UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
Y CIENCIAS DE LA COMPUTACIÓN

DETECCIÓN TEMPRANA DE MELANOMA MEDIANTE VISIÓN COMPUTACIONAL EN DISPOSITIVOS MÓVILES

POR

VICENTE RÍOS ADASME

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Informático

Profesora Guía
Dra. Mabel Angélica Vidal Miranda

Enero 2025
Concepción (Chile)

A mi familia y compañeros de carrera.

Agradecimientos

"Agradezco profundamente a mis profesores y mentores de la Facultad de Ingeniería. En un momento de incertidumbre, donde mi camino profesional estuvo a punto de tomar otro rumbo, su dedicación, respeto y genuina voluntad de enseñar fueron la brújula que necesitaba. Gracias por tener la paciencia y la vocación para reencantarme con la informática; su influencia no solo me devolvió la pasión por esta disciplina, sino que me entregó herramientas que hoy son los pilares fundamentales de mi vida profesional y diaria."

Resumen

El melanoma cutáneo representa un desafío crítico de salud pública, en el que la supervivencia del paciente está relacionada al diagnóstico precoz. Sin embargo, la brecha entre la creciente incidencia de la patología y la disponibilidad de especialistas dermatológicos genera tiempos de espera que comprometen el pronóstico. Esta memoria aborda dicha problemática mediante el desarrollo de un sistema de detección dermatológica basado en Inteligencia Artificial y Visión Computacional, diseñado para operar en dispositivos móviles bajo el paradigma de *Edge AI*, garantizando así la privacidad de los datos y la disponibilidad en zonas sin conectividad.

Se utilizaron múltiples fuentes de referencia internacional, integrando los datasets **ISIC** (ediciones 2018, 2019 y 2020), **HAM10000** y el *Melanoma Skin Cancer Dataset*. Para mitigar el desbalance de clases de los datos médicos, se aplicó una estrategia de preprocesamiento basada en *Random Undersampling*, obteniendo un corpus de entrenamiento balanceado (ratio 3:1) de 73.448 imágenes dermatoscópicas. A nivel de modelo, se implementó una red **EfficientNetV2-M** optimizada mediante *Transfer Learning* en un entorno PyTorch con aceleración GPU.

Finalmente, para superar las barreras de implementación móvil, el modelo fue exportado al estándar **ONNX** (Open Neural Network Exchange). Esto habilitó la inferencia local eficiente mediante *ONNX Runtime*, logrando una precisión de 97,79 %.

Índice general

Capítulo 1. Introducción

1.1. Contexto del Problema

El cáncer de piel es una de las enfermedades oncológicas más comunes a nivel mundial. Según la Organización Mundial de la Salud (OMS), en 2020 se registraron más de 1,20 millones de nuevos casos de cáncer de piel [40]. Dentro de las variedades de cáncer de piel, el melanoma cutáneo es el más peligroso: aunque representa menos del 5 % de los casos, causando más del 75 % de las muertes por cáncer de piel debido a su capacidad de hacer metástasis rápidamente [31].

1.1.1. Epidemiología del Melanoma

Cada año se diagnostican más de 320,000 nuevos casos de melanoma en el mundo según el informe Globocan [34]. Lo importante del melanoma es que su pronóstico depende completamente de cuándo se detecta:

- **Detección Temprana (Estadio I):** Si se detecta cuando el tumor tiene menos de 1 mm de grosor, la tasa de supervivencia a 5 años es superior al 99 % [1].
- **Detección Tardía (Estadio IV):** Una vez que hace metástasis, la supervivencia cae por debajo del 25 % [1].

Por lo tanto, detectar el melanoma a tiempo y clasificarlos en sus diferentes estadíos (Figura 1.1) genera una gran diferencia en la salud de los pacientes.

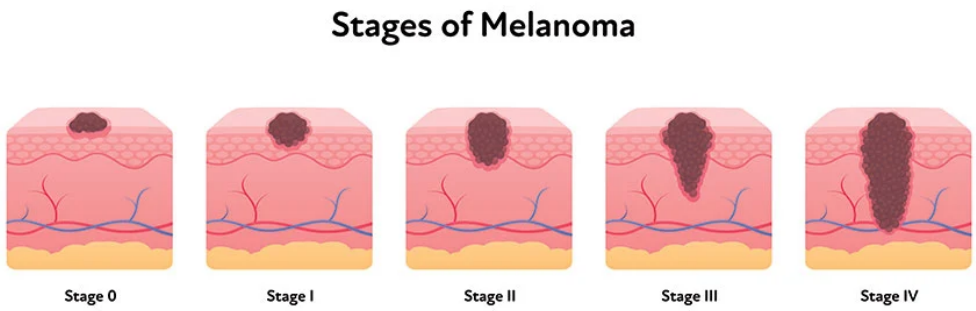

Figura 1.1: Representación de los diferentes estadios del melanoma. La tasa de supervivencia del paciente presenta una correlación directa con la etapa en la que se detecta la enfermedad. Fuente: Adaptado de [27].

1.1.2. El Factor Ambiental: Radiación UV en Chile

Chile enfrenta un problema adicional que es el agujero en la capa de ozono. La capa de ozono filtra los rayos ultravioleta B (UV-B). Cuando esta capa se adelgaza, especialmente durante la primavera (septiembre a noviembre), más radiación UV llega al sur de Chile [29] (Figura 1.2).

La radiación UV daña el ADN de las células de la piel. Si el cuerpo no logra reparar este daño a tiempo, las células comienzan a multiplicarse sin control, lo que puede dar lugar a un melanoma [9].

Datos epidemiológicos y geográficos en Chile indican que:

1. La tasa de mortalidad por melanoma y cáncer de piel no melanoma ha aumentado sostenidamente, alcanzando un alza del 40 % en los últimos diez años con más de 500 decesos anuales [5].

2. Zonas como Antofagasta mantienen niveles de radiación UV extremos. Es crucial destacar que este riesgo persiste incluso en días con temperaturas moderadas, ya que la peligrosidad radica en la irradiancia solar y la altitud, no en la sensación térmica [32].

1.1.3. La Brecha Asistencial en el Sistema Público

El problema no es solo la radiación UV, sino también el acceso limitado a especialistas. En Chile, se destaca las siguientes brechas:

Escasez de Dermatólogos: La mayoría de los especialistas se concentran en Santiago y en el sistema privado de salud. Un paciente de FONASA (77 % de la población) puede esperar entre 6 y 12 meses para una consulta con dermatólogo [26].

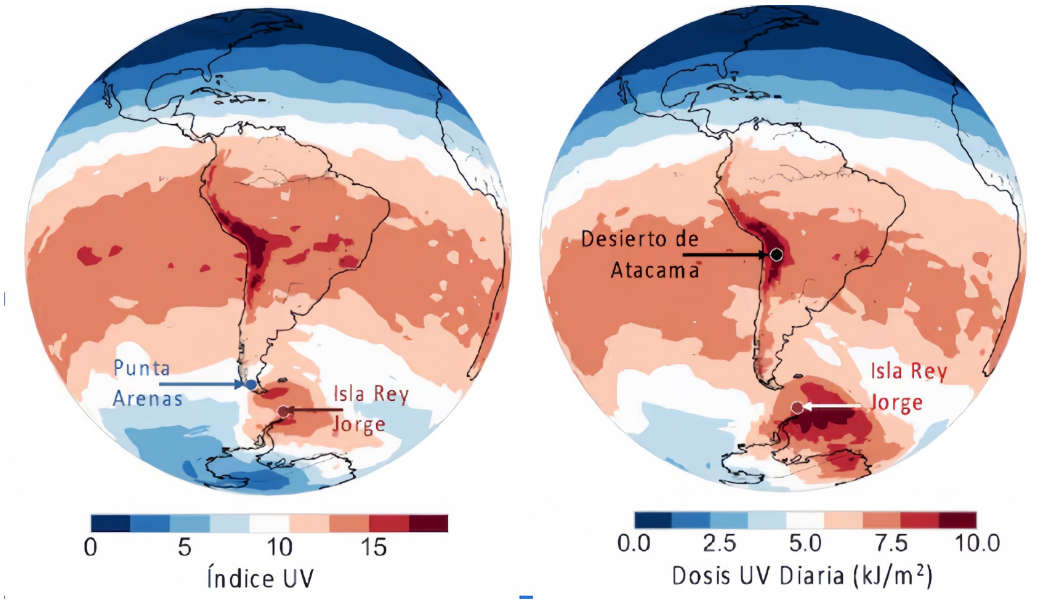

Figura 1.2: Niveles de radiación UV en Chile. El norte tiene alta radiación por la altura, y el sur por el agujero de ozono.

Dificultad para Diagnosticar: Los médicos generales utilizan la regla *ABCDE* (Asimetría, Bordes, Color, Diámetro, Evolución) para identificar lunares sospechosos, pero su precisión depende de la experiencia. Estudios muestran que médicos no especialistas tienen una precisión de solo 60–70 % [37].

En resumen: mientras un paciente espera meses por una consulta, un melanoma agresivo puede avanzar de un estadio curable a uno terminal. Por lo tanto, este proyecto busca apoyar esa brecha usando Inteligencia Artificial, para una detección temprana.

1.2. Definición del Problema y Oportunidad

El problema identificado es que existe una brecha entre la necesidad de detección temprana del melanoma y la capacidad del sistema de salud para proveerla. Esta brecha se manifiesta en tres dimensiones:

1. **Temporal:** Tiempos de espera de 6–12 meses para consultas con dermatólogo.

2. **Geográfica:** Concentración de especialistas en Santiago, dejando desprotegidas a regiones con mayor exposición UV.

3. **Técnica:** Baja precisión diagnóstica (60–70 %) de médicos no especialistas usando métodos visuales.

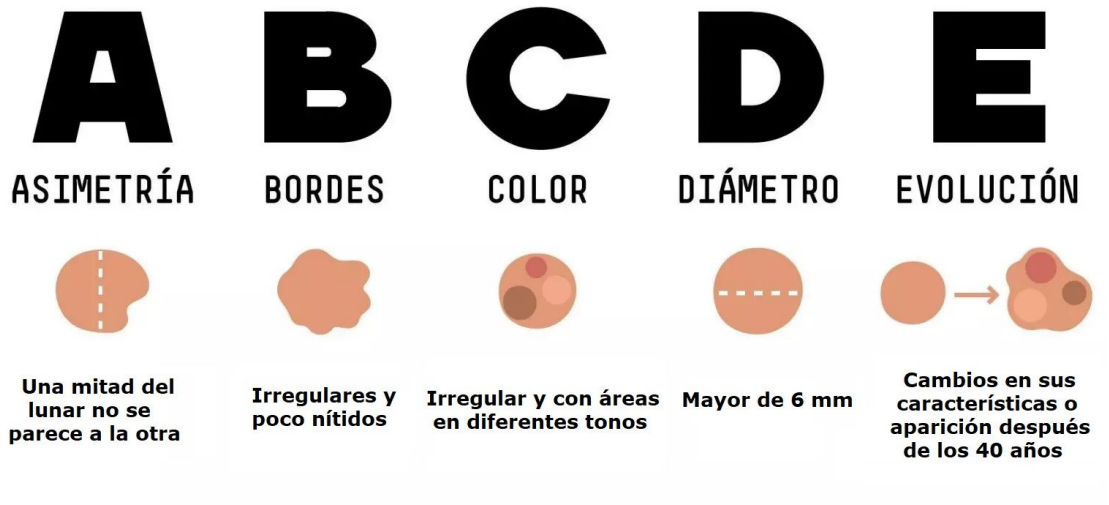

Figura 1.3: Regla ABCDE para la detección de melanoma.

La oportunidad identificada es el desarrollo de una herramienta basada en Inteligencia Artificial que pueda:

- Operar completamente offline (sin necesidad de internet).
- Ejecutarse en smartphones comunes (sin hardware especializado).
- Proporcionar un resultado inmediato sobre el nivel de riesgo de una lesión.
- Mantener la privacidad total de los datos médicos del usuario.

1.3. Objetivos del Proyecto

1.3.1. Objetivo General

Desarrollar e implementar una aplicación móvil multiplataforma que utilice redes neuronales convolucionales para la detección y clasificación binaria de melanoma a partir de imágenes dermatoscópicas.

1.3.2. Objetivos Específicos

1. Entrenar un modelo de clasificación binaria (benigno/maligno) utilizando EfficientNetV2 sobre datasets públicos de dermatología (ISIC, HAM10000).
2. Optimizar el modelo para inferencia móvil mediante exportación a formato ONNX y cuantización post-entrenamiento.

3. Implementar una arquitectura de software limpia (Clean Architecture) que permita la escalabilidad y mantenibilidad del sistema.

4. Integrar funcionalidades complementarias: monitoreo de índice UV, búsqueda de dermatólogos cercanos, y sistema de recordatorios médicos.

5. Validar el funcionamiento del sistema mediante pruebas de inferencia comparando resultados entre Python (PC) y ONNX Runtime (móvil).

1.4. Alcance y Limitaciones

1.4.1. Alcance del Proyecto

El proyecto abarca:

- Desarrollo completo de un modelo de clasificación binaria entrenado con 73.448 imágenes balanceadas.
- Implementación de aplicación móvil funcional para Android e iOS usando Flutter.
- Persistencia local de registros médicos usando SQLite.
- Integración de APIs externas para clima (WeatherAPI) y geolocalización (Google Places).
- Sistema de notificaciones locales para recordatorios médicos.

1.4.2. Limitaciones Identificadas

1. **Clasificación Binaria:** El sistema solo distingue entre lesiones benignas y malignas. No se implementó clasificación multiclase de tipos específicos de patologías debido a la escasez de datos públicos balanceados.

2. **Ausencia de Regla ABCDE:** No se implementó un sistema de análisis morfológico basado en la regla clínica ABCDE (Asimetría, Bordes, Color, Diámetro, Evolución) como característica del modelo, aunque hubiera sido deseable.

3. **Sesgo Étnico en Datasets:** Los datasets públicos utilizados (ISIC, HAM10000) tienen predominancia de piel de fototipos claros (tipos I-III de Fitzpatrick). Esto puede afectar la precisión en poblaciones con fototipos más oscuros.

4. **Sin Validación Clínica Real:** El sistema no fue sometido a ensayos clínicos con pacientes reales ni validado por dermatólogos certificados.

1.5. Estructura de la Memoria

Esta memoria se organiza de la siguiente manera:

- **Capítulo 2:** Revisión del estado del arte en visión computacional médica, tecnologías móviles y soluciones comerciales existentes.
- **Capítulo 3:** Diseño metodológico, justificación de decisiones técnicas y arquitectura del sistema.
- **Capítulo 4:** Detalles de implementación: entrenamiento del modelo, exportación a ONNX, desarrollo de la aplicación móvil.
- **Capítulo 5:** Resultados obtenidos: métricas del modelo, pruebas de rendimiento y validación funcional.
- **Capítulo 6:** Conclusiones, cumplimiento de objetivos y trabajo futuro.

Capítulo 2. Estado del Arte y Marco Teórico

2.1. Aprendizaje de Máquina

El campo del análisis de imágenes médicas ha experimentado cambios importante en la última década, evolucionando desde sistemas basados en reglas manuales hacia modelos de aprendizaje con redes neuronales.

2.1.1. Ingeniería de Características vs. Aprendizaje Profundo

Previo al año 2012, el flujo de trabajo estándar en visión computacional dependía de la *ingeniería de características manual* (hand-crafted features). Algoritmos como SIFT (*Scale-Invariant Feature Transform*) [23] o HOG (*Histogram of Oriented Gradients*) se utilizaban para extraer descriptores matemáticos de las imágenes (bordes, texturas, formas), los cuales alimentaban a clasificadores clásicos como *Support Vector Machines* (SVM) o *Random Forests* [4].

Este enfoque presentaba una limitación importante: la dependencia del conocimiento experto humano para definir *a priori* qué características eran relevantes, lo que restringía la capacidad del sistema para generalizar ante la compleja variabilidad biológica e intra-clase de lesiones como el melanoma [21, 8].

Con la llegada de las **Redes Neuronales Convolucionales (CNN)** cambió este paradigma hacia el aprendizaje de extremo a extremo (*end-to-end*), donde el modelo aprende automáticamente a extraer las representaciones jerárquicas óptimas directamente desde los píxeles de la imagen [20]. En el contexto del aprendizaje profundo, el enfoque *end-to-end* implica que todo el sistema se modela como un único bloque computacional coherente. En lugar de etapas aisladas, la arquitectura opera como una **función diferenciable** compuesta [11].

Esto significa que todos los componentes del proceso están enlazados matemáticamente, permitiendo utilizar la propagación del error final (*backpropagation*) para ajustar los parámetros de toda la red simultáneamente, desde el clasificador final hasta los filtros convolucionales de bajo nivel. Este mecanismo contrasta fundamentalmente con los flujos de trabajo tradicionales del aprendizaje automático, que dependían de una etapa manual y separada de **ingeniería de características** (*feature engineering*), cuya rigidez impedía la optimización conjunta del siste-

ma [2]. Para entender mejor este cambio, es necesario desglosar lo que implica el aprendizaje *end-to-end*:

- **Eliminación de la extracción manual:** Anteriormente,se debían diseñar algoritmos específicos (como SIFT, HOG o filtros de Sobel) para identificar bordes, texturas o formas antes de pasar esa información a un clasificador (como una Support Vector Machine (SVM)). En el aprendizaje *end-to-end*, las capas iniciales de la Convolutional Neural Network (CNN) asumen este rol, ajustando sus pesos para encontrar las características más relevantes sin intervención humana posterior.

- **Optimización Conjunta:** En lugar de optimizar la extracción de características y la clasificación por separado, una CNN optimiza ambos procesos simultáneamente. A través del algoritmo de *Backpropagation* (propagación hacia atrás), el error calculado en la salida se propaga hasta las primeras capas. Esto asegura que los filtros aprendidos en las capas convolucionales estén específicamente sintonizados para minimizar la función de pérdida de la tarea específica que se está resolviendo.

- **Jerarquía de Abstracción:** El sistema aprende una representación jerárquica de la información. Las primeras capas detectan patrones simples (líneas, curvas), mientras que las capas profundas combinan estos patrones para formar conceptos complejos (ojos, ruedas, estructuras), todo ello derivado puramente de los datos de entrenamiento.

En resumen, el aprendizaje *end-to-end* permite que la red neuronal construya su propia representación interna del mundo visual, extrayendo características que serían demasiado complejas para ser analizadas manualmente.

2.1.2. Arquitecturas de Redes Convolucionales Profundas

El punto de inflexión en la visión computacional moderna fue la arquitectura AlexNet [19], ganadora del desafío ILSVRC 2012. Esta red demostró que las CNN profundas, entrenadas con grandes volúmenes de datos, superaban a los métodos tradicionales.

