28].

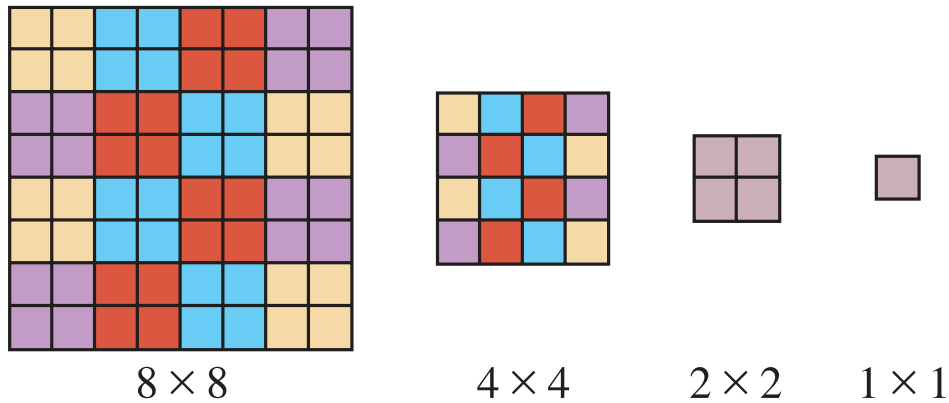


Figura VII: Operación convolución con un filtro de 3x3 [29].

Capas de agrupamiento: Las Capas de Agrupamiento (Pooling) en las redes neuronales convolucionales realizan la reducción de la dimensión espacial de los mapas de características, resultando en una disminución de parámetros y uso computacional. Ubicadas típicamente después de las capas convolucionales, estas capas aumentan la eficiencia y

ayudan a prevenir el sobreajuste [26]. El proceso de agrupamiento, ilustrado en la Figura VIII, se realiza mediante operaciones de agrupamiento promedio o máximo.

El agrupamiento promedio se define como:

$$P_{\text{avg}}(S) = \frac{1}{n} \sum_{i=1}^n S_i \quad (7)$$

donde P_{avg} es el valor resultante del agrupamiento promedio en una sub-región S del mapa de características, con n siendo el número de elementos en S .

El agrupamiento máximo, por otro lado, se define como:

$$P_{\text{max}}(S) = \text{máx}(S_i) \quad (8)$$

donde P_{max} es el valor resultante del agrupamiento máximo y $\text{máx}(S_i)$ es el valor más alto en la sub-región S .

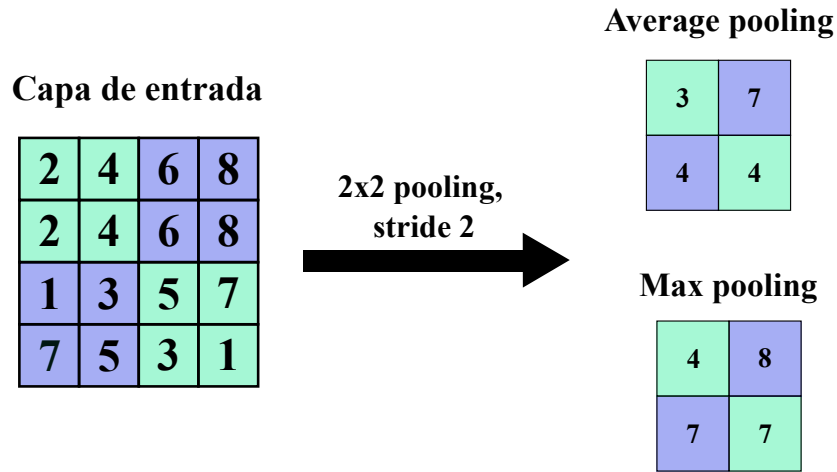


Figura VIII: Operación de agrupamiento promedio y máximo en una CNN.

3. Tipos de entrenamiento en redes neuronales

En las redes neuronales, el entrenamiento es el proceso mediante el cual un modelo aprende a realizar tareas como clasificación, regresión o reconocimiento de patrones a partir de datos de entrada. Existen varios tipos de entrenamiento, cada uno con su metodología y aplicación particular. A continuación se presentan algunos tipos de entrenamiento más comunes.

Entrenamiento supervisado : El entrenamiento supervisado es un enfoque central en el campo del aprendizaje automático, donde el modelo se entrena utilizando un conjunto de datos que incluye tanto las entradas como las salidas deseadas. Esta metodología implica proporcionar a la red neuronal ejemplos específicos (datos de entrada) junto con las respuestas correctas (etiquetas o salidas), permitiendo que el modelo aprenda la correlación entre ellos. Durante el proceso de entrenamiento, la red ajusta sus parámetros internos, generalmente mediante algoritmos de optimización, con el objetivo de minimizar una función de pérdida. Esta función cuantifica el error entre las predicciones de la red y las salidas reales proporcionadas en el conjunto de entrenamiento. [30]. Un esquema general de este tipo de entrenamiento se ilustra en la Figura IX.

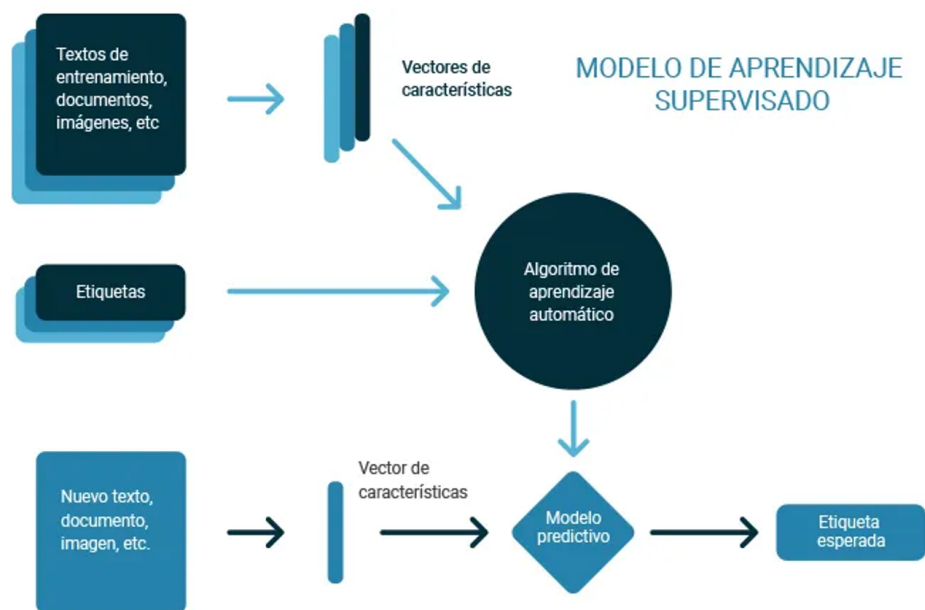


Figura IX: Flujo del modelo de aprendizaje supervisado [31].

Entrenamiento no supervisado : El entrenamiento no supervisado se centra en el análisis de datos no etiquetados para descubrir patrones y estructuras ocultas. A diferencia del entrenamiento supervisado, este método no requiere datos con etiquetas predefinidas, permitiendo que los algoritmos identifiquen de manera autónoma regularidades y relaciones intrínsecas. Las redes neuronales son una de las herramientas utilizadas en este enfoque,

pero existen también métodos clásicos igualmente efectivos. Entre estos se encuentran el k-NN (k-vecinos más cercanos) para tareas de agrupación, donde se identifican grupos o clusters basados en la proximidad de los datos [30]. Estas técnicas son fundamentales en inteligencia artificial para explorar y entender grandes conjuntos de datos, facilitando así el descubrimiento de insights valiosos y patrones significativos [32]. Un esquema general de este tipo de entrenamiento se ilustra en la Figura X.

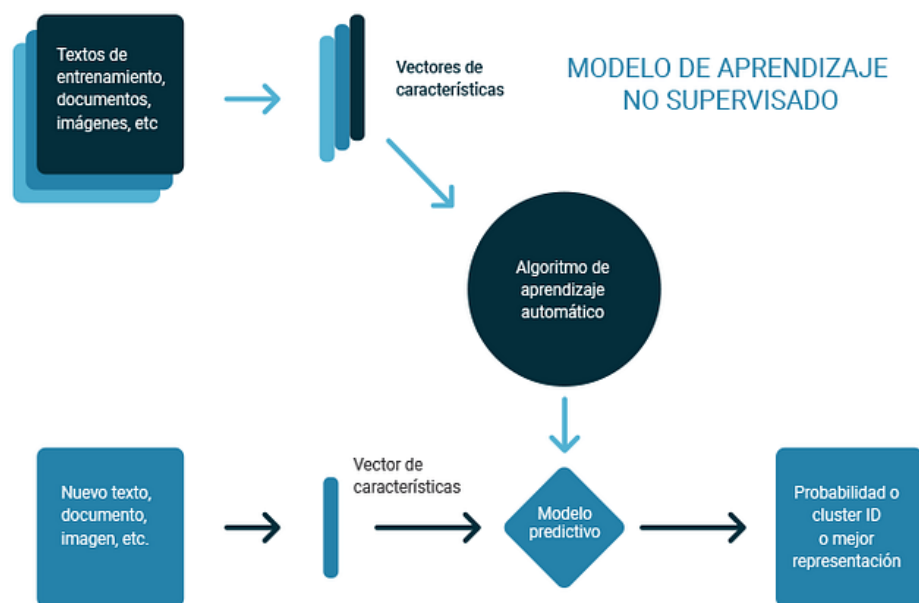


Figura X: Flujo del modelo de aprendizaje no supervisado [31].

Entrenamiento por refuerzo : En el aprendizaje por refuerzo, el modelo aprende a tomar decisiones a través de recompensas y penalizaciones en lugar de recibir respuestas correctas directas. En lugar de decirle explícitamente qué hacer, el modelo interactúa con su entorno, ajustando su comportamiento para obtener más recompensas y menos penalizaciones. Este enfoque es muy útil en situaciones donde la adaptabilidad y la toma de decisiones autónoma son esenciales, como en robótica y juegos [33]. Algunos modelos destacados en esta categoría incluyen algoritmos como Q-Learning y enfoques más avanzados basados en políticas, ofreciendo diversas estrategias de aprendizaje. Un esquema general de este tipo de entrenamiento se ilustra en la Figura XI.

MODELO DE APRENDIZAJE POR REFUERZO



Figura XI: Flujo del modelo de aprendizaje por refuerzo [31].

4. *Aprendizaje supervisado en redes neuronales*

El aprendizaje supervisado en redes neuronales se inicia con una asignación de los pesos iniciales de manera aleatoria a las conexiones entre neuronas. Este paso establece un punto de partida en el proceso de aprendizaje, que se centra en optimizar la precisión de las predicciones realizadas por la red.

Durante la fase de propagación hacia adelante (forward propagation), las entradas son procesadas secuencialmente por las neuronas de cada capa. Cada neurona aplica una función de activación a sus entradas y transmite el resultado a la capa siguiente [30]. Este proceso se ilustra en la Figura XII, donde el conjunto de datos de entrenamiento consta de entradas $x_1, x_2, \dots, x_m$ y una salida deseada z . Los pesos de las conexiones, denotados como w_{mn} , entre las entradas x_m y las neuronas n se ajustan durante el entrenamiento. La salida de cada neurona se denota por y_n .

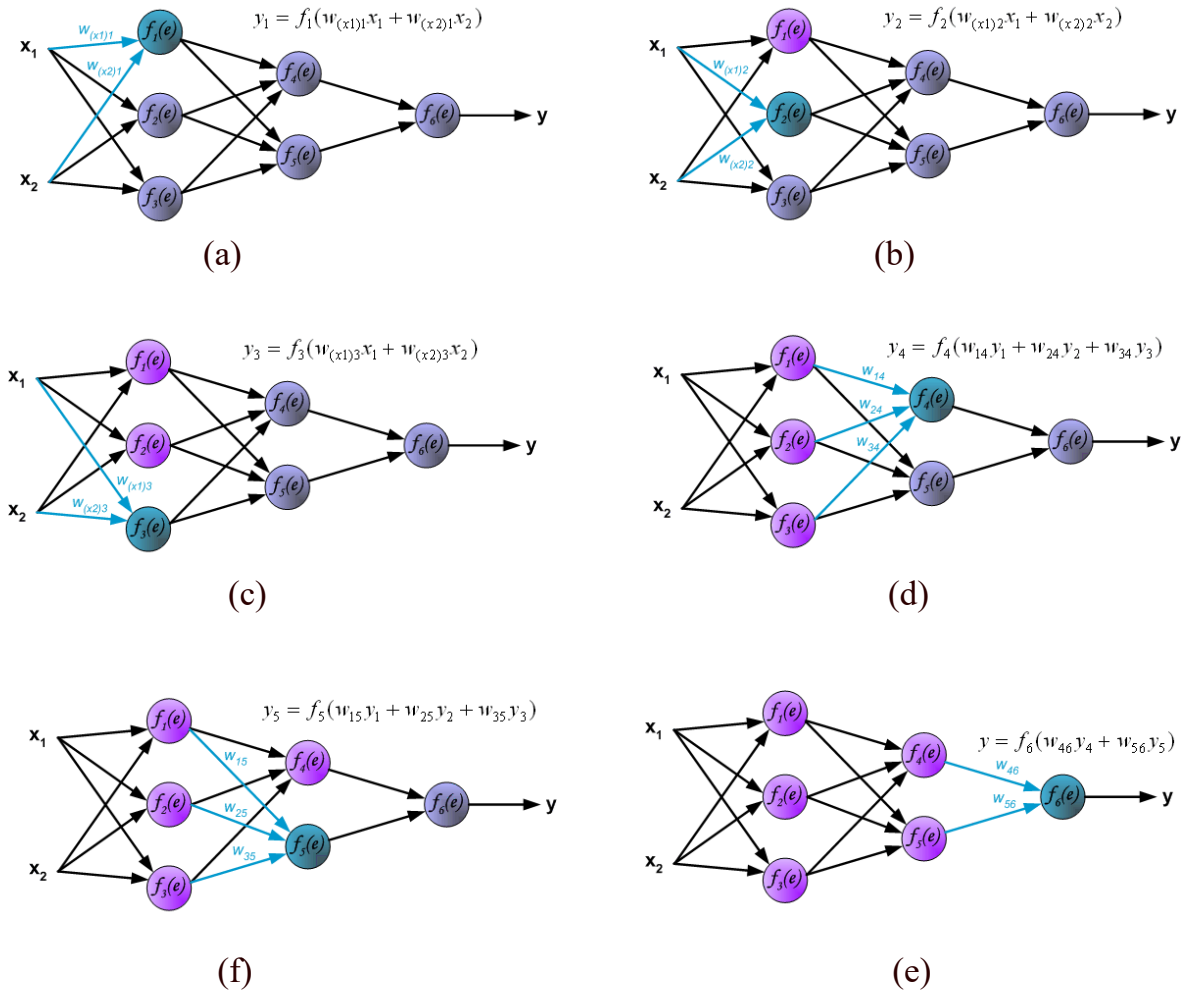


Figura XII: Procesamiento secuencial de entradas en una red neuronal [34].

Posteriormente, se evalúa la discrepancia entre las predicciones de la red y los valores reales a través de una función de pérdida. Este cálculo permite el ajuste de los parámetros de la red, que se realiza mediante el algoritmo de retropropagación (backpropagation). En este proceso, se calcula el error de predicción, representado por δ , y se propaga hacia atrás a través de la red para actualizar los pesos [30]. Este mecanismo se muestra en la Figura XIII, donde el valor de salida de la red y se compara con la salida deseada z , generando la señal de error δ .

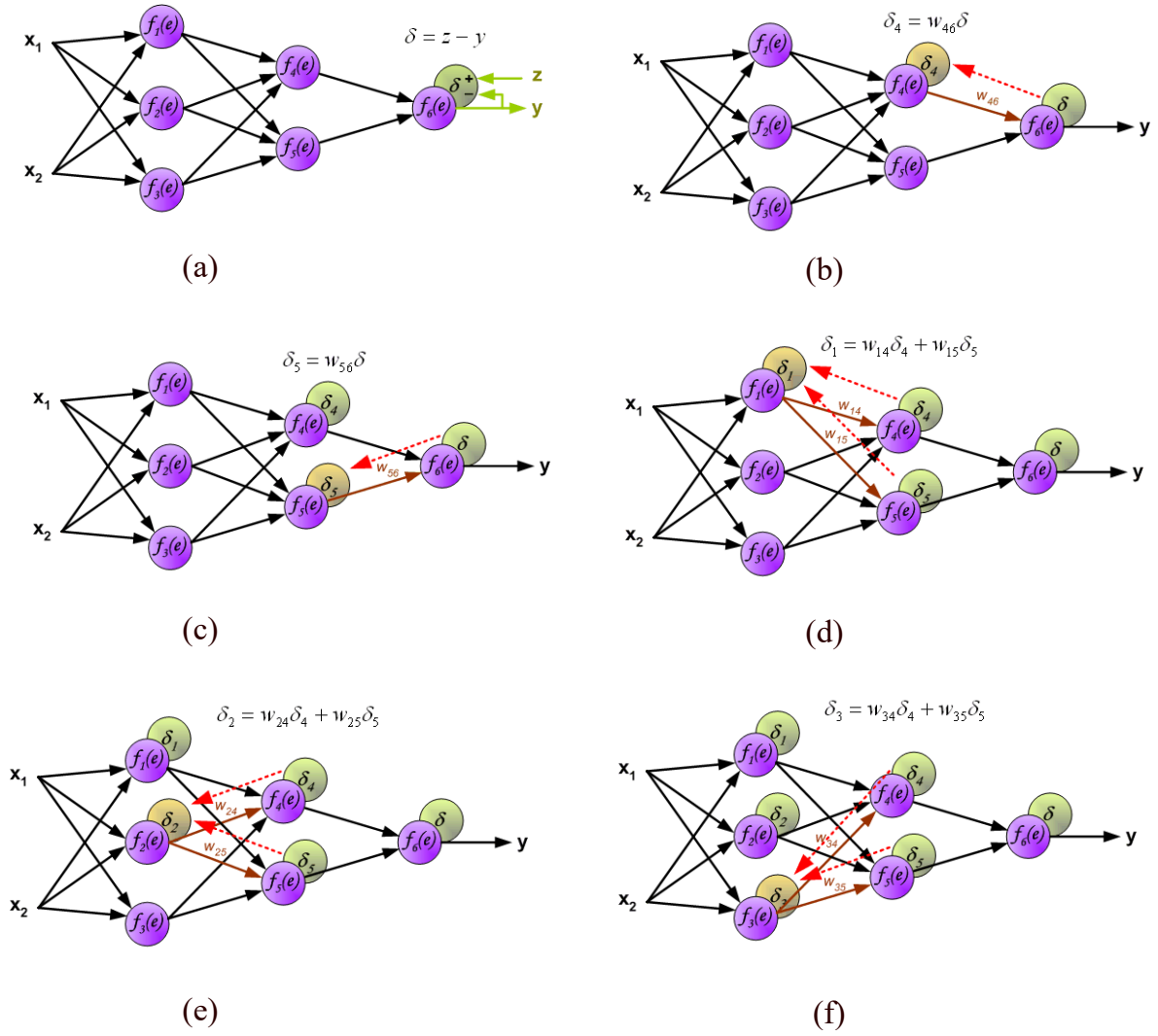


Figura XIII: Retropropagación del error para ajuste de pesos en la red [34].

La actualización de los pesos en una red neuronal se efectúa mediante algoritmos de optimización, como el método de descenso de gradiente. Cada neurona en la red se compone de dos unidades principales. La primera unidad suma los productos de los coeficientes de pesos y las señales de entrada, generando la señal de salida del sumador, denotada como e . La segunda unidad aplica una función no lineal, conocida como función de activación neuronal, a esta señal. La salida de esta función de activación, denotada como $y = f(e)$, también representa la señal de salida de la neurona.

Este proceso iterativo, ilustrado en la Figura XIV, mejora gradualmente la capacidad predictiva de la red ajustándose en base a los datos de entrenamiento. Al calcular la señal de error para cada neurona, es posible modificar los coeficientes de ponderación de cada

nodo de entrada a la neurona. En este contexto, la expresión $\frac{df(e)}{de}$ representa la derivada de la función de activación neuronal. El coeficiente de aprendizaje, denotado por η , determina la tasa a la que se ajustan estos pesos [30].

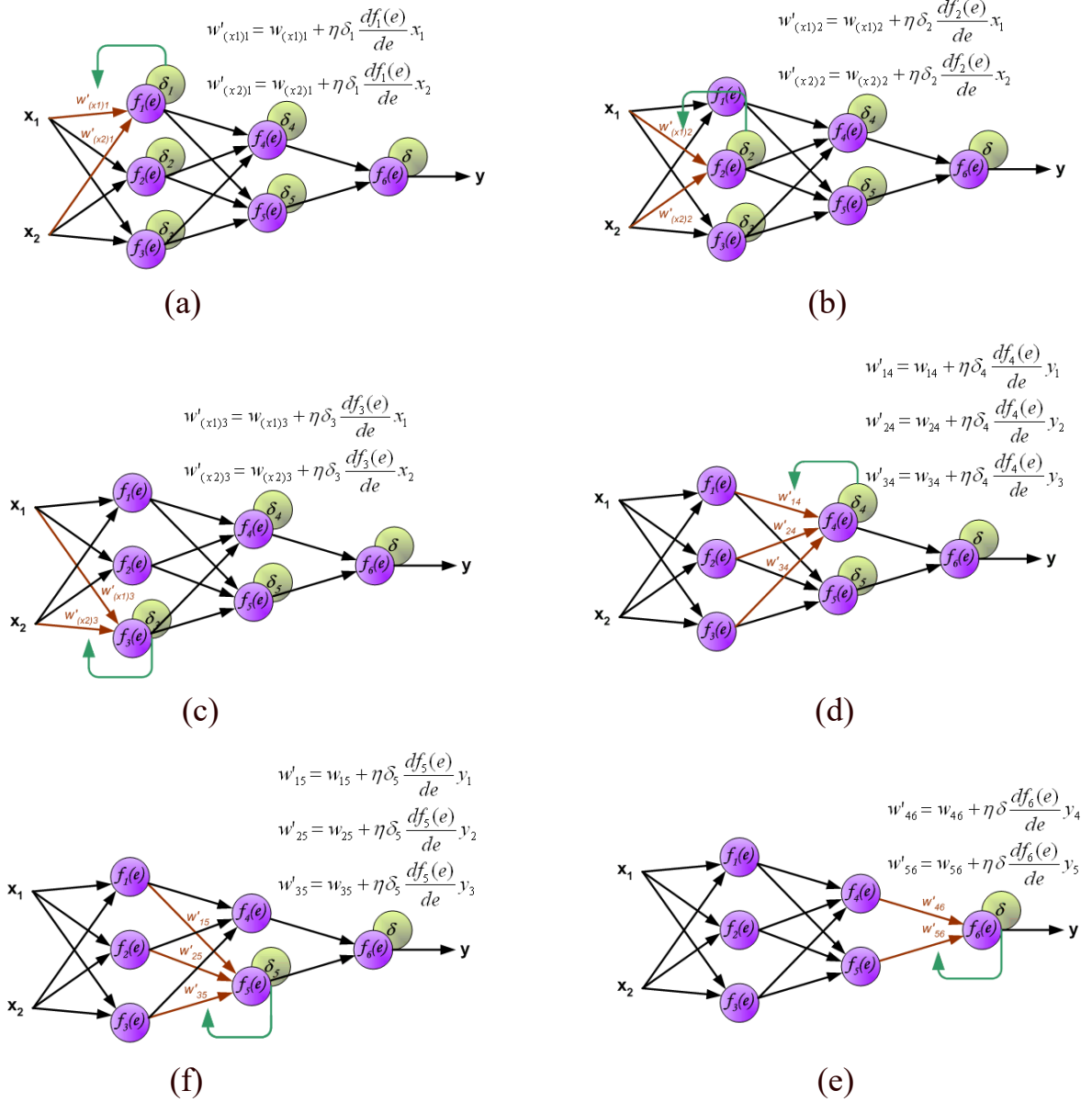


Figura XIV: Optimización de pesos en una red neuronal multicapa [34].

Aunque el ejemplo se centra en una red de tamaño reducido, los principios y métodos descritos son igualmente aplicables a redes neuronales de mayor escala y complejidad. Este proceso general de entrenamiento se ilustra en la Figura XV, donde se presenta un esquema del funcionamiento del entrenamiento supervisado en una red neuronal. Cabe destacar que cada componente de este proceso, desde la inicialización de pesos hasta la

optimización final, encierra una mayor complejidad en su implementación y teoría. Estos aspectos serán explorados con mayor detalle en los siguientes subtemas, proporcionando una comprensión más profunda de cada etapa del proceso de aprendizaje supervisado en redes neuronales.

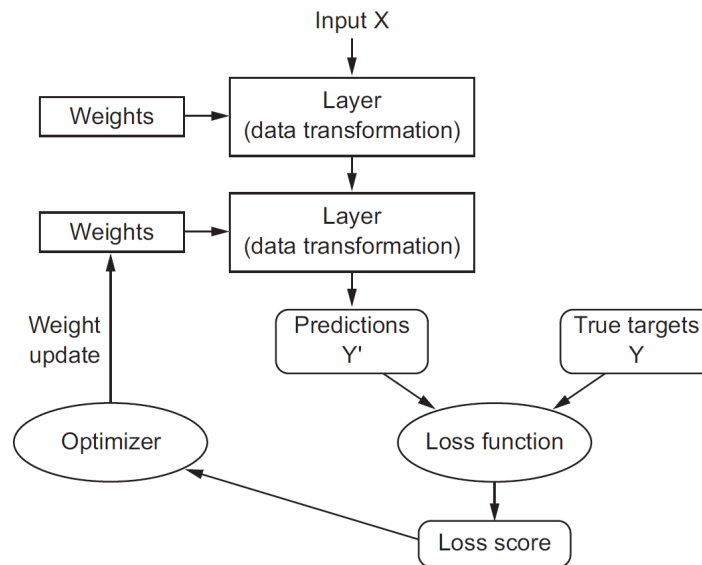


Figura XV: Flujo de aprendizaje en redes neuronales multicapa [35].

En la implementación de redes neuronales convolucionales (CNNs), la optimización de pesos es un proceso crítico que afecta a los filtros o kernels en las capas convolucionales. Inicialmente, estos filtros se configuran de manera aleatoria o mediante algoritmos de inicialización preestablecidos. A lo largo del entrenamiento, mediante el algoritmo de retropropagación, se ajustan para resaltar las características importantes de las imágenes de entrada. Este ajuste es evidente en la Figura XVI, donde se muestra cómo la CNN procesa una imagen a través de sus capas sucesivas. Las primeras capas se enfocan en características de bajo nivel como bordes y esquinas. Capas intermedias integran estas características en patrones más complejos, y las capas más avanzadas consolidan estos patrones en representaciones de alto nivel que son decisivas para la clasificación. Cada iteración de entrenamiento refina la habilidad de la red para reducir la función de pérdida y mejorar su precisión al mapear entradas a sus etiquetas de salida correctas [35].

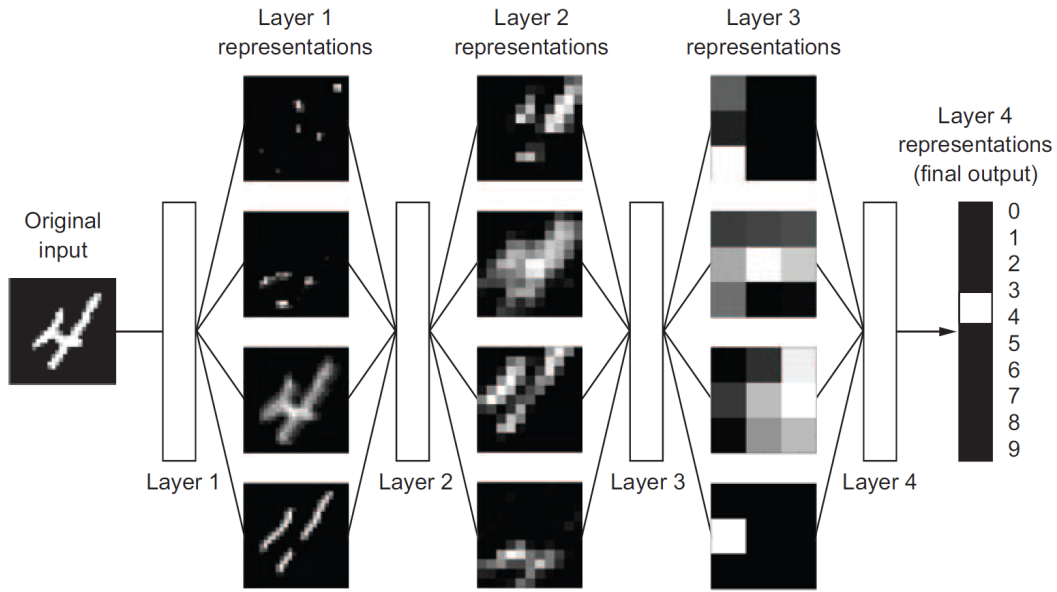


Figura XVI: Visualización de características en la red neuronal para clasificación de dígitos [35].

Función de activación : En el contexto de las redes neuronales artificiales, la función de activación desempeña un rol matemático crucial al determinar la salida de una neurona a partir de su entrada neta, también conocida como potencial postsináptico. Matemáticamente, se representa como

$$y = f(x) = f\left(\sum_{i=1}^n w_i x_i + b\right),$$

donde x_i son las entradas, w_i los pesos asociados, b el sesgo, y f la función de activación. Este mecanismo permite transformar una combinación lineal de las entradas en una salida no lineal, habilitando así la red neuronal para capturar relaciones complejas y patrones intrincados presentes en los datos.