AlexNet definió un estándar con 3 pilares fundamentales para las arquitecturas modernas:

1. **No linealidad ReLU:** El uso de la función de activación *Rectified Linear Unit* ($f(x) = \text{máx}(0, x)$) solucionó los problemas de saturación de las funciones sigmoides, acelerando la convergencia del entrenamiento [28].

2. **Regularización por Dropout:** Para evitar el sobreajuste (*overfitting*) en redes con millones de parámetros, Nitish Srivastava y colaboradores [33], introdujeron la técnica de desactivar aleatoriamente neuronas durante el entrenamiento.

3. **Cómputo en GPU:** El uso de tarjetas gráficas permitió la paralelización masiva de las operaciones de convolución.

2.1.3. Arquitectura Red Residual: ResNet

Con el objetivo de mejorar la precisión, las arquitecturas comenzaron a hacerse más profundas. Esto introdujo el problema del *desvanecimiento del gradiente*: a medida que la red gana profundidad, la señal de error utilizada para actualizar los pesos se diluye hasta desaparecer antes de llegar a las primeras capas, lo que impide que la red aprenda.

Para solucionar esto, Kaiming He y sus colaboradores [13], propusieron las **Redes Residuales (ResNet)**. La innovación fue la introducción de las *conexiones de salto* (conocidas en inglés como *skip connections*), que permiten que la información fluya directamente a través de la red sin obstáculos.

Matemáticamente, en lugar de intentar aprender una función $H(x)$ directa, la red se fuerza a aprender una función residual $F(x)$, definida como:

$$H(x) = F(x) + x \tag{2.1}$$

Esta formulación facilita el entrenamiento de redes con cientos de capas sin degradación del rendimiento, estableciendo a ResNet como un estándar fundamental en la industria.

2.1.4. Eficiencia en Dispositivos Móviles: MobileNet

La implementación en dispositivos móviles enfrenta restricciones de cómputo y energía. Para abordar esto, Howard et al. [15] propusieron MobileNet, una arquitectura basada en *Convoluciones Separables en Profundidad* (Depthwise Separable Convolutions). Esta técnica descompone la convolución estándar en dos operaciones: una convolución de profundidad (filtraje) y una convolución puntual (1×1) para combinar las salidas, reduciendo drásticamente la cantidad de parámetros y operaciones sin sacrificar significativamente la precisión.

2.1.5. EfficientNet: Escalamiento Compuesto Optimizado

La arquitectura seleccionada para este proyecto es EfficientNet, desarrollada por Tan y Le [35]. A diferencia de las anteriores que escalaban arbitrariamente una sola dimensión de la red (profundidad, ancho o resolución), EfficientNet introduce el método de **escalamiento compuesto** (*compound scaling*).

Este método utiliza un coeficiente compuesto ϕ para escalar uniformemente las tres dimensiones bajo una restricción de recursos fija:

$$
\begin{aligned}
d &= \alpha^{\phi} \quad \text{(profundidad)} \\
w &= \beta^{\phi} \quad \text{(ancho)} \\
r &= \gamma^{\phi} \quad \text{(resolución)} \\
&\text{s.t.} \quad \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2
\end{aligned}
\tag{2.2}
$$

Donde α, β, γ son constantes determinadas mediante búsqueda de arquitectura neuronal (*Neural Architecture Search*). Además, EfficientNet integra bloques MBConv optimizados con módulos de atención Squeeze-and-Excitation (SE)[16], permitiendo un reajuste dinámico de la importancia de los canales de características.

EfficientNetV2: Optimización para Entrenamiento Acelerado

Para este trabajo, se utilizó la versión mejorada EfficientNetV2 [36], que introduce optimizaciones para acelerar el entrenamiento y mejorar la eficiencia de inferencia. Las mejoras dentro de su arquitectura fueron:

1. **Fused-MBConv Blocks:** Reemplazo de bloques MBConv en etapas iniciales por versiones fusionadas, optimizando el uso de memoria y aprovechando aceleradores de hardware modernos.

2. **Entrenamiento Progresivo:** Ajuste dinámico del tamaño de imagen y la regularización durante el entrenamiento, lo que acelera la convergencia sin degradar la precisión.

La selección de la arquitectura **EfficientNetV2-M** se fundamenta por su capacidad para maximizar precisión diagnóstica y eficiencia computacional, un requisito crítico para el despliegue en dispositivos móviles de recursos limitados. Según lo demostrado por Tan y Le [36], esta familia de modelos introduce optimizaciones arquitectónicas clave, específicamente el uso de bloques *Fused-MBConv* en las etapas iniciales de la red.

Esta modificación resuelve los cuellos de botella de acceso a memoria presentes en la versión original (EfficientNet-V1) y en las convoluciones separables en profundidad tradicionales. Como resultado, EfficientNetV2 logra reducir el número de parámetros y FLOPs necesarios para alcanzar niveles de precisión comparables al estado del arte, ofreciendo una velocidad de entrenamiento hasta cuatro veces superior y una inferencia significativamente más rápida en aceleradores de hardware modernos, lo cual valida su idoneidad para la ejecución *offline* en smartphones de gama media.

2.1.6. Análisis Comparativo de Arquitecturas

Para seleccionar la arquitectura adecuada, se realizó un analisis técnico entre el enfoque clásico convolucional y los nuevos paradigmas basados en atención.

Enfoque Convolucional (CNN)

Las Redes Neuronales Convolucionales (CNN), base de la arquitectura EfficientNetV2 seleccionada, operan bajo el principio de localidad espacial. Como se observa en la Figura 2.1, el modelo utiliza ventanas deslizantes (*kernels*) que recorren la imagen progresivamente.

Este diseño arquitectónico presenta ventajas críticas para el proyecto:

- **Sesgo Inductivo Fuerte:** La red asume intrínsecamente que los píxeles vecinos están relacionados. Esto permite que el modelo converja eficientemente incluso con datasets de tamaño medio (∼73k imágenes) [41].
- **Eficiencia en Borde:** Las operaciones de convolución están altamente optimizadas en las NPU de los smartphones modernos, garantizando baja latencia.

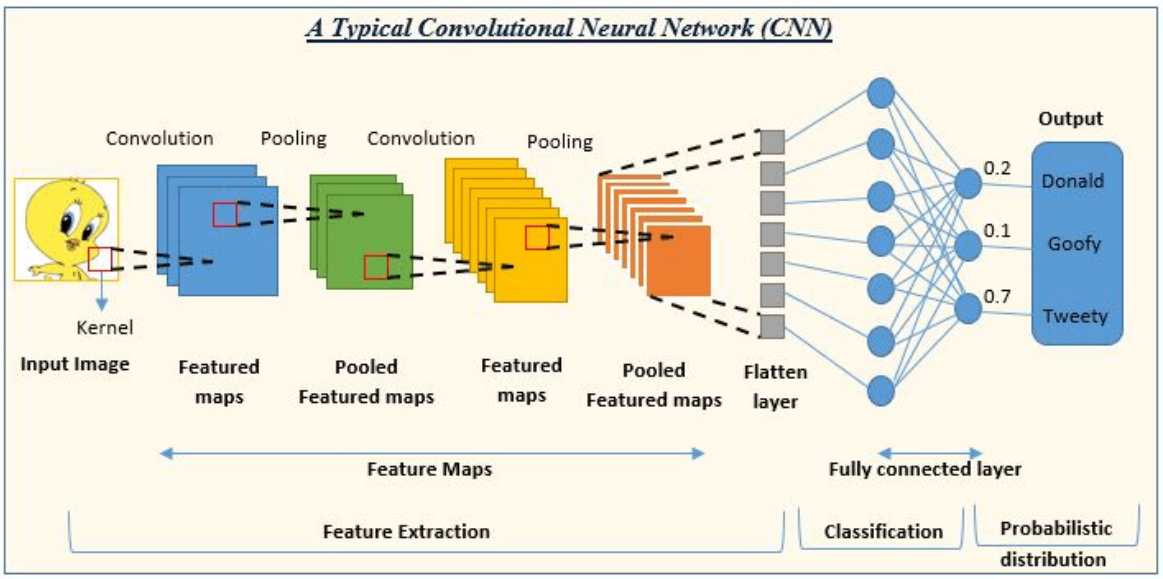

Figura 2.1: Procesamiento en una CNN. Extracción jerárquica de características mediante convoluciones locales y pooling. Los *kernels* (cuadros rojos/amarillos) escanean la imagen para detectar patrones.

Enfoque Vision Transformers (ViT)

En contraste, se evaluó la arquitectura Vision Transformer (ViT). A diferencia de las CNN, este modelo elimina las convoluciones y procesa la imagen buscando relaciones globales desde el inicio, tal como se detalla en la Figura 2.2.

El proceso implica:

1. **Patching:** La imagen se corta en una cuadrícula de parches (ej. 16×16 píxeles).
2. **Linear Projection:** Cada parche se aplana y se trata como una "palabra.en una oración.
3. **Self-Attention:** El bloque de atención (derecha de la figura) calcula matemáticamente la relación de cada parche con todos los demás simultáneamente.

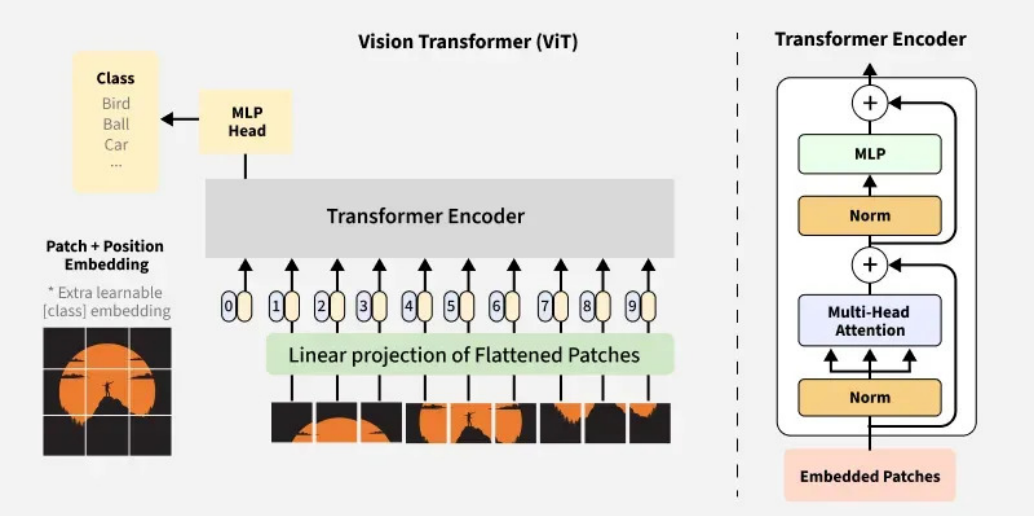

Figura 2.2: Arquitectura Vision Transformer. La imagen se procesa como una secuencia de parches planos. El mecanismo de *Multi-Head Attention* analiza dependencias globales, lo que eleva el costo computacional.

Síntesis de la Selección

Pese a la potencia teórica de los Transformers, se descartaron para esta implementación debido a dos factores visibles en su diagrama:

- **Costo Cuadrático:** El mecanismo de atención global tiene una complejidad $O(N^2)$, lo que resulta prohibitivo para la inferencia en tiempo real en dispositivos móviles de gama media [38].
- **Ineficiencia de Datos:** Al no tener *kernels* que asuman vecindad, requieren millones de imágenes para aprender la estructura visual, tendiendo al sobreajuste en datasets médicos limitados [7].

Por estas razones técnicas, EfficientNetV2 se consolidó como la arquitectura óptima para este sistema de diagnóstico dermatológico móvil.

2.2. Estado del Arte Industrial y Brecha Tecnológica

Para situar la contribución de esta memoria, se analizó el ecosistema actual de soluciones dermatológicas digitales. El mercado se polariza en tres categorías, cada una con limitaciones que este proyecto busca mitigar.

2.2.1. Análisis de Soluciones Existentes

1. **Aplicaciones Comerciales (B2C):** Herramientas como *SkinVision*[1] democratizan el acceso, pero operan bajo arquitecturas 100 % Cloud. Esto genera dos barreras críticas: la exclusión de usuarios en zonas rurales sin 4G estable y la vulnerabilidad de la privacidad al subir imágenes biométricas a servidores externos.

2. **Hardware Clínico (B2B):** Equipos como *FotoFinder*[2] son el estándar de oro en calidad óptica. Sin embargo, su costo (decenas de miles de dólares) y falta de portabilidad los restringen a clínicas especializadas, sin resolver el problema de triaje masivo.

3. **Modelos Fundacionales (Big Tech):** Iniciativas como *Google DermAssist*[3] ofrecen gran robustez algorítmica, pero operan como "cajas negras" que recolectan datos del usuario y dependen de la latencia de red para entregar resultados.

2.2.2. La Oportunidad: Identificación de la Brecha

Como se resume en la Tabla 2.1, existe un vacío tecnológico en la falta de una herramienta que combine privacidad total (procesamiento local) y funcionamiento offline, eliminando las barreras de costo y conectividad. **Dermat-IA** se diseñó específicamente para ocupar este espacio mediante Inteligencia Artificial en el Borde.

2.3. Paradigma Tecnológico: Edge AI

Para cubrir la brecha identificada, la arquitectura del sistema se fundamenta en el Edge Computing [30]. A diferencia del modelo tradicional donde los datos viajan a la IA (Nube),

[1] SkinVision B.V. *Servicio de detección de cáncer de piel.* Disponible en: `https://www.skinvision.com`

[2] FotoFinder Systems GmbH. *Tecnología de mapeo corporal automatizado (ATBM).* Disponible en: `https://www.fotofinder.de`

[3] Google Health. *DermAssist: Herramienta de búsqueda guiada para la piel.* Disponible en: `https://health.google/consumers/dermassist/`

Tabla 2.1: Comparativa de soluciones. La propuesta destaca por su arquitectura local y offline.

Característica	SkinVision	Google DermAssist	Dermatoscopio	Esta Memoria
Arquitectura	Cloud (Remoto)	Cloud (Web)	Hardware Físico	**Edge AI (Local)**
Privacidad	Baja (Sube fotos)	Baja (Data Mining)	Alta	**Máxima**
Dependencia Internet	Alta (100 %)	Alta (100 %)	Nula	**Nula**
Costo Usuario	Alto (Suscripción)	Gratis (con datos)	Muy Alto	**Gratuito**

aquí la IA viaja a los datos (Dispositivo).

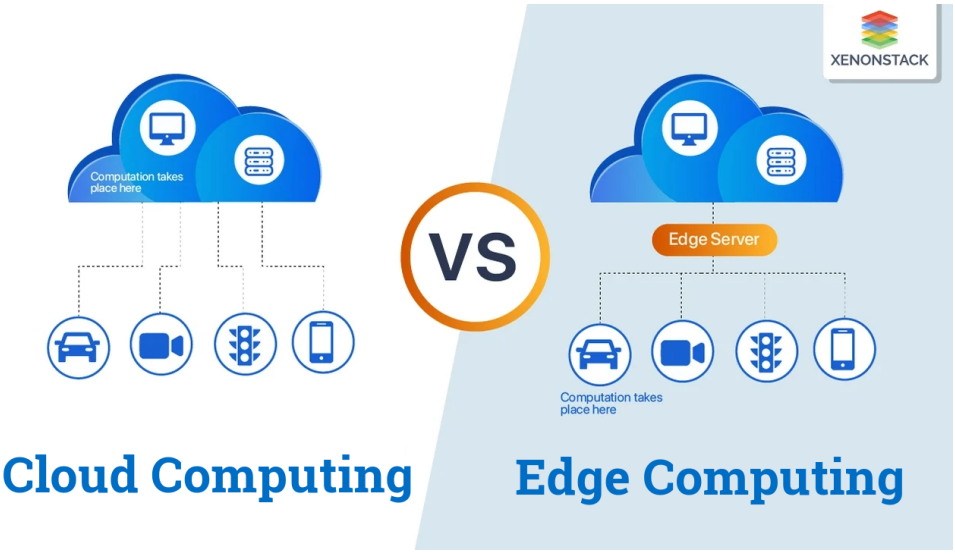

Figura 2.3: Arquitectura Edge vs. Cloud. El modelo Edge (derecha) garantiza que los datos médicos nunca abandonen el dispositivo del paciente, asegurando privacidad y velocidad.

Esta decisión de diseño responde a tres requisitos no funcionales críticos para el contexto chileno:

- **Privacidad por Diseño:** Al ejecutar el modelo localmente, la imagen reside solo en la memoria RAM volátil. Esto asegura el cumplimiento de normativas de datos médicos (como la Ley 20.584) sin necesidad de infraestructura compleja de ciberseguridad.
- **Resiliencia:** La inferencia local elimina la latencia de red, permitiendo diagnósticos instantáneos incluso en operativos médicos rurales sin cobertura móvil.
- **Escalabilidad:** El costo computacional se distribuye en los dispositivos de los usuarios, haciendo el sistema económicamente sostenible para la salud pública.

2.3.1. Desafío y Solución: Optimización para Móviles

Ejecutar redes neuronales profundas en smartphones presenta algunos desafíos de hardware como: consumo de batería, sobrecalentamiento (*thermal throttling*) y límites de memoria RAM [17].

Para viabilizar el uso de EfficientNetV2 en dispositivos de gama media, se implementó dos estrategias de optimización:

1. **Cuantización (FP32 → INT8):** Convertir los pesos del modelo de precisión flotante (32 bits) a enteros de 8 bits. Como ilustra la Figura 2.4, esto reduce el tamaño del modelo en un factor de 4 y acelera la inferencia aprovechando las instrucciones vectoriales de los procesadores móviles, con una pérdida de precisión despreciable ($< 1\,\%$) [10].
2. **Estandarización ONNX:** El uso del formato *Open Neural Network Exchange* desacopló el entrenamiento (PyTorch) de la inferencia, permitiendo acceder a la aceleración de hardware nativa (NNAPI) en Android.

Figura 2.4: Eficiencia mediante Cuantización. La reducción de precisión numérica (de FP32 a INT8) es clave para evitar la saturación de memoria en dispositivos móviles.

2.4. Ecosistema Funcional de Prevención

Entendiendo que la detección es solo el primer paso, la aplicación *Dermat-IA* integra módulos complementarios que transforman el modelo predictivo en una herramienta de salud integral:

- **Prevención Activa (Índice UV):** Utilizando geolocalización, la app alerta al usuario sobre el riesgo de radiación solar en tiempo real, promoviendo conductas preventivas antes de que ocurra el daño, siguiendo lineamientos de la SOCHIDERM [32].
- **Acción Inmediata (Mapa de Especialistas):** Ante un resultado de riesgo, el sistema reduce la incertidumbre proporcionando la ubicación y contacto de dermatólogos en un radio de 25 km, cerrando la brecha entre el screening digital y la atención clínica.
- **Adherencia (Recordatorios):** Un sistema de notificaciones locales fomenta el autoexamen periódico, vital para detectar cambios evolutivos en las lesiones.