Las funciones de activación tradicionales incluyen funciones continuas y monótonas crecientes como la función sigmoide:

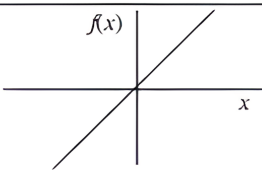
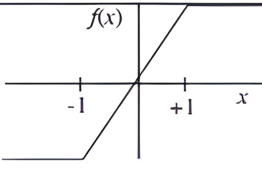
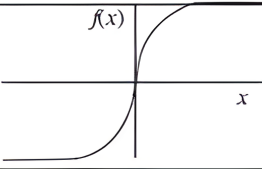
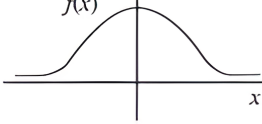
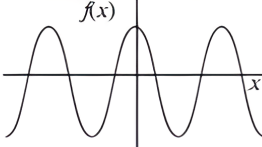
$$f(x) = \frac{1}{1 + e^{-x}},$$

y la tangente hiperbólica:

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Estas funciones poseen características útiles como la suavidad y derivabilidad, aspectos fundamentales para la optimización mediante algoritmos como la retropropagación [36]. La Tabla I resume algunas de las funciones de activación comunes.

Tabla I: Funciones de activación [29].

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = \text{sign}(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Lineal a tramos	$y = \begin{cases} -1, & \text{si } x < -l \\ x, & \text{si } -l \leq x \leq l \\ +1, & \text{si } x > l \end{cases}$	$[-1, +1]$	
Sigmoidea	$y = \frac{1}{1 + e^{-x}}$ $y = \text{tgh}(x)$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0, +1]$	
Sinusoidal	$y = A \text{sen}(\omega x + \varphi)$	$[-1, +1]$	

En el diseño de arquitecturas modernas de redes neuronales, una de las funciones más prominentes es la Unidad Lineal Rectificada (ReLU, por sus siglas en inglés), definida matemáticamente como:

$$f(x) = \text{máx}(0, x),$$

donde $f(x)$ mantiene una salida igual a la entrada cuando esta es positiva ($x > 0$) y resulta

en cero para valores negativos ($x \leq 0$). Esta función no es saturante, lo que significa que el gradiente no tiende a cero cuando $x \rightarrow \infty$, a diferencia de funciones como la sigmoide y la tangente hiperbólica. Esta propiedad es esencial para mitigar el problema del desvanecimiento del gradiente (*vanishing gradient*), facilitando el entrenamiento de redes profundas [30, 37].

Además de su simplicidad computacional, la ReLU introduce esparsidad en las activaciones, ya que un porcentaje significativo de las neuronas generan salidas nulas, lo que puede mejorar la eficiencia computacional y reducir el riesgo de sobreajuste.

Debido a ciertas limitaciones de la ReLU estándar, como el problema de unidades "muertas" (*dead neurons*), donde una neurona queda atrapada en una salida constante de cero, se han propuesto varias variantes:

ReLU con fugas (Leaky ReLU):

$$f(x) = \begin{cases} x & \text{si } x > 0, \\ \alpha x & \text{si } x \leq 0, \end{cases}$$

donde α es un pequeño valor positivo (por ejemplo, $\alpha = 0,01$). Esta variante introduce una pendiente no nula en la región negativa, lo que permite la propagación de gradientes incluso para $x \leq 0$ [38].

Unidad Lineal Exponencial (ELU):

$$f(x) = \begin{cases} x & \text{si } x > 0, \\ \alpha(e^x - 1) & \text{si } x \leq 0, \end{cases}$$

donde $\alpha > 0$ controla la suavidad en la región negativa. Esta función combina las ventajas de la ReLU con una salida más equilibrada cerca de cero, ayudando a acelerar la convergencia [39].

La elección de la función de activación tiene un impacto significativo en el rendimiento y la eficiencia de las redes neuronales. Las variantes de ReLU mejoran la capacidad de la red para manejar gradientes y contribuyen a una convergencia más rápida en redes profundas.

Sin embargo, su selección depende del problema específico y las características de los datos, así como de la arquitectura de la red.

Inicialización de pesos : La inicialización de pesos en una red neuronal, denotada matemáticamente como W , es importante para garantizar una convergencia efectiva durante el entrenamiento. La inicialización nula, es decir, $W = 0$, es desaconsejada ya que conduce a gradientes idénticos en las neuronas de una misma capa, representado como $\nabla W = 0$, lo que a su vez impide el aprendizaje diferencial.

Una práctica común es la inicialización aleatoria, donde los pesos se asignan siguiendo una distribución, por ejemplo, una distribución normal con media cero y una pequeña varianza σ^2 , es decir, $W \sim \mathcal{N}(0, \sigma^2)$. Esta estrategia ayuda a mantener la diversidad en el aprendizaje de las neuronas y evitar la simetría en la actualización de los pesos.

En el contexto de redes neuronales profundas, esta inicialización es fundamental para prevenir problemas como el desvanecimiento o la explosión de gradientes. Estos problemas se pueden describir como una disminución exponencial o un incremento exponencial de los gradientes (respectivamente) a lo largo de las capas, que matemáticamente se puede representar como $\|\nabla W\| \rightarrow 0$ (desvanecimiento) o $\|\nabla W\| \rightarrow \infty$ (explosión) durante la retropropagación. Por tanto, la elección de σ^2 y la distribución de inicialización dependen de la arquitectura de la red y la función de activación de las neuronas, con el objetivo de optimizar el flujo de gradientes [27].

Inicialización de pesos en redes neuronales : La inicialización de los pesos, W , en una red neuronal es un paso crítico para asegurar una convergencia eficiente durante el proceso de aprendizaje. La inicialización nula, es decir, establecer todos los pesos a cero ($W = 0$), es desaconsejable debido a que conduce a la homogeneidad en los gradientes de la red ($\nabla W = 0$), lo que resulta en una falta de aprendizaje diferenciado en las neuronas.

Para superar este problema, la práctica recomendada es la inicialización aleatoria

de pesos. Esta técnica consiste en asignar los valores de W siguiendo una distribución probabilística, comúnmente una distribución normal. La forma estándar es utilizar una media de cero y una varianza controlada (σ^2), representada como $W \sim \mathcal{N}(0, \sigma^2)$. Esta estrategia promueve un aprendizaje diversificado y evita la simetría en la actualización de los pesos, fundamental para el funcionamiento eficaz de la red.

En el contexto de las redes neuronales profundas, una adecuada inicialización de pesos es importante para prevenir el fenómeno del desvanecimiento o la explosión de gradientes. Estos se manifiestan como una reducción ($\|\nabla W\| \rightarrow 0$) o un aumento ($\|\nabla W\| \rightarrow \infty$) exponenciales de los gradientes durante la retropropagación. Por lo tanto, la selección precisa de σ^2 y la metodología de inicialización deben ser adaptadas específicamente a la arquitectura de la red y a las características de la función de activación utilizada, con el objetivo de mantener un flujo de gradientes adecuado y eficiente a través de las capas de la red [27, 30].

Método de optimización : En el entrenamiento de , los métodos de optimización tienen la tarea de ajustar los pesos W de la red. Su objetivo es reducir al mínimo la función de pérdida L , que representa la discrepancia entre las predicciones de la red y los valores objetivo. Para lograr esto, se emplea el algoritmo de retropropagación, un proceso que calcula el gradiente de la función de pérdida con respecto a cada peso individual w_i en el conjunto W [40].

El gradiente $\nabla L(w_i)$ proporciona la dirección en la que se debe ajustar cada peso w_i para disminuir el valor de L . En cada iteración del proceso de entrenamiento, los métodos de optimización aplican pequeños cambios a los pesos, guiados por este gradiente. La magnitud de estos ajustes está regulada por la tasa de aprendizaje, que determina qué tan grandes son los pasos que se dan en el espacio de parámetros hacia el mínimo de la función de pérdida [41]. Al continuar este proceso a lo largo de múltiples iteraciones, los pesos W se afinan gradualmente, lo que lleva a una mejora en la precisión de las predicciones de la red.

Uno de los algoritmos en este campo es el Descenso del Gradiente Estocástico (SGD). Este algoritmo actualiza los pesos según la ecuación:

$$W_{t+1} = W_t - \eta \nabla L(W_t) \quad (9)$$

donde η es la tasa de aprendizaje fija y t indica la iteración actual [42]. Aunque efectivo, el SGD puede enfrentar desafíos debido a su tasa de aprendizaje uniforme, que no se ajusta durante el entrenamiento.

Para mejorar esto, se han desarrollado algoritmos como Adam (Adaptive Moment Estimation), que combina las fortalezas del Descenso del Gradiente con Momento y la adaptabilidad de tasas de aprendizaje de RMSprop [43]. Adam se distingue por ajustar los pesos W de manera más efectiva, empleando estimaciones adaptativas de los momentos de primer y segundo orden de los gradientes. Esta característica hace que Adam sea eficiente en diversos escenarios de entrenamiento, adaptándose a las particularidades de los datos y la estructura de la red neuronal.

La actualización de los pesos en el optimizador Adam comienza con el cálculo del gradiente de la función de pérdida L con respecto a los pesos W en la iteración t , que se denota como $\nabla L(W_t)$ [43].

Luego, se actualizan los estimadores de los momentos de primer orden, conocidos como momentum m_t , y de segundo orden, las velocidades v_t . Esto se hace mediante las fórmulas:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla L(W_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla L(W_t))^2$$

Aquí, β_1 y β_2 son hiperparámetros que controlan estos momentos.

Para mitigar el sesgo inicial en estos estimadores, se aplican correcciones:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Finalmente, la actualización de los pesos W se realiza utilizando estos momentos corregidos, según la fórmula:

$$W_{t+1} = W_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

donde η es la tasa de aprendizaje y ϵ es un pequeño número para prevenir la división por cero [43].

Este conjunto de operaciones permite que Adam ajuste los pesos de manera eficaz, mejorando la eficiencia del proceso de entrenamiento.

Estos métodos avanzados, como Adam, han demostrado un rendimiento superior en términos de una convergencia más rápida y estable hacia soluciones óptimas, en comparación con métodos tradicionales como el SGD, tal como se muestra en la Figura XVII [30].

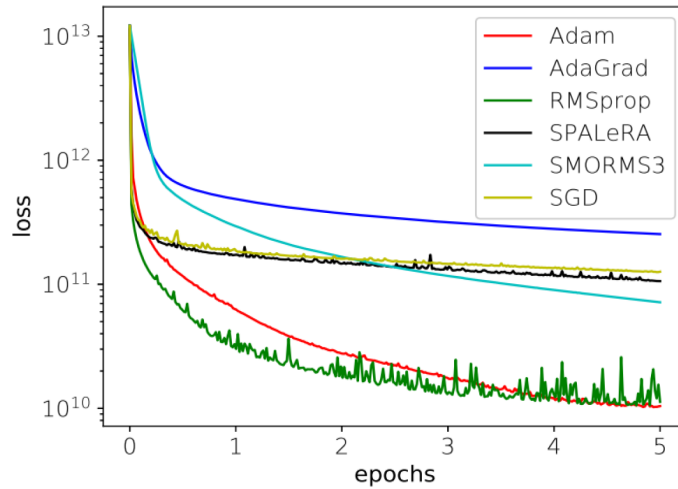


Figura XVII: Gráfica comparativo de la efectividad de métodos de optimización [44].

Tasa de aprendizaje (learning rate) La tasa de aprendizaje, simbolizada como η , es un hiperparámetro clave en el algoritmo de optimización, controlando el tamaño de los pasos en la actualización de los pesos W de la red. Este proceso se rige por la ecuación 9.

El valor de η influye directamente en la trayectoria de optimización en el espacio de parámetros. Un η excesivamente alto puede causar oscilaciones o incluso divergencia, alejándose de los mínimos de L . Por el contrario, un η muy bajo resulta en convergencias

lentas, requiriendo un número elevado de iteraciones para alcanzar el óptimo, lo que se puede representar como:

$$\lim_{t \rightarrow \infty} W_t = W^* \text{ si } \eta \text{ es adecuado}$$

donde W^* es el conjunto de pesos óptimos.

La adaptabilidad del learning rate, es decir, ajustar η durante el entrenamiento, es una estrategia para equilibrar la eficiencia y la precisión del proceso de optimización. Técnicas como el descenso de gradiente con tasa de aprendizaje adaptativa ajustan η en respuesta a ciertas condiciones durante el entrenamiento, lo que puede ser representado como:

$$\eta_{t+1} = f(\eta_t, \nabla L(W_t), t)$$

donde f es una función que ajusta η basándose en el comportamiento del gradiente y la iteración actual [30]. La Figura XVIII ilustra una representación visual de cómo diferentes tasas de aprendizaje impactan en el proceso de convergencia de una red neuronal hacia la solución óptima.

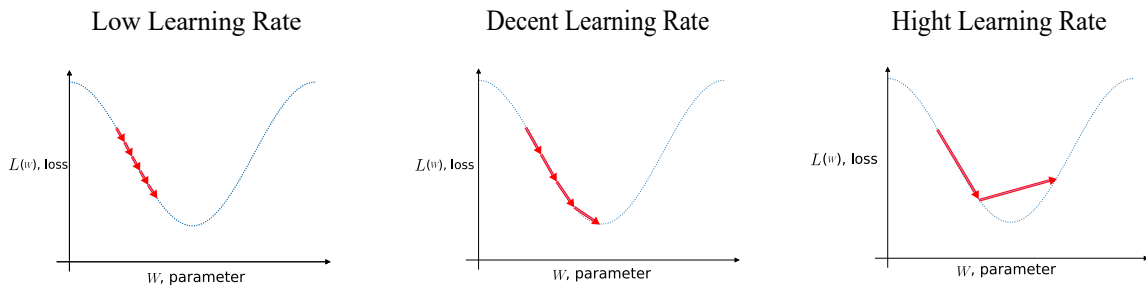


Figura XVIII: Diferencias al variar el learning rate [45].

Función de pérdida En el entrenamiento de redes neuronales, las funciones de pérdida, representadas por L , evalúan la diferencia entre las predicciones de la red, y , y los valores reales o esperados, z . Esta evaluación es crucial para determinar y mejorar la precisión de las predicciones de la red [30].

Entre las funciones de pérdida más utilizadas está el Error Cuadrático Medio (MSE), que calcula el promedio de los cuadrados de las diferencias entre los valores pronosticados

y los reales, ofreciendo una clara indicación de la variabilidad media de los errores en las predicciones. Se expresa mediante la fórmula:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (z_i - y_i)^2$$

donde N es el número total de muestras en el conjunto de datos. Minimizar el MSE durante el entrenamiento ajusta la red para realizar predicciones cada vez más exactas respecto a los valores reales [23].

En situaciones de clasificación, se emplea la Entropía Cruzada para evaluar la discrepancia entre las distribuciones de probabilidad predichas y las reales. En la clasificación binaria, la ecuación es:

$$\text{Cross-Entropy} = - \sum_{i=1}^N z_i \log(y_i) + (1 - z_i) \log(1 - y_i)$$

Mientras que para clasificación multiclase, la fórmula se amplía a:

$$\text{Cross-Entropy} = - \sum_{i=1}^N \sum_{c=1}^C z_{ic} \log(y_{ic})$$

donde C indica el número de categorías o clases. Esta función es útil para cotejar cómo las predicciones de la red se alinean con la distribución real de las etiquetas en los datos [37].

Tanto el MSE como la Entropía Cruzada, cuando se integran en el proceso de retropropagación, facilitan la calibración de los pesos W de la red para reducir los errores de predicción, orientando el aprendizaje hacia modelos más exactos y robustos.

pal1993review

Tamaño del lote (batch size) Debido a la gran cantidad de datos requeridos para el entrenamiento de redes neuronales, no es práctico ni eficiente procesar todos los datos de una sola vez. Por lo tanto, se fraccionan los datos en segmentos más manejables, conocidos como batches, que son procesados secuencialmente por la red [46]. Este proceso está asociado con el término "Época", que se refiere al ciclo completo de procesamiento de todos los batches del conjunto de datos original a través de la red. Cada vez que se completa

este ciclo, se considera que se ha finalizado una época. Dado que la optimización de los parámetros de la red es un proceso iterativo, normalmente se requieren múltiples épocas para alcanzar el entrenamiento deseado. El tamaño del batch, que es un hiperparámetro importante del modelo, varía según el tamaño del conjunto de entrenamiento. Para conjuntos pequeños (no mayores de 2000 elementos), es común que el tamaño del batch sea igual al tamaño del conjunto de entrenamiento, mientras que para conjuntos más grandes se suelen utilizar tamaños de batch que son potencias de 2 [47]. Es crucial asegurar que el tamaño del batch seleccionado sea compatible con la capacidad de memoria de la CPU o GPU [27].

5. *Aprendizaje por transferencia*

El aprendizaje por transferencia en aprendizaje automático se refiere a la estrategia de reutilizar modelos neuronales que ya han sido entrenados para aplicarlos a nuevas tareas. Esta técnica explota el conocimiento preexistente que las redes neuronales han desarrollado, particularmente en sus capas iniciales. En estas capas, la red ya ha aprendido a identificar patrones universales en los datos, tales como bordes y texturas en el caso de imágenes [48]. Al implementar el aprendizaje por transferencia, se usa este aprendizaje previo como fundamento, concentrando los esfuerzos de entrenamiento en las capas superiores. Las capas ajustadas en este proceso están diseñadas para discernir patrones más complejos y específicos que son cruciales para la nueva tarea en cuestión. Al hacerlo, se minimiza la cantidad de datos y recursos computacionales requeridos, ya que se evita el proceso de aprender estos patrones desde cero. Este enfoque ha demostrado ser especialmente eficaz en modelos entrenados con grandes conjuntos de datos, como ImageNet, donde la reutilización de las capas inferiores proporciona una transición fluida hacia la adaptación a nuevas tareas [49]. Este método se visualiza en la Figura XIX.

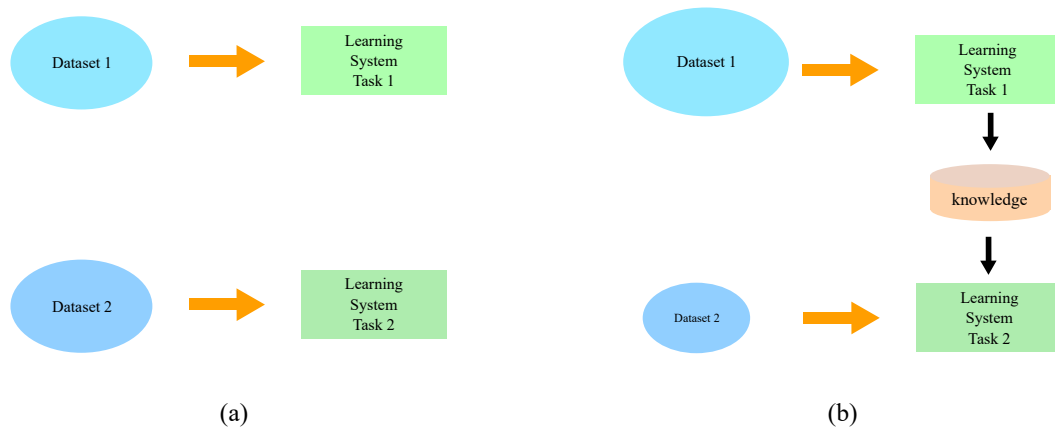


Figura XIX: Comparativa entre el entrenamiento desde cero y el entrenamiento mediante transfer learning [50].

6. *Fine tuning*

El fine-tuning es un paso adicional en el proceso de aprendizaje por transferencia. Mientras que el aprendizaje por transferencia puede involucrar el uso de un modelo preentrenado tal cual, el fine-tuning ajusta este modelo a una tarea específica mediante un proceso de entrenamiento adicional. En el fine-tuning, todas o algunas de las capas del modelo preentrenado son entrenadas nuevamente con un conjunto de datos específico de la nueva tarea [49].

Este proceso generalmente incluye los siguientes pasos:

1. **Inicialización del Modelo:** Comenzar con un modelo preentrenado, como aquellos entrenados en el conjunto de datos ImageNet [48].
2. **Congelación de Capas Iniciales:** Las primeras capas, que contienen características generales, se congelan para evitar que sus pesos se actualicen durante el entrenamiento [49].
3. **Entrenamiento de Capas Superiores:** Las capas superiores del modelo se entrenan con el nuevo conjunto de datos específico de la tarea. Esto permite que el modelo ajuste sus parámetros para la tarea específica, mejorando su rendimiento en dicha tarea [49].

4. **Ajuste Final:** Después de entrenar las capas superiores, el modelo completo puede ser ajustado finamente (fine-tuned), permitiendo que todas las capas, incluidas las iniciales, sean actualizadas si es necesario [49].

El fine-tuning permite aprovechar el conocimiento adquirido durante el entrenamiento original, mientras se adapta y optimiza el modelo para una nueva tarea específica, logrando así mejores resultados con menos datos y tiempo de entrenamiento comparado con entrenar un modelo desde cero [49].

E. Deep learning

El deep learning es una rama especializada del aprendizaje automático que enfatiza en aprender representaciones de datos a través de múltiples capas sucesivas, cada una añadiendo un nivel de comprensión más significativo. A diferencia del aprendizaje automático tradicional que se enfoca en pocas capas de representación, el deep learning utiliza muchas capas, aprendiendo automáticamente de los datos. Estas capas transforman progresivamente los datos en formas más informativas para tareas específicas, actuando como un proceso de destilación de información en múltiples etapas. Las redes neuronales en deep learning se componen de millones de parámetros (pesos), cuyo ajuste correcto se realiza a través de un proceso iterativo de entrenamiento. Este proceso utiliza una función de pérdida para medir la precisión de la red y un algoritmo de retropropagación para ajustar los pesos, optimizando así la red para realizar tareas como el reconocimiento de imágenes [35].

1. Red neuronal VGG16

La red neuronal VGG16, desarrollada por K. Simonyan y A. Zisserman de la Universidad de Oxford, es una red neuronal convolucional (CNN) ampliamente reconocida en el campo del procesamiento de imágenes. Esta red, una evolución de la arquitectura AlexNet, se caracteriza por utilizar múltiples capas de filtros convolucionales de tamaño 3x3,

a diferencia de los filtros más grandes usados en AlexNet. Esta estrategia de filtros más pequeños permite una mayor profundidad de la red con un menor número de parámetros, lo que se traduce en una extracción de características más eficiente y detallada [51].

Un aspecto clave de la VGG16 son los mapas de características generados por sus capas convolucionales. Estos mapas son representaciones de alto nivel de la imagen de entrada, donde cada capa sucesiva capta características más abstractas y complejas. A medida que la imagen pasa por las capas de la red, los primeros mapas de características tienden a identificar bordes y texturas simples, mientras que las capas más profundas capturan patrones más complejos, como formas o partes específicas de objetos [52].

La red VGG16, preentrenada en el amplio conjunto de datos ImageNet con más de 15 millones de imágenes en 22.000 categorías, está bien equipada para la clasificación de imágenes. Consta de 16 capas ponderadas, incluidas 13 capas convolucionales y 3 capas densas, optimizadas para imágenes de $224 \times 224 \times 3$ píxeles como se muestra en la Figura XX. Aunque este preentrenamiento proporciona una base de conocimiento general, la especialización en tareas concretas se logra mediante entrenamiento adicional con datos específicos, permitiendo así que la red se adapte y mejore su desempeño en aplicaciones más focalizadas [53].

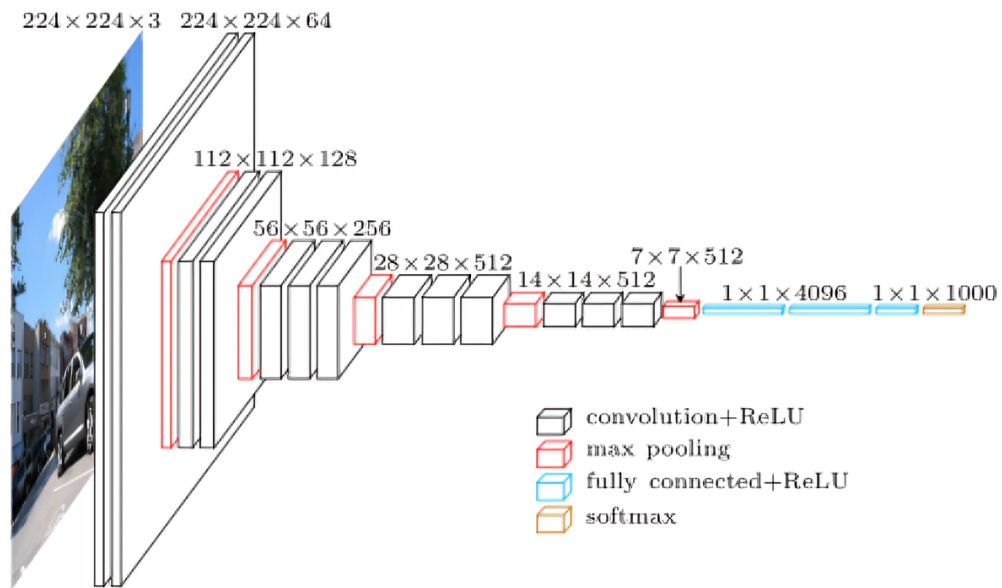


Figura XX: Arquitectura del modelo de red neuronal VGG 16 [54].

F. Detectores de objetos

En el campo de la visión por computadora, un área de estudio importante es la detección de objetos en imágenes. Esta disciplina se enfoca en identificar y categorizar varios objetos presentes en una imagen (ver Figura XXI). Esta tarea es esencial para comprender el contenido visual de manera automatizada [55].

Un enfoque significativo en este campo es el uso de detectores de objetos de una sola etapa. Estos sistemas se basan en una red convolucional compleja que realiza tres tareas de manera integrada y eficiente: identifica las características de los objetos, establece los límites de cada objeto mediante cuadros delimitadores y clasifica cada objeto en categorías específicas [56].

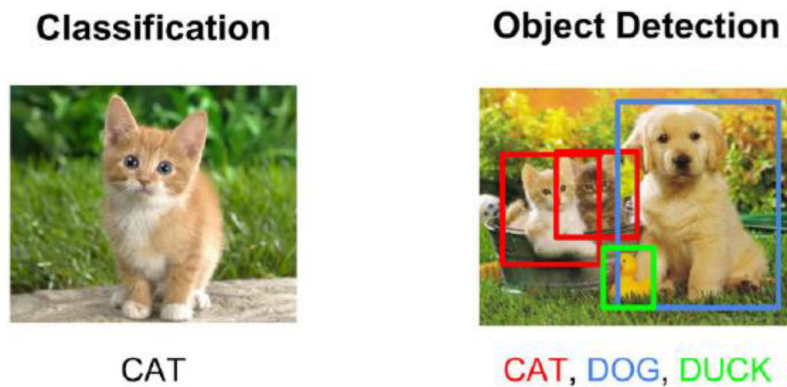


Figura XXI: Diferencia entre clasificación y detección de objetos [56].

1. *Detector de objeto YOLOv5*

YOLOv5, una iteración avanzada del sistema de detección de objetos YOLO, se destaca por su rapidez y alta precisión, adecuada para aplicaciones en tiempo real. Este modelo divide la imagen de entrada en una grilla de $S \times S$, tal y como se muestra en la Figura XXII, donde cada celda es responsable de predecir un objeto si su centro cae en ella. Cada celda predice B *bounding boxes* y un *confidence score*, que refleja la probabilidad de existencia de un objeto y es definido por el IoU (*intersection over union*) entre el *bounding box* predicho y el *ground truth*. Además, se predicen C probabilidades condicionales para cada clase, resultando en un tensor de salida de tamaño $S \times S \times (B * 5 + C)$ [57].

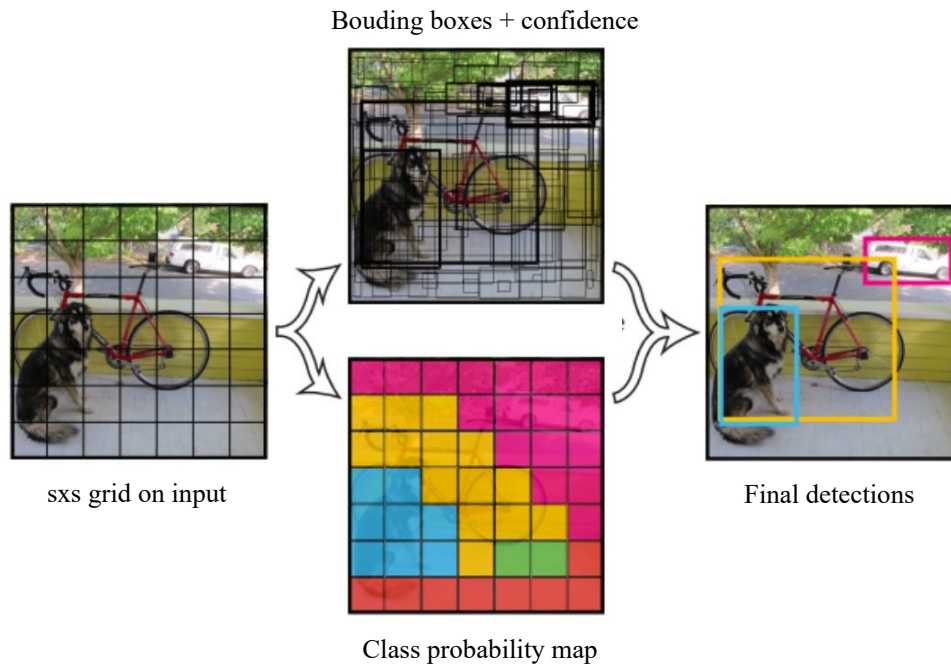


Figura XXII: Grilla de clasificación para detección de objetos del modelo YOLOv5 [58].

YOLOv5 se compone de:

- **Backbone:** Esta es la red convolucional primaria de YOLOv5. Su función es captar formas y características de la imagen a diferentes niveles de granularidad. Actúa como la base que extrae y procesa las características visuales fundamentales de las imágenes, desde los detalles más simples hasta los más complejos.
- **Neck:** Las capas que conforman el "Neck" sirven para combinar las características extraídas por el Backbone. Estas capas sintetizan la información recogida en diferentes niveles de abstracción y la preparan para la predicción precisa de objetos.
- **Head:** Esta sección de la arquitectura es responsable de la generación de las cajas delimitadoras *bounding boxes* que identifican la ubicación de los objetos en la imagen. Además, el Head se encarga de predecir las categorías de los objetos detectados y de calcular la confianza de estas clasificaciones.

La interacción y el flujo de datos entre estos componentes se detallan en la Figura

XXIII, mostrando cómo cada parte contribuye al proceso general de detección y clasificación.

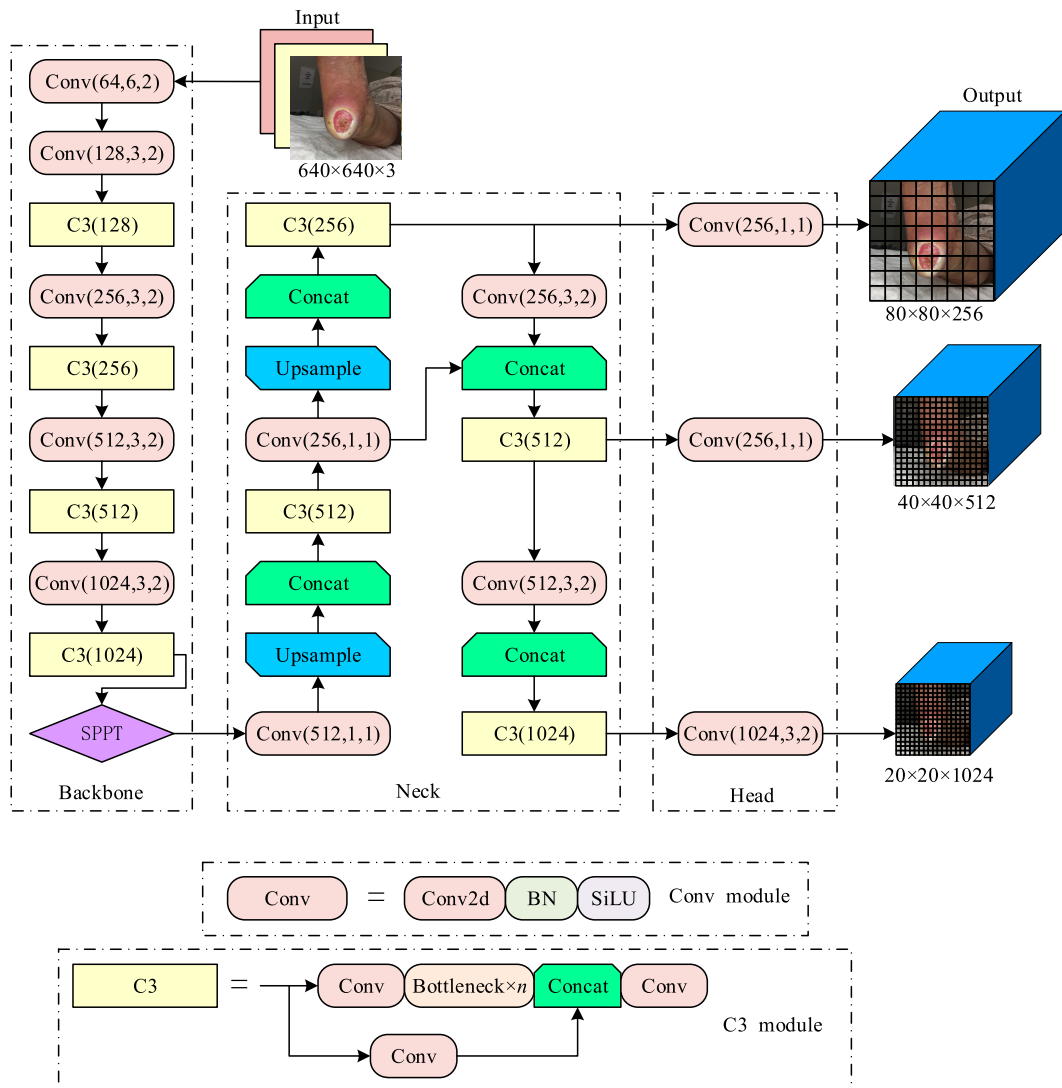


Figura XXIII: Arquitectura de YOLOv5 [59].

El detector YOLOv5, desarrollado por Ultralytics, es un modelo de detección de objetos en tiempo real que viene en varias variantes: YOLOv5s (small) para alta velocidad y bajo consumo de recursos, YOLOv5m (medium) que equilibra velocidad y precisión, YOLOv5l (large) para mayor precisión a costa de velocidad, y YOLOv5x (xlarge), que es el más preciso pero también el más lento y con mayor consumo de recursos. Estas variantes permiten su adaptación a diferentes necesidades y restricciones de hardware, desde dispositivos móviles hasta sistemas más potentes [60].

G. Segmentación de imágenes

La segmentación de imágenes es una técnica esencial en el procesamiento y análisis de imágenes, jugando un papel crucial en la identificación de objetos y fronteras (líneas, curvas, etc.) en imágenes. Su principal objetivo es dividir una imagen en segmentos o regiones para simplificar su representación y facilitar su análisis, lo cual es clave para la extracción y análisis precisos de objetos y regiones en una imagen [13]. Los métodos de segmentación de imágenes se pueden clasificar en varias categorías.

Este método implica seleccionar un valor de umbral para dividir los píxeles de una imagen en diferentes regiones. Es particularmente útil para imágenes con alto contraste entre el objeto y el fondo. La selección del umbral puede ser manual o automática, esta última a menudo involucrando métodos estadísticos o histogramas para determinar el valor óptimo de umbral [61].

Técnicas como K-means o Fuzzy C-means agrupan píxeles basándose en características como la intensidad del color o la textura. Estos algoritmos son efectivos para segmentar imágenes basadas en la similitud de colores o texturas en diferentes regiones de la imagen [62].

Este enfoque se enfoca en identificar los bordes de los objetos en una imagen detectando discontinuidades en la intensidad de los píxeles. Los algoritmos comunes incluyen el detector de bordes de Canny, Sobel, y Prewitt [63].

1. Segmentación basada en regiones

Este método agrupa píxeles adyacentes con propiedades similares, como el crecimiento de regiones o la división y fusión de regiones. Es efectivo para imágenes donde las regiones de interés tienen intensidades similares y se diferencian bien del fondo [64].

2. Superpíxeles en segmentación de imágenes

En el ámbito de la segmentación de imágenes mediante superpíxeles, se destaca el uso de algoritmos como SLIC para agrupar píxeles basándose en características como color y textura [19]. Este enfoque reduce la complejidad de la imagen, facilitando la extracción de vectores de características detallados de cada superpíxel. Estos vectores, que incluyen información sobre color, textura y posiblemente forma, son fundamentales para la clasificación posterior mediante algoritmos de aprendizaje supervisado, como Máquinas de Vectores de Soporte (SVM) o redes neuronales [37, 65]. Finalmente, la imagen se segmenta asignando a cada píxel la etiqueta de clasificación de su superpíxel correspondiente, lo que resulta en una identificación precisa y detallada de distintas regiones en la imagen. Este proceso constituye un método eficaz para la segmentación de imágenes, combinando técnicas avanzadas de agrupación, extracción de características y clasificación [66].

H. Limpieza y preparación de datos (data cleaning)

El proceso de limpieza de datos, es esencial en la ciencia de datos para asegurar la calidad y precisión del análisis posterior. Este procedimiento implica la detección y corrección (o eliminación) de errores e inconsistencias en los datos para mejorar su calidad [67]. Los métodos comunes incluyen la imputación de valores perdidos, la identificación y corrección de errores, y la armonización de las estructuras de datos [68]. La limpieza de datos no solo mejora la precisión de los resultados analíticos, sino que también es crucial para obtener modelos de aprendizaje automático robustos y confiables, como se discute en la literatura [69]. La relevancia de un conjunto de datos limpio y bien curado no puede ser subestimada, ya que los datos sucios o mal manejados pueden llevar a conclusiones erróneas y, en última instancia, a decisiones empresariales defectuosas [70].

I. Modelo de superresolución EDSR

El algoritmo Enhanced Deep Super-Resolution (EDSR), diseñado específicamente para mejorar la calidad de imágenes digitales, es un método avanzado en el campo del aprendizaje profundo. Su objetivo es incrementar la resolución de imágenes de baja calidad, transformándolas en versiones de alta resolución con detalles más nítidos y claros. EDSR está basado en una arquitectura de red neuronal profunda similar a SRResNet, pero con modificaciones que lo diferencian, especialmente en términos de eficiencia y resultados visuales [71]. La Figura XXIV muestra la arquitectura de este modelo.

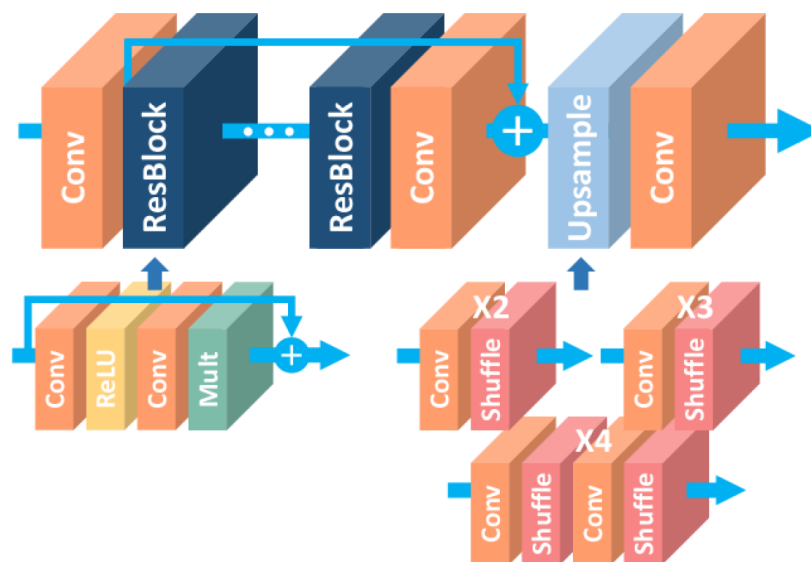


Figura XXIV: Arquitectura de la red de SR a escala única propuesta (EDSR) [71].

1. Modificaciones en la arquitectura residual

Una de las principales innovaciones de EDSR es la eliminación de las capas de normalización por lotes (*Batch Normalization*), que son comunes en redes profundas tradicionales como ResNet. En redes residuales clásicas, estas capas ayudan a estabilizar y acelerar el entrenamiento, pero en el contexto de superresolución de imágenes, pueden limitar la capacidad del modelo para producir imágenes detalladas y precisas.

En EDSR, la omisión de estas capas es una decisión estratégica que permite que los

valores de salida de cada capa convolucional tengan mayor flexibilidad, sin ser normalizados de manera constante a una distribución estándar. Esto es especialmente importante en la superresolución, donde la red necesita aprender una transformación precisa entre la imagen de baja resolución y su contraparte de alta resolución. Las capas de normalización por lotes tienden a cambiar las distribuciones de las características intermedias, lo que podría afectar negativamente la reproducción precisa de colores y detalles [71].

Además, la eliminación de estas capas también reduce significativamente el consumo de memoria GPU durante el entrenamiento, ya que las capas de normalización requieren la misma cantidad de memoria que las capas convolucionales previas [72]. Al evitar esta normalización, EDSR logra una mayor capacidad para aprender y mantener detalles sutiles y complejos en las imágenes de salida, asegurando que los detalles ampliados sean fieles al contenido original.

2. Bloques residuales

La estructura básica del modelo EDSR está compuesta por múltiples bloques residuales, que son importantes para el aprendizaje de la red. Cada bloque residual incluye dos capas convolucionales, y en lugar de simplemente apilar más capas, utiliza una conexión residual que suma la salida de las convoluciones con la entrada original del bloque, lo que permite que la red aprenda las diferencias (*residuos*) entre la imagen de baja resolución y la versión de alta resolución. Esto ayuda a preservar los detalles importantes de la imagen original mientras se genera una versión ampliada [71]. En términos matemáticos, cada bloque residual se puede describir como:

$$x_{\text{out}} = x_{\text{in}} + F(x_{\text{in}})$$

donde $F(x_{\text{in}})$ representa el resultado de las dos convoluciones aplicadas a la entrada

x_{in} .

3. Upsampling y expansión de la imagen

El modelo EDSR aumenta la resolución espacial de una imagen utilizando un proceso de upsampling o ampliación. En lugar de usar técnicas convencionales de interpolación (como el *bicúbico*), EDSR utiliza un enfoque basado en convoluciones y una técnica llamada *depth-to-space*, que reorganiza los canales de una imagen en el espacio, expandiendo su resolución [71].

El proceso de *depth-to-space* funciona aplicando convoluciones para aumentar el número de canales de la imagen de entrada, seguido de una reorganización que convierte esos canales adicionales en un incremento en las dimensiones espaciales (alto y ancho) de la imagen. Esta técnica permite que la red amplíe la imagen a una resolución más alta, por ejemplo 4x, sin perder calidad y preservando detalles importantes [72]. Este reordenamiento es clave para obtener una imagen de mayor resolución sin generar artefactos indeseados.

4. Normalización y preservación de los colores

Para garantizar que los colores de la imagen de salida sean consistentes con los de la imagen de entrada, el modelo aplica normalización al comienzo y al final del proceso. Al inicio, los valores de los píxeles de la imagen se normalizan a un rango de $[0, 1]$ utilizando una capa de *Rescaling*. Esto facilita el entrenamiento de la red y asegura que los valores numéricos sean manejables durante las operaciones convolucionales [72].

Una vez que la imagen ha sido procesada y ampliada, se aplica una segunda capa de *Rescaling* para devolver los valores a su rango original de $[0, 255]$, que es el estándar para imágenes en formato RGB. Adicionalmente, se emplea un proceso de *clipping* (recorte) para asegurar que los valores de los píxeles no se salgan de este rango, y se redondean los valores para obtener números enteros, lo que garantiza una representación adecuada de los colores en la imagen de salida [72].

5. Función de pérdida

Otra innovación clave en EDSR es el uso de la función de pérdida L1, o error absoluto medio, en lugar del error cuadrático medio (L2), que es más común en tareas de regresión. La pérdida L1 tiende a producir imágenes más nítidas y con mejor calidad visual, ya que penaliza grandes errores de manera lineal y no cuadrática, lo que evita suavizar demasiado los detalles en la imagen reconstruida [71]. Esto es particularmente importante en aplicaciones donde la nitidez es crucial, como en imágenes médicas, vigilancia o mejora de contenido multimedia.

6. Desafíos computacionales

A pesar de su eficacia, uno de los desafíos del modelo EDSR es su alta demanda de recursos computacionales debido al mayor número de parámetros, lo que puede dificultar su uso para imágenes de ultra alta resolución. Sin embargo, para imágenes de baja o media resolución, el modelo sigue siendo extremadamente efectivo y eficiente en términos de rendimiento [73].

J. Transformers en visión computacional

Los Transformers representan un cambio paradigmático en el procesamiento de secuencias, inicialmente en el campo del procesamiento del lenguaje natural. Un Transformer se basa en mecanismos de atención que permiten modelar dependencias sin importar la distancia en la secuencia de datos, una capacidad que lo hace particularmente adecuado para tareas donde el contexto y la relación entre elementos distantes son cruciales [74].

En el ámbito del procesamiento de imágenes, la adaptación de los Transformers para manejar datos visuales ha sido revolucionaria. Estos modelos, conceptualizados como secuencias de píxeles o parches de imágenes, emplean un enfoque novedoso, pues en lugar de trabajar directamente con píxeles individuales, los Transformers en visión por computadora operan sobre una serie de parches o bloques de una imagen, tratando cada

parche como un token en la secuencia, tal como se observa en la Figura XXV. El Transformer de Visión (ViT) ejemplifica esta técnica, dividiendo imágenes en parches y utilizando capas de atención múltiple para analizar su relevancia relativa [9]. A diferencia de las CNN, que se centran en características locales, los Transformers captan relaciones globales en la imagen, ofreciendo una comprensión más rica del contexto y las relaciones espaciales. Sin embargo, esta capacidad avanzada viene con la necesidad de extensos datos de entrenamiento y alta demanda computacional, especialmente en imágenes de alta resolución [7].

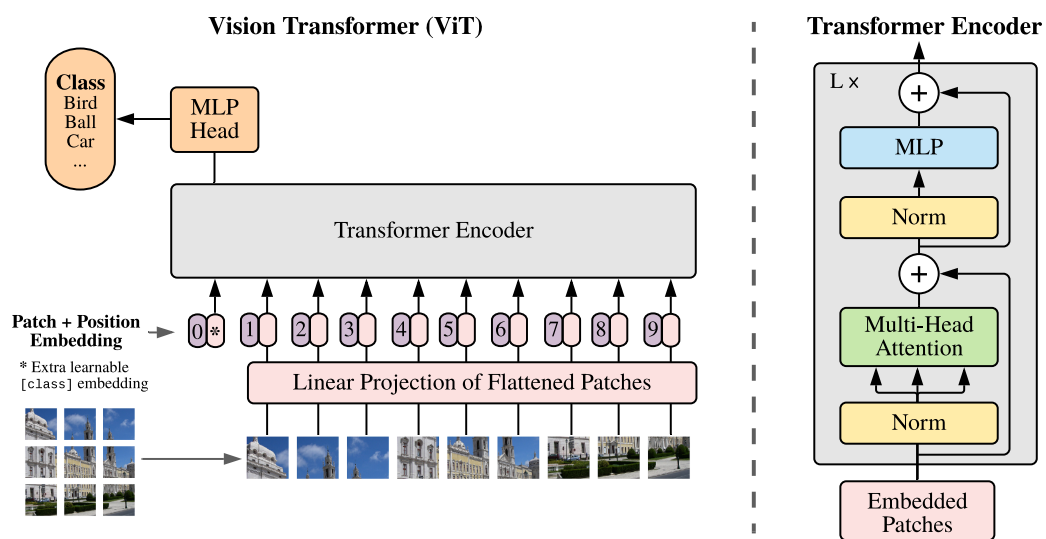


Figura XXV: Descripción del modelo basado en transformer para clasificación de imágenes [9].

1. Modelo SAM (segment anything model)

El modelo SAM se distingue en el campo de la visión por computadora como una solución avanzada y versátil para la segmentación de imágenes. Inspirado en técnicas de procesamiento del lenguaje natural (NLP), SAM traslada la idea de prompting (indicaciones o instrucciones) del NLP a la segmentación de imágenes. Esto implica usar prompts variados, como puntos de primer plano/fondo, cajas, texto libre o máscaras, para guiar la segmentación. Su capacidad para generar máscaras precisas incluso con prompts ambiguos es una de sus características más notables. Este enfoque innovador permite a SAM no solo segmentar objetos específicos de manera precisa, sino también adaptarse a una amplia gama de tareas

de segmentación, incluso aquellas desconocidas en el momento del diseño del modelo [10]. El modelo SAM se compone de varios elementos, como se ilustra en la Figura XXVI, esenciales que trabajan en conjunto para lograr una segmentación precisa y eficiente.

Codificador de imágenes : Este componente esencial emplea un Transformer de Visión pre-entrenado con MAE, optimizado para imágenes de alta resolución. Se puede configurar con variantes como ViT-H, ViT-L o ViT-B, según los requisitos de capacidad y escala. La elección de la variante influye en el rendimiento y la eficiencia, con ViT-H/16 siendo una opción poderosa para tareas exigentes. El codificador transforma la imagen en embeddings a una escala 16 veces menor, facilitando la tarea de segmentación avanzada. Esta reducción de escala es crucial para manejar eficientemente las entradas de alta resolución y las máscaras de segmentación complejas, especialmente en entornos con recursos computacionales limitados [10].

Codificador de prompts : Maneja prompts dispersos (puntos, cajas, texto) y densos (máscaras). Los prompts dispersos utilizan codificaciones posicionales y embeddings aprendidos, mientras que los densos se procesan mediante convoluciones para alinearse con el embedding de la imagen [10].

Decodificador de máscaras : Combina eficientemente el embedding de la imagen y los embeddings de los prompts, con auto-atención y atención cruzada, actualizando los embeddings con la información de los prompts en un diseño de dos capas [10].

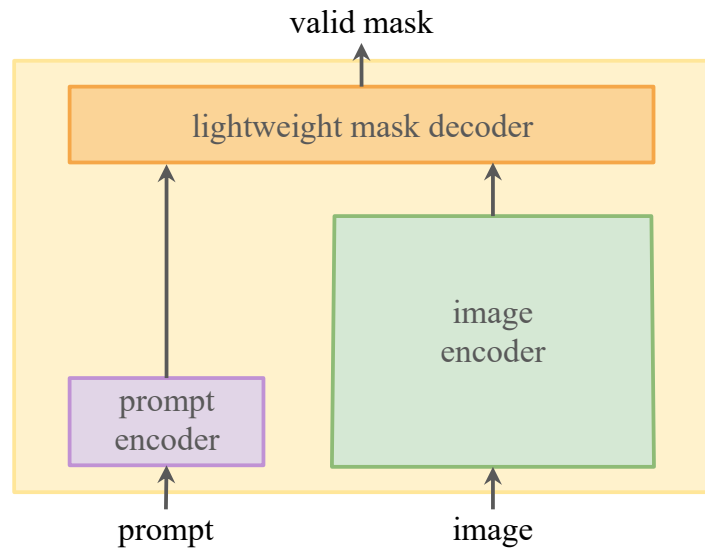


Figura XXVI: Arquitectura general del modelo SAM [10].

2. *Proceso de entrenamiento de SAM*

El entrenamiento de SAM, utilizando el conjunto de datos SA-1B con más de 1.1 mil millones de máscaras de segmentación, siguió un enfoque progresivo y adaptativo. Inicialmente, se implementó una etapa de anotación asistida donde anotadores profesionales etiquetaron máscaras en puntos de primer plano y fondo, utilizando herramientas interactivas avanzadas como pincel y borrador. Posteriormente, se avanzó a una fase semiautomática, en la que se aumentó la diversidad de las máscaras mediante el trabajo sobre imágenes pre-rellenadas con máscaras detectadas automáticamente, añadiendo 5.9 millones de máscaras más. La fase final del entrenamiento fue completamente automática, aprovechando las mejoras del modelo para generar de manera autónoma 1.1 mil millones de máscaras de alta calidad para todas las imágenes del conjunto de datos, este esquema de fases se ilustra en la Figura XXVII. Este proceso iterativo y detallado permitió a SAM desarrollar una capacidad sofisticada de adaptación a una amplia variedad de escenarios de segmentación, preparándolo para aplicaciones prácticas en el campo de la visión por computadora [10].

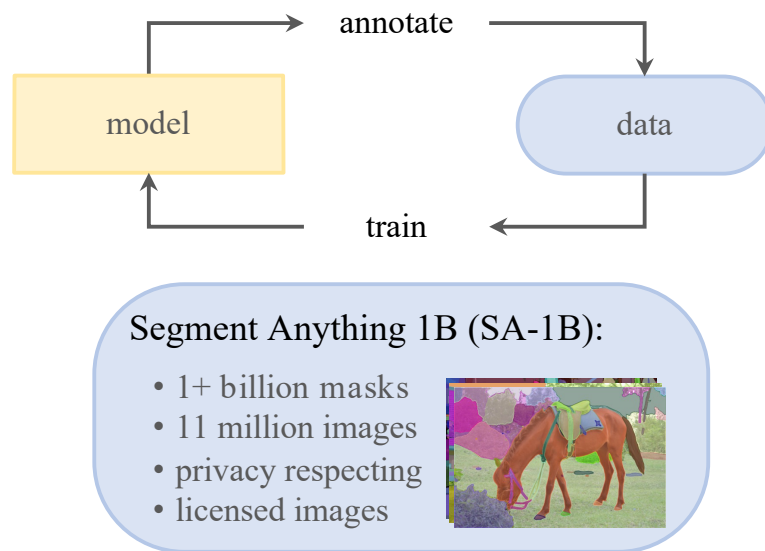


Figura XXVII: Metodología de entrenamiento y etiquetado automático de SAM [10].

3. *Prompting del modelo SAM*

SAM implementa un método de prompting inspirado en el procesamiento de lenguaje natural, procesando indicaciones como puntos, cajas delimitadoras o texto, vease la Figura XXVIII. Este enfoque permite a SAM interpretar y responder a diferentes prompts con precisión, generando máscaras de segmentación detalladas para objetos específicos. La versatilidad de SAM para adaptarse a variados prompts abre un amplio espectro de posibilidades en la segmentación de imágenes [10].

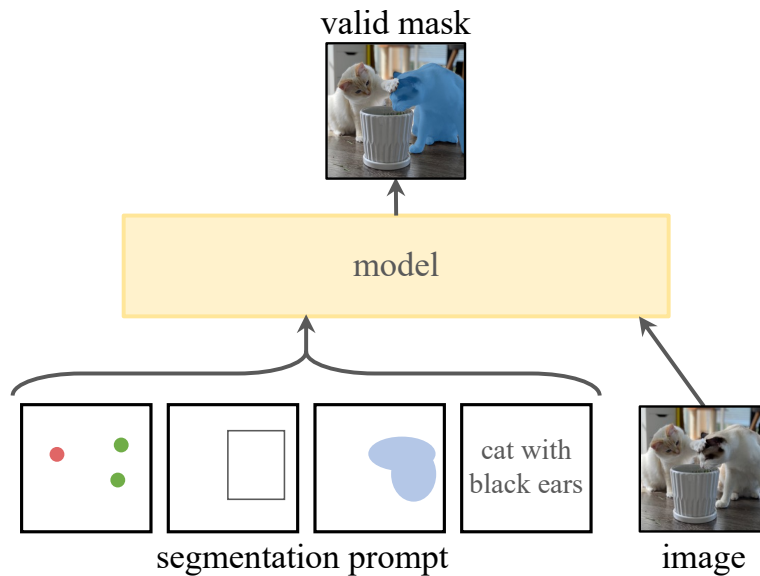


Figura XXVIII: Aplicación del modelo SAM en la segmentación de imágenes mediante prompts [10].

4. Limitaciones

SAM es un modelo versátil para el procesamiento de imágenes, pero presenta varias limitaciones. Aunque eficaz en general, tiende a fallar en la detección de estructuras finas y puede generar componentes desconectados, además de no definir los bordes con la precisión de métodos más avanzados. No supera a los métodos de segmentación interactiva especializados, especialmente en tareas que requieren alta precisión en la Intersección sobre la Unión (IoU). Aunque es capaz de procesar instrucciones rápidamente, su rendimiento disminuye con codificadores de imágenes pesados, y su enfoque en la conversión de texto a máscara sigue siendo experimental. Además, enfrenta desafíos en la implementación de segmentaciones semánticas y panópticas con instrucciones simples y es probable que sea superado por herramientas especializadas en dominios específicos [10].

K. Métricas de desempeño

En la evaluación de modelos de inteligencia artificial mediante la matriz de confusión, se distinguen cuatro elementos esenciales: Verdaderos Positivos (VP), cuando el modelo acierta identificando la presencia de una clase; Falsos Positivos (FP), indicando erróneamente

la presencia de una clase; Verdaderos Negativos (VN), acertando en predecir la ausencia de una clase; y Falsos Negativos (FN), fallando en detectar una clase presente. Esta matriz, como se muestra en la Tabla II. Su utilidad reside en identificar confusiones entre clases y en revelar sesgos en clasificadores, especialmente cuando la distribución de muestras es desigual. Esto permite evaluar no solo la precisión general del modelo, sino también su efectividad y limitaciones en diferentes escenarios [75].

Tabla II: Matriz de confusión.

	Clase Real	
Clase Predicha	Positiva (P)	Negativa (N)
Positiva (P)	Verdadero Positivo (VP)	Falso Positivo (FP)
Negativa (N)	Falso Negativo (FN)	Verdadero Negativo (VN)

A partir de esta matriz, se derivan varias métricas:

1. *Especificidad*

La Especificidad es una métrica que mide la habilidad del modelo para identificar correctamente casos negativos. Es crucial para entender cómo el modelo maneja las instancias que no pertenecen a la clase objetivo [76].

$$\text{Especificidad} = \frac{VN}{FP + VN}$$

2. *Precisión*

La Precisión se refiere a la exactitud con la que el modelo predice la clase positiva. Una alta precisión indica que el modelo es eficaz en identificar correctamente los casos positivos [77].

$$\text{Precisión} = \frac{VP}{VP + FP}$$

3. *Exactitud*

La Exactitud mide la proporción total de predicciones correctas (positivas y negativas) del modelo, proporcionando una visión general de su rendimiento [77].

$$\text{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN}$$

4. *Sensibilidad o Recall*

La Sensibilidad, o Recall, evalúa la capacidad del modelo para detectar todos los casos relevantes de la clase positiva [76]. Es una métrica importante para modelos donde es crucial no perder ningún caso positivo .

$$\text{Sensibilidad} = \frac{VP}{VP + FN}$$

5. *Medida-F*

La Medida-F es una métrica que combina la Precisión y la Sensibilidad, ofreciendo un balance entre ambas [77]. Es especialmente útil en situaciones con clases desbalanceadas.

$$\text{Medida-F} = \frac{2 \cdot VP}{2 \cdot VP + FP + FN}$$

6. *Curvas ROC*

Las curvas ROC (Receiver Operating Characteristic) son herramientas gráficas fundamentales utilizadas para evaluar la calidad de los sistemas de clasificación binaria. Representan gráficamente la relación entre la tasa de verdaderos positivos (Sensibilidad) y la tasa de falsos positivos (1-Especificidad) a diferentes umbrales de decisión. Su importancia radica en su capacidad para mostrar cómo la capacidad de discriminación de un modelo varía con el cambio en el umbral, proporcionando así una comprensión integral de su rendimiento. Las curvas ROC son particularmente útiles para comparar diferentes modelos de clasificación y para elegir el umbral más adecuado para la clasificación [76].

7. Intersection over union (IoU)

La métrica Intersection over Union (IoU), también conocida como índice Jaccard, es fundamental en la evaluación de algoritmos de segmentación y detección de objetos en visión por computadora. Representa la relación entre la intersección y la unión de dos conjuntos, típicamente el área del objeto detectado y el *ground truth*. Se calcula como:

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

donde A es el área del *bounding box* predicho y B es el área del *bounding box* del *ground truth*. Un IoU alto indica una mayor precisión en la detección o segmentación. Esta métrica es ampliamente reconocida por su claridad y efectividad, como se destaca en desafíos como el Pascal VOC [78].

8. Coeficiente de Dice

El Coeficiente de Dice es otra métrica importante en la evaluación de la precisión en segmentación de imágenes y detección de objetos. Similar al IoU, este coeficiente mide la similitud entre dos conjuntos, pero pone un énfasis mayor en la intersección al duplicar su valor en el numerador:

$$Dice = \frac{2|A \cap B|}{|A| + |B|}$$

donde A representa el conjunto de píxeles del *bounding box* predicho y B el conjunto de píxeles del *bounding box* del *ground truth*. Un valor de Dice cercano a 1 indica una alta precisión en la superposición entre la predicción y la verdad fundamental. Debido a su sensibilidad a la intersección, es especialmente útil en situaciones donde las clases están desbalanceadas [30].