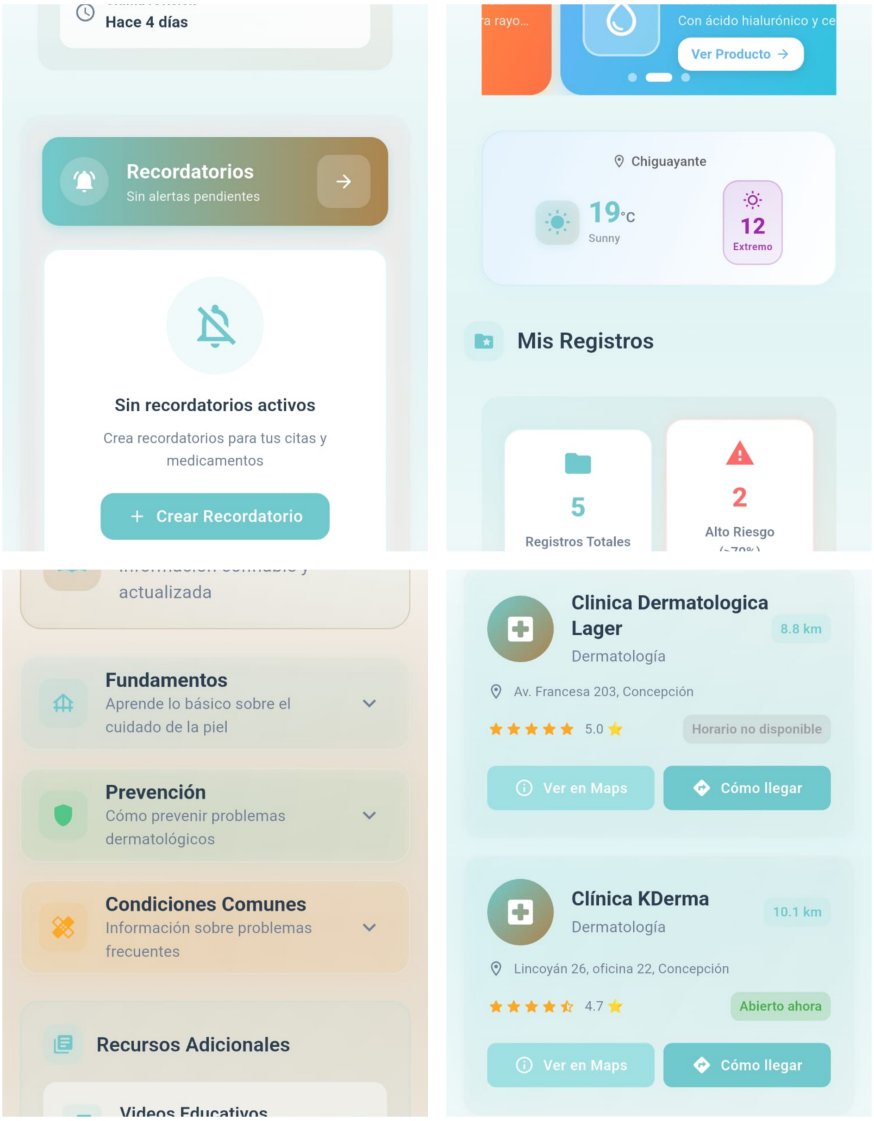

Figura 2.5: Enfoque Integral. La aplicación combina el diagnóstico por IA con herramientas de prevención (UV), acción (Mapa) y seguimiento (Recordatorios).

2.4.1. Enfoque Clásico vs. Deep Learning

Hace poco más de una década, para detectar melanoma por computadora, era necesario programar manual las reglas. Los ingenieros debían diseñar software específico para saber qué buscar: "¿Tiene bordes irregulares?", "¿Es el color asimétrico?". Esto se conoce como *ingeniería de características* [24]. El problema era que el sistema no podía aprender nada que no se le hubiese enseñado previamente. Si aparecía un lunar con un nuevo patrón, el sistema fallaba.

A partir de 2012, con la llegada del Deep Learning (Aprendizaje Profundo), esto cambió. En lugar de darle las reglas, al algoritmo se le entrena con miles o varios ejemplos (fotos de lunares sanos y enfermos) y la red neuronal descubre por sí sola qué características importan.

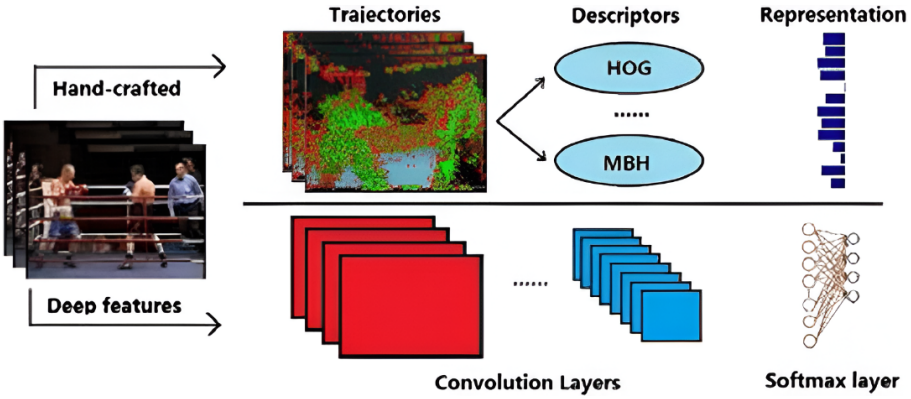

Figura 2.6: Cambio de paradigma. Antes (arriba) debíamos extraer características manualmente. Ahora (abajo), la red neuronal aprende sola .

2.4.2. Análisis de arquitecturas de redes neuronales

Durante esta investigación se evaluaron varias arquitecturas. A continuación, se detalla el proceso de selección:

El problema de la profundidad: ResNet

Al principio, las redes neuronales tenían un límite: si eran muy profundas, dejaban de aprender. ResNet (2015) solucionó esto creando "atajos"(*skip connections*) que permiten que la información fluya más fácil por la red [13].

- **Conclusión:** Aunque ResNet-50 es un estándar en la industria, no fue la mejor alternativa para analizar imágenes desde un celular. Requiere mucha memoria y batería, lo que la hacía inviable el objetivo de crear una app rápida y ligera.

La eficiencia ante todo: MobileNet

MobileNet, diseñada por Google específicamente para teléfonos. Usan un modelo matemático llamado çonvoluciones separables"que reduce drásticamente los cálculos necesarios [15].

- **Conclusión:** MobileNet es rápida, pero en nuestras pruebas preliminares se notó que reducía la precisión. En una app para detectar cáncer, la precisión es más importante que la velocidad.

El equilibrio ideal: EfficientNet

EfficientNet (2019). Esta arquitectura resultó ser la mejor alternativa porque no solo escala la profundidad de la red, sino también el ancho y la resolución de la imagen de forma balanceada (*compound scaling*) [35].

Por lo tanto, para este proyecto, se decidió utilizar específicamente la versión EfficientNetV2-M. ¿Por qué?

1. **Entrena más rápido:** Tiene optimizaciones que permitieron que el entrenamiento en la GPU no durara semanas.

2. **Es el punto medio:** Ofrece una precisión superior a ResNet, pero sigue siendo lo suficientemente ligera para correr en un celular moderno mediante cuantización.

¿Por qué no se utilizó Vision Transformers (ViT)?

Hoy en día, la arquitectura Transformers, es de las más populares aplicados a imágenes. Si bien, se consideró usarla, ya que son excelentes capturando el contexto global de una imagen [6],

se decidió descartarlos por dos razones de ingeniería práctica: (a) Los Transformers necesitan grandes volúmenes de imágenes para su entrenamiento. En esta memoría, el dataset consolidado tiene alrededor de 73.000 imágenes; lo cual no es suficiente y genera sobreajuste (*overfitting*). Por otro lado, las operaciones matemáticas de los Transformers son del orden ($O(N^2)$). En un celular sin hardware específico, la app resulta lenta. Por lo tanto, las CNNs como EfficientNet siguen siendo una de las mejores alternativas en dispositivos móviles.

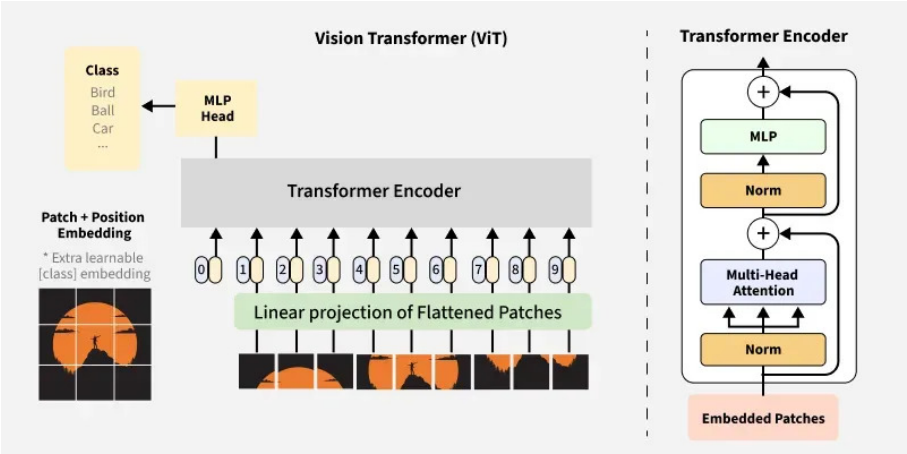

Figura 2.7: Vision Transformers. Aunque son potentes, requieren demasiados recursos computacionales para este proyecto.

2.5. Inteligencia Artificial en el Móvil: Edge Computing

Tradicionalmente, las apps de IA funcionan siguiendo este orden: se toma una foto, se sube a un servidor en la nube, se procesa y devuelven el resultado. Para este proyecto, se decidió tomar el camino opuesto, conocido como Edge AI (IA en el Borde) [30].

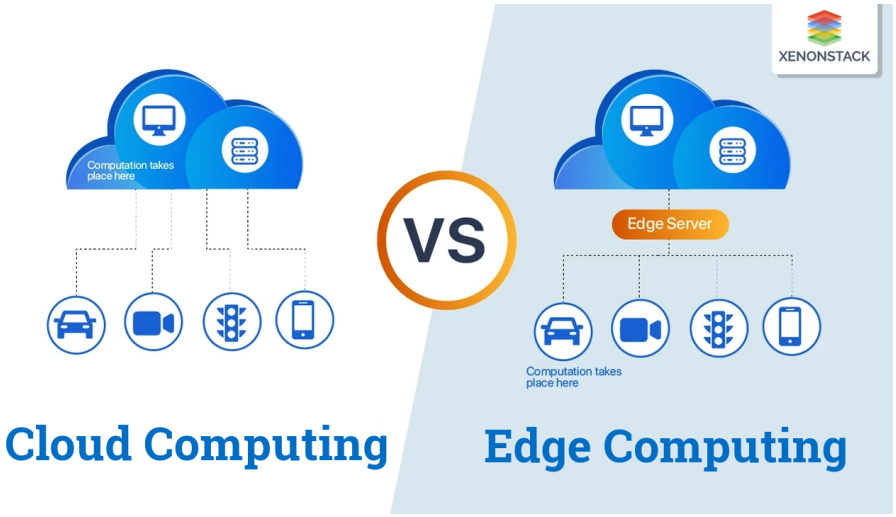

Figura 2.8: Tu teléfono es el servidor. A la izquierda, el modelo tradicional (nube) que arriesga tus datos. A la derecha, mi enfoque (Edge): todo pasa en tu mano.

2.5.1. Procesamiento desde en el celular

Se tomó esta decisión técnica basándose en tres problemas de la salud en Chile:

Privacidad Real (*Privacy-by-Design*): Una foto de una lesión en la piel es información médica sensible. Al usar Edge AI, la foto no sale del teléfono. El análisis ocurre en la memoria RAM y se borra apenas se cierra la app. Esto cumple por diseño con las leyes de protección de datos (como la Ley 20.584), porque simplemente no hay datos viajando por internet para ser interceptados.

Funciona sin conexión (Offline): Al correr el modelo localmente, la respuesta es inmediata y funciona incluso en modo avión.

Sin Costo: Mantener servidores GPU en la nube requiere una inversión económica. Al usar el procesador del teléfono del usuario, no habría un costo de operación para el sistema de salud.

2.5.2. El desafío técnico: Optimización

Los teléfonos tienen batería limitada y sube la temperatura si el procesador trabaja mucho. Para lograrlo, se aplicaron dos técnicas clave en el Capítulo 4:

1. Cuantización: Se convertió los pesos del modelo de números decimales complejos (32 bits) a enteros simples (8 bits). Es como comprimir un archivo de audio: pierdes una cantidad imperceptible de calidad, pero el archivo pesa cuatro veces menos y corre mucho más rápido [10].

2. Uso de ONNX: Se exporté el modelo a un formato estándar que permite que Android use su aceleración de hardware nativa, en lugar de depender de librerías más lentas de Python.

2.6. Estado del Arte Industrial: ¿Qué existe hoy?

se cargaes de desarrollar esta solución, se analizaron las alternativas que existen en el mercado para evaluar dónde aportaría valor este proyecto.

Aplicaciones Comerciales (Ej: SkinVision)

Existen soluciones consolidadas en el mercado, siendo **SkinVision**[4] un ejemplo representativo. Aunque demuestran un alto rendimiento operativo, su modelo de negocio SaaS (*Software as a Service*) impone barreras de acceso: requieren suscripciones de alto costo y la obligatoriedad de cargar imágenes a la nube para su procesamiento. Si bien la usabilidad móvil es un punto fuerte, la dependencia de conexión y el costo recurrente limitan su impacto como herramienta de salud pública masiva.

Hardware Clínico Especializado (Ej: FotoFinder)

En el extremo opuesto se encuentra el equipamiento clínico de referencia, ejemplificado por los sistemas **FotoFinder**[5]. Estos dispositivos representan el "estándar de oro" en imagenología dermatológica, ofreciendo una iluminación polarizada y resolución estandarizada inalcanzable para la óptica de un smartphone convencional.

Sin embargo, esta tecnología presenta barreras de entrada significativas: costos de implementación que superan los 20.000 USD y la necesidad de infraestructura física dedicada. La propuesta de este proyecto no busca competir en fidelidad óptica contra este hardware, sino ofrecer una alternativa de *triaje* descentralizada que complemente al equipamiento hospitalario, cubriendo la brecha de acceso en la atención primaria.

2.6.1. La propuesta de valor de esta memoria

Revisando el estado del arte, se identificó una brecha vacío clara: No existía una herramienta gratuita, privada y que funcionara sin internet. La Tabla 2.2 resume cómo este proyecto se diferencia de lo existente:

[4]SkinVision B.V. Service. *Melanoma detection app*. Disponible en: `https://www.skinvision.com`

[5]FotoFinder Systems GmbH. *Mapeo Corporal Total Automatizado (ATBM)*. Especificaciones técnicas disponibles en: `https://www.fotofinder.de/es`

Tabla 2.2: Comparación de soluciones. Mi propuesta destaca por ser la única local y offline.

Característica	**Apps Comerciales**	**Dermatoscopio**	**Esta Memoria**
Arquitectura	Nube (Lento)	Hardware Físico	**Local (Rápido)**
Privacidad	Baja (Sube fotos)	Alta	**Máxima**
Dependencia Internet	Alta	Nula	**Nula**
Costo	Alto	Muy Alto	**Gratis**

Capítulo 3. Diseño metodológico e ingeniería del sistema

3.1. Marco Metodológico Híbrido

Dada la complejidad del proyecto, que involucra investigación aplicada en *Deep Learning* e ingeniería de software, se diseñó un marco metodológico híbrido. Este enfoque integra el estándar industrial CRISP-DM (*Cross-Industry Standard Process for Data Mining*) para el flujo de datos, con los principios de Arquitectura Limpia (*Clean Architecture*) para el desarrollo del aplicativo.

Esta convergencia permitió asegurar la validez del modelo predictivo mientras se gestionaban las restricciones de hardware al despliegue en dispositivos móviles (*Edge AI*).

3.1.1. Adaptación de la Metodología CRISP-DM

El ciclo de vida del modelo siguió las seis fases de CRISP-DM, ajustadas para un entorno de optimización computacional:

Fase 1: Comprensión del Negocio (Business Understanding)

El objetivo principal se definió como la maximización de la Sensibilidad (*Recall*) para minimizar los falsos negativos en la deteccion del melanoma, un requisito importante en aplicaciones de teledermatología. Se definió como restricción de diseño la operatividad offline, obligando al uso de técnicas de compresión de modelos y cuantización post-entrenamiento.

Fase 2: Preparación de Datos (Data Preparation)

Consolidación de Múltiples Datasets La construcción del modelo se fundamentó en la consolidación de múltiples fuentes de referencia internacional, integrando los datasets **ISIC** (ediciones 2018, 2019 y 2020), **HAM10000** (*Human Against Machine with 10000 training images*) y el **Melanoma Skin Cancer Dataset**. Esta decisión se tomó para maximizar la diversidad fenotípica de lesiones cutáneas y mejorar la capacidad de generalización del modelo frente a variabilidad en condiciones de captura y demografía de pacientes.

Datasets integrados:

- **ISIC 2018:** 10 015 imágenes con 7 clases diagnósticas

- **ISIC 2019:** 25 331 imágenes con 8 clases diagnósticas
- **ISIC 2020:** 33 126 imágenes con diagnósticos binarios y multiclase
- **HAM10000:** 10 015 imágenes con 7 clases de lesiones pigmentadas
- **Melanoma Skin Cancer Dataset:** Colección complementaria con casos documentados

Total aproximado: ∼78 000+ imágenes dermatoscópicas provenientes de múltiples instituciones médicas.

Simplificación a Clasificación Binaria Una decisión importante de diseño fue la **simplificación del problema multiclase a clasificación binaria**. Los datasets originales contienen múltiples clases diagnósticas (melanoma, nevus melanocítico, queratosis seborreica, carcinoma basocelular, queratosis actínica, dermatofibroma, lesión vascular, etc.), pero presentan un desbalance multiclase extremadamente severo que dificulta el entrenamiento efectivo.

Reclasificación binaria implementada:

- **Clase MALIGNO (1):** Melanomas y lesiones malignas confirmadas.
- **Clase BENIGNO (0):** Todas las demás lesiones benignas (nevus, queratosis, dermatofibromas, etc.)