9. Mean average precision (mAP)

En el contexto de detección de objetos, se fundamenta matemáticamente en el uso del Índice de Superposición entre Áreas (IoU). El IoU compara el área de superposición y el área de unión entre el bounding box predicho y el real, sirviendo como indicador de exactitud de la localización de objetos. Una detección se considera precisa si el IoU supera un umbral determinado; por ejemplo, un $\text{IoU} \leq 0.3$ implica una detección inexacta. El mAP se calcula promediando la Precisión Media (AP) para diferentes clases y para varios umbrales de IoU, típicamente en el rango de 0.5 a 0.95, incrementando en pasos de 0.05. Esta medida integral proporciona una evaluación exhaustiva del rendimiento del modelo en términos de precisión de detección en varios niveles de rigurosidad [56].

L. Gasto computacional en modelos de aprendizaje automático

El análisis de FLOPs (Floating Point Operations per Second) es una medida esencial para comprender y evaluar el gasto computacional en modelos de aprendizaje automático. Las FLOPs indican el número de operaciones de punto flotante que un sistema informático puede realizar en un segundo, lo cual es un indicador de la intensidad computacional de un modelo [79].

En el contexto del aprendizaje automático, un modelo que realiza un número elevado de FLOPs generalmente implica una mayor complejidad computacional. Esto se debe a que los modelos más avanzados, especialmente en el campo del aprendizaje profundo, requieren una cantidad significativa de cálculos de punto flotante para tareas como la multiplicación de matrices y la convolución en redes neuronales [67]. Por lo tanto, un alto número de FLOPs puede ser indicativo de un modelo más complejo y potencialmente más preciso, pero también implica un mayor consumo de recursos computacionales y energéticos [80].

La optimización de FLOPs, por lo tanto, se convierte en una tarea crítica para mejorar la eficiencia de los modelos de aprendizaje automático. Reducir el número de FLOPs

necesarios para realizar una tarea determinada puede llevar a un menor uso de recursos, tiempos de entrenamiento más cortos y, en última instancia, a una mayor viabilidad de los modelos en entornos con recursos limitados [81]. Además, en el ámbito del aprendizaje automático, se ha demostrado que la optimización de FLOPs puede contribuir a la reducción de la huella de carbono, lo que es cada vez más importante en la era de la sostenibilidad y la informática ecológica [82].

Se emplea la siguiente fórmula estándar para calcular las operaciones en capas convolucionales, expresadas en FLOPs:

$$\text{FLOPs} = 2 \times K^2 \times N \times O^2 \times C \quad (10)$$

Donde:

K = Tamaño del filtro (kernel)

N = Número de filtros

O = Tamaño de salida (dimensión espacial)

C = Número de canales de entrada

Esta fórmula estima el número total de multiplicaciones y sumas requeridas para una operación de convolución, siendo un componente fundamental en el cálculo del gasto computacional de redes neuronales convolucionales [30].

III. Desarrollo del sistema de segmentación

La aparición de modelos de segmentación avanzados como SAM ha representado un avance significativo en el campo de visión computacional. Este modelo se adapta a nuevas distribuciones de imágenes y tareas utilizando una variedad de prompts, como puntos de primer plano/fondo y cajas delimitadoras, para generar máscaras de segmentación [10].

La integración del modelo SAM con técnicas tradicionales de aprendizaje automático marca un avance novedoso en la segmentación de imágenes. Este método propuesto se fundamenta en una combinación del modelo de detección YOLOv5 y métodos clásicos de clasificación de machine learning, aprovechando así la flexibilidad y eficacia del modelo SAM. Para enriquecer este enfoque, implementamos una estrategia de doble prompt, que consiste en la utilización de puntos ubicados tanto dentro como fuera del área de interés, en conjunto con los cuadros delimitadores generados por YOLOv5.

Para identificar los puntos que pertenecen a la lesiones o no, implementamos la técnica de segmentación por superpíxeles SLIC con algoritmos de clasificación binaria de machine learning. La estrategia se enfoca en el entrenamiento de un modelo de regresión logística, diseñado específicamente para evaluar y seleccionar los cinco superpíxeles con la mayor probabilidad de corresponder a una lesión, así como otros diez con la menor probabilidad. Los puntos de interés, que son los centroides de estos superpíxeles, se convertirán en los inputs para el modelo SAM.

En la Figura XXIX, se ilustra el flujo completo de integración del modelo SAM con técnicas tradicionales de aprendizaje automático para la detección y segmentación de lesiones. El proceso comienza con la utilización del modelo YOLOv5, que detecta la lesión en la imagen y genera un cuadro delimitador alrededor de la región de interés. A partir de esta región, se mejora la resolución de la imagen utilizando un modelo de superresolución (EDSR), lo que permite obtener características mas precisas. Posteriormente, se aplica una segmentación en superpíxeles, dividiendo la imagen en pequeñas regiones homogéneas.

Para cada superpíxel, se calcula un vector de características que incluye estadísticas como la media, la varianza y el centroide, basado en los mapas de características de VGG16 y los canales RGB. Estos vectores son analizados mediante un modelo de regresión logística, que clasifica cada superpíxel como perteneciente a piel sana o lesión. Los puntos seleccionados a partir de esta clasificación, junto con las cajas delimitadoras y la imagen mejorada, se utilizan como entrada para el modelo SAM. Finalmente, el modelo SAM genera una máscara de segmentación precisa, refinando la detección de la lesión. Este enfoque híbrido propone aprovechar la eficiencia de detección de YOLOv5 y la sofisticación de segmentación del modelo SAM.

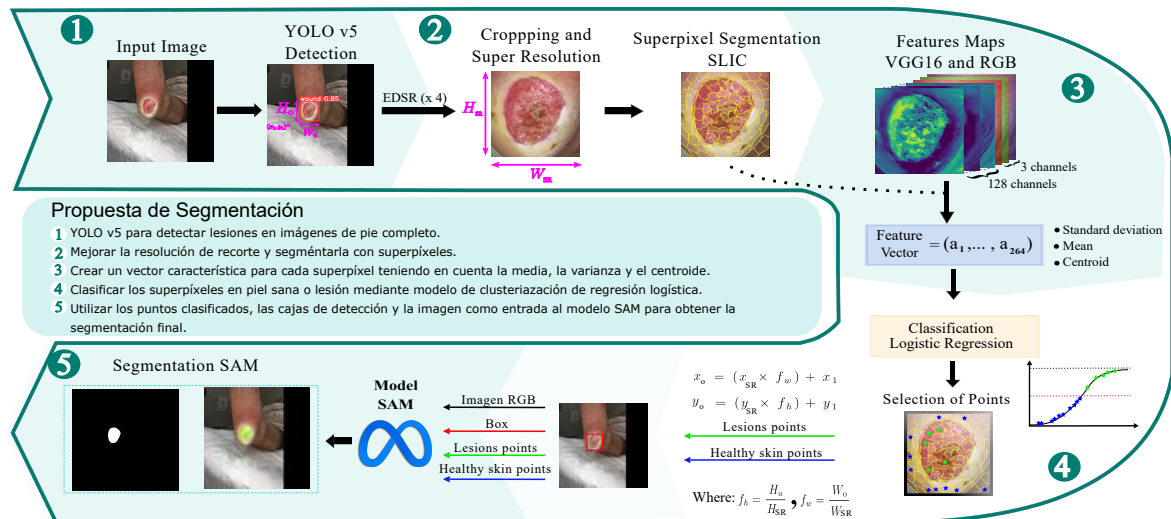


Figura XXIX: Flujo de trabajo para la detección y segmentación de heridas combinando YOLOv5, SLIC, Regresión Logística y el Modelo SAM.

A. Recursos de desarrollo

El proyecto fue desarrollado utilizando Python como lenguaje de programación, elegido por su amplia gama de herramientas y bibliotecas diseñadas específicamente para aplicaciones de inteligencia artificial. Las principales bibliotecas utilizadas fueron scikit-learn, torch y keras, que son esenciales para implementar y manejar modelos avanzados de aprendizaje automático y redes neuronales.

En cuanto al hardware, se utilizó una laptop Dell Alienware M15 R5 con un procesador

Ryzen 9, 16 GB de RAM, 1 TB de almacenamiento SSD y una tarjeta gráfica NVIDIA RTX 3070, lo cual fue adecuado para el procesamiento de imágenes y el desarrollo de algoritmos. No obstante, para el entrenamiento específico del modelo YOLOv5, se optó por Google Colab Pro, aprovechando su acceso a recursos computacionales avanzados como la GPU Tesla T4 y un procesador Intel(R) Xeon(R). Esta elección se basó en la superior capacidad de procesamiento y memoria de la Tesla T4, ideal para aprendizaje automático y tareas de AI, en contraste con la RTX 3070, más enfocada en gráficos y juegos. Así, el uso de Google Colab Pro facilitó el entrenamiento del modelo YOLOv5.

Los algoritmos de inteligencia artificial utilizados para el proyecto son reconocidos en la literatura académica, disponibles a través de librerías especializadas. Para la implementación del modelo YOLO v5, se recurrió al código fuente y a las guías de implementación disponibles en su repositorio oficial en GitHub. Para la integración del algoritmo de regresión logística, se utilizó la librería scikit-learn, reconocida por su robustez y flexibilidad en análisis de datos. Respecto a la segmentación de imágenes, se implementó el algoritmo de superpíxeles mediante la herramienta SLIC de skimage, y el modelo VGG16 fue incorporado utilizando Keras. Finalmente, para la segmentación avanzada, se empleó el algoritmo SAM, obtenido del repositorio 'segment-anything' de Facebook Research, disponible en GitHub. Cada uno de estos componentes fue seleccionado para alinearse con los objetivos específicos del proyecto.

B. Dataset

La búsqueda de bases de datos médicas enfocadas en el pie diabético no es una tarea sencilla, ya que este tipo de datos requiere cumplir con estrictas restricciones éticas y legales. Además, no solo es necesario que las imágenes estén disponibles, sino que también deben estar adecuadamente etiquetadas por especialistas, cumpliendo con todos los estándares de calidad y ética necesarios.

En el trabajo desarrollado por Zhang et al. [6], se presenta un estado del arte de los

métodos de clasificación, detección y segmentación de lesiones en general. En este estudio, se reportan diversas bases de datos utilizadas en investigaciones, las cuales se resumen en la Tabla III. Entre estas, solo dos bases de datos están específicamente enfocadas en lesiones del pie diabético, lo que resalta la limitada disponibilidad de recursos especializados para este tipo de patología.

Tabla III: Resumen de bases de datos de lesiones, incluyendo tipos, tamaños y si están etiquetadas [6].

Dataset	Wound type	Size	Labeled
Medetec dataset	Mixed	-	No
BIP_US database	Burn	94	Yes
FUSeg dataset	Diabetic foot ulcer	1210	Yes
Sårwebben	Mixed	-	No
Chronic wound database	Chronic wound	188	Yes
AHZ dataset	Diabetic foot ulcer	1109	Yes
AHZ&UWM dataset	Mixed	538	Yes

Como se observa en la Tabla III, las bases de datos FUSeg y AHZ están específicamente diseñadas para el estudio de úlceras del pie diabético y cumplen con los requisitos de etiquetado por especialistas.

La base de datos FUSeg, utilizada en este estudio, fue creada por el Centro de Heridas y Vasculares Advancing the Zenith of Healthcare en Milwaukee, WI, y es presentada en detalle en el trabajo de Wang et al. [83]. Las imágenes fueron capturadas utilizando una cámara Canon SX 620 HS y un iPad Pro, asegurando tomas en primeros planos con iluminación adecuada para resaltar los detalles relevantes de las lesiones. Este conjunto incluye 810 imágenes de entrenamiento, 200 de validación y 200 de prueba, todas acompañadas de sus respectivas máscaras de segmentación, como se ilustra en la Figura XXX. El proceso de anotación inicial fue llevado a cabo por estudiantes de posgrado mediante la herramienta *VGG Image Annotator (VIA)*, y las anotaciones fueron posteriormente refinadas por especialistas clínicos bajo la supervisión del Dr. Jeffrey Niezgoda, MD. Como

parte del preprocesamiento, todas las imágenes fueron ajustadas mediante un relleno de ceros (*zero-padding*) para estandarizar su resolución a 512x512 píxeles.

Una de las principales ventajas del conjunto FUSeg es su acceso público, lo que facilita su uso por cualquier investigador sin restricciones. Por ello, este estudio se realizó exclusivamente con esta base de datos, que además fue utilizada en el Challenge [84], donde se convirtió en una referencia para la evaluación de modelos de segmentación de úlceras del pie diabético. En este desafío, el trabajo desarrollado por Mahbod et al. [85] alcanzó el primer premio, destacando la relevancia de FUSeg en investigaciones relacionadas con la segmentación de lesiones.

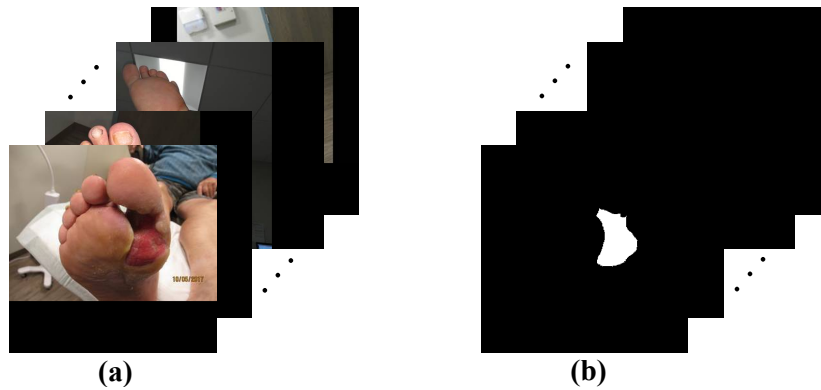


Figura XXX: (a) Ejemplo de imagen de entrenamiento con resolución de 512x512 píxeles. (b) Máscara de segmentación correspondiente, mostrando las áreas de interés etiquetadas.

C. Entrenamiento del modelo de detección YOLOv5

En esta sección, describiremos el proceso necesario para el entrenamiento del modelo YOLO v5, específicamente la variante YOLOv5s, elegido por su equilibrio entre eficiencia y precisión en la detección de objetos en tiempo real.

1. *Preprocesamiento*

El entrenamiento de un modelo de segmentación o detección, como YOLOv5, requiere una preparación de los datos para garantizar la compatibilidad con su formato

de entrada. En este caso, fue necesario estructurar los datos en carpetas separadas: una para las imágenes de entrenamiento y otra para las etiquetas correspondientes. Cada imagen debe estar asociada con un archivo de texto (.txt) que contenga la descripción de los cuadros delimitadores en el formato requerido por YOLOv5. Este formato incluye cinco elementos por línea: el primer elemento indica la clase del objeto, el segundo y tercero representan las coordenadas normalizadas (X , Y) del centro del cuadro delimitador, y el cuarto y quinto corresponden al ancho y la altura del cuadro delimitador, también normalizados con respecto al tamaño de la imagen, como se ilustra en la Figura XXXI.

En el caso del conjunto de datos FUSeg, las etiquetas originales se presentan en forma de máscaras de segmentación, donde los píxeles blancos indican las regiones de la lesión y los píxeles negros corresponden al fondo. Para adaptar estas etiquetas al formato de YOLOv5, se diseñó un algoritmo que procesa las máscaras y genera automáticamente los cuadros delimitadores. Este algoritmo localiza los límites superior, inferior, izquierdo y derecho de la región de interés analizando los píxeles blancos en la máscara. El límite superior se define como la primera fila que contiene al menos un píxel blanco, mientras que el límite inferior corresponde a la última fila con píxeles blancos. De manera análoga, los límites izquierdo y derecho se determinan buscando la primera y última columna con píxeles blancos, respectivamente.

Una vez identificados los límites de la región de interés, el algoritmo expande estos bordes en un 5 % del tamaño total de la imagen en cada dirección. Esta expansión asegura que el cuadro delimitador no solo abarque la lesión, sino también una zona periférica adyacente, proporcionando un contexto adicional que puede ser relevante para el análisis. Finalmente, las coordenadas de los cuadros delimitadores se normalizan dividiendo cada valor por las dimensiones de la imagen (ancho y altura) para cumplir con los requisitos del formato YOLOv5. Esto genera una etiqueta en el formato adecuado para cada imagen, donde las coordenadas del cuadro están completamente adaptadas al sistema de entrada del modelo.

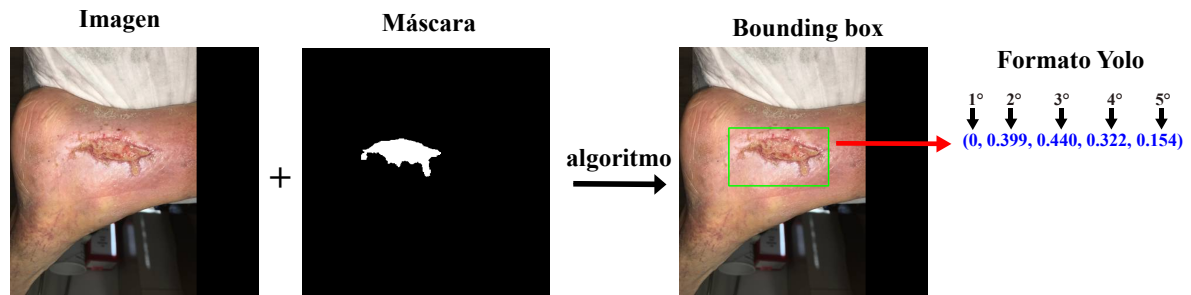


Figura XXXI: Creación de la etiqueta en formato YOLOv5.

2. Entrenamiento

Para el entrenamiento del modelo YOLOv5, preparamos los datos, asegurando que cada imagen en el conjunto de entrenamiento estuviera emparejada con su correspondiente archivo txt. Estos archivos contienen la información necesaria para la detección de objetos. Además, con el fin de facilitar el proceso de entrenamiento y aprovechar la integración entre plataformas de Google, todas las imágenes y archivos de texto se subieron a Google Drive. Esta configuración permitió utilizar Google Colab como plataforma de entrenamiento, beneficiándose de su acceso directo a los datos almacenados en Drive. La distribución de las imágenes para las fases de entrenamiento, validación y prueba se detalla en la Tabla IV, proporcionando una visión clara de la organización y estructura del conjunto de datos utilizado.

Tabla IV: Distribución de imágenes para entrenamiento de YOLOv5.

Dimensión	Train	Validación	Total
-	810	200	1010

El entrenamiento del modelo YOLOv5 se llevó a cabo a lo largo de 200 épocas, con un tamaño de lote de 64. Sin embargo, el proceso finalizó en la época 169 gracias a la implementación de un criterio de detención temprana (*early stopping*). Este mecanismo monitorea las métricas de validación y detiene automáticamente el entrenamiento cuando no se observa una mejora significativa durante un número predefinido de épocas consecutivas, optimizando así los recursos computacionales y evitando problemas de sobreajuste. En

total, el entrenamiento se completó en un tiempo de 51 minutos y 24 segundos, utilizando el optimizador *descenso de gradiente estocástico* (SGD) con las configuraciones predeterminadas del modelo.

La Figura XXXII ilustra la evolución de las curvas de precisión y sensibilidad (*recall*) durante el entrenamiento. Se observa un rápido incremento en ambas métricas durante las primeras 25 épocas, seguido de una estabilización en valores cercanos a 0.9, lo que refleja una convergencia eficiente del modelo. De manera complementaria, las curvas de pérdida (*loss*) para las cajas delimitadoras, tanto en el conjunto de entrenamiento como en el de validación (representadas en líneas punteadas), muestran una disminución consistente hasta alcanzar valores mínimos estables. Este comportamiento indica que el modelo logró un equilibrio adecuado entre el aprendizaje de características relevantes y su capacidad de generalización, evitando un ajuste excesivo a los datos de entrenamiento.

Hiperparámetros del entrenamiento

- **Épocas de calentamiento (*warmup epochs*):** El número de épocas de calentamiento se estableció en 3, durante las cuales la tasa de aprendizaje se incrementó progresivamente para evitar fluctuaciones bruscas en las primeras iteraciones.
- **Tasa de aprendizaje (*learning rate*, *lr0*):** La tasa de aprendizaje inicial fue de 0.01, ajustándose dinámicamente mediante un planificador de tasa de aprendizaje lineal (*scheduler*). Esta tasa se incrementó gradualmente durante las primeras épocas (*warmup*) para estabilizar el entrenamiento.
- **Momentum:** Se utilizó un valor de 0.937, ayudando a mantener la dirección de los gradientes a lo largo del proceso de optimización.
- **Pesos preentrenados (*weights*):** El modelo fue inicializado utilizando los pesos preentrenados *yolov5s.pt*, que corresponden a la versión más ligera de los modelos

YOLOv5. Estos pesos se descargaron automáticamente si no estaban presentes en el sistema local.

- **Decaimiento de los pesos (*weight decay*):** Se aplicó un valor de 0.0005 para el *weight decay*, lo que reduce la magnitud de los pesos y ayuda a regularizar el modelo, minimizando el riesgo de sobreajuste.

Funciones de activación

- **Leaky ReLU:** Se aplicó la función de activación *Leaky ReLU* en las capas ocultas del modelo. Esta función permite que los valores negativos de activación pasen con un pequeño valor (*leak*), evitando que los nodos queden inactivos.
- **Sigmoide:** En la capa de detección final, se utilizó la función de activación sigmoide para calcular las probabilidades de detección de los objetos.

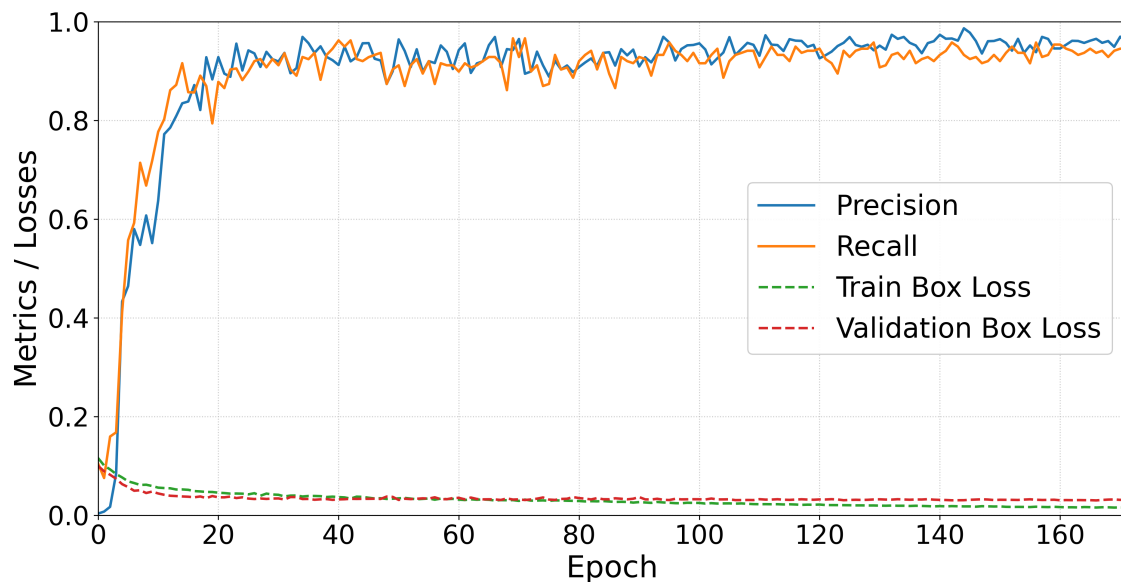


Figura XXXII: Evolución de la precisión, *recall* y pérdidas de las cajas delimitadoras durante el entrenamiento.

Por su parte, la Tabla V resume las métricas de desempeño alcanzadas por el modelo YOLOv5 durante las pruebas. Entre los indicadores clave se incluyen la sensibilidad,

la precisión, mAP@.5 y mAP@.95. En particular, el modelo alcanzó una sensibilidad de 0.966 y una precisión de 0.895, lo que demuestra su capacidad para identificar correctamente las lesiones minimizando los falsos positivos. Asimismo, las métricas mAP@.5 (0.972) y mAP@.95 (0.61) ofrecen una visión detallada de la precisión promedio en diferentes umbrales de intersección sobre unión (*IoU*), destacando el desempeño robusto del modelo en la localización precisa de las lesiones.

Tabla V: Métricas de desempeño del algoritmo YOLOv5.

Métodos	Sensibilidad	Precisión	mAP@.5	mAP@.95
YOLOv5	0.966	0.895	0.972	0.61

D. Sistema de segmentación de lesiones

Para el desarrollo del sistema propuesto, se utilizó el modelo de segmentación SAM, que emplea como entrada *bounding boxes* (BBoxes) y puntos marcados como pertenecientes al plano (lesión) o al fondo (piel sana) para segmentar una imagen. Los BBoxes fueron generados previamente por el modelo YOLOv5, y a partir de ellos se identificaron cinco puntos dentro de la región de la lesión y diez puntos fuera de ella. Para seleccionar estos puntos, se implementó una técnica de segmentación basada en superpíxeles, que agrupa píxeles en clústeres homogéneos según su similitud en color y proximidad espacial, reduciendo la complejidad de la imagen y dividiéndola en regiones con características similares. Los centroides de los superpíxeles clasificados como pertenecientes a la lesión se utilizaron para determinar las coordenadas de los puntos dentro de ella, mientras que los centroides de los superpíxeles clasificados como fondo proporcionaron los puntos fuera de la lesión. La Figura XXXIII muestra cómo la segmentación basada en superpíxeles organiza las regiones homogéneas, facilitando la posterior segmentación mediante el modelo SAM.

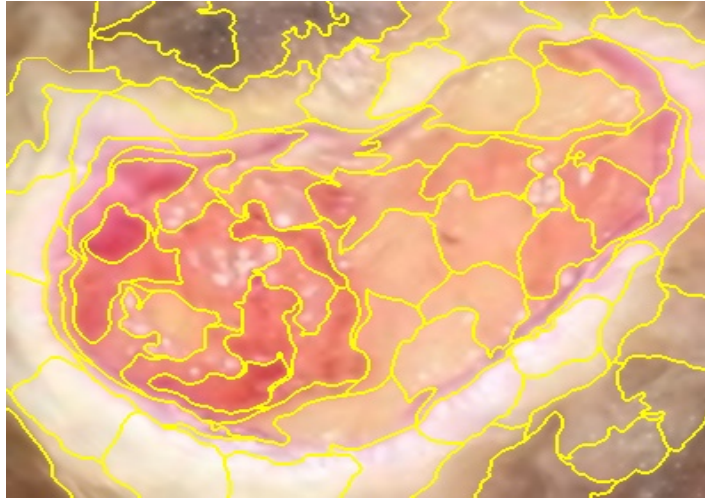


Figura XXXIII: Superpíxeles generados utilizando el método SLIC para la segmentación de imágenes.

Una vez generados los superpíxeles, se construye un vector de características para cada uno, basado en propiedades clave como color, textura y forma. Estos vectores se utilizan como entrada para entrenar un modelo de clasificación basado en regresión logística, que asigna a cada superpíxel una probabilidad de pertenecer o no a una lesión. A partir de esta clasificación, los centroides de los superpíxeles identificados como pertenecientes a la lesión se utilizan para determinar las coordenadas de los puntos dentro de la misma. De manera similar, los centroides de los superpíxeles clasificados como no pertenecientes a la lesión proporcionan las coordenadas de los puntos fuera de ella.

En el ámbito médico, los superpíxeles han demostrado ser una herramienta efectiva en tareas como la detección de cáncer de piel [86], la segmentación de tejidos en úlceras dermatológicas [87] y la identificación de lesiones cutáneas [3]. Estas aplicaciones evidencian la utilidad de los superpíxeles para abordar problemas similares en contextos de análisis médico.

No obstante, la baja resolución de las imágenes disponibles representa un desafío importante para la implementación efectiva de esta técnica. Las imágenes suministradas por el conjunto de datos FUSeg tienen una resolución base de 512 x 512 píxeles, lo cual se ve aún más limitado al obtener recortes (*crops*) de las regiones de interés mediante YOLOv5.

Estos recortes de baja resolución carecen de detalles suficientes, afectando la precisión de los métodos de generación de superpíxeles. Como resultado, la segmentación es menos efectiva, dificultando la identificación precisa de las regiones relevantes, tal como se ilustra en la Figura XXXIV(a).

1. Superresolución con el método EDSR

Para abordar el desafío de la baja resolución en los recortes (crops) generados por YOLOv5, se implementó la Red de Superresolución Profunda Mejorada (Enhanced Deep Super-Resolution Network, EDSR) presentado en el trabajo de **paper** 13. Este modelo ha demostrado ser altamente eficiente y preciso al aumentar la resolución de imágenes mientras preserva las características originales, como bordes y texturas. En este proyecto, el modelo fue configurado para incrementar la resolución de los recortes en un factor de 4, lo que permite un análisis más detallado de las lesiones, tal como se muestra en la Figura XXXIV (b). A continuación, se describe el flujo del proceso en detalle.

Preparación de las imágenes: Las imágenes y máscaras fueron cargadas desde sus respectivos directorios y convertidas de su formato original BGR al formato RGB requerido por el modelo EDSR. Este paso asegura la compatibilidad con los estándares de entrada del modelo. Posteriormente, cada imagen se transformó en matrices tridimensionales.

Configuración del modelo: El modelo preentrenado EDSR_x4.pb, diseñado para escalar imágenes por un factor de 4, fue cargado utilizando el módulo de superresolución DnnSuperResImpl de OpenCV. Este modelo realiza un escalado tanto en ancho como en alto, preservando los detalles estructurales y visuales de las regiones de interés. La configuración del modelo incluyó la asignación explícita del factor de escalado.

Aplicación de superresolución: Cada imagen y su máscara correspondiente fueron procesadas de manera iterativa mediante el modelo EDSR. Este procedimiento generó versiones ampliadas de las imágenes de entrada, incrementando significativamente su resolución y mejorando la definición de bordes y texturas. Las imágenes ampliadas

y las máscaras resultantes se almacenaron en directorios separados para mantener la organización de los datos y facilitar su análisis posterior.

Validación del proceso: Una vez completada la etapa de superresolución, se verificaron las dimensiones de las imágenes y máscaras ampliadas, asegurando que fueran consistentes entre sí. Este paso es crucial para garantizar la integridad de los datos y su compatibilidad con los métodos de segmentación subsiguientes.

Los detalles específicos sobre el modelo EDSR se detallan en la sección I.

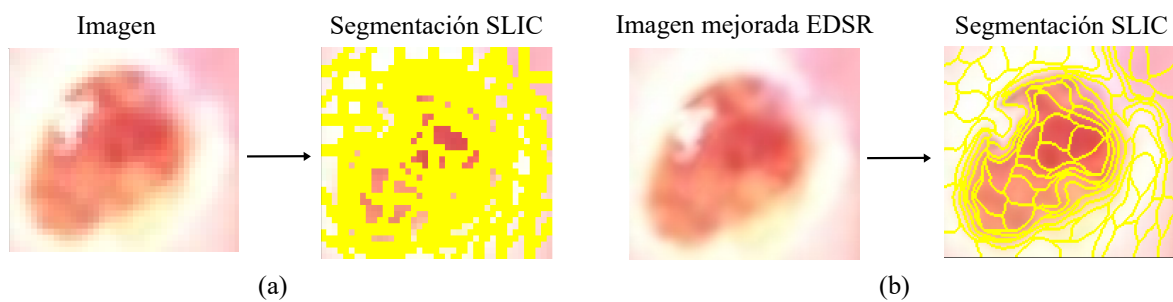


Figura XXXIV: (a) Imagen inicial y extracción de características de baja calidad, (b) Mejora con EDSR y extracción avanzada.

2. *Generación de superpíxeles:*

En este proyecto, se emplea el método SLIC (Simple Linear Iterative Clustering) para la generación de superpíxeles debido a su capacidad para equilibrar la eficiencia computacional con la precisión en la segmentación. Este método agrupa píxeles en regiones homogéneas considerando tanto su proximidad espacial como su similitud espectral, logrando una representación coherente de los patrones presentes en la imagen [19].

El procedimiento comienza procesando los recortes proporcionados por YOLO, los cuales son mejorados mediante técnicas de superresolución antes de aplicar el método SLIC. A cada superpíxel generado se le asocia un vector de características que captura información relevante de la región correspondiente. Estos vectores son utilizados para entrenar un modelo de clasificación binaria, diseñado para distinguir entre superpíxeles que pertenecen a una lesión y aquellos que no.

Sin embargo, un desafío clave en el análisis de imágenes médicas es la presencia de regiones de reflexión especular (Specular Reflection Regions, SR). Estas áreas, caracterizadas por su alta intensidad, saturación y contraste, aparecen como zonas de brillo blanco que pueden ocultar detalles importantes y distorsionar los valores reales de los píxeles en dichas regiones [3]. Si estas reflexiones no se corrigen adecuadamente, podrían alterar las características asociadas a los superpíxeles afectados, provocando errores en la clasificación. Por ejemplo, si un vector de características incluye información de una región especular, el modelo podría confundir estas características con las de una lesión, afectando la precisión de la clasificación.

3. *Data cleaning:*

Para abordar este problema, se realizó un preprocesamiento destinado a identificar y corregir las regiones especulares antes de la extracción de características. Este paso asegura que los vectores asociados a los superpíxeles reflejen únicamente información relevante para la segmentación de lesiones. Con ello, se reduce la probabilidad de interpretaciones erróneas.

Diversos enfoques han sido propuestos en la literatura para mitigar la reflexión especular en imágenes médicas. Un ejemplo notable se encuentra en el trabajo publicado por Kudva et al. [88], donde se empleó un filtro en imágenes cervicales con resultados satisfactorios. La fórmula original para este filtro se expresó como:

$$F = ((1 - S) \cdot G \cdot L)^3, \quad (11)$$

donde S , G y L representan los componentes de saturación, canal verde y luminosidad de la imagen, respectivamente, normalizados en un rango de 0 a 1. Sin embargo, la aplicación de esta técnica en imágenes de lesiones del pie demostró ser menos eficaz. Este resultado reveló una fuerte dependencia del rendimiento del filtro respecto a la media del canal verde (G), además de una notable variabilidad en la sensibilidad del filtro debido a fluctuaciones en el valor exponencial.

Adaptación el filtro especular El filtro original no se ajusta adecuadamente a nuestras imágenes de lesiones, ya que algunas de ellas presentan una alta media en el canal verde. Esto provoca que prácticamente toda el área de la lesión se considere como región especular, como se puede observar en la Figura XXXVII. Para abordar esta limitación, se propuso modificar el parámetro exponencial en la fórmula original del filtro descrito en Kudva et al. [88].

Durante nuestras pruebas, se observó que el filtro se atenuaba a medida que aumentábamos el valor del exponente dentro de un rango de aproximadamente 3 a 10. Este rango fue seleccionado para ajustar dinámicamente el exponente en función de la media del canal verde, un parámetro importante debido a su relevancia en la percepción de luminosidad. La Figura XXXV muestra un análisis del histograma del canal verde en 200 imágenes, donde la media varía entre 17.95 y 198.82. Sin embargo, decidimos analizar el exponente en función de un rango más amplio de la media del canal verde, específicamente entre 8 y 200, de esta forma evitábamos tener como exponente un valor de 3.

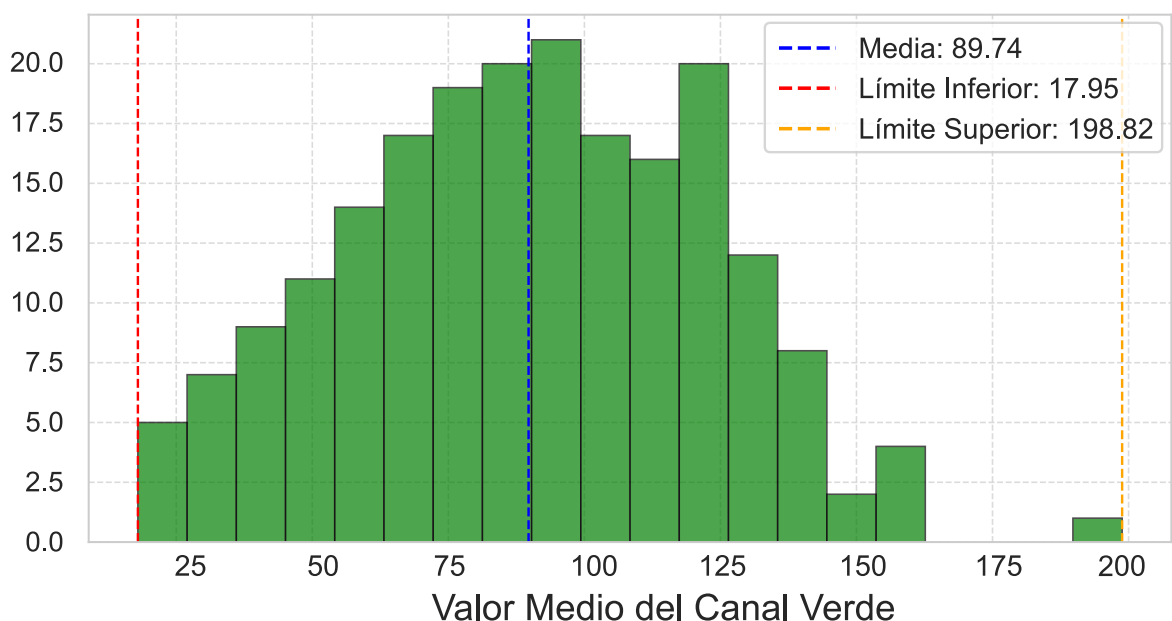


Figura XXXV: Histograma del canal verde de las imágenes analizadas. La variabilidad observada justifica la elección del rango 8-200 para el cálculo del exponente en el filtro especular.

Para adaptar el exponente de manera más eficaz, se introdujo una función logarítmica que mapea la media del canal verde al rango deseado. Esta modificación se expresa en la siguiente fórmula adaptada del filtro:

$$F = ((1 - S) \cdot G \cdot L)^{x_2}, \quad (12)$$

donde x_2 se calcula mediante la relación logarítmica:

$$x_2 = e^{(a \cdot \ln(x_1) + b)}, \quad (13)$$

siendo x_1 la media del canal verde. La selección de una transformación logarítmica se justifica por su eficacia en el manejo de un amplio rango de valores, proporcionando una escala más flexible para el ajuste del exponente.

Los valores de a y b se determinaron para escalar el rango de la media del canal verde entre 8 y 200 a un rango de exponentes entre 3 y 10. Específicamente, los valores se calcularon como:

$$a = \frac{\ln(10) - \ln(3)}{\ln(200) - \ln(8)}, \quad b = \ln(3) - a \cdot \ln(8) \quad (14)$$

Esto asegura que para un valor de $x_1 = 8$, el exponente x_2 sea aproximadamente 3, y para un $x_1 = 200$, el exponente x_2 sea aproximadamente 10.

Una vez identificadas las áreas de reflexión especular utilizando este enfoque mejorado, se aplica el algoritmo de inpainting de OpenCV para rellenar estas zonas de manera coherente con el entorno circundante. Este proceso de inpainting restaura la información de la imagen, integrando armoniosamente las áreas afectadas con las regiones adyacentes, preservando así la integridad estructural y la continuidad visual de la imagen. Un ejemplo ilustrativo de este proceso se muestra en la Figura XXXVI, donde se utiliza una máscara específica para identificar las áreas con alta reflexión especular y luego se aplica el inpainting para su corrección.

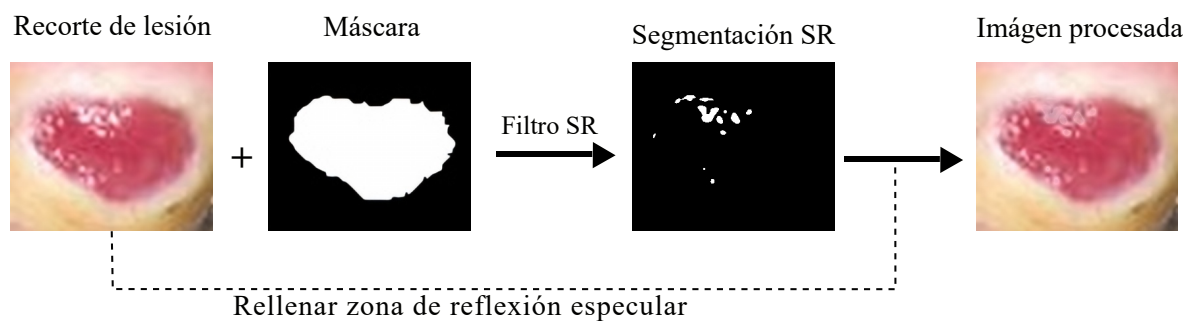


Figura XXXVI: Ejemplo del proceso de eliminación de reflexión especular utilizando un filtro mejorado y completación de imagen mediante el método de inpainting de OpenCV.

El pseudocódigo para la implementación del filtro especular está detallado en el ANEXO 1, véase 1 para más detalles.

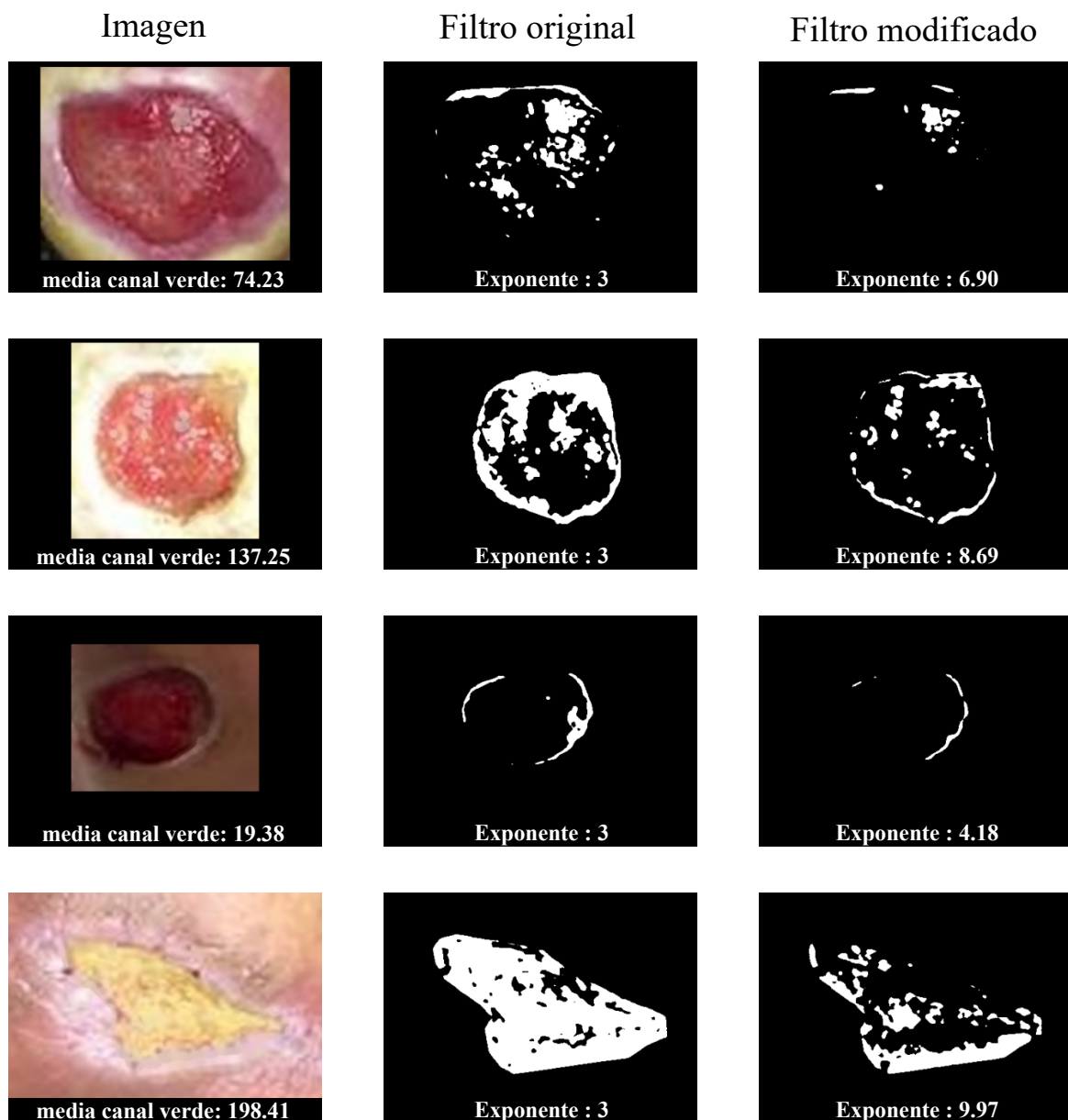


Figura XXXVII: Comparación de la aplicación del filtro de reflexión especular en imágenes con diferentes medias del canal verde.

4. *Etiquetado de superpíxeles*

Para entrenar un modelo de clasificación binaria, como una regresión logística, es importante estructurar los datos de forma que sean consistentes con las necesidades del modelo. En este contexto, los superpíxeles generados mediante el método SLIC constituyen la base para la extracción de vectores de características. A continuación, se describe detalladamente el proceso de etiquetado de superpíxeles.

En primer lugar, SLIC se configura utilizando tres parámetros clave, tales como el

número de segmentos, el factor de compactación y el parámetro sigma. Para nuestras imágenes, se definió un número objetivo de 100 segmentos, seleccionado empíricamente. Aunque el número exacto de superpíxeles puede variar, este valor guía al algoritmo hacia una partición aproximada de la imagen en 100 regiones.

El factor de compactación, establecido en 10, regula el equilibrio entre la cohesión de color y la proximidad espacial de los píxeles dentro de un superpíxel. Este valor asegura que los superpíxeles generados respeten tanto las fronteras espaciales como las características visuales de las áreas de interés. El parámetro sigma, fijado en 1, introduce un suavizado previo a la segmentación, lo que ayuda a minimizar el impacto del ruido y las pequeñas variaciones de color en la imagen.

El etiquetado de los superpíxeles se realiza evaluando su intersección con la máscara binaria de la imagen. Para ello, se calcula la cantidad de píxeles de la máscara que se solapan con cada superpíxel. Si este solapamiento excede un umbral de 180 píxeles, el superpíxel se clasifica como perteneciente a la lesión; en caso contrario, se etiqueta como no perteneciente a la lesión. Este umbral se determinó empíricamente, considerando la escala y las características típicas de las lesiones en nuestras imágenes.

El proceso de etiquetado garantiza que cada superpíxel reciba una etiqueta binaria en función de su correspondencia con la región de interés definida por la máscara de la lesión. La Figura XXXVIII (c) muestra el resultado de este proceso. Los superpíxeles delimitados por contornos amarillos corresponden a regiones que no superaron el umbral de intersección de 180 píxeles con la máscara original, por lo que se etiquetan como fondo. En contraste, los superpíxeles con contornos rojos superaron dicho umbral y se clasificaron como pertenecientes a la lesión.

Este etiquetado permite identificar claramente qué superpíxeles pertenecen a la lesión y cuáles no. Con esta información, el siguiente paso consiste en caracterizar cada superpíxel extrayendo sus vectores de características. Estas características, junto con las etiquetas generadas, se emplean para entrenar un modelo de clasificación logística,

obteniendo un modelo que nos indica la probabilidad de que un superpíxel pertenezca a una lesión o no.

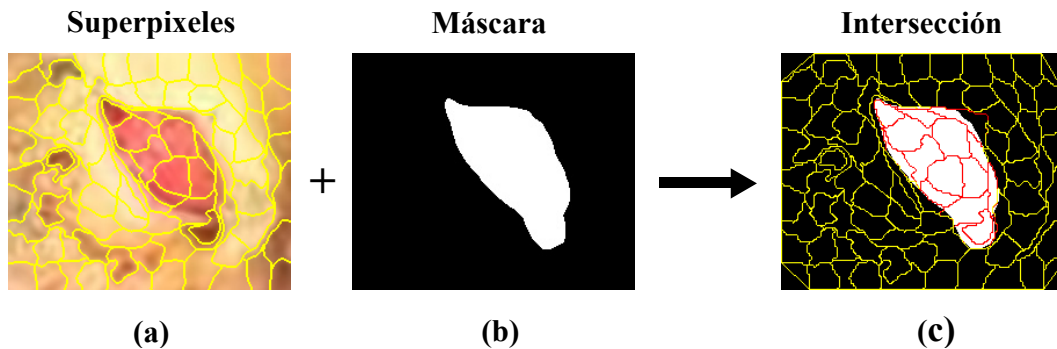


Figura XXXVIII: (a) Recorte de la lesión aplicando el método SLIC, (b) Máscara asociada al recorte, (c) Imagen con superpíxeles etiquetados de amarillo (no pertenecientes a la lesión) y rojo (pertenecientes a la lesión).

5. Extracción de características

En la fase de extracción de características, se ha desarrollado un enfoque que combina conocimientos obtenidos de estudios previos para construir una representación de datos efectiva. Siguiendo las directrices de Silva y Machado [3], se diseñó un vector de características que incorpora descriptores de color. Además, basándonos en los hallazgos reportados en Giraldo et al. [89], se integró un enfoque avanzado que aprovecha los mapas de características generados por una red neuronal convolucional preentrenada. Estos mapas, como se describe en Zeiler y Fergus [52], poseen propiedades de composicionalidad que permiten capturar patrones complejos y sutiles en las imágenes, incluso aquellos que no son perceptibles de forma directa. A medida que los datos progresan a través de las capas de la red, las características se vuelven más abstractas y representativas, enriqueciendo el conjunto de descriptores disponibles.

El método propuesto se centra en la construcción de un vector de características para cada superpíxel identificado, integrando tanto información espectral como espacial. Inicialmente, se extraen 128 mapas de características de la quinta capa convolucional del modelo VGG16, preentrenado utilizando la base de datos ImageNet. Estos mapas, que

capturan patrones abstractos y jerárquicos en las imágenes, se complementan añadiendo los tres canales RGB originales de la imagen, generando un total de 131 canales de datos, tal como se observa en la Figura XXXIX(c).

Para cada canal, se calculan dos métricas estadísticas, la media y la varianza, limitadas a los píxeles contenidos dentro de los bordes de cada superpíxel. Este enfoque extrae propiedades internas de cada región, resultando en un conjunto inicial de 262 características derivadas de los 131 canales. Adicionalmente, se incorporan dos características espaciales basadas en las coordenadas x e y del centroide geométrico de cada superpíxel, lo que eleva el número total de características por superpíxel a 264, esto puede visualizarse en la Figura XXXIX(d).

Esta representación, que integra información espacial, espectral y características de alto nivel extraídas de los mapas generados por la red neuronal, constituye un vector descriptivo completo para cada superpíxel. Este vector, junto con la etiqueta asignada a cada superpíxel basada en los píxeles contenidos dentro de sus límites (lesión o no lesión), se emplea como entrada para entrenar un modelo de clasificación supervisado.

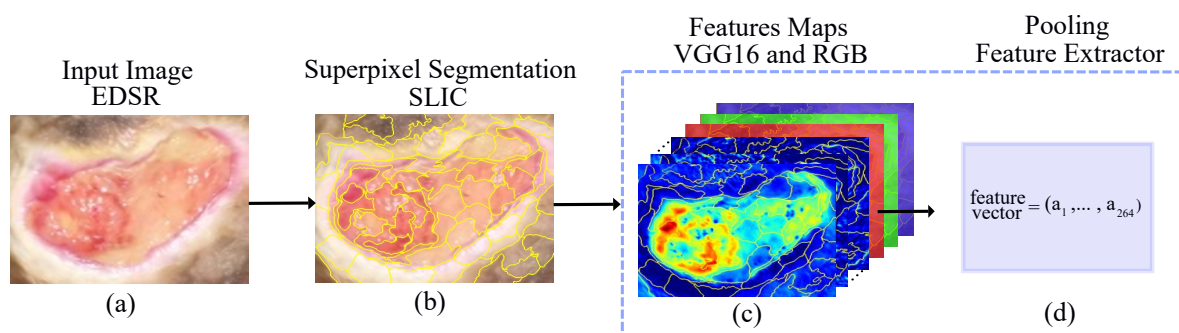


Figura XXXIX: (a) Aplicación de superresolución EDSR, (b) método SLIC para la generación de superpíxeles, (c) extracción de características con mapas de características VGG16 y canales RGB, (d) obtención de un vector de características de 264 elementos para cada superpíxel.

El pseudocódigo utilizado en la extracción de características se encuentra en el ANEXO 1, véase 3 para más detalles.

6. Clasificación binaria

Para determinar si un superpíxel pertenece o no a una lesión, se utilizó un modelo de regresión logística. Este modelo fue seleccionado debido a su capacidad para gestionar un gran número de variables independientes y su adecuación en problemas de clasificación binaria, donde la salida es una probabilidad que se mapea a dos clases discretas. La regresión logística modela la probabilidad de pertenencia a una clase a partir de una combinación lineal de las características del superpíxel, aplicando una transformación logística para garantizar que las probabilidades se encuentren en el intervalo $[0, 1]$.

La probabilidad de que un superpíxel pertenezca a la clase de lesión, denotada como p , se modela matemáticamente mediante el logit o razón de verosimilitud logarítmica, definido en la Ecuación 15:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_{264} X_{264} \quad (15)$$

En esta ecuación, X_i representa las características individuales asociadas a cada superpíxel, mientras que β_i son los coeficientes del modelo que se estiman durante el proceso de entrenamiento. Para convertir esta relación lineal en una probabilidad dentro del intervalo $[0, 1]$, se utiliza la función logística, como se muestra en la Ecuación 16:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_{264} X_{264})}} \quad (16)$$

El vector de características, $X = [X_1, X_2, \dots, X_{264}]$, incluye valores extraídos de la imagen, específicamente, la media y la varianza de los 131 canales, donde 128 corresponden a mapas de características obtenidos del modelo VGG16 preentrenado y 3 a los canales RGB de la imagen original. Además, se incorporan las coordenadas espaciales x e y del centroide del superpíxel, permitiendo al modelo capturar tanto información espectral como espacial.

La Ecuación 15 describe cómo las características del superpíxel se combinan

linealmente para formar el logit, mientras que la Ecuación 16 transforma este valor en una probabilidad que se encuentra dentro del intervalo unitario. Con base en esta probabilidad, el modelo clasifica al superpíxel como perteneciente a la lesión si $p \geq 0,5$, y como fondo en caso contrario ($p < 0,5$).

Conjunto de datos y proceso de entrenamiento: El conjunto de datos utilizado consta de 18,133 superpíxeles extraídos de 200 imágenes. La etiqueta binaria de cada superpíxel se definió en función de su intersección con la máscara de la lesión, el superpíxel se etiquetaba como perteneciente a la lesión ($wound = 1$); en caso contrario, se etiquetaba como fondo ($wound = 0$). Esto resultó en un conjunto de datos con 11,988 superpíxeles etiquetados como fondo y 6,145 como lesión.

El conjunto de datos fue dividido en subconjuntos de entrenamiento y prueba mediante una partición estratificada, asignando el 80 % para entrenamiento (X_{train}, Y_{train}) y el 20 % para prueba (X_{test}, Y_{test}), preservando la proporción de clases en ambos conjuntos. Se utilizó un modelo de regresión logística con un límite de 100 iteraciones.

El proceso de entrenamiento fue eficiente, con un tiempo de CPU de 1.58 segundos, ejecutado en una computadora equipada con un procesador AMD Ryzen 9 5900HX, 16 GB de RAM y una GPU NVIDIA RTX 3070.

Evaluación del sistema de segmentación El desempeño del modelo fue evaluado utilizando métricas estándar para problemas de clasificación binaria, cuyos valores se resumen en la Tabla VI. Los resultados indican que el modelo alcanzó una exactitud de 0,86, con una precisión de 0,78, una sensibilidad de 0,83 y un valor F1 de 0,80.

7. Prompting SAM

En esta sección, exploramos en detalle el proceso por el cual los parámetros obtenidos, tales como los cuadros delimitadores y los puntos de lesiones o no lesiones, son integrados en el modelo SAM.

Tabla VI: Métricas de desempeño del modelo de regresión logística para la clasificación de superpíxeles.

Métrica	Valor
Exactitud	0.86
Precisión	0.78
Sensibilidad	0.83
Valor F1	0.80

El proceso inicia con la identificación de lesiones en las imágenes originales mediante el modelo YOLOv5. Este avanzado modelo localiza las áreas de interés en la imagen, marcándolas con cuadros delimitadores o 'bboxes', los cuales se ubican en la imagen original.

Tras la identificación de los cuadros delimitadores, procedemos a extraer recortes de la imagen correspondientes a estas áreas específicas. Estos recortes son posteriormente sometidos a un proceso de superresolución utilizando el modelo EDSR. Dicho proceso mejora la calidad de los recortes, incrementando su resolución y revelando detalles más finos. Sin embargo, este paso altera las dimensiones originales de los recortes, debido a la falta de correspondencia en tamaño y escala entre los recortes mejorados y las imágenes originales.

Para abordar este desafío, se calculan factores de escala, denominados f_h (factor de altura) y f_w (factor de anchura). Estos factores son determinantes para conciliar las diferencias en las dimensiones entre los recortes mejorados mediante superresolución y las imágenes originales. Los factores f_h y f_w se calculan mediante las siguientes ecuaciones.

$$f_h = \frac{H_{\text{original}}}{H_{\text{mejorado}}} \quad (17)$$

$$f_w = \frac{W_{\text{original}}}{W_{\text{mejorado}}} \quad (18)$$

Una vez calculados los factores de conversión f_h y f_w , se pueden ajustar las coordenadas obtenidas de los recortes con superresolución. El modelo de regresión logística proporciona las coordenadas de los puntos identificados como lesiones o no lesiones. Para reescalar

estas coordenadas a la escala original de la imagen, se ajustan utilizando los factores de escala y las coordenadas del cuadro delimitador. Por ejemplo, para ajustar la coordenada x' proporcionada por el modelo de regresión a su escala original, se utiliza la siguiente fórmula.

$$x_{\text{ajustada}} = (x' \times f_w) + x_1 \quad (19)$$

donde x_1 es el punto en el eje x en el extremo inferior izquierdo del cuadro delimitador (bbox). Este proceso asegura que la coordenada ajustada x_{ajustada} refleje con precisión su posición real en la imagen completa.

De manera análoga, para las coordenadas en el eje y , se utiliza el factor de escala f_h .

$$y_{\text{ajustada}} = (y' \times f_h) + y_1 \quad (20)$$

donde y_1 representa el punto en el eje y en el extremo inferior izquierdo del cuadro delimitador. Este enfoque garantiza que las coordenadas, una vez procesadas y ajustadas, reflejen con precisión su posición real en la imagen completa, manteniendo la integridad del análisis y la interpretación, este proceso completo se ilustra en la Figura XL.

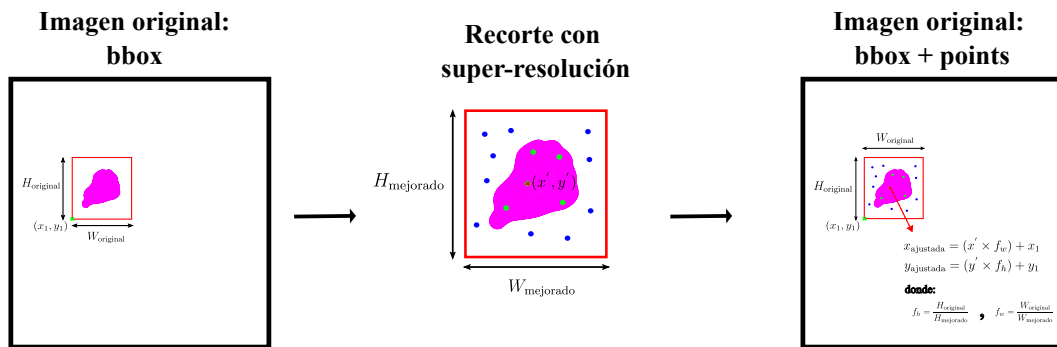


Figura XL: Diagrama del cálculo de factores de escala y ajuste de coordenadas en recortes con superresolución.

Después de determinar las coordenadas de los píxeles que corresponden a la lesión, así como el cuadro delimitador (bbox) de la región de interés, estos datos se introducen en el modelo SAM. Este modelo proporciona una segmentación basada tanto en los inputs

proporcionados como en el contexto general de la imagen, un proceso que se ilustra en la Figura XLI.