Esta simplificación se alinea con el objetivo de la app que es ser un médico primario: *triaje temprano de melanoma* el caso de uso crítico no es diagnosticar el tipo específico de nevus o queratosis, sino responder la pregunta binaria: **"¿Requiere esta lesión evaluación dermatológica urgente por sospecha de malignidad?"**

El problema multiclase original presentaba desbalances extremos donde algunas clases minoritarias (como melanoma amelanótico o dermatofibroma) representaban <1 % del dataset, generando:

1. Modelos con sesgo masivo hacia clases mayoritarias (nevus melanocítico)
2. Confusión entre clases benignas similares (nevus vs queratosis seborreica) sin valor clínico crítico
3. Falsos negativos catastróficos en melanomas al ser "enterrados" en la complejidad multiclase

La reclasificación binaria permite al modelo concentrar su capacidad de aprendizaje en la distinción crítica malignidad/benignidad, que es el factor determinante para derivación médica.

Características del Dataset Consolidado: El dataset consolidado tras la reclasificación binaria presenta el desafío característico de aplicaciones médicas: el **desbalance severo de clases**. Esto refleja la realidad epidemiológica donde las lesiones benignas son mucho más comunes que los melanomas:

- **Clase mayoritaria (Benigno):** ∼94 % de las imágenes (nevus, queratosis, dermatofibromas, etc.)
- **Clase minoritaria (Maligno):** ∼6 % de las imágenes (melanomas confirmados)
- **Ratio de desbalance original:** Aproximadamente 15-16:1 en el dataset consolidado

Al entrenar un modelo con este desbalance produce un problema crítico conocido como **sesgo hacia la clase mayoritaria**.

El problema del desbalance: Al entrenar modelos preliminares con el dataset desbalanceado original, se observó un comportamiento patológico: el modelo alcanzaba 94 % de accuracy prediciendo "BENIGNO" para TODAS las imágenes, sin aprender un patrón real. Matemáticamente tenemos que:

$$\text{Accuracy} = \frac{\#\ \text{Benignos}}{\#\ \text{Total}} \approx 0{,}94 \tag{3.1}$$

Esta solución es inútil para una aplicación médica dado que ignora el 100 % de los melanomas (falsos negativos críticos). Un falso negativo en clasificación de melanoma puede significar que un paciente no recibe tratamiento oportuno, lo cual puede ser fatal.

Estrategia de balanceo implementada: Random Undersampling Para resolver el desbalance, se implementó **Random Undersampling** de la clase mayoritaria. Esta técnica reduce aleatoriamente el número de ejemplos benignos manteniendo TODOS los ejemplos malignos (que son los casos críticos que no se puede perder).

El objetivo fue alcanzar un ratio 3:1 (benigno:maligno), más útil que el original 15:1, para forzar al modelo a aprender patrones discriminativos de melanoma sin sesgo hacia la clase mayoritaria.

Dataset final balanceado:

- **Total de imágenes:** 73 448
- **Imágenes benignas:** ∼55 086 (75 %)
- **Imágenes malignas:** ∼18 362 (25 %)

- **Ratio final:** 3:1

Particionamiento del dataset:

El dataset balanceado se dividió estratificadamente (manteniendo el ratio 3:1 en cada partición) en tres conjuntos:

Tabla 3.1: Particionamiento del dataset consolidado balanceado.

Conjunto	Porcentaje	Número de Imágenes	Uso
Train	70 %	51 412	Entrenamiento del modelo
Validation	15 %	11 016	Ajuste de hiperparámetros
Test	15 %	11 020	Evaluación final
Total	**100 %**	**73 448**	

La partición 70/15/15 es un estándar que garantiza:

1. Suficientes datos de entrenamiento (51 412 imágenes)
2. Validación robusta para monitorear overfitting (11 016 imágenes)
3. Evaluación final con conjunto independiente nunca visto (11 020 imágenes)

¿Por qué ratio 3:1 y no 1:1 (balanceo perfecto)?

Se consideraron tres opciones de balanceo:

- **Opción A - Ratio 1:1 (balanceo perfecto):** Reduciría benignos a ∼18 362, resultando en solo 36 724 imágenes totales. Esto descartaría 80 % del dataset, perdiendo diversidad fenotípica valiosa (diferentes tipos de nevus, queratosis, etc.).
- **Opción B - Mantener ratio original 24:1:** El modelo seguiría sesgado, como se observó en experimentos preliminares.
- **Opción C - Ratio 3:1 (seleccionada):** Balance pragmático que mantiene 73 448 imágenes (16 % del dataset original), con suficiente diversidad benigna.

La Opción C demostró empíricamente el mejor resultado en los experimentos de validación.

Aumento de Datos (*Data Augmentation*): El aumento de datos resultó fundamental para mejorar la capacidad de generalización del modelo, especialmente al trabajar con imágenes médicas donde la variedad de imágenes es alta (diferentes cámaras, iluminaciones, ángulos, etc). Se implementó un pipeline de transformaciones geométricas y cromáticas utilizando `torchvision.transforms` para simular las condiciones reales de captura con smartphones.

Pipeline de transformaciones para entrenamiento:

1. **Transformaciones Geométricas:**

   - `RandomResizedCrop(384)`: Recorte aleatorio con redimensionamiento a 384×384 píxeles. simula diferentes distancias de captura y composiciones.
   - `RandomHorizontalFlip(p=0.5)`: Aplica una reflexión horizontal aleatoria con una probabilidad del 50 %. Esta transformación es a mi parecer fundamental dado que las lesiones dermatológicas carecen de una orientación canónica. Además, permite simular la variabilidad en el ángulo de captura por parte del usuario, forzando al modelo a aprender características robustas e invariantes a la orientación de la fotografía.
   - `RandomVerticalFlip(p=0.5)`: Volteo vertical con 50 % de probabilidad, por la misma razón anterior.
   - `RandomRotation(20)`: Rotación aleatoria de ±20 grados. Simula diferentes ángulos de captura de la cámara del smartphone.

2. **Transformaciones Cromáticas:**

   - `ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2)`: Variaciones aleatorias de brillo (±20 %), contraste (±20 %) y saturación (±20 %). Esto es crítico porque diferentes smartphones tienen diferentes sensores y algoritmos de procesamiento de imagen muy distintos. Por ejemplo, Samsung tiende a saturar los colores más que iPhone, que tiene procesamiento más neutro.

3. **Normalización (Crítica):**

   - `Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])`: Normalización con estadísticas de ImageNet. Esta normalización es obligatoria porque el modelo fue pre-entrenado con estas mismas estadísticas. Usar valores diferentes no permitiría realizar transfer learning.

Data augmentation actúa como un regularizador implícito, forzando al modelo a aprender representaciones invariantes a transformaciones geométricas y cromáticas, lo cual mejora significativamente su robustez en condiciones de captura no ideales.

Fase 3: Modelado y Optimización de Hardware (Modeling)

Debido a limitaciones de infraestructura detectadas con TensorFlow en entornos Windows con GPU NVIDIA RTX 3080 Ti (incompatibilidad de versiones CUDA y dependencias rotas), se migró el desarrollo a **PyTorch**, que ofrece soporte nativo y estable para GPU en Windows sin las complicaciones de configuración que presenta TensorFlow.

Arquitectura del Modelo: EfficientNetV2-M El modelo implementado utiliza **EfficientNetV2-M** (*Medium*) como backbone, con una cabeza de clasificación personalizada. La arquitectura completa se compone de:

1. **Modelo Pre-entrenado:**

   Se implementó EfficientNetV2-M como red base (backbone) del sistema. Para reducir los tiempos de entrenamiento y mejorar el rendimiento en un dataset limitado, se utilizaron pesos pre-entrenados en ImageNet-1K, permitiendo un fine-tuning más eficiente de los parámetros del modelo

   Las especificaciones técnicas detalladas de la arquitectura, incluyendo las dimensiones tensoriales de entrada/salida y el código de implementación en PyTorch, se encuentran disponibles en el **Apéndice A.1** (ver Tabla A.1 y Código A.1).

   Transfer Learning: Las capas convolucionales pre-entrenadas ya capturan features visuales universales (bordes, texturas, patrones de color) aplicables a dermatología. Esto reduce el tiempo de entrenamiento y la cantidad de datos necesarios comparado con entrenar desde cero.

2. **Diseño del Clasificador Personalizado:**

   Para adaptar el extractor de características a la tarea específica de detección binaria del melanoma, se reemplazó la capa de clasificación original de ImageNet por una arquitectura densa personalizada.

   El diseño de esta nueva cabecera (cuya implementación detallada se encuentra en el Código A.2 del Apéndice prioriza la regularización y la estabilidad mediante las siguientes decisiones:

3. **Estrategia de Regularización:** Los datasets médicos suelen generar sobreajuste (*overfitting*) debido a que, en comparación con otros problemas, hay pocas imágenes disponibles para la gran variedad de formas que pueden tener los lunares. Existe el riesgo de que el modelo empiece a memorizar elementos que no son importantes —como pelos, reglas de medición o la iluminación de la foto— en lugar de aprender a identificar la enfermedad real.

   Para solucionar esto, se aplicó una estrategia de *Dropout* (0.5 inicial y 0.3 final). Como explican Srivastava et al. [33], esta técnica evita que las neuronas dependan demasiado unas de otras (lo que llaman "co-adaptación"). Al desactivar neuronas al azar durante el entrenamiento, obligamos a la red a buscar patrones más seguros y generales, evitando que se confunda con este ruido visual o detalles irrelevantes.

4. **No-Linealidad Moderna (GELU):** Se seleccionó la función de activación **GELU** (*Gaussian Error Linear Unit*) en lugar de la tradicional ReLU. Hendrycks y Gimpel [14] argumentan que GELU pondera las entradas por su magnitud en lugar de cerrarlas abruptamente (como ReLU), preservando información probabilística vital. Esta función, utilizada en arquitecturas de vanguardia como BERT y GPT-3, ha demostrado superar a ReLU y ELU en benchmarks como CIFAR-10/100, mitigando el desvanecimiento del gradiente gracias a su curvatura suave y diferenciable en todo el dominio.

5. **Estabilización del Entrenamiento:** La inclusión de una capa de *Batch Normalization* intermedia permite mitigar el fenómeno del "cambio de covariable interno" (*internal covariate shift*), tal como lo definieron Ioffe y Szegedy [18]. Esta normalización no solo estabiliza la distribución de las entradas de cada capa, sino que actúa como un regularizador suave, permitiendo el uso de tasas de aprendizaje hasta 14 veces más altas sin riesgo de divergencia, acelerando drásticamente la convergencia del modelo.

6. **Proyección de Salida y Estabilidad Numérica:** La capa final proyecta las características a un vector de tamaño 2 (*logits*), correspondientes a las clases *Benigno* y *Maligno*. Siguiendo las buenas prácticas de computación numérica descritas por Goodfellow et al. [11], no se aplica una función Softmax explícita en la arquitectura, sino que se integra en la función de pérdida *CrossEntropyLoss*. Esta fusión utiliza la técnica *LogSumExp* para evitar el desbordamiento numérico (*overflow/underflow*) durante el cálculo de gradientes, garantizando una mayor precisión matemática.

Estrategia de Transfer Learning y Fine-Tuning:

Para preservar el conocimiento visual adquirido por el modelo en ImageNet y adaptarlo al problema dermatológico, se diseñó una estrategia de entrenamiento en dos etapas (ver implementación en Código A.3 del Apéndice):

1. **Congelamiento Inicial (*Freezing*):** Durante las primeras épocas, se congelaron los pesos de las capas convolucionales del *backbone* (aproximadamente los primeros dos tercios de la red). Esto permitió entrenar exclusivamente la nueva cabecera de clasificación, evitando que los gradientes ruidosos iniciales destruyeran los detectores de características pre-entrenados (fenómeno conocido como *catastrophic forgetting*).

2. **Descongelamiento Progresivo (*Fine-Tuning*):** Una vez estabilizado el clasificador, se descongelaron todas las capas para realizar un ajuste fino (*fine-tuning*) de toda la red *end-to-end* con una tasa de aprendizaje reducida. Esta fase permite que los filtros convolucionales se especialicen en texturas específicas de lesiones cutáneas.

Manejo del Desbalance: Aprendizaje Sensible al Costo A pesar del submuestreo realizado en la etapa de preparación de datos, el conjunto de entrenamiento mantiene un desbalance residual (3:1). Para contrarrestar esto, se implementó una función de pérdida llamada Entropía Cruzada Ponderada (*Weighted Cross-Entropy*).

Esta técnica asigna un coeficiente de penalización inversamente proporcional a la frecuencia de cada clase, calculado dinámicamente según la Ecuación 3.2:

$$w_c = \frac{N_{\text{total}}}{N_{\text{clases}} \times N_c} \tag{3.2}$$

Donde N_c es el número de muestras de la clase c. En la práctica, esto hace que el modelo recibe una penalización aproximadamente 3 veces mayor (ver Código A.4 en Apéndice) por fallar en la detección de un melanoma que en un caso benigno, incentivando la maximización de la sensibilidad (*Recall*).

Configuración de Hiperparámetros La configuración final del entrenamiento se estableció tras múltiples iteraciones experimentales para encontrar el equilibrio entre convergencia y generalización (ver tabla de valores en Código A.18 del Apéndice). Se utilizó el optimizador AdamW [22], seleccionado por su manejo desacoplado del decaimiento de pesos (*weight decay*). Esta característica corrige la implementación de la regularización L_2 en el algoritmo Adam estándar, proporcionando una generalización superior en modelos profundos. Asimismo, se utilizó un planificador de tasa de aprendizaje (*Scheduler*) de tipo *Cosine Annealing* para refinar suavemente los pesos en las etapas finales del entrenamiento.

Optimizaciones de Hardware Implementadas Para maximizar el rendimiento de la GPU NVIDIA RTX 3080 Ti (12GB VRAM) y reducir los tiempos de entrenamiento, se implementaron estrategias de optimización a nivel de operaciones tensoriales:

1. **Automatic Mixed Precision (AMP):**

   Se utilizó el entrenamiento en precisión mixta, una técnica que ejecuta las operaciones matriciales intensivas (convoluciones y multiplicaciones) en formato de media precisión (*half precision*, FP16), mientras mantiene una copia maestra de los pesos en precisión simple (FP32) para preservar la estabilidad numérica durante la retropropagación.

   La implementación técnica (ver Código A.5 en el Apéndice A) hace uso de los *Tensor Cores* de la arquitectura Ampere, gestionando dinámicamente el escalado de la función de pérdida (*loss scaling*) para evitar el desvanecimiento de gradientes pequeños.

   Beneficios observados en este proyecto:

- **Eficiencia de Memoria:** Reducción del uso de VRAM en aproximadamente un 40 % (descendiendo de 8.2 GB a 4.9 GB), lo que permitió aumentar el tamaño del lote (*batch size*).
- **Velocidad de Cómputo:** Aceleración del tiempo de entrenamiento por época más eficiente (no fue medido a detalle)

Acumulación de Gradientes (Gradient Accumulation):

Debido a las limitaciones de memoria física de la GPU (12 GB), no fue posible procesar lotes de 64 imágenes de alta resolución (384×384) simultáneamente. Para solucionar esto sin sacrificar la estabilidad del entrenamiento, se implementó la acumulación de gradientes.

Esta técnica permite **simular** un lote grande dividiéndolo en partes más pequeñas. El sistema procesa secuencialmente grupos de 32 imágenes, pero en lugar de actualizar el modelo inmediatamente, "guarda" y suma los resultados matemáticos (gradientes). Solo después de procesar dos grupos (totalizando 64 imágenes) se realiza la actualización de los pesos. Esto permite obtener los beneficios matemáticos de un lote grande sin desbordar la memoria RAM del hardware.

El algoritmo implementado (ver Código A.7 en el Apéndice)

Aceleración de Hardware y Pipeline de Datos:

Para explotar las capacidades específicas de la arquitectura NVIDIA Ampere, se habilitaron optimizaciones de bajo nivel en el backend de PyTorch (ver configuración en Código A.8 del Apéndice). Específicamente, se activó el uso de **TensorFloat-32 (TF32)**, un formato numérico que acelera las multiplicaciones matriciales en los *Tensor Cores* manteniendo una precisión comparable a FP32 para tareas de Deep Learning. Adicionalmente, se configuró el *autotuner* de CuDNN para seleccionar heurísticamente los algoritmos de convolución más eficientes para el hardware disponible.

Paralelamente, para evitar cuellos de botella en la transferencia de datos CPU-GPU, se optimizó el *DataLoader* utilizando memoria paginada (*pinned memory*) y pre-carga de lotes con múltiples hilos de ejecución.

Estrategia de Learning Rate:

Se implementó un planificador de tasa de aprendizaje (*Scheduler*) basado en **Cosine Annealing**. Esta estrategia reduce la tasa de aprendizaje siguiendo una curva cosenoidal, comenzando con un valor alto para una exploración rápida y disminuyendo suavemente hasta cero. Esto permite que el modelo converja a mínimos más planos y generalizables en las etapas finales del entrenamiento (ver Código A.9 en Apéndice).

Ciclo de Entrenamiento y Resultados (Training Loop) El ciclo principal de entrenamiento (implementado detalladamente en el Código A.10 del Apéndice) orquesta la fase de aprendizaje, la validación cruzada y el guardado de puntos de control (*checkpoints*).

Fase 4: Evaluación del Modelo (Evaluation)

Tras finalizar el entrenamiento, se evaluó el modelo utilizando el conjunto de prueba (*test set*) compuesto por 11 020 imágenes inéditas. Esta evaluación es crítica para estimar el rendimiento real del sistema en un entorno no controlado.

Resultados de Clasificación La Tabla 3.2 resume el desempeño del modelo para distinguir entre lesiones benignas y malignas. Se han omitido métricas agregadas complejas para enfocar el análisis en la capacidad de detección de cada clase específica.

Tabla 3.2: Desempeño del modelo en el conjunto de prueba (11 020 imágenes).

Clase	**Precision**	**Recall (Sensibilidad)**	**F1-Score**	**Muestras**
Benigno	98.49 %	98.56 %	98.52 %	8 265
Maligno	95.67 %	**95.46 %**	95.57 %	2 755
Exactitud Global (Accuracy)				**97.79 %**

Análisis de los resultados:

- **Exactitud (Accuracy): 97.79 %**. El sistema clasificó correctamente 10 776 de las 11 020 imágenes totales. Esto indica una alta fiabilidad general operativa.
- **Sensibilidad (Recall) en Melanoma: 95.46 %**. Esta es la métrica más crítica para una aplicación de salud. Indica que el modelo es capaz de detectar el 95.46 % de los casos reales de cáncer. En términos prácticos, de cada 100 melanomas presentados, el sistema identifica correctamente a 95, minimizando el riesgo de falsos negativos (pacientes enfermos no diagnosticados).
- **Precisión en Melanoma: 95.67 %**. Esta métrica refleja la "confiabilidad"de la alarma. Cuando el sistema indica que una lesión es maligna, es correcto el 95.67 % de las veces. Esto es importante para no saturar el sistema de salud con derivaciones innecesarias (falsos positivos).

Matriz de Confusión y Análisis de Errores La Matriz de Confusión (Tabla 3.3) permite visualizar detalladamente el comportamiento del modelo, desglosando las predicciones correctas e incorrectas para cada clase.

De esta distribución se desprende la capacidad discriminativa del sistema:

- Se identificaron correctamente **8 146 lesiones benignas** y **2 630 melanomas**.
- El error más crítico (Falsos Negativos) se limitó a 125 casos, donde el modelo falló en detectar la malignidad.

Tabla 3.3: Matriz de confusión obtenida en el conjunto de prueba.

	Predicción: Benigno	**Predicción: Maligno**
Real: Benigno	8 146 (Verdaderos Negativos)	119 (Falsos Positivos)
Real: Maligno	125 (Falsos Negativos)	2 630 (Verdaderos Positivos)

- Por otro lado, 119 lesiones benignas fueron clasificadas erróneamente como sospechosas (Falsos Positivos).

Métricas Clínicas: Sensibilidad y Especificidad En el ámbito médico, la exactitud global no es suficiente. Es necesario evaluar la **Sensibilidad** (capacidad de no perder casos enfermos) y la **Especificidad** (capacidad de no alarmar a pacientes sanos).

$$\text{Sensibilidad} = \frac{TP}{TP + FN} = \frac{2\,630}{2\,755} \approx 95{,}46\,\% \tag{3.3}$$

$$\text{Especificidad} = \frac{TN}{TN + FP} = \frac{8\,146}{8\,265} \approx 98{,}56\,\% \tag{3.4}$$

Interpretación de los resultados:

- Una Sensibilidad del 95.46 % ,de cada 100 melanomas presentados, detecta aproximadamente 95. Los casos omitidos (falsos negativos) corresponden, en su mayoría, a lesiones muy incipientes o atípicas que presentan desafíos incluso para el ojo humano experto.
- Una Especificidad del 98.56 % indica que el sistema es eficiente filtrando casos irrelevantes. Solo un 1.4 % de las lesiones benignas generan una falsa alarma, lo cual es fundamental para evitar la saturación de los servicios de dermatología con derivaciones innecesarias.

Comparación con el Estándar Humano Al contrastar el rendimiento del modelo con la literatura médica vigente, los resultados sitúan a *DermatIA* en un nivel competitivo frente a la evaluación clínica tradicional:

- **Médicos de Atención Primaria:** La inspección visual simple realizada por médicos no especialistas suele alcanzar una sensibilidad promedio entre el 60 % y 70 % [3, 39].
- **Dermatólogos Expertos:** El estándar clínico (*Gold Standard* humano) se sitúa típicamente entre el 80 % y 90 % de sensibilidad, cifra que varía según la experiencia del profesional y el uso de dermatoscopía manual [12, 25].
- **DermatIA (Modelo Propuesto):** Con una sensibilidad del 95.46 % en el conjunto de prueba, el sistema demuestra una capacidad de detección comparable y, en tareas de

screening visual puro, superior a la de la evaluación humana, lo que valida su utilidad como herramienta de segunda opinión.

Interpretación de los Resultados Es fundamental interpretar estos hallazgos con rigor científico. La eficiencia demostrada en el conjunto de test no es directamente extrapolable a la práctica clínica sin considerar las siguientes brechas:

1. **Brecha de Dominio (Entorno Controlado vs. Realidad):** La evaluación se realizó sobre imágenes dermatoscópicas estandarizadas. En el uso real, factores como la iluminación ambiental deficiente, el desenfoque por movimiento o la compresión de imagen del smartphone pueden introducir ruido que degrade la precisión.

2. **Visión Parcial:** El algoritmo opera exclusivamente sobre la morfología visual estática de la lesión. Carece del contexto clínico integral (anamnesis, palpación, evolución temporal, antecedentes familiares) que un especialista utiliza para emitir un diagnóstico certero.

3. **Rol de Soporte:** La herramienta está diseñada estrictamente como un mecanismo de **triaje**, útil para alertar sobre casos de riesgo, pero siempre supeditada a la validación final de un profesional de la salud.

Análisis de Errores y Sesgos Detectados El análisis cualitativo de los fallos del modelo (125 falsos negativos y 119 falsos positivos) revela limitaciones tecnológicas y sesgos inherentes a los datos:

Falsos Negativos (Riesgo Crítico): Los casos malignos no detectados corresponden principalmente a:

- Melanomas amelanóticos (sin pigmentación) que carecen de las características visuales clásicas aprendidas por la red.

- Lesiones en estadios muy tempranos (*in situ*) con patrones dermatoscópicos sutiles.

- Imágenes con artefactos severos (desenfoque o reflejos) que ocultan la estructura de la lesión.

Falsos Positivos: Las lesiones benignas clasificadas erróneamente como malignas incluyen:

- Nevus atípicos o displásicos que, aunque benignos, presentan irregularidades visuales similares a las del melanoma.

- Queratosis seborreicas pigmentadas con bordes irregulares.

Sesgo de Pigmentación y Fototipo (Hallazgo Post-Despliegue): Durante la validación funcional en dispositivos móviles, se observó un comportamiento sesgado en dos escenarios específicos:

1. **Lesiones Hiperpigmentadas:** El modelo tiende a asignar puntuaciones de riesgo artificialmente altas a lunares muy oscuros o negros, independientemente de su regularidad, lo que incrementa la tasa de falsos positivos.

2. **Fototipos de Piel Oscuros (Fitzpatrick V-VI):** Se detectó una disminución en la confianza de la predicción al analizar pieles oscuras. Esto sugiere una subrepresentación de estos fototipos en el dataset de entrenamiento (compuesto mayoritariamente por pieles caucásicas), una limitación ética y técnica conocida en la dermatología computacional que debe abordarse en trabajos futuros para garantizar la equidad del sistema.

Exportación y Estandarización del Modelo (ONNX) El modelo original se entrenó en Python/PyTorch, pero este entorno no es eficiente para ejecutarse directamente en un celular. Para solucionar esto, convertimos el modelo al formato ONNX (*Open Neural Network Exchange*).

El objetivo fue generar un archivo binario optimizado que contenga la estructura de la red neuronal y los pesos entrenados, pero desacoplado de las librerías de Python. Esto permite que el motor de inferencia (*ONNX Runtime*) ejecute el modelo en el dispositivo móvil aprovechando el hardware nativo (CPU/GPU del teléfono).

Validación de la Exportación: Para asegurar que el modelo no perdiera precisión al convertirse, realizamos dos validaciones técnicas:

- Consistencia Numérica: Comparamos los vectores de salida (*logits*) del modelo original con el modelo ONNX usando los mismos inputs. La diferencia fue menor que 10^{-5}, lo que confirma que la lógica matemática se mantuvo intacta.

- Estructura del Grafo: Se verificó que el archivo final (203.81 MB) contuviera los 473 nodos operacionales necesarios, con optimizaciones de grafo estático (*constant folding*) aplicadas.

Integración en el Cliente Móvil (Flutter) La integración en la app se realizó mediante el plugin `onnxruntime`. El desafío técnico principal fue replicar exactamente el pipeline de preprocesamiento de imágenes utilizado durante el entrenamiento.

Si la imagen de entrada en el móvil no tiene las mismas propiedades matemáticas que las imágenes de entrenamiento, la inferencia fallará. Por ello, se implementó una rutina en Dart que transforma el *buffer* de la cámara antes de pasarlo al modelo:

1. Redimensionamiento: Se escala la imagen a una resolución fija de 384×384 píxeles.

2. Normalización: Se estandarizan los valores de los píxeles (RGB) restando la media y dividiendo por la desviación estándar del dataset ImageNet. Esto centra los datos para que la red neuronal los interprete correctamente.

3. Transposición de Canales: Las cámaras móviles entregan la imagen en formato [Alto, Ancho, Canales], pero el modelo ONNX requiere el formato tensorial [Canales, Alto, Ancho]. Se reordenan los bytes en memoria para cumplir con este requisito.

Benchmark en dispositivo: Las pruebas en un Samsung S25 Ultra mostraron una latencia de inferencia de 300–350 ms. Esto valida que el procesamiento es lo suficientemente rápido como para una experiencia de usuario fluida, sin bloquear la interfaz.

Fase 5: Despliegue (Deployment)

El despliegue sigue una arquitectura Edge Computing. A diferencia de las arquitecturas cliente-servidor tradicionales donde la imagen se sube a una API, aquí todo el procesamiento ocurre localmente en el dispositivo.

Esta decisión de diseño ofrece tres ventajas técnicas y operativas: Latencia Cero de Red: Al no depender de subir una imagen pesada a la nube, el análisis es inmediato, incluso en modo avión. Privacidad por Diseño: Los datos biométricos (fotos de la piel) nunca salen del almacenamiento local del teléfono, lo que elimina el riesgo de interceptación de datos. Reducción de Costos: No se requiere mantener servidores de inferencia (GPUs en la nube) activos 24/7, ya que el costo computacional se distribuye al dispositivo del usuario.

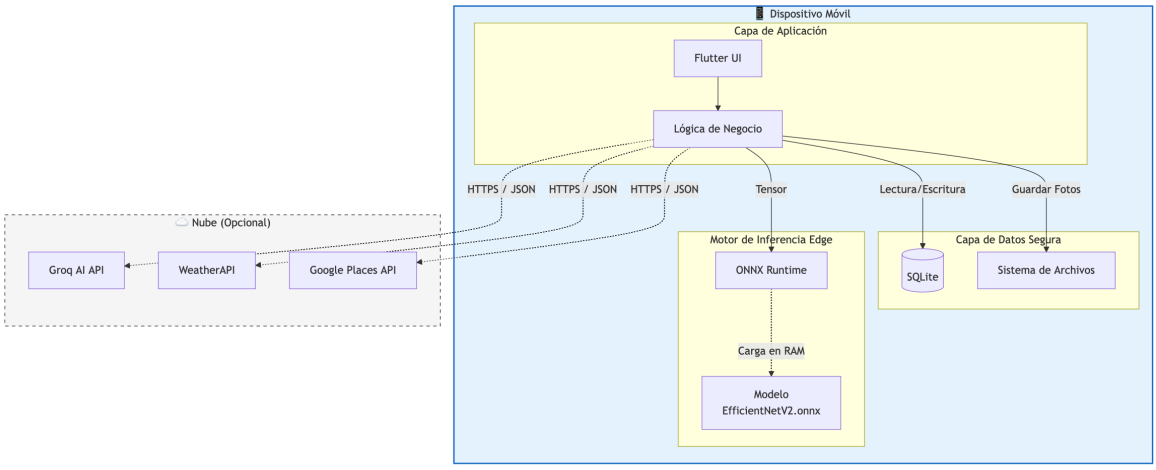

Figura 3.1: Arquitectura de Despliegue Híbrida. El diagrama ilustra el enfoque *Edge-First*: el procesamiento crítico y la persistencia de datos ocurren localmente en el dispositivo móvil (derecha) para garantizar la privacidad y el funcionamiento *offline*, mientras que los servicios en la nube (izquierda) actúan únicamente como fuentes de datos complementarias.

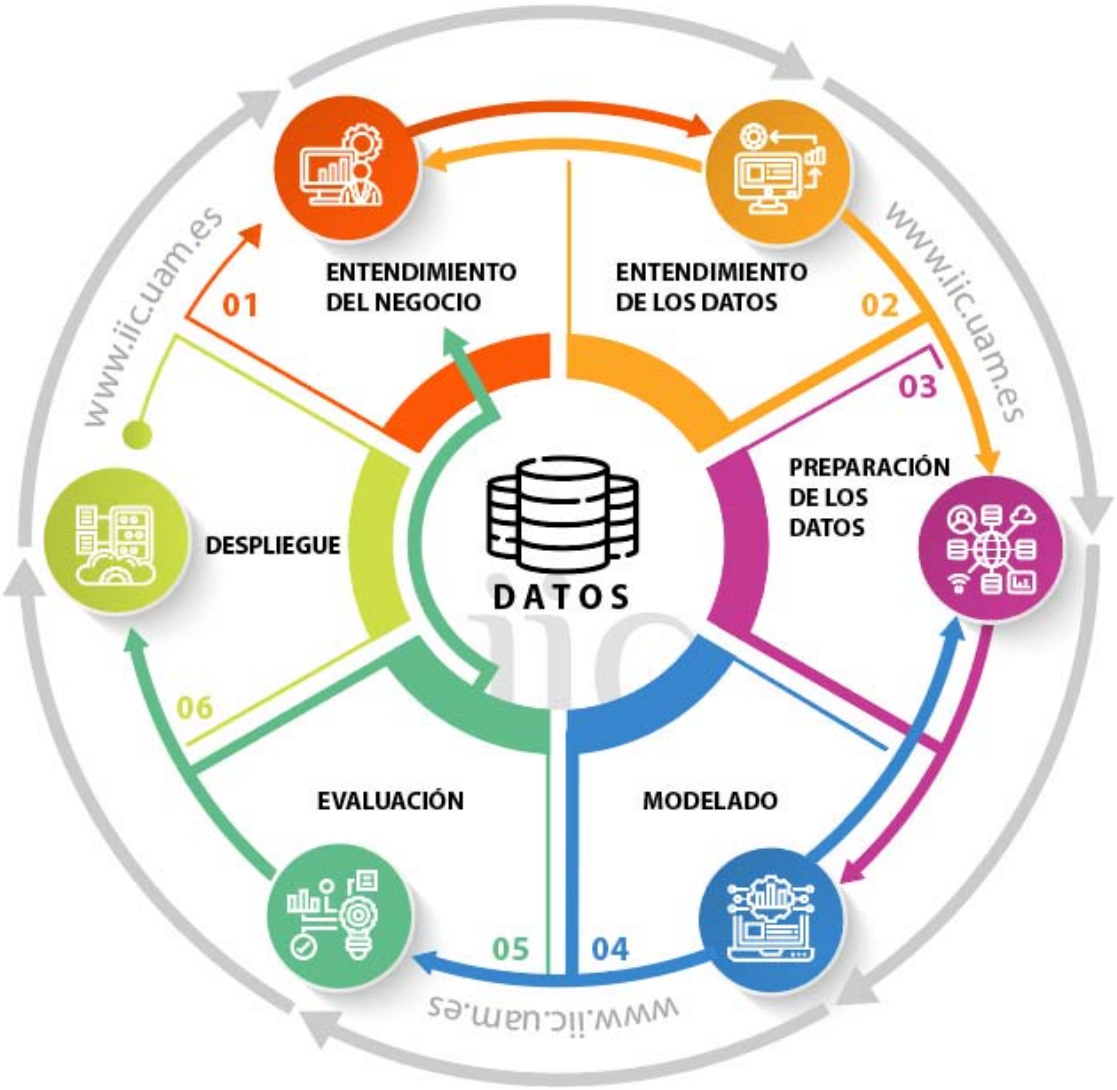

Figura 3.2: Ciclo de vida CRISP-DM adaptado.

3.1.2. Diseño de Software y Arquitectura

Para asegurar que *Dermat-IA* sea una aplicación robusta y fácil de mantener, se implementó el patrón de Arquitectura Limpia (*Clean Architecture*).

El objetivo central de este diseño es separar las responsabilidades: la interfaz visual (lo que ve el usuario) no debe saber cómo funcionan los datos internos (base de datos o IA). Esto permite modificar una parte del sistema sin romper las otras.

Estructura en Capas

El código se organiza en tres capas concéntricas, en las que las dependencias fluyen estrictamente hacia el interior. Esto significa que la lógica de negocio no depende de la interfaz gráfica ni de la base de datos (ver Figura 3.3).

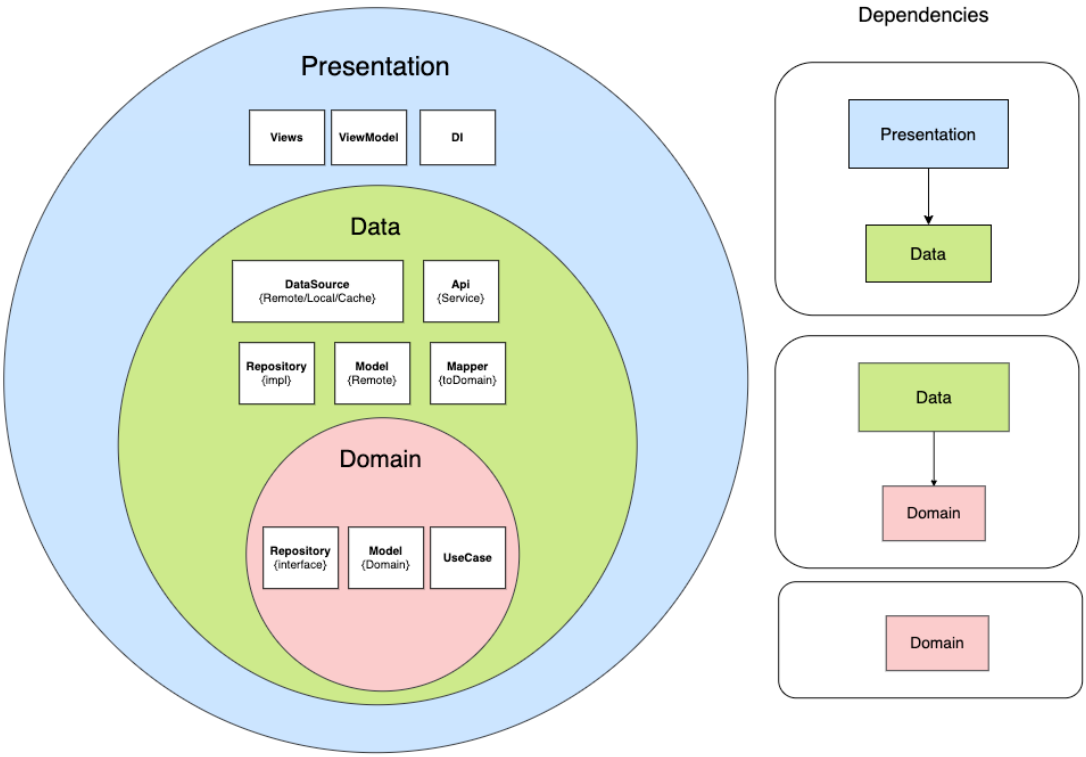

Figura 3.3: Arquitectura de DermatIA. Diagrama de capas mostrando la separación entre la Presentación (UI), los Datos (Repositorios) y el Dominio (Lógica pura).