En situaciones donde YOLOv5 identifica múltiples lesiones en una sola imagen, se repite el mismo procedimiento para cada detección. Posteriormente, se combinan las máscaras de segmentación generadas para cada lesión detectada, resultando en una representación comprensiva de todas las lesiones identificadas en la imagen.

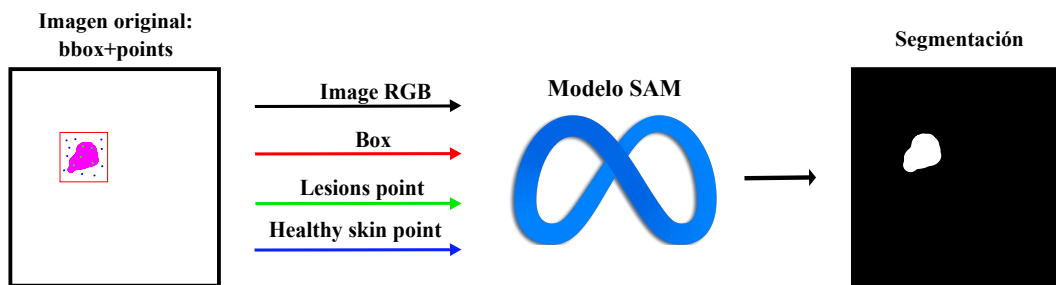


Figura XLI: Segmentación del modelo SAM utilizando cuadros delimitadores y puntos de lesión/no lesión.

El pseudocódigo que se empleó en la implementación final del algoritmo está detallado en el ANEXO 1 véase 4 para más detalles.

IV. Evaluación del sistema de segmentación

Esta sección se enfoca en detallar las pruebas realizadas para evaluar la precisión y eficiencia del modelo de segmentación propuesto.

A. Evaluación del sistema con referencias bibliográficas

Como se discutió en la sección B, la disponibilidad de bases de datos públicas para la segmentación de úlceras del pie diabético es limitada, lo que dificulta la comparación directa entre modelos si no se parte de un punto de referencia común. En trabajos como los presentados por Cui et al. [90], Muñoz et al. [91] y Rania et al. [92], se hace uso de datasets propios obtenidos a través de colaboraciones con laboratorios médicos participantes. Sin embargo, estos conjuntos de datos no están disponibles públicamente, lo que impide realizar comparaciones directas con los resultados obtenidos en este estudio.

En la Tabla VII se presentan los resultados de diferentes modelos, incluyendo aquellos entrenados con bases de datos públicas, como los trabajos realizados por Monroy et al. [93] y Mahbod et al. [85]. No obstante, cabe destacar que, aunque el dataset CW-Colombia [93] incluye imágenes de lesiones crónicas, estas no se enfocan específicamente en úlceras del pie diabético, lo que limita su aplicabilidad para comparaciones directas.

Por otro lado, el trabajo de Mahbod et al. [85] utiliza una combinación de varias bases de datos, entre las que se incluye el conjunto de datos FUSeg, el cual también fue empleado en este estudio como dataset base para el entrenamiento del modelo. La técnica propuesta por Mahbod et al. [85] combina un ensemble de modelos CNN, específicamente LinkNet y U-Net, que fueron pre-entrenados con EfficientNetB1 y EfficientNetB2, respectivamente. Además, el enfoque incluye técnicas avanzadas de aumento de datos basadas en propiedades morfológicas y de color, validación cruzada de cinco pliegues, y test time augmentation con fusión de resultados. Estas estrategias les permitieron alcanzar un rendimiento sobresaliente al ser evaluados en conjuntos como Chronic Wound y MICCAI 2021 FUSeg Challenge.

De este modo, la comparación más relevante en este trabajo se centra en los modelos que, al igual que el nuestro, emplean el dataset FUSeg como punto de partida, permitiendo evaluar de manera consistente la efectividad de las técnicas aplicadas en el presente estudio.

Tabla VII: Resumen de modelos de aprendizaje profundo para segmentación de úlceras del pie diabético, tamaño de los datasets, métricas de desempeño, acceso y fuentes de datos.

Dataset	Size	Acceso	Modelo	IoU (%)	DSC (%)	Precisión (%)
NYU dataset [90]	445	Privada	U-Net	76.1	84.5	83
ICIMAF-DFU [91]	1176	Privada	Mask R-CNN	-	>94.04	>92.17
DFU-Small [92]	188	Privada	U-Net	94.86	97.25	-
CW-Colombia [93]	188	Pública	U-Net	88.8 ± 1	90.2 ± 1	-
Medetec, CW, FUSeg [85]	1210	Pública	LinkNet + U-Net	85.51	92.07	92.68

Tabla VIII: Comparativa de métricas de desempeño. La evaluación involucró un conjunto de prueba de 83 imágenes.

Modelo	IoU	Dice	Precisión	F1
SAM: box + points	0,941 ± 0,054	0,969 ± 0,030	1,000 ± 0,000	0,969 ± 0,030
LinkNet + U-Net	0,720 ± 0,140	0,828 ± 0,110	0,827 ± 0,124	0,828 ± 0,110

B. Evaluación con enfoques similares

Para evaluar la efectividad de nuestro método frente a alternativas existentes, realizamos un análisis comparativo con la estrategia descrita en Ma et al. [11]. Este estudio introduce un protocolo específico para entrenar el modelo SAM en el contexto de imágenes médicas, denominando a la versión resultante como MedSAM. Es relevante mencionar que MedSAM fue entrenado con imágenes médicas que no incluían lesiones de pie diabético, lo que plantea un desafío adicional para una comparación directa. Siguiendo los pasos propuestos en dicho estudio, adaptamos esta técnica de ajuste fino a nuestro conjunto de datos, comenzando con la descarga del modelo preentrenado basado en Vision Transformer (ViT) tipo B y enfocando la optimización en los parámetros del decodificador durante el ajuste fino. Los detalles específicos de este entrenamiento se describen en la sección 4.1 del Anexo 4.

Adicionalmente, realizamos una evaluación considerando distintas estrategias de entrada y representaciones del modelo SAM. Estas incluyen el uso exclusivo de cuadros delimitadores generados automáticamente, la utilización de puntos de referencia específicos, y la integración de ambas estrategias. Además, exploramos alternativas que varían la dimensionalidad del vector característico utilizado en el proceso de segmentación. Una de estas alternativas emplea exclusivamente los canales RGB, resultando en un vector característico de dimensión 8, mientras que otra se basa en los 131 mapas de características extraídos mediante VGG16, configurando un vector característico de dimensión 262. Estas variantes permiten analizar el impacto de la representación de características sobre el desempeño del modelo.

Para garantizar la comparabilidad entre enfoques, se normalizó el proceso de segmentación en cada estrategia evaluada utilizando como entrada las bounding boxes generadas por YOLOv5. Los resultados cuantitativos obtenidos y las comparaciones detalladas se presentan en la Tabla IX, donde se evidencia el desempeño de cada modelo en términos de métricas como IoU, Dice, precisión y F1.

Tabla IX: Comparativa de métricas de desempeño en la generación de máscaras de segmentación aplicando cuadros delimitadores, puntos, SAM con fine-tuning y MedSAM. La evaluación involucró un conjunto de prueba de 83 imágenes.

Modelo	IoU	Dice	Precisión	F1
MedSAM	0.568 ± 0.206	0.702 ± 0.174	0.995 ± 0.007	0.702 ± 0.174
SAM: Fine Tuning	0.873 ± 0.101	0.928 ± 0.068	0.999 ± 0.001	0.928 ± 0.068
SAM: box	0.809 ± 0.242	0.869 ± 0.190	0.998 ± 0.005	0.869 ± 0.190
SAM: points	0.937 ± 0.040	0.967 ± 0.022	1.000 ± 0.000	0.967 ± 0.022
SAM: box + points	0.941 ± 0.054	0.969 ± 0.030	1.000 ± 0.000	0.969 ± 0.030

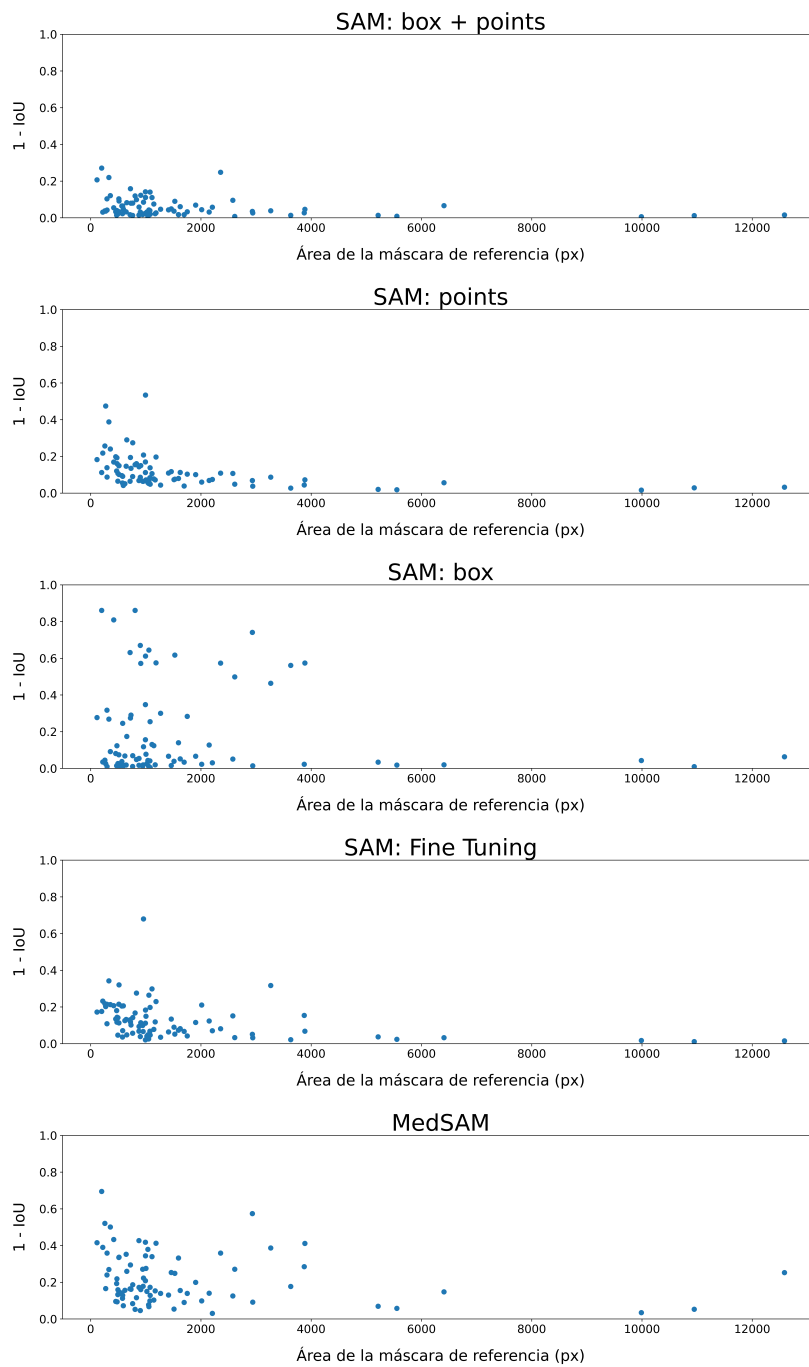


Figura XLII: Gráficos comparativos del error por modelo en la data de test utilizando el complemento del índice IoU frente al área de la etiqueta en píxeles.

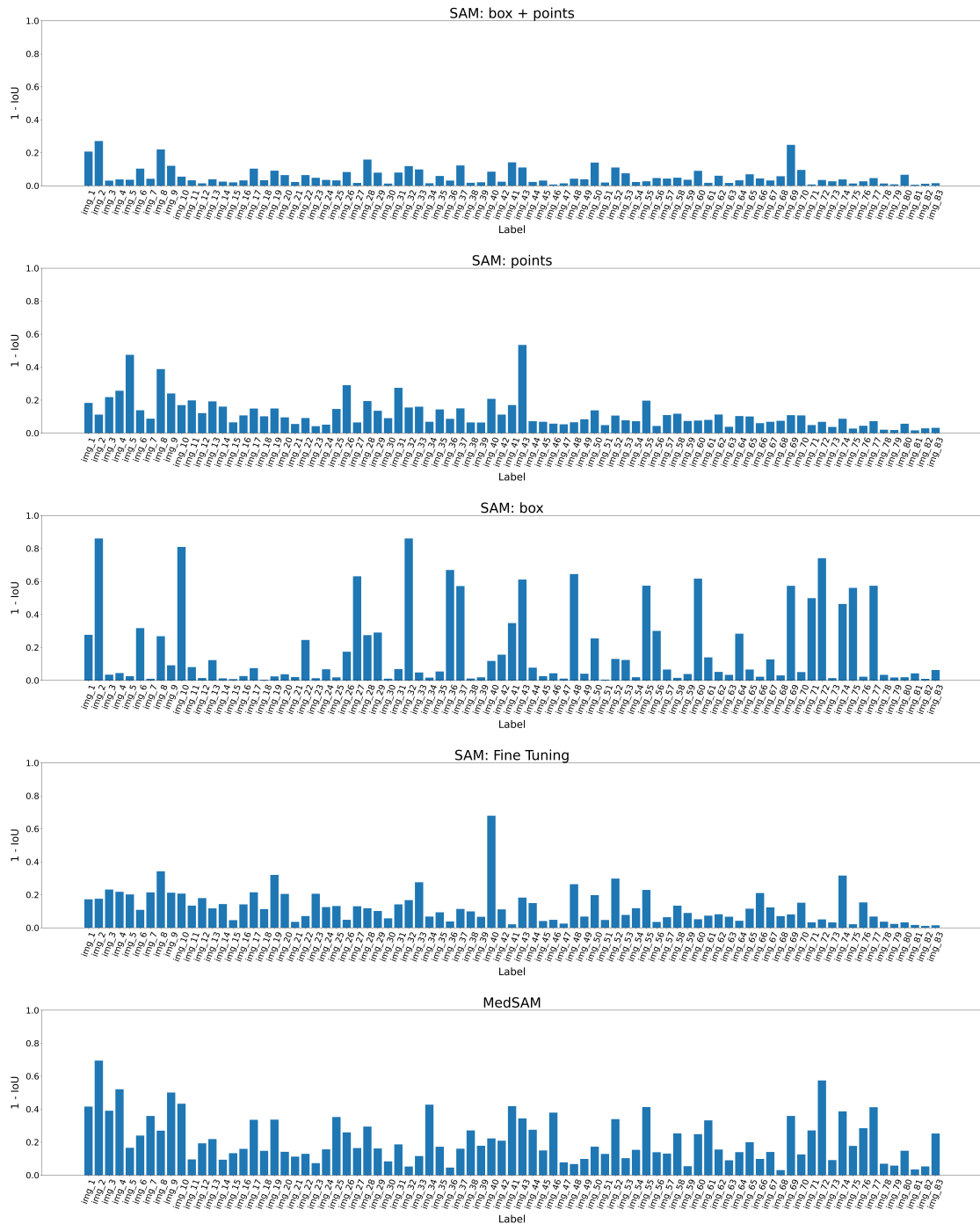


Figura XLIII: Gráficos comparativos del desempeño de los enfoques propuestos utilizando el complemento del IoU por imagen, ordenados de forma ascendente según el área, destacando la img_83 como la de mayor área.

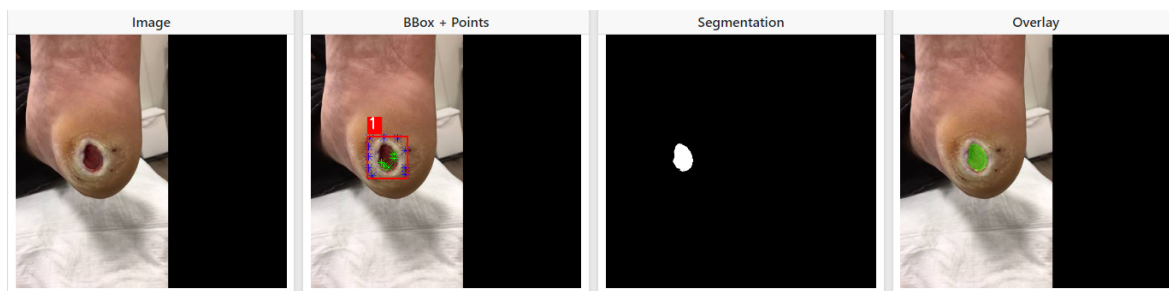
C. Evaluación de la eficiencia

La eficiencia del modelo para la segmentación de lesiones fue probada a través de una interfaz web implementada en un entorno local, con una explicación detallada disponible

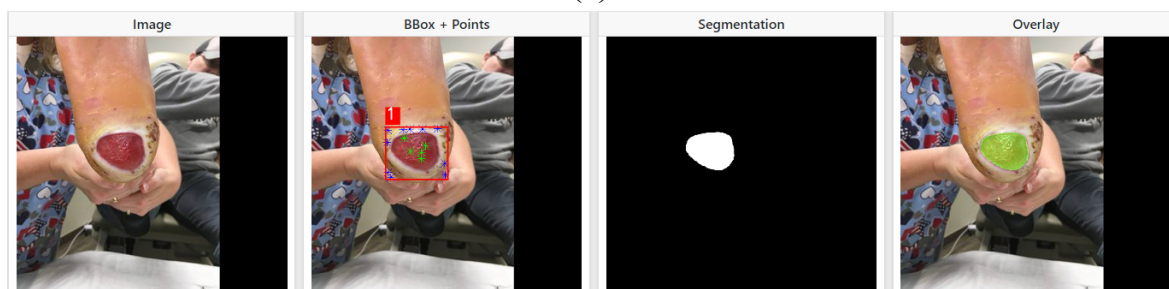
en la Sección 3.1 del Anexo 3. Se ejecutaron pruebas para evaluar el tiempo en cada etapa de segmentación. La prueba preliminar, documentada en la Figura XLIV(a), implicó el procesamiento de una imagen prototípica del dataset utilizado durante la fase de validación. Esta evaluación tenía como fin verificar la estabilidad y fiabilidad del modelo, asegurando la invariancia de su rendimiento al migrar de una plataforma de pruebas controladas en una interfaz web local.

En la Figura XLIV (b), se probó el modelo con una imagen que tenía un fondo con colores rojo presentes, un desafío particular para evaluar cómo el modelo distingue las lesiones en condiciones de color complejas. Seguidamente, como se muestra en la Figura XLIV (c), se seleccionó una imagen para probar la capacidad del modelo en la segmentación de múltiples lesiones.

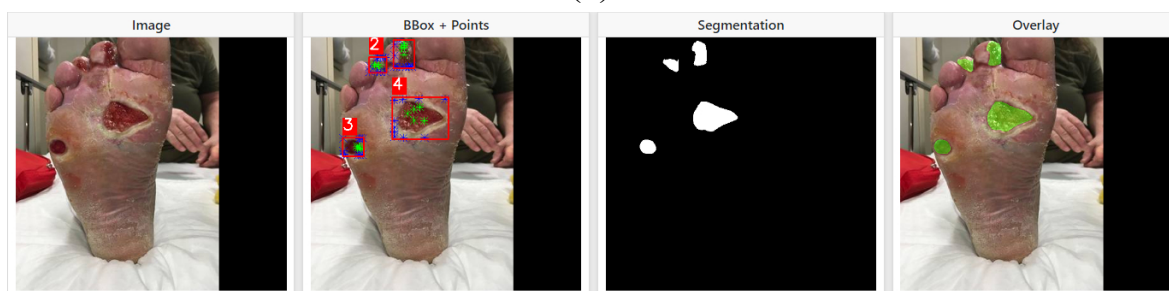
Finalmente, para evaluar la capacidad de generalización de nuestro modelo, seleccionamos una imagen de alta resolución de una lesión ubicada en una región diferente a la que se enfocó durante el entrenamiento, como se muestra en la Figura XLIV(d). Esta elección permitió probar la efectividad del modelo en condiciones distintas a las de su configuración inicial. Los resultados de estas pruebas se detallan en la Tabla X.



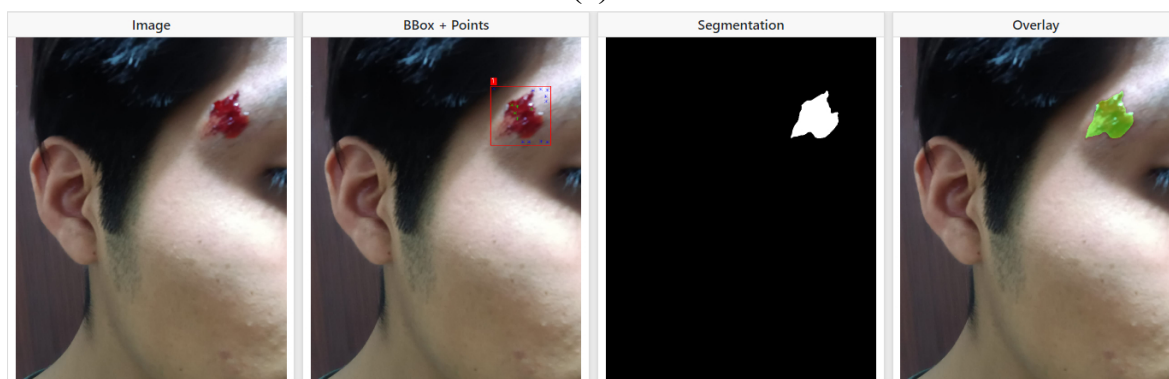
(a)



(b)



(c)



(d)

Figura XLIV: Capturas de pantalla enfocadas en la segmentación de lesiones para diversos tipos de casos.

Tabla X: Desglose detallado de los tiempos de procesamiento en las diferentes fases del sistema de segmentación.

Img	Resolución (px)	YOLOv5 (s)	EDSR (s)	VGG16+RL (s)	SAM (s)	Total (s)
(a)	512x512	0.90	5.46	1.88	3.02	11.30
(b)	512x512	0.89	9.42	2.81	3.06	16.22
(c)	512x512	0.84	10.10	6.22	3.20	20.43
(d)	1200x1600	0.86	0.00	1.97	2.99	5.99

1. Gasto computacional

El sistema desarrollado emplea como detector el modelo YOLOv5s, una variante optimizada y de tamaño reducido. A pesar de su escala compacta, el YOLOv5s mantiene una demanda significativa de capacidad de cómputo. De acuerdo con su documentación oficial de Ultralytics [60], este modelo necesita alrededor de 16.5 GFLOPs por inferencia para procesar imágenes de 640×640 píxeles.

El modelo de superresolución EDSR fue implementada de la biblioteca OpenCV, el cual ajusta su complejidad según la resolución de la imagen procesada. Aunque el estudio original Lim et al. [71] no especifica la cantidad exacta de GFLOPs, hemos realizado un cálculo basado en su arquitectura propuesta. Para imágenes de 100×100 píxeles, la demanda computacional se estima en 1.49 GFLOPs. Los detalles de este cálculo se encuentra en la sección 2.1 Anexo 3.

En cuanto a VGG16, al utilizar únicamente sus primeras cinco capas, los FLOPs necesarios se estiman en aproximadamente 18.495 GFLOPs. El cálculo de esta estimación puede verse en la sección 2.2 Anexo 3.

El modelo de generación de superpíxeles SLIC, destaca por su eficiencia, capaces de segmentar una imagen en superpíxeles con un bajo número de operaciones. a diferencia de los GFLOPs usados en modelos más complejos. Igualmente, un modelo de regresión logística con 264 características requiere solo 531 operaciones, resaltando su sencillez y eficacia. Estos valores, detallados en la sección 2.3 Anexo 3.

Finalmente, el modelo SAM utilizado incorporó el transformer ViT-H/16, destacándose por una arquitectura robusta con 636 millones de parámetros . La información detallada sobre su arquitectura no está disponible públicamente debido a consideraciones de confidencialidad. Sin embargo, basándonos en la información accesible en Kirillov et al. [10], se puede estimar que la sección del modelo denominada *image encoder*, que emplea el transformer ViT-H, requiere aproximadamente 3.4 TFLOPs para su operación, según se detalla en Li et al. [94]. Cabe destacar que, a pesar de ser una operación computacionalmente intensiva, esta se lleva a cabo únicamente una vez por cada imagen procesada, lo que permite que las tareas subsiguientes del modelo SAM se ejecuten casi en tiempo real.

V. Discusión de Resultados y Conclusiones

A. Discusión de resultados

En esta sección, presentaremos un análisis de las observaciones y experiencias obtenidas a lo largo del desarrollo del sistema de segmentación.

1. *Entrenamiento del modelo YOLOv5*

Para el entrenamiento del modelo YOLOv5, específicamente su variante YOLOv5s, se eligió esta versión debido a su equilibrio entre velocidad y precisión. A pesar de que la técnica de aumento de datos (data augmentation) era una opción viable para enriquecer el conjunto de entrenamiento, se optó por no utilizarla, ya que los resultados preliminares demostraron ser satisfactorios sin esta medida. Este proceso de entrenamiento se llevó a cabo en Google Colab, siguiendo los pasos indicados en la documentación oficial de YOLOv5.

Posteriormente, se realizaron modificaciones en el archivo detect.py del sistema YOLOv5 para adaptarlo mejor a las necesidades de nuestro entorno específico. Originalmente, YOLOv5 está más enfocado en operar mediante la compilación de scripts independientes. Sin embargo, para fines de este proyecto se ajustó para que funcionase como un módulo dentro de nuestro sistema de segmentación.

2. *Data cleaning*

Durante el proceso de data cleaning, enfocado en las imágenes destinadas al entrenamiento del modelo de regresión logística, se prestó especial atención a corregir anomalías, como los reflejos especulares en las lesiones. Esta corrección resultó ser un paso importante para minimizar la posibilidad de que puntos inexactos fueran introducidos al modelo SAM durante la segmentación. La ajuste de la función de filtrado, para eliminar reflejos especulares, se realizó a través de una serie de pruebas de procesamiento de

imágenes. En estas pruebas, se examinó visualmente la efectividad del filtro en una variedad de imágenes, lo que nos permitió ajustar adecuadamente la función original y definir los parámetros más adecuados para su aplicación.

3. Superresolución con el método EDSR

El modelo de superresolución EDSR presenta retos en términos de consumo computacional, especialmente notable al tratar con imágenes de gran tamaño. En nuestro caso, al manejar imágenes de 512 x 512 píxeles, la implementación de EDSR resultó ser manejable debido a que los recortes utilizados se mantuvieron dentro de un tamaño razonable. Cabe mencionar que, dado que OpenCV ejecuta sus modelos principalmente en CPU, la implementación de EDSR, obtenida de esta biblioteca, implicó un mayor tiempo de cómputo en comparación con modelos más intensivos en recursos como SAM o YOLOv5. Para contrarrestar esta demanda de recursos, se introdujo una condición previa que limita las dimensiones de los recortes a 100 x 100 píxeles. Esta medida asegura que el EDSR se aplique solo cuando sea necesario.

4. Etiquetado de superpíxeles

Durante el proceso de etiquetado de superpíxeles, se observó que algunos superpíxeles incluían píxeles de piel sana en superpíxeles designadas para representar lesiones, y viceversa, como se ilustra en la Figura XLV. Sin embargo, esta variación no impacta de manera significativa en el entrenamiento del modelo de regresión logística. La razón es que el proceso de extracción de características promedia los valores dentro de cada superpíxel. Por lo tanto, las pequeñas inclusiones de áreas no correspondientes tienen un impacto mínimo en las medidas estadísticas, como la media y la varianza, de cada superpíxel. Adicionalmente, en la selección de puntos para el modelo SAM, se eligen los cinco centroides de los superpíxeles que con mayor probabilidad representan una lesión. Este criterio selectivo reduce considerablemente el riesgo de incorporar puntos que no reflejen con precisión su

clase real.

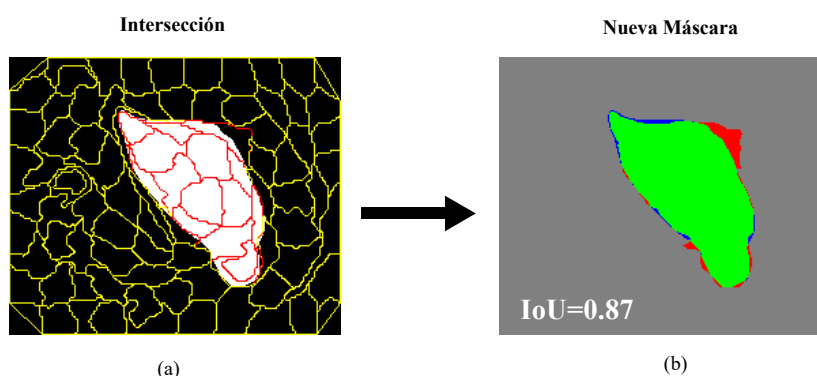


Figura XLV: (a) Clasificación de superpíxeles en lesión (rojo) y no lesión (amarillo), (b) identificación de verdaderos positivos (verde), falsos negativos (azul), falsos positivos (rojo), verdaderos negativos (gris) con un índice de 0.87 en la unión sobre intersección (IoU).

5. Extracción de características

En el proceso de extracción de características, exploramos varias metodologías para construir vectores de características más complejos, incluyendo opciones como la entropía, la varianza, y la distribución de color por canal. A pesar de la diversidad de estos enfoques, para este estudio, optamos por la estimación de solo dos parámetros estadísticos. Esta elección se fundamentó en la observación de que la integración de mapas de características extraídas de VGG16 proporcionaba un equilibrio adecuado en nuestro análisis, compensando la necesidad de descriptores más complejos.

6. Prompting SAM

En la aplicación del modelo SAM, seguimos las sugerencias establecidas en el paper original que presenta este modelo Kirillov et al. [10], donde se recomienda el uso de cinco puntos para la segmentación eficaz de las regiones de interés. No obstante, en el caso de las áreas no lesionadas, optamos por emplear diez puntos, basando esta decisión en la premisa de que estas zonas suelen ser más extensas y, por tanto, requieren una delimitación más detallada para una segmentación precisa. Esta estrategia fue el resultado

de un análisis de las particularidades de nuestro conjunto de datos y buscaba encontrar un balance óptimo entre la precisión y la claridad en la identificación y diferenciación de las regiones no lesionadas.