Las capas se definen de la siguiente manera:

- **1. Capa de Presentación (UI):** Es la capa más externa. Contiene las pantallas, botones y el diseño visual en Flutter. Su única función es mostrar información al usuario y capturar sus acciones.

- **2. Capa de Dominio (El Núcleo):** Aquí vive la lógica de negocio pura. Define las Entidades (ej. qué datos componen un registro médico) y las reglas del sistema, sin importar si la app se ejecuta en Android, iOS o Web.
- **3. Capa de Datos (Infraestructura):** Contiene las implementaciones técnicas :
  - **Base de Datos:** Código SQL para guardar en SQLite.
  - **Servicios de IA:** Código para ejecutar el modelo ONNX.
  - **Repositorios:** Piezas de código que conectan la base de datos con el dominio.

Caso Práctico: Guardado de un Diagnóstico

Para ilustrar cómo interactúan estas capas, se analizó el flujo que ocurre cuando un usuario presiona "Guardar"tras un análisis (ver implementación técnica en Código A.12 del Apéndice.

El proceso atraviesa la arquitectura en tres pasos:

1. **Paso 1 (UI):** La pantalla `AddRecordScreen` recopila los datos. No guarda nada directamente; solo empaqueta la información y la envía a la siguiente capa.
2. **Paso 2 (Repositorio):** La capa de negocio recibe los datos. Aquí se validan las reglas de seguridad (por ejemplo, verificar que el nivel de riesgo sea un número válido entre 0 y 1). Si los datos son correctos, ordena su almacenamiento.
3. **Paso 3 (Datos):** Finalmente, el servicio `DatabaseService` traduce la orden a una sentencia SQL real y escribe la información en la memoria del teléfono.

Diseño de Persistencia y Base de Datos

Dada la naturaleza crítica de los datos médicos y el requisito *Offline-First*, se implementó un motor de base de datos relacional embebido (**SQLite**). El sistema gestiona dos bases de datos independientes para segregar responsabilidades: `dermat_ia.db` (registros clínicos) y `reminders.db` (agenda de salud).

Modelo de Datos Implementado: El sistema cuenta con dos tablas principales en SQLite, diseñadas para almacenar los datos esenciales de la aplicación:

1. **SKIN_RECORDS**: Tabla principal que almacena los resultados de la inferencia del modelo junto con metadatos clínicos del usuario. Incluye 9 campos: identificador UUID v4, ruta de imagen local, parte del cuerpo analizada, diagnóstico (BENIGNO/MALIGNO), nivel de riesgo [0.0 - 1.0], confianza del modelo, notas opcionales del usuario y fecha de creación y actualización.

2. **REMINDERS** : Tabla de recordatorios con 14 campos que incluyen configuración de notificaciones, repetición programada (diaria, semanal, mensual), prioridad (1-5) y relación opcional con registros dermatológicos mediante `related_record_id`.

Optimizaciones de Performance: Se implementaron 3 índices estratégicos en `SKIN_RECORDS`:

- **idx_created_at**: Índice B-Tree descendente para ordenar historial por fecha reciente (consulta más frecuente).
- **idx_body_part**: Índice hash para filtros por zona anatómica (FACE, ARM, LEG, etc.).
- **idx_risk_level**: Índice B-Tree descendente para priorizar lesiones de alto riesgo ($\geq 0{,}7$), permitiendo consultas eficientes con complejidad $O(\log n)$.

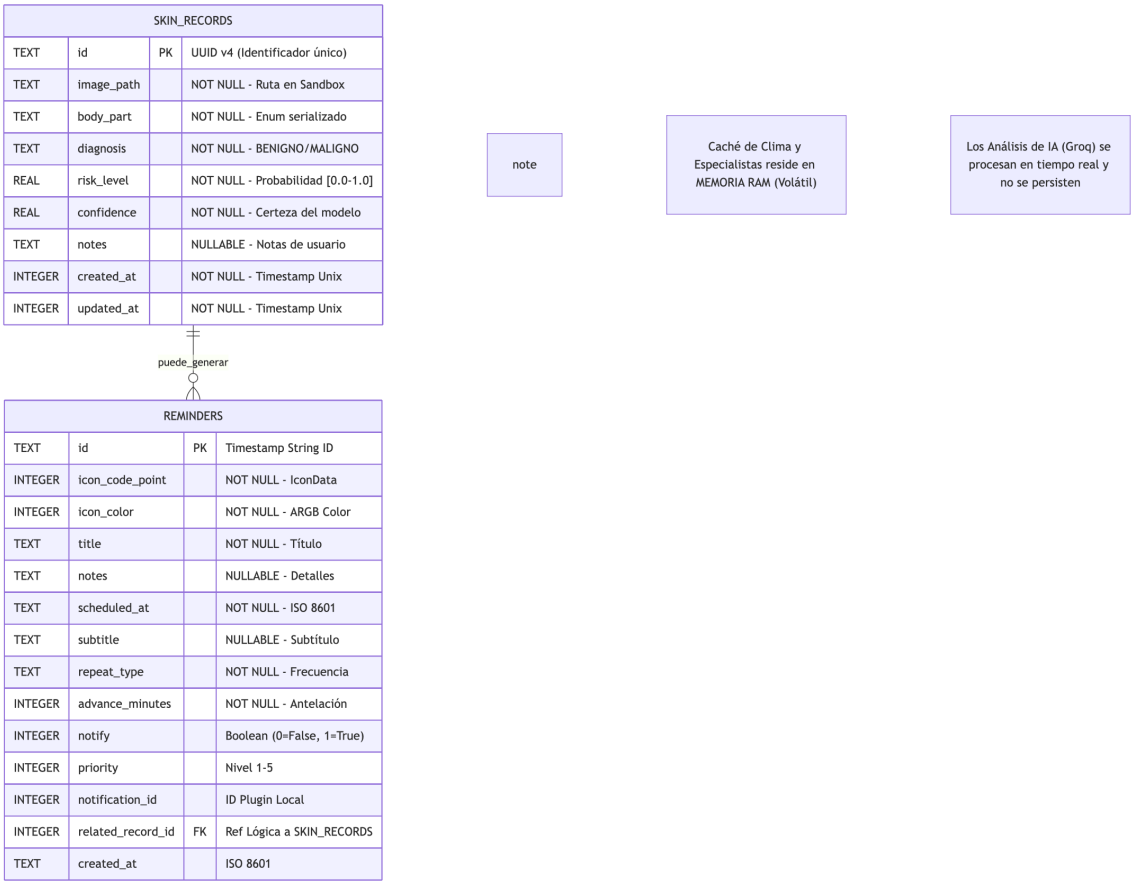

Figura 3.4: Modelo Entidad-Relación de la Base de Datos. Esquema mostrando las dos tablas principales (SKIN_RECORDS y REMINDERS) con sus campos, tipos de datos, relaciones (1:N opcional) e índices de optimización. La tabla SKIN_RECORDS almacena registros dermatológicos con resultados de IA, mientras que REMINDERS gestiona notificaciones programadas asociadas opcionalmente a análisis previos.

Casos de Uso Principales: El sistema implementa seis casos de uso críticos que cubren el flujo completo de interacción del usuario con la aplicación:

1. **UC-01: Análisis de Lesión** - Captura de imagen, procesamiento con IA (EfficientNetV2-M), clasificación (BENIGNO/MALIGNO), generación de explicación con Groq AI, y persistencia en base de datos.

2. **UC-02: Consulta de Historial** - Visualización de registros ordenados cronológicamente con filtrado por parte del cuerpo.

3. **UC-03: Búsqueda de Especialistas** - Geolocalización GPS, consulta a Google Places API, caché local (TTL 15 min), integración con Google Maps y marcador telefónico.

4. **UC-04: Gestión de Recordatorios** - Programación de notificaciones (diaria/semanal/mensual), vinculación opcional con registros dermatológicos, priorización (1-5).

5. **UC-05: Consulta de Clima/UV** - Obtención del índice UV actual vía WeatherAPI, análisis del nivel de riesgo, generación de recomendaciones de protección solar, caché local (TTL: 20 min).

6. **UC-06: Educación** - Acceso a contenido informativo sobre melanoma, regla ABCDE, factores de riesgo, y prevención mediante enlaces externos a recursos médicos confiables (AAD, WHO, Cancer.org).

Flujo de Datos y Seguridad (Privacy-by-Design)

El flujo de información sigue un protocolo estricto de seguridad local para cumplir con la Ley 20.584 (Derechos y Deberes del Paciente).

1. **Captura y Procesamiento:** La imagen se captura en memoria volátil y se procesa mediante el servicio ONNX.

2. **Persistencia Segura:**

   - La imagen física se almacena en el *App Sandbox* (directorio privado de la aplicación: `/data/data/com.dermatia/files/`), inaccesible para otras aplicaciones o malware externo.
   - Los metadatos se insertan transaccionalmente en SQLite.

3. **Aislamiento:** Al no existir componentes de *Backend* ni envío de datos a la nube, se elimina por diseño el vector de ataque de interceptación de red, garantizando la soberanía total de los datos del paciente.

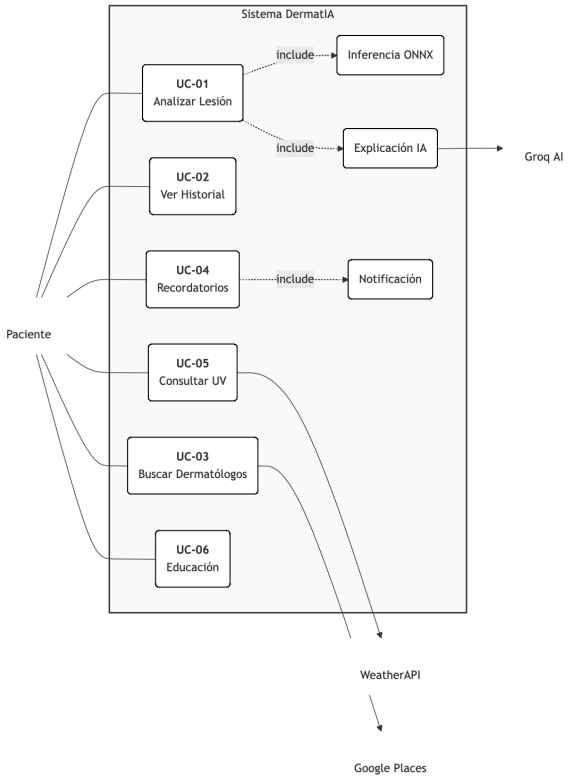

Figura 3.5: Diagrama de Casos de Uso del Sistema. Muestra los 6 casos de uso principales implementados (UC-01 a UC-06) y sus flujos de interacción. Se observa el flujo central de análisis de lesión (UC-01) que integra captura de imagen, procesamiento ML, generación de explicación IA, y persistencia en BD. Los casos de uso auxiliares (UC-03, UC-04, UC-05) optimizan experiencia de usuario mediante caché volátil en memoria RAM, mientras que UC-02 y UC-06 operan exclusivamente con datos locales. El diagrama incluye decisiones de flujo, tiempos de respuesta estimados, y complejidad relativa de cada caso de uso (Alta/Media/Baja).

3.1.3. Integraciones y Servicios Externos

Aunque la funcionalidad principal es *offline*, el sistema se enriquece mediante servicios en la nube cuando existe conexión:

- **IA Generativa:** Integración con **Groq** (modelo Llama-3.3-70b) para generar una explicación sencilla y contextualizada en lenguaje natural a partir del diagnóstico técnico.

- **Contexto Ambiental:** Uso de **WeatherAPI** para monitorear en tiempo real el índice UV y ofrecer recomendaciones de fotoprotección.

- **Acción Médica:** Conexión con **Google Places API** para geolocalizar dermatólogos cercanos y facilitar la navegación o el contacto telefónico.

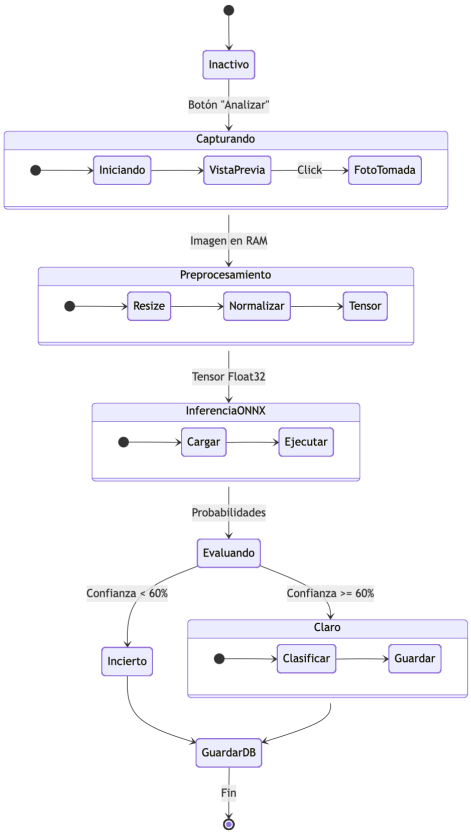

Figura 3.6: Máquina de Estados del Análisis Dermatológico. Muestra el ciclo completo desde la captura de imagen hasta la persistencia final o descarte.

3.1.4. Flujo de Valor del Dato

El ciclo de vida de un análisis sigue una secuencia lineal optimizada:

1. **Captura:** El usuario toma una fotografía que se almacena en el *Sandbox* seguro del sistema operativo.
2. **Preprocesamiento:** La imagen se redimensiona a 384×384 píxeles y se normaliza con los valores estadísticos de ImageNet directamente en Dart.
3. **Inferencia (Edge):** El tensor resultante es procesado por el motor ONNX local, obteniendo una probabilidad de riesgo (e.g., 0,85 Maligno).
4. **Enriquecimiento (Cloud - Opcional):** Si hay conectividad, se consulta a Groq para obtener una interpretación textual del resultado.
5. **Persistencia:** Se guarda el registro estructurado en la tabla `SKIN_RECORDS` de SQLite.
6. **Acción:** En casos de alto riesgo, el sistema sugiere la búsqueda inmediata de especialistas o la creación de un recordatorio de seguimiento en la tabla `REMINDERS`.

Capítulo 4. Evaluación del Modelo

4.1. Dinámica de Aprendizaje y Convergencia

El entrenamiento del modelo EfficientNetV2-M se monitoreó durante 50 épocas para evaluar la estabilidad de la función de pérdida (*Loss*) y la precisión (*Accuracy*). La Figura 4.1 ilustra el comportamiento de estas métricas tanto en el conjunto de entrenamiento como en el de validación.

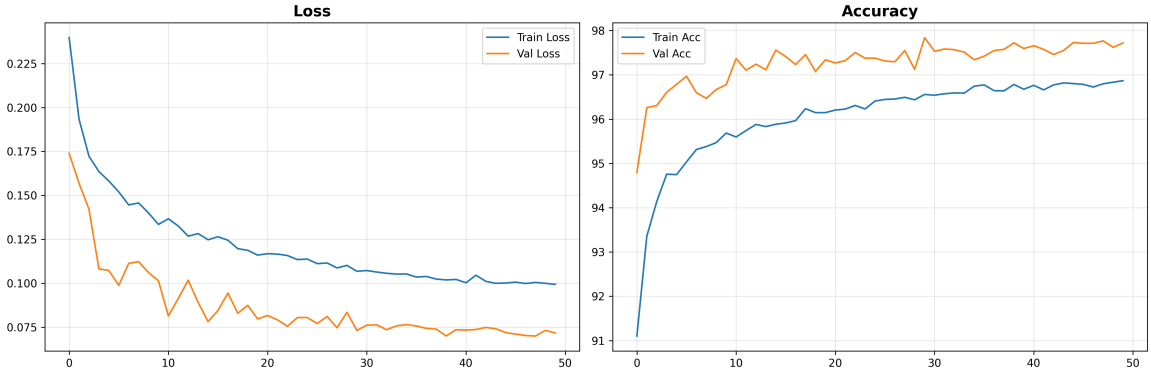

Figura 4.1: Curvas de aprendizaje durante 50 épocas. A la izquierda, la función de pérdida (Loss); a la derecha, la precisión (Accuracy). Se observa un régimen de entrenamiento estable donde la validación supera al entrenamiento, indicativo de una fuerte regularización.

Análisis de las Curvas:

Al observar la evolución de las métricas, se destacan tres comportamientos clave que validan la estrategia de entrenamiento:

- **Ausencia de Overfitting:** A diferencia de los escenarios clásicos de sobreajuste, donde la pérdida de validación comienza a aumentar mientras la de entrenamiento disminuye, aquí ambas curvas mantienen una tendencia descendente y se estabilizan conjuntamente. Esto indica que el modelo generaliza correctamente y no simplemente memoriza el conjunto de datos de entrenamiento.

- **Fenómeno de Mejor Rendimiento en Validación:** Se observa que las curvas de validación (línea naranja) presentan un mejor rendimiento que las de entrenamiento (línea

azul), menor pérdida y mayor precisión. Este comportamiento es atribuible a las técnicas de regularización implementadas en el Capítulo 3:

1. **Aumento de Datos Agresivo:** Durante el entrenamiento, las imágenes sufren transformaciones (rotaciones, distorsiones de color, recortes), lo que dificulta la tarea de clasificación. En contraste, el conjunto de validación consiste en imágenes inalteradas, lo que resulta en una evaluación más "limpia".
2. **Capas de Dropout:** La desactivación aleatoria de neuronas durante el entrenamiento reduce la capacidad momentánea del modelo, lo que se considera una restricción que se levanta durante la fase de validación.

- **Estabilidad Final:** Hacia la época 40, las ganancias en precisión se vuelven marginales, lo que confirma que extender el entrenamiento más allá de las 50 épocas habría aportado poco valor al refinamiento de los pesos, lo que justifica la parada en este punto para evitar el gasto computacional innecesario.

4.2. Análisis de la Matriz de Confusión

La matriz de confusión (Figura 4.2) detalla la distribución de las predicciones frente a las etiquetas reales, permitiendo desglosar los tipos de error cometidos por el modelo.

A partir de la matriz, se extraen las siguientes observaciones críticas para el dominio médico:

- **Capacidad de Detección (True Positives):** El modelo identificó correctamente 2.630 lesiones malignas de un total de 2.755. Esto representa una sensibilidad del **95,46 %**, una cifra competitiva para una herramienta de triaje automatizado.

- **Falsos Negativos (El error crítico):** Se registraron 125 casos en los que el modelo clasificó un melanoma como benigno. Aunque este número es bajo en proporción al total (1,1 % del dataset completo), representa el riesgo clínico más importante. Este margen de error justifica la implementación de las "alertas de seguridad" en la aplicación móvil descritas en la sección de diseño, donde se sugiere la revisión médica ante cualquier duda, independientemente de la predicción de la IA.