7. *Análisis del sistema de segmentación*

El modelo propuesto logra una segmentación con alta precisión mediante la integración secuencial de múltiples técnicas avanzadas. Una de las innovaciones en nuestro enfoque es la incorporación de tecnologías recientes, como los transformers, que representan un avance significativo en el campo de la visión computacional y no se habían aplicado previamente en la segmentación de lesiones en pie. Aunque el modelo YOLOv5 puede presentar desafíos en términos de detección precisa en ciertos casos, vemos un considerable potencial para mejorar su rendimiento, ya sea a través de un entrenamiento más exhaustivo o mediante la implementación de detectores más avanzados.

Los resultados obtenidos hasta el momento son competitivos y, en varios casos, superan a otros modelos reportados en la literatura, como se muestra en la Tabla VII. Sin embargo, es importante destacar que los conjuntos de datos utilizados por los modelos comparados varían en tamaño y, en su mayoría, corresponden a bases de datos privadas, lo que limita la posibilidad de realizar comparaciones directas de los valores de precisión alcanzados. A pesar de estas limitaciones, los valores obtenidos por nuestro sistema en métricas como IoU, DSC y precisión, que se presentan en la Tabla IX, demuestran un rendimiento superior en comparación con los modelos previamente evaluados.

Con el objetivo de realizar una comparación más justa, se seleccionó un modelo disponible públicamente en GitHub, desarrollado por Mahbod et al. [85], para segmentar las mismas 83 imágenes de prueba utilizadas en nuestro sistema. Los resultados de esta comparación se encuentran en la Tabla VIII, donde se evidencia que el sistema propuesto logró un mejor desempeño en todas las métricas evaluadas. Es relevante mencionar que, según Mahbod et al. [85], su modelo fue preentrenado utilizando imágenes de lesiones

crónicas provenientes de las bases de datos Medetec y Chronic Wound, además de aplicar técnicas de aumento de datos durante el entrenamiento. No obstante, su modelo presentó un desempeño inferior al nuestro. La ventaja de nuestro sistema radica en el uso del modelo SAM, preentrenado con un conjunto masivo de 1.1 mil millones de imágenes. Este preentrenamiento le permite comprender de manera intrínseca el contexto de las imágenes y los objetos, refinando de forma eficiente las búsquedas mediante *prompts*. Sin embargo, esta mayor precisión se alcanzó a costa de una menor eficiencia computacional. Mientras que el modelo de Mahbod et al. [85] procesó las 83 imágenes en 1 minuto y 12 segundos, nuestro sistema requirió 23 minutos y 25 segundos en una laptop Dell Alienware M15 R5 equipada con un procesador Ryzen 9, 16 GB de RAM, 1 TB de almacenamiento SSD y una tarjeta gráfica NVIDIA RTX 3070. Esto demuestra que, aunque el sistema propuesto alcanzó una alta precisión, su tiempo de procesamiento es significativamente mayor.

Adicionalmente, se compararon diferentes variantes de nuestro sistema para identificar la mejor configuración. Como se muestra en la Tabla IX, el método de segmentación propuesto presenta un rendimiento superior en comparación con las variantes evaluadas. Esta diferencia se evidencia en las Figuras XLII y XLIII, donde se observa un menor error en la segmentación del conjunto de prueba. Aunque se exploró la relación entre el tamaño de la lesión y la precisión, los resultados no demostraron una correlación directa. Esto podría atribuirse a la distribución desigual de tamaños de lesiones en las imágenes analizadas, lo que subraya la importancia de validar el sistema con un conjunto de datos más diverso en términos de tamaño y características de las lesiones.

Es relevante mencionar que la estrategia que emplea exclusivamente puntos como entradas para el modelo SAM demuestra un desempeño comparable con el método que integra tanto cajas delimitadoras como puntos. No obstante, es importante destacar que ambos métodos empiezan con la identificación de cajas delimitadoras. Así, la incorporación de cajas delimitadoras en nuestra metodología no implica un incremento notable en el costo computacional, dado que estas cajas son parte integral del proceso inicial de detección.

En consecuencia, el mejoramiento en precisión obtenido con nuestro enfoque resulta ser eficiente en términos de recursos computacionales, ya que no incurre en un aumento sustancial del gasto computacional.

La comparativa entre el proceso de fine-tuning del modelo SAM y la metodología empleada por el modelo propuesto revela diferencias significativas en términos de demanda computacional y eficiencia. Por un lado, el fine-tuning de SAM, enfocado en la variante tipo B con sus 91 millones de parámetros, es intensivo en recursos, requiriendo 25 GB de RAM y aproximadamente 22.77 horas para entrenar 1010 imágenes en una GPU 3070 Ti, como se detalla en la sección 4.1 del Anexo 4. Este proceso busca especializar un modelo ya de por sí ligero, culminando en un segmentador eficiente y especializado. Por otro lado, nuestro enfoque se centra en el SAM tipo H, una versión más robusta con 636 millones de parámetros. Aunque este modelo exige una mayor capacidad de procesamiento, se complementa con puntos y cajas delimitadoras para lograr una segmentación más detallada. Además, nuestro método solo requiere entrenar un clasificador de regresión logística, proceso que se completó en tan solo 1.58 segundos utilizando una laptop con GPU RTX 3070, ofreciendo una alternativa rápida y eficiente al proceso de fine-tuning más prolongado.

El modelo SAM con fine-tuning, una vez entrenado, se convierte en una solución eficiente y rápida, integrando únicamente YOLOv5 y SAM, sin la necesidad de etapas adicionales. En contraste, aunque nuestro modelo involucra solo el entrenamiento de un clasificador de regresión logística, que es un proceso rápido, la segmentación de imágenes requiere varias etapas adicionales. Este enfoque puede resultar en un tiempo de procesamiento más extenso. Sin embargo, en situaciones con imágenes de alta resolución, la activación del modelo EDSR se omite, como se observa en la Tabla X para la imagen (d), donde el tiempo de segmentación se incrementa solo en 1.97 segundos adicionales. Este tiempo extra es compensado por la mayor precisión en la segmentación que ofrece nuestro modelo, aprovechando al máximo las capacidades del modelo SAM tipo H para obtener

resultados detallados y precisos. La Figura XLVI proporciona una visualización detallada que facilita la comparación de la precisión en la segmentación obtenida por los distintos métodos. Esta representación gráfica ofrece una perspectiva más profunda sobre la efectividad de cada enfoque.

La evaluación entre el ajuste fino del modelo SAM y nuestra propuesta subraya una distinción crítica en el consumo computacional. El ajuste fino del SAM tipo B, que integra 91 millones de parámetros, demanda una inversión significativa de recursos, necesitando 25 GB de RAM y alrededor de 22.77 horas para entrenar 1010 imágenes en una GPU 3070 Ti, como se describe en la sección 4.1 del Anexo 4. A pesar del elevado costo computacional inicial, este modelo ajustado se caracteriza por una eficiencia superior en la segmentación, consumiendo menos tiempo una vez que está entrenado. Por contraste, nuestro enfoque, que se concentra en el entrenamiento de un clasificador de regresión logística, es notablemente más rápido en la fase de entrenamiento, finalizando en solo 1.58 segundos en una GPU RTX 3070. Sin embargo, al emplear el SAM tipo H en la segmentación, con 636 millones de parámetros y la adición de puntos y cajas delimitadoras para una precisión mejorada, nuestro modelo incurre en un mayor costo computacional durante su operación normal en comparación con el SAM ajustado. Aunque este enfoque aumenta el tiempo de procesamiento, como se evidencia en la Tabla X para la imagen (d), donde el incremento es de 1.97 segundos, la inversión se justifica con una precisión de segmentación superior, aprovechando plenamente las capacidades del SAM tipo H. La Figura XLVI ilustra esta ventaja en términos de precisión, proporcionando una comparación visual de la eficacia entre los diferentes métodos.

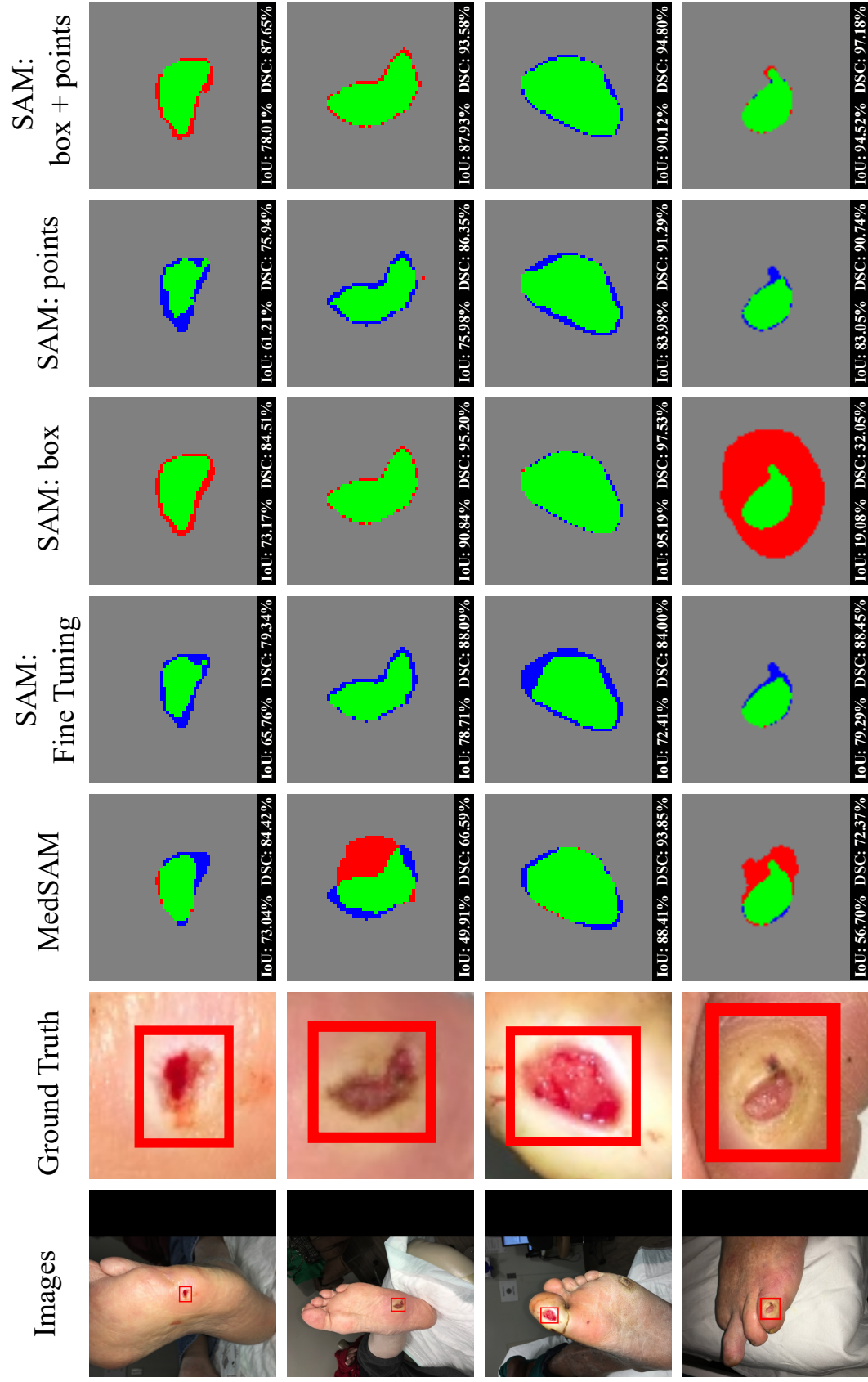


Figura XLVI: Comparativa de segmentación con diferentes enfoques de SAM, donde las imágenes en colores representan verdaderos positivos en verde, falsos negativos en azul, verdaderos negativos en gris, junto con métricas IoU y DSC.

VI. Conclusiones

En cumplimiento de uno de nuestros objetivos, este estudio ha realizado una búsqueda exhaustiva y ha seleccionado con éxito el conjunto de datos de la Diabetic Foot Ulcer Challenge 2021, que cuenta con la aprobación ética del Comité de Ética de Investigación del NHS del Reino Unido. Este conjunto, compuesto por 810 imágenes para entrenamiento y 400 para validación y pruebas, no solo ha proporcionado una base sólida para nuestro análisis, sino que también ha asegurado el cumplimiento de los estándares éticos y de calidad requeridos en la investigación científica, garantizando que nuestro estudio sobre las lesiones del pie diabético se realice en un marco ético y responsable.

En relación con el objetivo de entrenar eficazmente el modelo YOLOv5 para la detección de lesiones en imágenes de pie diabético, el estudio ha implementado un proceso de preprocesamiento y entrenamiento. A lo largo de 200 épocas, utilizando optimizadores como SGD y ADAM, el modelo YOLOv5 ha alcanzado un rendimiento sobresaliente, con una sensibilidad de 0.966 y un mAP@.5 de 0.972. Estos resultados no solo cumplen con nuestro objetivo de lograr una alta precisión y eficiencia en la identificación de áreas lesionadas, sino que también destacan la capacidad del YOLOv5 para segmentar de manera efectiva las lesiones en el pie diabético.

En línea con nuestro objetivo de desarrollar un método avanzado de extracción de características para imágenes de pie diabético, se implementó una estrategia innovadora que combina técnicas de procesamiento de imágenes y aprendizaje automático. Utilizando el método SLIC, se generaron superpíxeles que luego se etiquetaron basándose en su intersección con las máscaras de lesiones, empleando un umbral de 180 píxeles para diferenciar entre áreas lesionadas y no lesionadas. Cada superpíxel se caracterizó con un vector de atributos derivado de los 128 mapas de características de la quinta capa de VGG16, además de los canales RGB originales, resultando en 264 características distintas, estas características esta compuesta por la desviación estándar y la media, así como

las coordenadas del centroide del superpíxel. Esta técnica avanzada de extracción de características ha permitido capturar una amplia información relevante, facilitando así una segmentación más precisa y efectiva de las lesiones en el pie diabético.

En el marco de nuestro objetivo de implementar técnicas efectivas para identificar y extraer puntos representativos en imágenes de pie diabético, se seleccionó un modelo de regresión logística. Este modelo resultó ser un método adecuado en la clasificación binaria de los superpíxeles, aprovechando su habilidad para analizar 264 características por superpíxel, que incluyen elementos como color y posición. Esta capacidad analítica permitió una distinción clara entre lesiones y áreas no lesionadas. El modelo fue entrenado específicamente para seleccionar los cinco superpíxeles con la mayor probabilidad de representar una lesión y otros diez con la menor probabilidad. Estos puntos seleccionados se utilizaron luego como datos de entrada para el modelo SAM en su proceso de segmentación, cumpliendo efectivamente con el objetivo propuesto de utilizar puntos representativos para mejorar la precisión en la segmentación.

En cumplimiento del objetivo de integrar y adaptar eficazmente el modelo SAM para la segmentación de lesiones en pie diabético, desarrollamos el siguiente proceso. Inicialmente, el modelo YOLOv5 identificó áreas de interés en las imágenes, marcándolas con cuadros delimitadores. Estas áreas marcadas fueron posteriormente procesadas mediante superresolución con el modelo EDSR, lo que mejoró notablemente la calidad y el detalle de las imágenes. Ajustamos las coordenadas de los recortes mejorados para que coincidieran con la escala de las imágenes originales. Luego, el modelo de regresión logística proporcionó coordenadas precisas de puntos representativos de lesiones y áreas no lesionadas. Estos puntos, en conjunto con los cuadros delimitadores, se utilizaron como entradas para el modelo SAM. Este enfoque permitió una segmentación final altamente precisa y detallada, basándose en una combinación de inputs específicos y el contexto general proporcionado por la imagen, alineándose así con el objetivo de generar máscaras de segmentación precisas.

Este estudio ha logrado su objetivo principal al desarrollar un sistema avanzado y automatizado para la segmentación precisa de lesiones cutáneas en el pie diabético, destacando por la integración del modelo SAM, una técnica basada en los recientes avances de los transformers. Originalmente aplicados en el procesamiento de lenguaje natural, como en sistemas tipo Chat GPT, los transformers representan el vanguardismo en el campo de la visión computacional. La efectividad de combinar el modelo SAM con técnicas de Deep Learning y Machine Learning no solo mejora el rendimiento en métricas como IoU y el coeficiente de Dice, sino que también equilibra la precisión con la complejidad computacional. Esta metodología avanzada no solo subraya el potencial de nuestro enfoque en el monitoreo del pie diabético, sino que también destaca su capacidad para extenderse a otras áreas de imágenes médicas, marcando así un avance significativo en el desarrollo de tecnologías de segmentación de imágenes médicas.

VII. Referencias bibliográficas

- [1] G. Seguel, «¿Por qué debemos preocuparnos del pie diabético? Importancia del pie diabético,» *Revista Médica de Chile*, vol. 141, n.º 11, págs. 1464-1469, 2013. DOI: 10.4067/S0034-98872013001100014.
- [2] Yorgi, V. Gil, J. Pacheco, I. Benítez y M. Sánchez, *Evaluación y tratamiento del pie diabético. Protocolo del Servicio de Endocrinología del Instituto Autónomo Hospital Universitario de Los Andes*. 2012. dirección: http://ve.scielo.org/scielo.php?script=sci_arttext&pid=S1690-31102012000300008&lng=es&nrm=iso&tlng=es.
- [3] R. H. L. e Silva y A. M. C. Machado, «Automatic measurement of pressure ulcers using Support Vector Machines and GrabCut,» *Computer Methods and Programs in Biomedicine*, vol. 200, pág. 105867, 2021. DOI: 10.1016/j.cmpb.2020.105867.
- [4] X. Xie, J. Niu, X. Liu, Z. Chen, S. Tang y S. Yu, «A survey on incorporating domain knowledge into deep learning for medical image analysis,» *Medical Image Analysis*, vol. 69, 2021. DOI: 10.1016/j.media.2021.101985.
- [5] D. H. Keast, C. K. Bowering, A. W. Evans, G. L. Mackean, C. Burrows y L. D'Souza, «MEASURE: A Proposed Assessment Framework for Developing Best Practice Recommendations for Wound Assessment,» *Wound Repair and Regeneration*, vol. 12, n.º s1, S1-S17, 2004. DOI: 10.1111/j.1067-1927.2004.012351.x.
- [6] R. Zhang, D. Tian, D. Xu, W. Qian e Y. Yao, «A Survey of Wound Image Analysis Using Deep Learning: Classification, Detection, and Segmentation,» *IEEE Access*, vol. 10, 2022. DOI: 10.1109/ACCESS.2022.3194529.
- [7] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov y S. Zagoruyko, «End-to-End Object Detection with Transformers,» en *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, Glasgow, UK, 2020, págs. 213-229. DOI: 10.1007/978-3-030-58452-8_13.
- [8] M. Goyal, N. D. Reeves, A. K. Davison, S. Rajbhandari, J. Spragg y M. H. Yap, «DFUNet: Convolutional Neural Networks for Diabetic Foot Ulcer Classification,» *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, n.º 5, págs. 728-739, 2018. DOI: 10.1109/tetci.2018.2866254.
- [9] A. Dosovitskiy et al., «An Image Is Worth 16×16 Words: Transformers for Image Recognition at Scale,» *arXiv preprint arXiv:2010.11929*, 2021. DOI: 10.48550/arXiv.2010.11929.

- [10] A. Kirillov et al., «Segment Anything Model,» *arXiv preprint arXiv:2304.02643*, abr. de 2023. dirección: <https://arxiv.org/abs/2304.02643>.
- [11] J. Ma, Y. He, F. Li, L. Han, C. You y B. Wang, «Segment Anything in Medical Images,» *arXiv preprint arXiv:2304.12306*, 2023. DOI: 10.48550/arXiv.2304.12306.
- [12] C. Solomon y T. Breckon, *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. John Wiley & Sons, Ltd, 2011.
- [13] R. C. Gonzalez y R. E. Woods, *Digital Image Processing*, 4th. New York, NY: Pearson, 2018.
- [14] G. Sharma, *Digital Color Imaging Handbook*. USA: CRC Press, 2002.
- [15] MathWorks, *Understanding Color Spaces and Color Space Conversion*, feb. de 2024. dirección: <https://www.mathworks.com/help/images/understanding-color-spaces-and-color-space-conversion.html>.
- [16] A. R. Smith, «Color gamut transform pairs,» *Communications of the ACM*, vol. 12, n.º 3, págs. 12-19, 1978. DOI: 10.1145/965139.807361.
- [17] J.-D. Chang, S.-S. Yu, H.-H. Chen y C.-S. Tsai, «HSV-based Color Texture Image Classification using Wavelet Transform and Motif Patterns,» *Journal of Computers*, vol. 20, n.º 4, págs. 1-10, 2010. dirección: http://www.csroc.org.tw/journal/Published%20Vol_20_No_4_FULL_files/6.%20joc%2098032.pdf.
- [18] M. A. Carmona Concha, «Análisis mineralógico de escoria de fundición usando sistema de visión,» Informe de memoria de título para optar al título de Ingeniero Civil Electrónico, Universidad de Concepción, Concepción, Chile, 2012. dirección: <https://repositorio.udec.cl/server/api/core/bitstreams/e4c581f8-e887-4c22-be02-2da697f1b2db/content>.
- [19] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua y S. Süsstrunk, «SLIC Superpixels Compared to State-of-the-Art Superpixel Methods,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, n.º 11, págs. 2274-2281, 2012. DOI: 10.1109/TPAMI.2012.120.
- [20] X. Ren y J. Malik, «Learning a Classification Model for Segmentation,» *Proceedings of the Ninth IEEE International Conference on Computer Vision*, págs. 10-17, 2003. DOI: 10.1109/ICCV.2003.1238308.
- [21] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson y K. Siddiqi, «TurboPixels: Fast Superpixels Using Geometric Flows,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, n.º 12, págs. 2290-2297, 2009. DOI: 10.1109/TPAMI.2009.96.

- [22] A. C. Müller y S. Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists*, 1st. Sebastopol, CA: O'Reilly Media, 2016.
- [23] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY: Springer, 2006.
- [24] M. Negnevitsky, *Artificial Intelligence: A Guide to Intelligent Systems*. Edinburgh: Pearson Education Limited, 2005.
- [25] D. de Matemática Aplicada de la Universidad Politécnica de Madrid, *Introducción al Aprendizaje Automático*, 2021. dirección: https://dcain.etsin.upm.es/~carlos/bookAA/05.1_RedesNeuronalesIntroduccion.html.
- [26] F. J. Ascarza Mendoza, «Segmentación automática de textos, mediante redes neuronales convolucionales en imágenes de documentos históricos,» Tesis de Magíster, Pontificia Universidad Católica del Perú, Lima, Perú, 2018. dirección: <http://tesis.pucp.edu.pe/repositorio/handle/123456789/13050>.
- [27] D. Erroz Arroyo, *Visualizando neuronas en Redes Neuronales Convolucionales*, 2019. dirección: https://academica-e.unavarra.es/xmlui/bitstream/handle/2454/33694/memoria_TFG.pdf?sequence=1&isAllowed=y.
- [28] M. M. Taye, «Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions,» *Computation*, vol. 11, n.º 3, pág. 52, 2023. DOI: 10.3390/computation11030052.
- [29] A. Moreno, «Clasificación de imágenes usando redes neuronales convolucionales en Python,» Trabajo de suficiencia profesional, Universidad de Sevilla, Sevilla, España, 2019. dirección: <https://idus.us.es/server/api/core/bitstreams/c22e9088-218a-4554-9b27-6c0ebf89d0e8/content>.
- [30] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.
- [31] J. L. Gonzalez, *Tipos de Aprendizaje Automático*, 2018. dirección: <https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2>.
- [32] T. Hastie, R. Tibshirani y J. Friedman, *The Elements of Statistical Learning*, 2nd. New York, NY: Springer, 2009.
- [33] R. S. Sutton y A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd. Cambridge, MA; London, England: The MIT Press, 2018.
- [34] M. Bernacki, P. Włodarczyk y A. Gólda, *Principles of training multi-layer neural network using backpropagation*, sep. de 2004. dirección: http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html.
- [35] F. Chollet, *Deep Learning with Python*. Shelter Island, NY: Manning Publications Co., 2018.

- [36] S. Haykin, *Neural Networks and Learning Machines*, 3rd. Canada: Pearson, 2009.
- [37] Y. LeCun, Y. Bengio y G. Hinton, «Deep Learning,» *Nature*, vol. 521, n.º 7553, págs. 436-444, 2015. DOI: 10.1038/nature14539.
- [38] A. L. Maas, A. Y. Hannun y A. Y. Ng, «Rectifier Nonlinearities Improve Neural Network Acoustic Models,» en *Proceedings of the 30th International Conference on Machine Learning (ICML)*, Atlanta, USA, 2013.
- [39] D.-A. Clevert, T. Unterthiner y S. Hochreiter, «Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),» 2016. arXiv: 1511.07289 [cs.LG]. dirección: <https://arxiv.org/abs/1511.07289>.
- [40] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning Representations by Back-Propagating Errors,» *Nature*, vol. 323, págs. 533-536, 1986. DOI: 10.1038/323533a0.
- [41] H. Robbins y S. Monro, «A Stochastic Approximation Method,» *The Annals of Mathematical Statistics*, vol. 22, n.º 3, págs. 400-407, 1951. DOI: 10.1214/aoms/1177729586.
- [42] L. Bottou, «Large-Scale Machine Learning with Stochastic Gradient Descent,» en *Proceedings of COMPSTAT'2010*, 2010, págs. 177-186. DOI: 10.1007/978-3-7908-2604-3_16.
- [43] D. P. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization,» *arXiv preprint arXiv:1412.6980*, 2014. dirección: <https://arxiv.org/abs/1412.6980>.
- [44] S. Bhardwaj, R. R. Curtin, M. Edel, Y. Mentekidis y C. Sanderson, *ensmallen: a flexible C++ library for efficient function optimization*, dic. de 2018. dirección: <https://doi.org/10.5281/zenodo.2008650>.
- [45] R. Ng., *Learning Rate Scheduling – Deep Learning Wizard*, feb. de 2024. dirección: https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/lr_scheduling/.
- [46] D. Masters y C. Luschi, «Revisiting Small Batch Training for Deep Neural Networks,» 2018. arXiv: 1804.07612 [cs.LG]. dirección: <https://arxiv.org/abs/1804.07612>.
- [47] L. N. Smith, «Don't Decay the Learning Rate, Increase It,» en *Proceedings of the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Santa Rosa, CA, 2017, págs. 464-472. DOI: 10.1109/WACV.2017.58.
- [48] O. Russakovsky et al., «Imagenet large scale visual recognition challenge,» *International Journal of Computer Vision*, vol. 115, n.º 3, págs. 211-252, 2015. DOI: <https://doi.org/10.1007/s11263-015-0816-y>.

- [49] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang y C. Liu, «A survey on deep transfer learning,» *In the 27th International Conference on Artificial Neural Networks*, págs. 270-279, 2018. DOI: https://doi.org/10.1007/978-3-030-01424-7_27.
- [50] C. A. M. Victoria y G. A. D. Narvaes, «Detección y seguimiento de múltiples objetos en tiempo real para vehículos autónomos,» *J Chem Inf Model*, vol. 53, n.º 9, págs. 1689-1699, 2019. dirección: <http://red.uao.edu.co/bitstream/10614/12295/5/T09147.pdf>.
- [51] K. Simonyan y A. Zisserman, «Very deep convolutional networks for large-scale image recognition,» *arXiv preprint arXiv:1409.1556*, 2015. dirección: <https://www.robots.ox.ac.uk/~vgg/publications/2015/Simonyan15/>.
- [52] M. D. Zeiler y R. Fergus, «Visualizing and Understanding Convolutional Networks,» en *European Conference on Computer Vision (ECCV)*, 2014, págs. 818-833.
- [53] M. A. Bravo-Ortiz, H. B. Arteaga Arteaga, R. Tabares Soto, J. I. Padilla Buriticá y S. Orozco-Arias, «Cervical cancer classification using convolutional neural networks, transfer learning and data augmentation,» *Revista EIA*, vol. 18, n.º 35, págs. 100-111, nov. de 2021, ISSN: 1794-1237. DOI: 10.24050/reia.v18i35.1462.
- [54] D. Frossard, *VGG in TensorFlow*, jun. de 2016. dirección: <https://www.cs.toronto.edu/~frossard/post/vgg16/>.
- [55] S. R. S. Osses, «Detección de prendas de vestir utilizando modelos de detección de objetos basados en Deep Learning,» Memoria para optar al título de Ingeniero Civil Eléctrico, Universidad de Chile, Santiago, Chile, 2020. dirección: <https://repositorio.uchile.cl/handle/2250/176546>.
- [56] F. E. Souto Herrera, «Deep Local Directional Embeddings para detección de objetos,» Tesis de Magíster, Universidad de Chile, Santiago, Chile, 2020. dirección: <https://repositorio.uchile.cl/handle/2250/176745>.
- [57] J. Yao, J. Qi, J. Zhang, H. Shao, J. Yang y X. Li, «A Real-Time Detection Algorithm for Kiwifruit Defects Based on YOLOv5,» *Electronics*, vol. 10, n.º 17, pág. 2111, 2021. DOI: 10.3390/electronics10141711.
- [58] R. Xu, H. Lin, K. Lu et al., «A forest fire detection system based on ensemble learning,» *Forests*, vol. 12, n.º 2, pág. 217, 2021. DOI: 10.3390/f12020217.
- [59] Z. Yu, Y. Lei, F. Shen y S. Zhou, «Application of Improved YOLOv5 Algorithm in Lightweight Transmission Line Small Target Defect Detection,» *Electronics*, vol. 13, n.º 2, pág. 305, 2024. DOI: 10.3390/electronics13020305.
- [60] Ultralytics, *YOLOv5*, 2020. dirección: <https://github.com/ultralytics/yolov5>.