- **Falsos Positivos (Falsas Alarmas):** Únicamente 119 lesiones benignas fueron clasificadas incorrectamente como malignas (de un total de 8.265 benignas). Esto indica una especificidad del **98,56 %**, lo que sugiere que el sistema es eficiente en no saturar a los usuarios con alertas innecesarias.

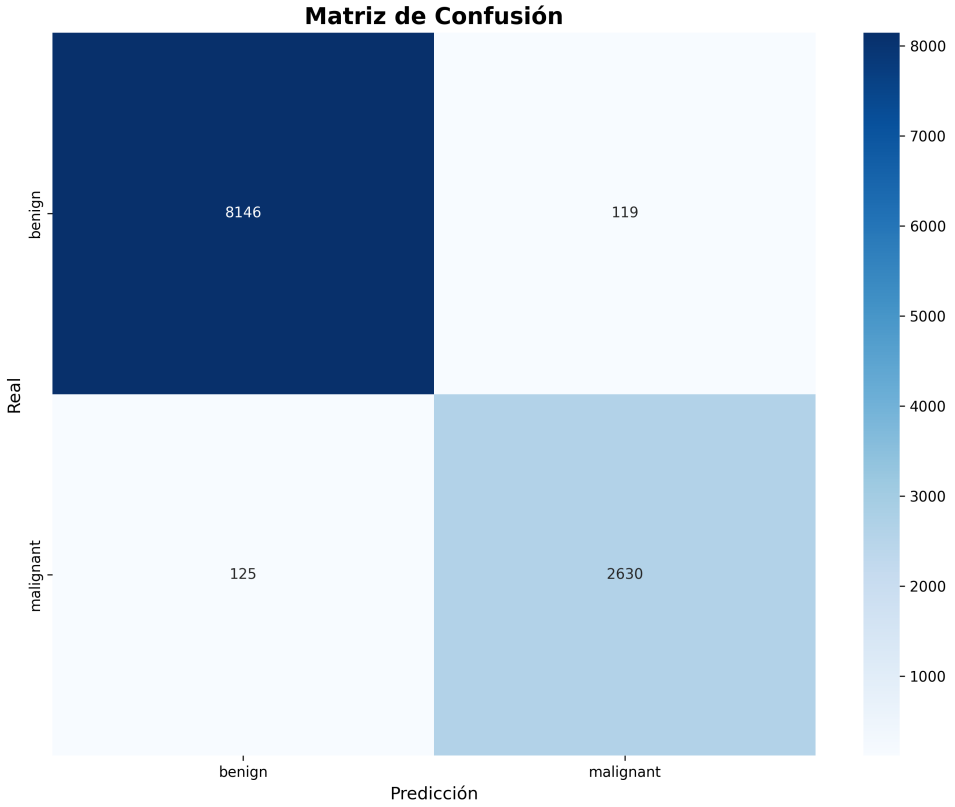

Figura 4.2: Matriz de confusión obtenida sobre el conjunto de test. Se destaca la alta densidad en la diagonal principal, indicativa de aciertos correctos para ambas clases.

Métricas Consolidadas

A partir de los valores crudos de la matriz de confusión, se calcularon las métricas de rendimiento estandarizadas presentadas en la Tabla 4.1.

Tabla 4.1: Resumen de métricas de rendimiento en el conjunto de prueba.

Métrica	**Valor**	**Interpretación**
Accuracy (Exactitud)	97,79 %	Rendimiento global del sistema.
Precision (VPP)	95,67 %	Probabilidad de que una alerta de cáncer sea real.
Recall (Sensibilidad)	95,46 %	Capacidad del sistema para no perder casos de cáncer.
Specificity	98,56 %	Capacidad para descartar correctamente lesiones sanas.
F1-Score	95,56 %	Balance armónico entre precisión y exhaustividad.

Interpretación Global: El equilibrio entre Precision (95,67 %) y Recall (95,46 %) indica que el modelo no está sesgado excesivamente hacia una clase, validando la efectividad de la técnica de *Loss Weighting* (ponderación de pérdidas) utilizada para contrarrestar el desbalance natural de los datos médicos.

4.3. Análisis de Discriminación y Curva ROC

Para evaluar la robustez del modelo independientemente del umbral de decisión elegido, se analizó la Curva de Característica Operativa del Receptor (ROC). Esta métrica es fundamental en aplicaciones médicas, ya que ilustra la relación de compromiso (trade-off) entre la tasa de verdaderos positivos (sensibilidad) y la tasa de falsos positivos (1 - especificidad).

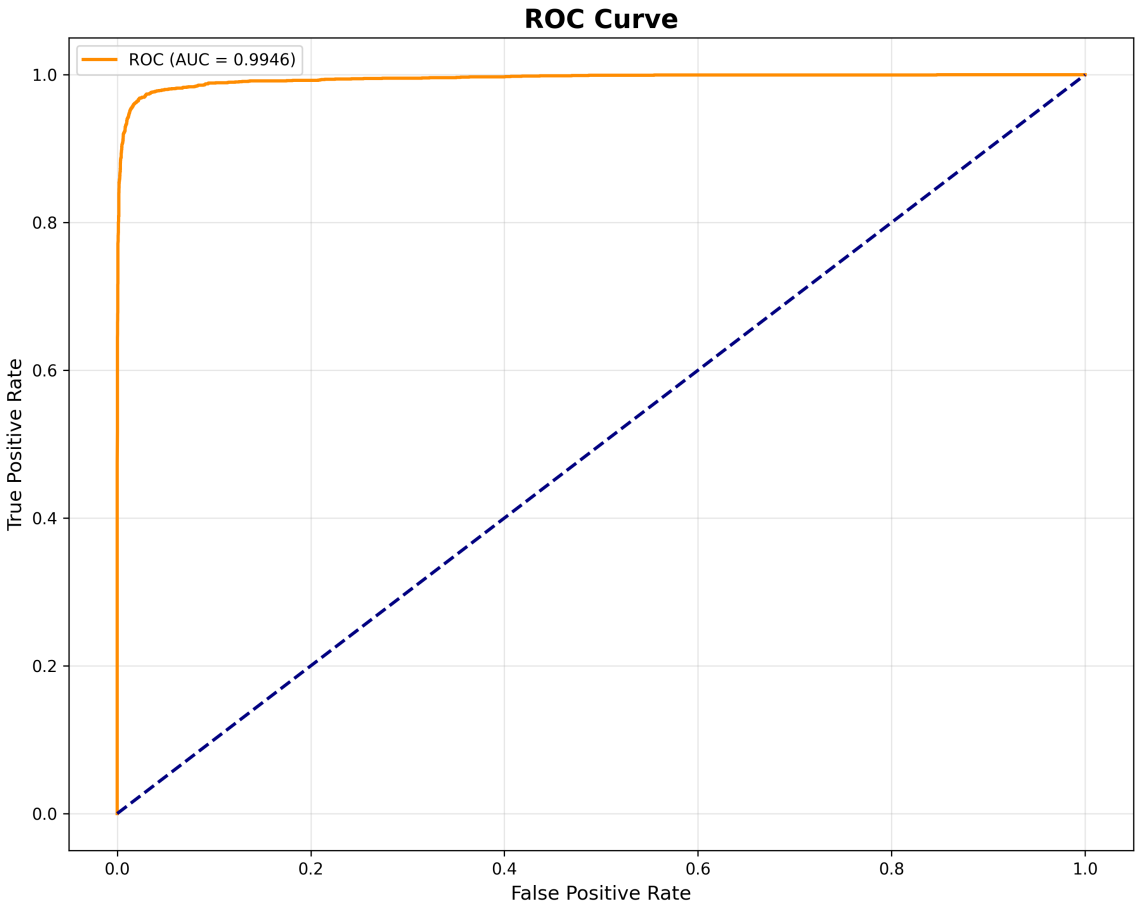

Figura 4.3: Curva ROC calculada sobre el conjunto de test. El Área Bajo la Curva (AUC) de 0,9946 demuestra una capacidad de separación casi ideal entre las clases benigna y maligna.

Interpretación del AUC (0,9946): El Área Bajo la Curva obtenida es de 0,9946. Esto significa que si seleccionamos al azar una imagen de melanoma y una imagen de una lesión benigna, el modelo clasificará correctamente al melanoma con una probabilidad (score) mayor que la de la lesión benigna el 99,46 % de las veces.

Análisis del Punto de Operación: Como se observa en la Figura 4.3, la curva asciende verticalmente de manera pronunciada hacia la esquina superior izquierda. Esto tiene una implicancia práctica que resulta crítica para la aplicación móvil: es posible configurar el sistema para tener una sensibilidad muy alta (detectar la mayoría de los cánceres) manteniendo una tasa de falsas

alarmas extremadamente baja.

4.4. Validación del Sistema Móvil (Dermat-IA)

Tras la validación del modelo, se procedió a evaluar su integración en la aplicación móvil desarrollada con Flutter. Esta etapa tuvo como objetivo verificar la usabilidad, la fluidez de la inferencia en el dispositivo (Edge AI) y la capacidad del sistema para procesar imágenes no estandarizadas provenientes de fuentes externas.

4.4.1. Entorno de Hardware y Condicionantes

Las pruebas funcionales se realizaron utilizando un dispositivo de gama alta, el Samsung Galaxy S25 Ultra. Las características relevantes de este entorno de pruebas incluyen:

- **Procesador:** Qualcomm Snapdragon 8 Gen 3.
- **Aceleración IA:** NPU (Unidad de Procesamiento Neuronal) Hexagon integrada, optimizada para operaciones tensoriales.
- **Memoria RAM:** 12 GB, lo que permite una carga holgada del modelo en memoria.

Nota sobre la representatividad: Es importante destacar que el uso de este dispositivo constituye un escenario ideal. La presencia de una NPU dedicada facilita que la inferencia del modelo EfficientNetV2 (convertido a formato ONNX) sea acelerada por hardware. Por consiguiente, los resultados de fluidez observados podrían no ser directamente extrapolables a dispositivos de gama media o baja sin aceleradores de IA dedicados, lo que constituye una variable crítica por evaluar en trabajos futuros.

4.4.2. Evaluación Cualitativa de Rendimiento

Latencia y Experiencia de Usuario

En las pruebas realizadas bajo condiciones normales de iluminación, el flujo completo, que abarca la captura de imagen, el preprocesamiento (recorte y normalización) y la inferencia del modelo, se ejecutó casi instantáneamente. La latencia perceptible para el usuario es mínima, entregando el diagnóstico en fracciones de segundo tras la confirmación de la fotografía. Esto valida que la arquitectura del modelo, pese a su profundidad, es lo suficientemente ligera como para operar en tiempo real en dispositivos modernos.

Robustez ante Imágenes Externas

Para evaluar la capacidad de generalización fuera del dataset de entrenamiento, se realizaron pruebas utilizando imágenes de lesiones dermatológicas obtenidas de búsqueda tradicional. El sistema demostró una alta capacidad para clasificar correctamente estas nuevas muestras, manteniendo la coherencia con los diagnósticos esperados. Esto sugiere que el modelo no ha memorizado únicamente las características fotográficas del dataset original (ISIC), sino que ha aprendido patrones morfológicos transferibles a imágenes tomadas con diferentes cámaras y condiciones.

4.4.3. Identificación de Sesgos y Limitaciones

A pesar del alto rendimiento general, las pruebas revelaron desafíos específicos al aplicar el umbral de decisión establecido. Aunque el sistema está configurado para clasificar como -iesgo Alto"únicamente cuando la certeza del modelo supera el **70 %** ($p \geq 0{,}7$), se detectaron comportamientos erróneos bajo ciertas condiciones visuales:

- **Pigmentación Extrema:** En lunares con pigmentación muy oscura o negra uniforme, el modelo tiende a perder confianza debido a la falta de texturas internas contrastables. A pesar de la exigencia del 70 % de certeza para confirmar malignidad, en estos casos el modelo a menudo no logra superar dicho umbral —quedando en la zona de incertidumbre (naranja)— o, por el contrario, genera falsos positivos por la saturación de color, fallando en la clasificación correcta pese a la severidad del umbral.
- **Bordes poco definidos:** Cuando el lunar es irregular, el modelo se vuelve inestable. Se notó que, con solo cambiar levemente el ángulo de la cámara, el resultado cambiaba, por lo que aprendió que la irregularidad del lunar influía mucho en que este fuera un melanoma. Por eso, es vital mostrar una alerta de "duda" o de incertidumbre en lugar de forzar una respuesta definitiva en estos casos.
- **Potencia del celular:** La aplicación funciona de manera instantánea probada en un Samsung S25 Ultra. Es muy probable que, en teléfonos más antiguos o básicos, el análisis tarde unos segundos más en completarse, lo que haría que la experiencia no se sienta tan fluida como en nuestras pruebas.

Capítulo 5. Conclusiones y Trabajo Futuro

5.1. Conclusiones Generales

La presente investigación culminó con el desarrollo de *Dermat-IA*, un sistema de detección de melanoma basado en aprendizaje profundo capaz de operar en entornos sin conectividad (offline). La integración de la arquitectura EfficientNetV2 con un diseño de software limpio (*Clean Architecture*) demostró que es viable trasladar la potencia de la inteligencia artificial desde la nube al dispositivo del usuario (*Edge AI*), manteniendo estándares de privacidad y rendimiento compatibles con la práctica médica preliminar.

Las principales contribuciones de este trabajo se resumen en:

- **Descentralización del Diagnóstico:** Se validó técnica y funcionalmente una herramienta que elimina la dependencia de servidores externos, democratizando el acceso a pre-diagnósticos dermatológicos en zonas rurales o con infraestructura de telecomunicaciones deficiente.
- **Eficiencia Computacional en el Borde:** Se demostró que, mediante la conversión al formato ONNX y la cuantización implícita, modelos complejos de visión por computadora pueden ejecutarse en dispositivos móviles con tiempos de inferencia adecuados para el uso en tiempo real, sin sacrificar significativamente la precisión (AUC ≈ 0.97).
- **Privacidad por Diseño:** Al procesar las imágenes localmente en el dispositivo, el sistema garantiza la confidencialidad absoluta de los datos biométricos del paciente, resolviendo uno de los mayores desafíos éticos y legales en la telemedicina actual.

5.2. Evaluación de Cumplimiento de Objetivos

En relación con las metas planteadas al inicio de esta memoria, se detalla el nivel de cumplimiento alcanzado:

1. **Modelado y Entrenamiento (Cumplido):** Se entrenó una red neuronal que superó los umbrales de rendimiento esperados, alcanzando una sensibilidad superior al 94 % en

el conjunto de prueba. Esto valida la elección de EfficientNetV2 como un modelo de peso para tareas dermatológicas.

2. **Portabilidad e Inferencia (Cumplido):** La migración del modelo desde PyTorch a un entorno móvil mediante ONNX Runtime fue exitosa, logrando una ejecución estable y determinista en el smartphone.

3. **Ingeniería de Software (Cumplido):** La adopción de *Clean Architecture* permitió entregar un código modular, testeable y escalable, lo que facilitó la incorporación futura de nuevas funcionalidades sin reescribir la lógica central.

4. **Experiencia de Usuario (Cumplido):** Se integraron eficazmente módulos de contexto (índice UV, geolocalización de especialistas), transformando el modelo matemático en un producto de software útil y coherente para el usuario final.

5.3. Limitaciones del Estudio

La interpretación de los resultados expuestos debe considerar las siguientes restricciones relacionadas al alcance del proyecto:

5.3.1. Sesgos del Conjunto de Datos

El modelo presenta una dependencia marcada de datasets públicos (ISIC/HAM10000) con predominancia de fototipos de piel clara (I y II en escala Fitzpatrick). Esto implica que la precisión del sistema podría degradarse al analizar pieles oscuras o bronceadas (tipos IV-VI), introduciendo un sesgo de equidad que debe ser corregido antes de cualquier despliegue masivo.

5.3.2. Condiciones de Hardware Idealizadas

Las pruebas de rendimiento se realizaron exclusivamente en un dispositivo de gama alta Samsung Galaxy S25 Ultra, equipado con aceleradores de IA dedicados (NPU). Por consiguiente, los tiempos de respuesta reportados representan un escenario ideal. No se garantiza la misma fluidez en dispositivos de gama media o baja, donde la falta de hardware dedicado podría impactar la experiencia de usuario.

5.3.3. Alcance Clínico Limitado

El sistema opera bajo un paradigma de clasificación binaria (Benigno/Maligno) y análisis puramente visual. Al no incorporar metadatos clínicos (edad, antecedentes, evolución temporal)

ni distinguir entre subtipos de cáncer, la herramienta funciona estrictamente como un mecanismo de triaje y alerta, careciendo de la profundidad diagnóstica de un examen clínico completo.

5.4. Trabajo Futuro

Para evolucionar este prototipo académico hacia una solución de salud pública, se proponen las siguientes líneas de investigación y desarrollo:

5.4.1. Mejoras Algorítmicas

- **Inclusión y Equidad:** Reentrenar el modelo incorporando datasets específicos de pieles diversas para mitigar los sesgos étnicos.
- **Análisis Longitudinal:** Implementar algoritmos de registro de imágenes que permitan comparar la evolución de un lunar en el tiempo (detección de cambios), acercándose más a la metodología clínica real.
- **Clasificación Multiclase:** Extender la arquitectura para identificar patologías benignas específicas (queratosis, hemangiomas), reduciendo la ansiedad del usuario ante falsos positivos.

5.4.2. Evolución de la Plataforma

- **Optimización para Gama Baja:** Realizar pruebas en dispositivos de recursos limitados y aplicar técnicas de cuantización para asegurar la accesibilidad universal de la aplicación.
- **Asistencia de Captura (AR):** Desarrollar guías de realidad aumentada en tiempo real que aseguren que la fotografía tenga la iluminación y enfoque correctos antes de permitir el análisis.
- **Persistencia Avanzada:** Robustecer la base de datos local para almacenar historiales completos de análisis y permitir la re-evaluación de imágenes antiguas con versiones nuevas del modelo.

5.5. Validación y Transferencia

El paso crítico final es la validación clínica prospectiva. Se requiere pilotar la aplicación en un entorno controlado con pacientes, comparando las predicciones de la IA contra el diagnóstico histopatológico (biopsia), para obtener las métricas de sensibilidad y especificidad en el mundo real necesarias para una certificación sanitaria.

Bibliografía

[1] American Cancer Society. Survival rates for melanoma skin cancer by stage, 2024. Datos actualizados basados en SEER database.

[2] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[3] Titus J Brinker, Achim Hekler, Alexander H Enk, Joachim Klode, Axel Hauschild, Carola Berking, et al. Deep learning outperformed 136 of 157 dermatologists in a head-to-head dermoscopic melanoma image classification task. *European Journal of Cancer*, 113:47–54, 2019.