- [61] M. Sezgin y B. Sankur, «Survey over image thresholding techniques and quantitative performance evaluation,» *Journal of Electronic Imaging*, vol. 13, n.º 1, págs. 146-165, 2004. DOI: 10.1117/1.1631315.
- [62] A. K. Jain, M. N. Murty y P. J. Flynn, «Data clustering: a review,» *ACM computing surveys (CSUR)*, vol. 31, n.º 3, págs. 264-323, 1999.
- [63] J. Canny, «A Computational Approach to Edge Detection,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, n.º 6, págs. 679-698, 1986. DOI: 10.1109/TPAMI.1986.4767851.
- [64] A. Trémeau y P. Colantoni, «A region growing and merging algorithm to color segmentation,» *Pattern Recognition*, vol. 33, n.º 7, págs. 1191-1203, 2000. DOI: [https://doi.org/10.1016/S0031-3203\(96\)00147-1](https://doi.org/10.1016/S0031-3203(96)00147-1).
- [65] C. J. C. Burges, «A Tutorial on Support Vector Machines for Pattern Recognition,» *Data Mining and Knowledge Discovery*, vol. 2, n.º 2, págs. 121-167, 1998. DOI: 10.1023/A:1009715923555.
- [66] N. R. Pal y S. K. Pal, «A Review on Image Segmentation Techniques,» *Pattern Recognition*, vol. 26, n.º 9, págs. 1277-1294, 1993. DOI: 10.1016/0031-3203(93)90135-J.
- [67] J. Dongarra, P. Luszczek y A. Petit, «The LINPACK Benchmark: Past, Present and Future,» *Concurrency and Computation: Practice and Experience*, vol. 15, n.º 9, págs. 803-820, 2003. DOI: 10.1002/cpe.728.
- [68] H. Wickham, «Tidy data,» *Journal of Statistical Software*, vol. 59, n.º 10, págs. 1-23, 2014. DOI: 10.18637/jss.v059.i10.
- [69] X. Chu, I. F. Ilyas, S. Krishnan y J. Wang, «Data Cleaning: Overview and Emerging Challenges,» en *Proceedings of the 2016 International Conference on Management of Data*, 2016, págs. 2201-2206. DOI: 10.1145/2882903.2912574.
- [70] S. Kandel et al., «Research Directions in Data Wrangling: Visualizations and Transformations for Usable and Credible Data,» *Information Visualization*, vol. 10, n.º 4, págs. 271-288, 2011. DOI: 10.1177/1473871611415994.
- [71] B. Lim, S. Son, H. Kim, S. Nah y K. M. Lee, «Enhanced Deep Residual Networks for Single Image Super-Resolution,» en *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Honolulu, USA, 2017, págs. 1132-1140. DOI: 10.1109/CVPRW.2017.151.
- [72] G. Chawda, *Enhanced Deep Residual Networks for Single Image Super-Resolution*, abr. de 2022. dirección: <https://keras.io/examples/vision/edsr/>.

- [73] S. M. A. Bashir, Y. Wang, M. Khan e Y. Niu, «A Comprehensive Review of Deep Learning–Based Single Image Super-Resolution,» *PeerJ Comput Sci*, vol. 7, págs. 1-56, 2021. DOI: 10.7717/PEERJ-CS.621.
- [74] A. Vaswani et al., «Attention Is All You Need,» en *Advances in Neural Information Processing Systems*, I. Guyon et al., eds., vol. 30, Long Beach California, USA: Curran Associates, Inc., 2017. dirección: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- [75] S. Amidi y A. Amidi. «CS 229 - Machine Learning Tips and Tricks Cheatsheet.» dirección: <https://web.stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricks>.
- [76] T. Fawcett, «An Introduction to ROC Analysis,» *Pattern Recognition Letters*, vol. 27, n.º 8, págs. 861-874, 2006. DOI: 10.1016/j.patrec.2005.10.010.
- [77] D. M. W. Powers, «Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation,» *Journal of Machine Learning Technologies*, vol. 2, n.º 1, págs. 37-63, 2011. dirección: <https://arxiv.org/abs/2010.16061>.
- [78] M. Everingham, L. Van Gool, C. K. Williams, J. Winn y A. Zisserman, «The Pascal Visual Object Classes (VOC) Challenge,» *International Journal of Computer Vision*, vol. 88, n.º 2, págs. 303-338, 2010. DOI: 10.1007/s11263-009-0275-4.
- [79] G. Hager y G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton, FL: CRC Press, 2010.
- [80] J. A. Stratton et al., «Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing,» en *2012 IEEE 18th International Symposium on High-Performance Computer Architecture*, 2012, págs. 1-11. DOI: 10.1109/HPCA.2012.6168960.
- [81] S. Williams, A. Waterman y D. Patterson, «Roofline: An insightful visual performance model for multicore architectures,» *Communications of the ACM*, vol. 52, n.º 4, págs. 65-76, 2009. DOI: 10.1145/1498765.1498785.
- [82] R. Schwartz, J. Dodge, N. A. Smith y O. Etzioni, «Green AI,» *Communications of the ACM*, vol. 63, n.º 12, págs. 54-63, 2020. DOI: 10.1145/3381831.
- [83] C. Wang et al., «Fully Automatic Wound Segmentation with Deep Convolutional Neural Networks,» *Scientific Reports*, vol. 10, pág. 21 897, 2020. DOI: 10.1038/s41598-020-79006-1.
- [84] M. 2. F. U. S. Challenge, *Foot Ulcer Segmentation Challenge 2021*, 2021. dirección: <https://fusc.grand-challenge.org/>.

- [85] A. Mahbod, G. Schaefer, R. Ecker e I. Ellinger, «Automatic Foot Ulcer Segmentation Using an Ensemble of Convolutional Neural Networks,» *arXiv preprint arXiv:2109.01408*, 2022. DOI: 10.48550/arXiv.2109.01408.
- [86] Q. Zhou, T. He e Y. Zou, «Superpixel-Oriented Label Distribution Learning for Skin Lesion Segmentation,» *Diagnostics*, vol. 12, n.º 4, 2022. DOI: 10.3390/diagnostics12040938.
- [87] G. B. et al., «A superpixel-driven deep learning approach for the analysis of dermatological wounds,» *Comput Methods Programs Biomed*, vol. 183, 2020. DOI: 10.1016/j.cmpb.2019.105079.
- [88] V. Kudva, K. Prasad y S. Guruvare, «Detection of Specular Reflection and Segmentation of Cervix Region in Uterine Cervix Images for Cervical Cancer Screening,» *IRBM*, vol. 38, n.º 5, 2017. DOI: 10.1016/j.irbm.2017.08.003.
- [89] J. H. Giraldo, V. Scarrica, A. Staiano, F. Camastra y T. Bouwmans, «Hypergraph Convolutional Networks for Weakly-Supervised Semantic Segmentation,» en *2022 IEEE International Conference on Image Processing (ICIP)*, Bordeaux France, 2022, págs. 16-20. DOI: 10.1109/ICIP46576.2022.9897774.
- [90] C. Cui et al., «Diabetic Wound Segmentation using Convolutional Neural Networks,» *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 2019, págs. 1002-1005, 2019. DOI: 10.1109/EMBC.2019.8856665.
- [91] P. L. Muñoz, R. Rodríguez y N. Montalvo, «Automatic Segmentation of Diabetic Foot Ulcer using Mask Region-Based Convolutional Neural Networks,» *Journal of Biomedical Research and Clinical Investigation*, vol. 1, n.º 1, pág. 1006, 2020. DOI: 10.31546/2633-8653.1006.
- [92] N. Rania, H. Douzi, Y. Lucas y S. Treuillet, «Semantic Segmentation of Diabetic Foot Ulcer Images: Dealing with Small Dataset in DL Approaches,» en *International Conference on Image and Signal Processing (ICISP 2020)*, Orléans, France, 2020, págs. 162-169. DOI: 10.1007/978-3-030-51935-3_17.
- [93] B. Monroy, J. Bacca, K. Sánchez, H. Arguello y S. Castillo, «Two-Step Deep Learning Framework for Chronic Wounds Detection and Segmentation: A Case Study in Colombia,» en *2021 IEEE International Conference on Biomedical Imaging (ISBI)*, Bucaramanga, Colombia, 2021. DOI: 10.1109/ISBI.2021.9453802.
- [94] Y. Li, H. Mao, R. Girshick y K. He, «Exploring Plain Vision Transformer Backbones for Object Detection,» *arXiv preprint arXiv:2203.16527*, 2022. DOI: 10.48550/arXiv.2203.16527.
- [95] Keras, *Enhanced Deep Residual Networks for Super-Resolution*, ene. de 2024. dirección: <https://keras.io/examples/vision/edsr/>.

- [96] Keras, *VGG models for Keras*, ene. de 2024. dirección: <https://keras.io/api/applications/vgg/>.

Anexos

Anexo 1: Algoritmos	1
Anexo 2: Análisis computacional	6
Anexo 3: Interfaz web	10
Anexo 4: Fine-tuning SAM	14

Anexo 1: Algoritmos

1.1. Algoritmo de filtro especular

Para información detallada y código fuente del algoritmo de filtro especular, visite el repositorio de GitHub.

Algorithm 1 Filtro Especular Detallado

Require: Imagen original (img), Máscara (msk)

Ensure: Imagen filtrada (img_comp), Máscara umbralizada (img_umbral)

Inicio:

Umbralizar msk:

$umbral, msk_fil \leftarrow umbralizar(msk, 100, 255)$

Aplicar msk a img:

$img_seg \leftarrow aplicarMascara(img, msk_fil)$

Convertir img_seg a HSV, RGB, LAB:

$hsv \leftarrow aHSV(img_seg)/255$

$verde \leftarrow aRGB(img_seg)[: , : , 1]/255$

$lum \leftarrow aLAB(img_seg)[: , : , 0]/100$

Calcular media de verde y aplicar función exponencial:

$verde_med \leftarrow media(verde, msk_fil)$

$exp \leftarrow \text{CalcularExponente}(verde_med)$ (ver Algoritmo 2)

Imagen de características:

$img_car \leftarrow ((1 - hsv[: , : , 1]) \times verde \times lum)^{exp}$

Suavizar imagen de características (img_car):

Aplicar un filtro Gaussiano

$img_fil \leftarrow suavizar(img_car, kernel = (3, 3))$

Normalizar imagen filtrada (img_fil):

Calcular la media y desviación estándar de la imagen suavizada

$media, std \leftarrow calcularMediaYDesviacion(img_fil)$

Normalizar restando la media y dividiendo por la desviación estándar

$img_fil \leftarrow (img_fil - media)/std$

Umbralizar img_fil:

Aplicar umbral binario con valores 0.3 y 1 para destacar áreas de interés

$img_umbral \leftarrow umbralizar(img_fil, 0.3, 1)$

Inpainting en img con img_umbral:

Rellenar las áreas umbralizadas en la imagen original usando inpainting con un radio de 10 píxeles.

$img_comp \leftarrow inpainting(img, img_umbral, 10)$

Fin.

return img_comp, img_umbral

Algorithm 2 **CalcularExponente:** Cálculo del exponente para el ajuste de escala del canal verde

Require: x_1 (media del canal verde)

Ensure: Valor del exponente calculado

Inicio:

Calcular el exponente basado en x_1 :

$$a \leftarrow (\log(10) - \log(3)) / (\log(200) - \log(8))$$

$$b \leftarrow \log(3) - a * \log(8)$$

$$exponente \leftarrow \exp(a * \log(x_1) + b)$$

Fin.

return $exponente$

1.2. Extracción de vectores de características

Para información detallada y código fuente del algoritmo de extracción de vectores de características, visite el repositorio de GitHub.

Algorithm 3 Extracción Compacta de Características de Imagen (RGB y VGG)

Require: Img, Msk, Modelo VGG16, Parámetros SLIC (n_seg, cmpct, sig), Umbral

Ensure: Características de Img

Inicio:

Cargar VGG16

Inicializar listas para características y etiquetas

Segmentación SLIC en Img (n_seg, cmpct, sig)

Calcular máscaras y centroides de segmentos

Clasificar regiones con Msk y Umbral

Extraer características RGB (varianza, media, etc.)

Procesar con VGG16 (capas profundas)

Extraer características de canales VGG16

Acumular características y etiquetas

Crear DataFrame con datos acumulados

if guardar imágenes/resultados **then**

 Guardar imágenes de segmentos y clasificaciones

 Opcional: Guardar imágenes VGG16 si aplica

end if

Fin.

return DataFrame de Img

1.3. Sistema de segmentación de lesiones

Para información detallada y código fuente del sistema completo de segmentación, visite el repositorio de GitHub.

Algorithm 4 Segmentación Automatizada de Lesiones en Imágenes

Require: Imágenes, Modelo YOLOv5, Modelo SAM, Modelo Reg. Logística y EDSR

Ensure: Segmentaciones

Inicio:

Preparar imágenes para YOLOv5

Cargar modelos YOLOv5 y SAM

Configurar YOLOv5 y SAM (checkpoint, device, etc.)

Crear directorios para resultados

for cada imagen en el dataset **do**

 Mover imagen a directorio de YOLOv5

 Detectar áreas de interés con YOLOv5

for cada área detectada **do**

 Extraer y almacenar recorte

if tamaño del recorte < 100x100px **then**

 Aplicar super-resolución (EDSR) al recorte

end if

 Segmentar con Reg. Logística y SAM

 Ajustar y visualizar segmentaciones

 Generar y combinar máscaras de segmentación

 Superponer máscara en imagen original

 Guardar imagen con máscara y segmentaciones

end for

 Limpiar archivos temporales

end for

Fin.

return Segmentaciones almacenadas

Anexo 2: Análisis computacional

2.1. Cálculo del gasto computacional del modelo EDSR

Descripción del Modelo

El modelo EDSR se compone de varias capas, incluyendo una capa convolucional inicial, $N_{\text{res}} = 16$ bloques residuales, una capa convolucional posterior a los bloques residuales, un bloque de submuestreo y una capa convolucional final. La imagen de entrada es de 100×100 píxeles con 3 canales (RGB) [95].

Parámetros del Modelo

- Número de Filtros: $N_{\text{filtros}} = 64$
- Tamaño del Filtro: $F = 3 \times 3$ (donde $K = 3$)
- Factor de Submuestreo: $s = 2$

Cálculo de FLOPs

Utilizando la ecuación 10 calcularemos el gasto computacional, en GFLOPs (G) se calcula como:

$$G = \frac{1}{10^9} (G_{\text{inicial}} + G_{\text{residual}} + G_{\text{post-residual}} + G_{\text{submuestreo}} + G_{\text{final}})$$

- Capa Convolucional Inicial:

$$G_{\text{inicial}} = 2 \times 3^2 \times 64 \times 100^2 \times 3$$

- Bloques Residuales:

$$G_{\text{residual}} = N_{\text{res}} \times 2 \times 3^2 \times 64 \times 100^2 \times 64$$

- Capa Convolucional Después de los Bloques Residuales:

$$G_{\text{post-residual}} = 2 \times 3^2 \times 64 \times 100^2 \times 64$$

- Bloque de Submuestreo:

$$G_{\text{submuestreo}} = 2 \times 3^2 \times 64 \times (100/s)^2 \times 64$$

- Capa Convolutiva Final:

$$G_{\text{final}} = 2 \times 3^2 \times 64 \times 100^2 \times 3$$

Resultado

El gasto computacional total del modelo EDSR es:

$$G_{\text{total}} = \frac{1}{10^9} (G_{\text{inicial}} + G_{\text{residual}} + G_{\text{post-residual}} + G_{\text{submuestreo}} + G_{\text{final}}) \approx 1,49 \text{ GFLOPs}$$

2.2. Cálculo del gasto computacional del modelo VGG16

Descripción del Modelo

El modelo VGG16 es una red convolutiva profunda que incluye 13 capas convolucionales y 3 capas completamente conectadas. Las primeras cinco capas convolucionales, que son el foco de este cálculo, utilizan filtros de 3×3 y aumentan el número de filtros progresivamente [96].

Parámetros del Modelo

- Tamaño de la Imagen de Entrada: 224×224 píxeles
- Número de Filtros en las Capas Convolucionales 1 y 2: $N_{\text{filtros1}} = 64$
- Número de Filtros en las Capas Convolucionales 3, 4 y 5: $N_{\text{filtros2}} = 128$
- Tamaño del Filtro: $F = 3 \times 3$ (donde $K = 3$)

Cálculo de FLOPs

Utilizando la ecuación 10, el gasto computacional en GFLOPs (G) para las primeras cinco capas se calcula como:

$$G = \frac{1}{10^9} (G_{\text{conv1}} + G_{\text{conv2}} + G_{\text{conv3}} + G_{\text{conv4}} + G_{\text{conv5}})$$

- Para las Capas Convolucionales 1 y 2:

$$G_{\text{conv1}} = G_{\text{conv2}} = 2 \times K^2 \times N_{\text{filtros1}} \times 224^2 \times 3$$

- Para las Capas Convolucionales 3, 4 y 5:

$$G_{\text{conv3}} = G_{\text{conv4}} = G_{\text{conv5}} = 2 \times K^2 \times N_{\text{filtros2}} \times 112^2 \times 64$$

Resultado

El gasto computacional total para las primeras cinco capas del modelo VGG16 es:

$$G_{\text{total}} = \frac{1}{10^9} (2 \times G_{\text{conv1}} + 3 \times G_{\text{conv3}}) \approx 18,495 \text{ GFLOPs}$$

2.3. Cálculo del gasto computacional del modelo de Regresión Logística

Descripción del modelo

El modelo de **regresión logística** implementado utiliza la clase ‘LogisticRegression’ de la biblioteca Scikit-Learn (‘sklearn’). Se emplean $N = 264$ características de entrada para realizar predicciones. Las operaciones matemáticas involucradas en cada predicción son las siguientes:

Parámetros del modelo

- Número de Características (N): 264
- Operaciones en la Función Sigmoide: Exponenciación, Suma, División

Cálculo de operaciones

El número total de operaciones matemáticas (T_{ops}) requerido para una predicción se calcula como sigue:

■ Operaciones Lineales:

- Multiplicaciones: $N = 264$ (cada característica se multiplica por su peso correspondiente)

- Sumas: $N = 264$ (cada resultado se suma al acumulado)

■ **Término de Sesgo:**

- Sumas adicionales: 1 (se añade el término de sesgo a la suma acumulada)

■ **Función Sigmoide ($\sigma(x)$):**

- Exponenciaciones: 1 (para calcular e^{-x})
- Sumas: 1 (para calcular $1 + e^{-x}$)
- Divisiones: 1 (para obtener el valor final de $\sigma(x)$)

Resultado

El número total de operaciones matemáticas para realizar una predicción en el modelo de regresión logística de Scikit-Learn es:

$$T_{\text{ops}} = 2 \times N + 3 = 2 \times 264 + 3 = 531 \text{ operaciones}$$

Este cálculo proporciona una estimación del gasto computacional para una predicción individual, sin incluir el costo de implementación de la función exponencial, que puede variar a diferencias en la implementación del software, la precisión numérica, las optimizaciones del hardware, las técnicas de procesamiento paralelo, las estrategias para garantizar la estabilidad numérica y el tamaño de los datos de entrada.

Anexo 3: Interfaz web

3.1. Implementación web

Para demostrar el potencial del sistema de segmentación de lesiones desarrollado, se implementó una interfaz web intuitiva y funcional. Actualmente, el sistema opera en un entorno local, pero está diseñado para ser fácilmente adaptable a un entorno de nube. La implementación en la nube implicaría costos adicionales asociados con máquinas virtuales equipadas con GPU. Dado que esta expansión no forma parte del objetivo principal del proyecto, se decidió mantener la implementación en un entorno local. La implementación se llevó a cabo en una laptop Alienware M15 R5 equipada con un procesador Ryzen 9, 16 GB de RAM, 1 TB de almacenamiento SSD y una tarjeta gráfica NVIDIA RTX 3070 de 16 GB.

3.2. Funciones de la interfaz

La interfaz web cuenta con las siguientes funcionalidades:

1. **Selección de Archivos:** Un botón *Choose File* permite al usuario cargar una imagen desde su dispositivo local.
2. **Proceso de Segmentación:** Al hacer clic en el botón *Segment*, el sistema inicia la segmentación de la imagen y muestra los resultados en tres secciones: 'BBox + Points', 'Segmentation' y 'Overlay'.
3. **Resultados y Análisis:** Se presenta información detallada del proceso, incluyendo el tiempo de segmentación y el conteo de píxeles de la lesión, en un cuadro de texto debajo de los cuadros de imágenes.
4. **Descarga de Datos:** Un botón *Download* facilita la descarga de las imágenes procesadas y los datos de análisis.

La representación visual del sistema se ilustra en la Figura XLVII. Esta imagen proporciona una visión

clara y detallada de la interfaz de usuario, donde cada número marcado corresponde directamente a una de las funciones descritas en la lista de funcionalidades previamente mencionada.

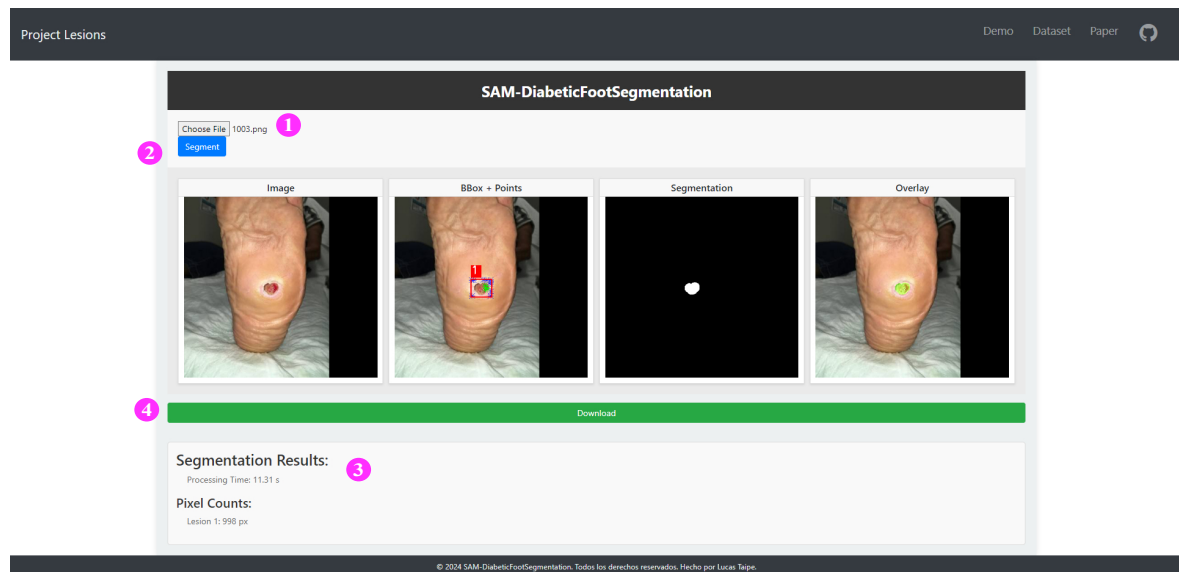


Figura XLVII: Interfaz web para la segmentación de lesiones.

3.3. Framework y lenguaje de programación

El modelo se desarrolló en Python, aprovechando la versatilidad de Flask como framework para una rápida integración de los modelos y algoritmos del proyecto. Flask, utilizado principalmente en el backend, maneja las solicitudes del cliente, procesa los datos y comunica los resultados, desempeñando un papel fundamental en la lógica de la aplicación. Los detalles técnicos adicionales incluyen:

- **Backend:** Python y Flask constituyen el núcleo del backend, gestionando las interacciones entre el servidor y el cliente, la lógica de la aplicación y el procesamiento de datos.
- **Front-End:** Se utilizó HTML, JavaScript y CSS para desarrollar una interfaz de usuario atractiva y responsive. Esta capa del frontend se encarga de la interacción directa con el usuario, presentando una interfaz gráfica intuitiva y recogiendo las entradas del usuario para su procesamiento.
- **Bootstrap:** La implementación de Bootstrap facilita un diseño coherente y adaptable a distintos dispositivos, mejorando la accesibilidad y usabilidad de la interfaz.

3.4. Integración backend y frontend

En el desarrollo de nuestro sistema, el backend, implementado con Flask, un micro-framework de Python, procesa las solicitudes del frontend, ejecuta los algoritmos de segmentación de imágenes y retorna los resultados para su visualización. Se establecieron rutas específicas para manejar acciones como cargar imágenes y realizar la detección de lesiones.

El frontend, desarrollado con JavaScript, interactúa con estas rutas del servidor. Por ejemplo, al cargar una imagen, se envía una solicitud POST a '/detect_lesion' en Flask, donde se procesa la imagen utilizando modelos de inteligencia artificial. Los resultados se devuelven al cliente, incluyendo las imágenes procesadas y los datos de segmentación.

Las llamadas AJAX en el frontend permiten solicitar y recibir datos del servidor sin recargar la página, mejorando la experiencia del usuario y permitiendo una interacción intuitiva con el sistema para el análisis detallado de imágenes médicas.

3.5. Guía de instalación

Para el óptimo funcionamiento del sistema, se requiere un hardware específico, incluyendo una GPU con al menos 4 TFlops de capacidad y 16 GB de RAM. Además, es necesario tener instalado Anaconda con Python versión 3.8.17.

1. Clonar el repositorio de GitHub desde este enlace.
2. Navegar a la carpeta 'app_segmentation_lesion'.
3. Abrir el Command Prompt de Anaconda y dirigirse a la carpeta 'app_segmentation_lesion'.
4. Crear un entorno virtual en Python e instalar las dependencias desde 'requirements.txt'.
5. Asegurarse de tener acceso a la GPU del dispositivo.
6. Descargar los modelos EDSR y SAM y colocarlos en la carpeta 'app_segmentation_lesion'.

Enlaces para descargar los modelos:

- EDSR disponible en GitHub.
- SAM Vitth disponible en GitHub.

7. Finalmente, ejecutar ‘app.py’ desde el entorno virtual. Se generará un enlace para abrir el localhost y probar el modelo.

Anexo 4: Fine-tuning SAM

4.1. Fine tuning del modelo SAM

Este documento ofrece una descripción del proceso de ajuste fino (fine-tuning) aplicado al modelo SAM original, enfocado específicamente en la segmentación de imágenes médicas con un conjunto de datos exclusivamente compuesto por imágenes de lesiones. A diferencia del ajuste previo conocido como MedSAM, que se especializó en un espectro más amplio de imágenes médicas sin incluir lesiones en pie, este entrenamiento tiene como objetivo mejorar la precisión del modelo en la identificación y segmentación de lesiones específicas. Este proceso se ejecutó en un sistema con las siguientes especificaciones técnicas:

- Procesador: Intel(R) Core(TM) i9-10940X CPU @ 3.30GHz
- Memoria RAM: 131072MB
- GPU: NVIDIA GeForce RTX 3070 Ti con 8032MB de VRAM

4.2. Metodología de entrenamiento

El modelo fue entrenado con 1010 imágenes durante 200 épocas utilizando el optimizador Adam con una tasa de aprendizaje de 1×10^{-5} y sin decaimiento de peso (weight decay). El entrenamiento se completó en un tiempo total de 22.77 horas usando 25GB de la memoria RAM.

4.3. Proceso de aprendizaje

Durante el entrenamiento, el modelo SAM aprende a afinar los parámetros de su decodificador de máscaras mediante la retroalimentación de la función de pérdida, una combinación de la pérdida de Dice y la Entropía Cruzada. Este ajuste permite que el modelo haga predicciones de segmentación más precisas al comparar las máscaras predichas con las verdaderas (ground truth). Para obtener más información sobre el proceso de entrenamiento y la implementación del modelo, se puede consultar el tutorial en el repositorio de GitHub en <https://github.com/bowang-lab/MedSAM>.

4.4. Resultados

El entrenamiento resultó en una disminución progresiva de la pérdida, lo cual indica una mejora continua en la capacidad del modelo SAM para segmentar imágenes médicas. El mejor modelo obtenido, determinado por la pérdida más baja, fue guardado para comparaciones futuras. La Figura XLVIII ilustra la evolución de la pérdida durante el proceso de entrenamiento.

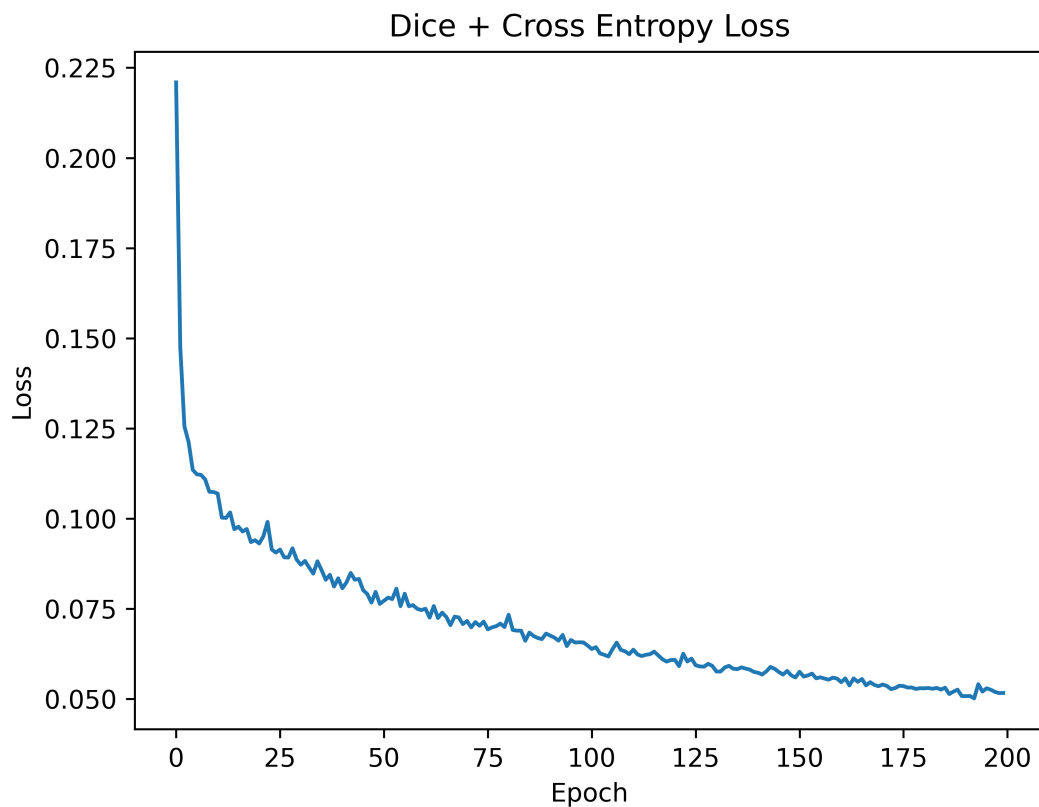


Figura XLVIII: Gráfica de la evolución de la pérdida durante el entrenamiento de SAM durante 200 épocas.