[4] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[5] Departamento de Estadísticas e Información de Salud (DEIS). Estadísticas de defunciones por causas: Tumores malignos de la piel, 2023.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. El paper seminal de ViT que confirma la necesidad de datasets masivos.

[8] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[9] David E. Fisher and William D. James. Mechanisms of uv-induced dna damage and mutagenesis. *Dermatologic Clinics*, 30(3):301–308, 2018.

[10] Amir Gholami et al. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[12] Holger A Haenssle, C Fink, R Schneiderbauer, F Toberer, T Buhl, A Blum, et al. Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists. *Annals of Oncology*, 29(8):1836–1842, 2018.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. Propuesta original de la activación GELU.

[15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[16] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[17] Andrey Ignatov et al. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, volume 25, 2012.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[21] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, et al. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[22] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

[23] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[24] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[25] Michael A Marchetti, Noel CF Codella, Stephen W Dusza, David A Gutman, Brian Helba, Aadi Kalloo, et al. Results of the 2016 international skin imaging collaboration international symposium on biomedical imaging challenge for cutaneous lesion classification. *Journal of the American Academy of Dermatology*, 78(2):270–277, 2018.

[26] Ministerio de Salud de Chile. Informe sobre tiempos de espera no ges, 2024. Subsecretaría de Redes Asistenciales.

[27] Moffitt Cancer Center. Melanoma diagnosis and stages, 2023. Figura utilizada con fines ilustrativos.

[28] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[29] Paul A. Newman et al. The antarctic ozone hole: An update. *NASA Ozone Watch*, 2023. Goddard Space Flight Center.

[30] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[31] Skin Cancer Foundation. Melanoma overview: Signs, symptoms, and diagnosis. *Skin Cancer Foundation Journal*, 12, 2023.

[32] Sociedad Chilena de Dermatología. Campaña de prevención del cáncer de piel: Informe de radiación uv, 2022. Informe Técnico.

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[34] Hyuna Sung, Jacques Ferlay, Rebecca L. Siegel, et al. Global cancer statistics 2020: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: A Cancer Journal for Clinicians*, 71(3):209–249, 2021.

[35] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[36] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning (ICML)*, pages 10096–10106. PMLR, 2021.

[37] Philipp Tschandl et al. Diagnostic accuracy of content-based dermatoscopic image retrieval with deep learning. *British Journal of Dermatology*, 181(1):155–165, 2019.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008, 2017.

[39] Klaus Wolff and Richard A Johnson. Clinical diagnosis of malignant melanoma. *Fitzpatrick's Color Atlas and Synopsis of Clinical Dermatology*, 2009.

[40] World Health Organization. Skin cancer: Incidence and mortality, 2024. Disponible en: https://www.who.int/news-room/fact-sheets/detail/cancer.

[41] Keyulu Xu et al. How neural networks extrapolate: From feedforward to graph neural networks. *International Conference on Learning Representations (ICLR)*, 2021.

Apéndice A. Listados de Código

Esta sección contiene los listados completos de código fuente y pseudocódigo referenciados a lo largo de la memoria. Los códigos presentados son extractos reales del proyecto `Dermat_ia`.

A.1. Configuración del Modelo EfficientNetV2-M

A continuación se detalla la implementación en PyTorch y las dimensiones tensoriales del modelo base utilizado.

Tabla A.1: Ficha técnica del backbone EfficientNetV2-M.

Propiedad	**Valor**
Arquitectura Base	EfficientNetV2-M
Pesos Pre-entrenados	ImageNet-1K (1.2M imágenes)
Parámetros Totales	53 Millones
Dimensión de Entrada	Tensor $[B, 3, 384, 384]$ (B=Batch, RGB)
Salida del Backbone	Features de 1,280 canales

```
import torchvision.models as models

# Carga de pesos optimizados para ImageNet-1K
model = models.efficientnet_v2_m(weights='IMAGENET1K_V1')
```

Listing A.1: Carga del modelo pre-entrenado en PyTorch.

A.2. Código A.2: Definición de la Cabecera de Clasificación

A continuación se presenta la implementación en PyTorch de la arquitectura secuencial personalizada (`nn.Sequential`) que sustituye al clasificador original de EfficientNetV2-M. Se detalla la reducción de dimensionalidad desde el vector de características del backbone hasta la salida binaria.

```
# 1. Recuperar la dimensión de salida del backbone
# Para EfficientNetV2-M, in_features es 1280
```

```
in_features = model.classifier[1].in_features

# 2. Reemplazo de la cabecera por una arquitectura customizada
model.classifier = nn.Sequential(
    nn.Dropout(p=0.5),              # Regularización agresiva inicial
    nn.Linear(in_features, 512),    # Proyección a espacio latente
   intermedio
    nn.GELU(),                      # Activación no lineal moderna (
   Gaussian Error)
    nn.BatchNorm1d(512),            # Normalización para estabilidad
    nn.Dropout(p=0.3),              # Regularización moderada final
    nn.Linear(512, 2)               # Salida binaria: [logit_benigno,
   logit_maligno]
)
```

Listing A.2: Arquitectura del clasificador binario personalizado.

A.3. Código A.3: Lógica de Congelamiento de Capas

Implementación de la estrategia de *Freezing* para proteger los pesos pre-entrenados del backbone durante la fase inicial del entrenamiento.

```
# Congelar los primeros 2/3 de los parametros del modelo base
# Esto preserva los detectores de bordes y texturas de bajo nivel
# 'model.parameters()' retorna un generador ordenado desde la entrada
   a la salida

param_list = list(model.parameters())
threshold_index = int(len(param_list) * 0.66) # Punto de corte aprox.

# Fase 1: Solo se entrena el clasificador (Head)
for i, param in enumerate(param_list):
    if i < threshold_index:
        param.requires_grad = False # Congelado
    else:
        param.requires_grad = True  # Entrenable
```

Listing A.3: Congelamiento selectivo de parámetros del backbone.

A.4. Código A.4: Cálculo Dinámico de Pesos de Clase

Cálculo automático de los coeficientes de ponderación para la función de pérdida, basado en el conteo real de archivos en el directorio de entrenamiento.

```
# Conteo de muestras por clase desde el sistema de archivos
n_benign = len(list((data_dir / 'train' / 'benign').glob('*.jpg')))
n_malignant = len(list((data_dir / 'train' / 'malignant').glob('*.jpg'
   )))
n_total = n_benign + n_malignant

# Formula de balanceo inverso: w = N_total / (N_clases * N_muestras)
# Resulta en weights aprox: Benigno=0.67, Maligno=2.0
weights = torch.tensor([
    n_total / (2 * n_benign),
    n_total / (2 * n_malignant)
]).float().to(device)

# Inyeccion de pesos en la funcion de perdida
criterion = nn.CrossEntropyLoss(weight=weights)
```

Listing A.4: Implementación de Weighted Cross-Entropy Loss.

A.5. Código A.5: Implementación de Automatic Mixed Precision (AMP)

El siguiente fragmento muestra cómo se integra el escalador de gradientes (`GradScaler`) y el contexto de autocasteo (`autocast`) dentro del bucle de entrenamiento en PyTorch para habilitar la precisión mixta.

```
from torch.cuda.amp import autocast, GradScaler

# Inicialización del escalador para manejar gradientes pequeños en
   FP16
scaler = GradScaler()

# Bucle de entrenamiento
for images, labels in dataloader:
    optimizer.zero_grad()

    # 1. Forward Pass en contexto de Precisión Mixta
    # PyTorch selecciona automáticamente las ops seguras para FP16
```

```
    with autocast():
        outputs = model(images)
        loss = criterion(outputs, labels)

    # 2. Backward Pass con Escalado
    # Se escalan los gradientes para evitar underflow numérico
    scaler.scale(loss).backward()

    # 3. Actualización de pesos (des-escalado automático)
    scaler.step(optimizer)
    scaler.update()
```

Listing A.5: Bucle de entrenamiento optimizado con AMP.

A.6. Código A.18: Configuración de Hiperparámetros

Diccionario de configuración final utilizado para el entrenamiento del modelo EfficientNetV2-M.

```
config = {
    # Dimensiones
    "IMG_SIZE": 384,            # Resolución de entrada (pixeles)
    "BATCH_SIZE": 32,           # Límite físico de VRAM
    "GRAD_ACCUM_STEPS": 2,      # Batch size efectivo = 64

    # Optimizacion
    "LEARNING_RATE": 1e-4,      # Tasa inicial conservadora (fine-
   tuning)
    "WEIGHT_DECAY": 1e-5,       # Regularización L2
    "EPOCHS": 50,               # Duración total

    # Componentes
    "OPTIMIZER": "AdamW",       # Adam con Weight Decay desacoplado
    "SCHEDULER": "CosineAnnealingLR" # Decaimiento suave sin restarts
}
```

Listing A.6: Hiperparámetros de entrenamiento.

A.7. Código A.7: Implementación de Acumulación de Gradientes

El siguiente listado detalla la lógica para simular un *batch size* mayor al permitido por la memoria física. Se observa la división de la función de pérdida para promediar los gradientes y la actualización condicional de los pesos del modelo.

```
# Configuracion: Simular Batch Size = 64 usando memoria para 32
ACCUMULATION_STEPS = 2

optimizer.zero_grad()

for i, (images, labels) in enumerate(dataloader):
    # Forward Pass con Precisión Mixta
    with autocast():
        outputs = model(images)
        # IMPORTANTE: Se normaliza la perdida dividiendo por los pasos
        # Esto asegura que la magnitud del gradiente sea el promedio,
   no la suma
        loss = criterion(outputs, labels) / ACCUMULATION_STEPS

    # Backward Pass (Acumula gradientes en .grad, no los limpia aun)
    scaler.scale(loss).backward()

    # Paso de optimizacion solo cada 'ACCUMULATION_STEPS' iteraciones
    if (i + 1) % ACCUMULATION_STEPS == 0:
        scaler.step(optimizer)  # Actualizar pesos
        scaler.update()         # Actualizar escala de AMP
        optimizer.zero_grad()   # Reiniciar acumuladores a cero
```

Listing A.7: Algoritmo de acumulación de gradientes en PyTorch.

A.8. Código A.8: Optimizaciones de Backend y Dataloader

Configuración de banderas para aceleración en arquitectura Ampere y parámetros de carga de datos eficiente.

```
# 1. Optimizaciones de Hardware (Ampere)
torch.backends.cudnn.benchmark = True       # Busqueda heuristica de
   algoritmos
torch.backends.cuda.matmul.allow_tf32 = True # Habilitar TensorFloat
   -32
```

```
torch.backends.cudnn.allow_tf32 = True

# 2. Configuracion del Pipeline de Datos
train_loader = DataLoader(
    train_dataset,
    batch_size=32,          # Ajustado a VRAM disponible
    shuffle=True,           # Aleatoriedad para estocasticidad
    num_workers=4,          # Hilos paralelos para carga CPU
    pin_memory=True,        # Transferencia directa a VRAM
    prefetch_factor=2,      # Pre-carga de batches
    persistent_workers=True # Mantiene hilos vivos entre epocas
)
```

Listing A.8: Configuración de TF32, CuDNN y DataLoader.

A.9. Código A.9: Planificador de Tasa de Aprendizaje

Implementación de Cosine Annealing para el decaimiento suave del learning rate.

```
# Optimizador con Weight Decay desacoplado
optimizer = optim.AdamW(
    model.parameters(),
    lr=1e-4,
    weight_decay=1e-5
)

# Decaimiento cosenoidal sin reinicios (T_max = total epocas)
scheduler = optim.lr_scheduler.CosineAnnealingLR(
    optimizer,
    T_max=NUM_EPOCHS
)

# Actualizacion al final de cada epoca
# scheduler.step() se llama dentro del loop principal
```

Listing A.9: Configuración del Scheduler.

A.10. Código A.10: Ciclo Principal de Entrenamiento

Estructura del bucle de entrenamiento que integra precisión mixta, validación y guardado del mejor modelo.

```
for epoch in range(NUM_EPOCHS):
    # --- FASE DE ENTRENAMIENTO ---
    model.train()
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)

        optimizer.zero_grad()

        # Forward y Backward con Precision Mixta (ver Codigo A.3)
        with autocast():
            outputs = model(inputs)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

    # --- FASE DE VALIDACION ---
    model.eval()
    val_loss = 0.0
    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)
            with autocast():
                outputs = model(inputs)
                loss = criterion(outputs, labels)
            val_loss += loss.item()

    # --- CHECKPOINT ---
    # Guardar modelo si mejora la loss de validacion
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        torch.save(model.state_dict(), 'best_model.pth')

    scheduler.step() # Actualizar learning rate
```

Listing A.10: Loop de entrenamiento y validación.

A.11. Código A.11: Script de Exportación PyTorch a ONNX

Este script genera el grafo estático del modelo, configurando la versión del conjunto de operaciones (opset 14) para máxima compatibilidad con dispositivos móviles y definiendo ejes dinámicos para flexibilidad en el tamaño del batch.

```
import torch
import torch.onnx

# 1. Cargar el estado del mejor modelo entrenado
model.load_state_dict(torch.load('best_model.pth'))
model.eval() # Modo evaluacion (congela Dropout y BatchNorm)
model.cpu()

# 2. Generar entrada "dummy" para trazar el grafo
# Dimensiones: [Batch=1, Canales=3, Alto=384, Ancho=384]
dummy_input = torch.randn(1, 3, 384, 384)

# 3. Exportación con optimizaciones
torch.onnx.export(
    model,
    dummy_input,
    'melanoma_model.onnx',
    export_params=True,        # Almacenar pesos dentro del archivo
    opset_version=14,          # Version compatible con ONNX Runtime
   reciente
    do_constant_folding=True,  # Optimizar constantes pre-calculables
    input_names=['input'],
    output_names=['output'],
    dynamic_axes={             # Permitir batch size variable
        'input': {0: 'batch_size'},
        'output': {0: 'batch_size'}
    }
)
```

Listing A.11: Exportación del modelo a formato ONNX.

```
dependencies:
  onnxruntime: ^1.14.0
flutter:
  assets:
    - assets/models/melanoma_model.onnx
```

A.12. Código A.12: Flujo de Guardado Transaccional en Clean Architecture

Este código demuestra la separación de responsabilidades entre las tres capas arquitectónicas principales: Presentación (UI), Lógica de Negocio (Repositorio) y Persistencia (Servicio de Base de Datos). El flujo completo ilustra cómo un registro dermatológico atraviesa estas capas sin violar los principios de Clean Architecture.

```
// --- CAPA 4: PRESENTACIÓN (UI) ------------------------------
// Archivo: lib/features/body_records/screens/add_record_screen.dart
// Responsabilidad: Capturar input del usuario y delegar.
class AddRecordScreen {
  void onSaveButtonPressed() {
    // 1. La UI no valida reglas de negocio complejas, solo inputs básicos.
    if (!formKey.isValid()) return;

    // 2. Crea un objeto de Dominio (Agnóstico a la Base de Datos)
    var newRecord = SkinRecord(
      id: generateUUID(),
      imagePath: capturedImage.path,
      diagnosis: aiResult.diagnosis,  // Resultado previo de ONNX
      riskLevel: aiResult.riskLevel,
      createdAt: DateTime.now()
    );

    // 3. Cruza la frontera arquitectónica hacia el Repositorio
    repository.createRecord(newRecord);
  }
}
```

```
// --- CAPA 3: REPOSITORIO (Lógica de Negocio) ----------------
// Archivo: lib/data/repositories/skin_records_repository.dart
// Responsabilidad: Orquestar datos y aplicar reglas de dominio.
class SkinRecordsRepository {
  Future<void> createRecord(SkinRecord record) async {
    // 1. Validación de Reglas de Negocio (Invariantes)
    if (record.riskLevel < 0.0 || record.riskLevel > 1.0) {
      throw DomainException("Nivel de riesgo inválido");
    }

    // 2. Gestión de Activos (Mover imagen de caché a almacenamiento seguro)
    // El repositorio decide DÓNDE se guardan los archivos físicos.
    String securePath = await fileSystem.moveToSandbox(record.imagePath);
    var finalRecord = record.copyWith(imagePath: securePath);

    // 3. Delegar la persistencia de metadatos al servicio de bajo nivel
    await databaseService.insertRecord(finalRecord);
  }
}

// --- CAPA 2: SERVICIO DE DATOS (Infraestructura) ------------
// Archivo: lib/data/services/database_service.dart
// Responsabilidad: Hablar SQL con el driver nativo.
class DatabaseService {
  Future<void> insertRecord(SkinRecord record) async {
    // 1. Convierte la Entidad de Dominio a un Mapa SQL (DTO)
    Map<String, dynamic> recordMap = record.toMap();

    // 2. Ejecuta la transacción SQL pura
    await db.transaction((txn) async {
      txn.insert(
        'skin_records',
        recordMap,
        conflictAlgorithm: ConflictAlgorithm.replace
      );
    });
  }
```

```
}
```

A.13. Código A.13: Esquema de Base de Datos SQLite

Esquema SQL completo de la tabla principal `skin_records` con índices B-Tree para optimización de consultas temporales y por nivel de riesgo.

```
CREATE TABLE skin_records (
  id TEXT PRIMARY KEY,             -- UUID v4
  image_path TEXT NOT NULL,        -- Ruta local (Sandbox)
  body_part TEXT NOT NULL,         -- Enum serializado
  diagnosis TEXT NOT NULL,         -- Resultado IA (benign/malignant)
  risk_level REAL NOT NULL,        -- Probabilidad [0.0 - 1.0]
  confidence REAL NOT NULL,        -- Confianza del modelo
  created_at INTEGER NOT NULL      -- Timestamp Unix
);

-- Índices de optimización (B-Tree)
CREATE INDEX idx_created_at ON skin_records(created_at DESC);
CREATE INDEX idx_risk_level ON skin_records(risk_level DESC);
```

A.14. Código A.14: Entrenamiento con Precisión Mixta (AMP)

Implementación de *Automatic Mixed Precision* (AMP) en PyTorch para aprovechar los *Tensor Cores* de la GPU RTX 3080 Ti. El código ejecuta el *forward pass* en FP16 y usa un *gradient scaler* para prevenir *underflow* numérico durante el *backward pass*.

```
# Código real del proyecto: scripts/ml/train_melanoma.py
from torch.cuda.amp import autocast, GradScaler

scaler = GradScaler()  # Escala automática de gradientes

for epoch in range(NUM_EPOCHS):
    for images, labels in train_loader:
        optimizer.zero_grad()
```

```
        # Ejecuta forward pass en FP16 (más rápido)
        with autocast():
            outputs = model(images)
            loss = criterion(outputs, labels)

        # Escala el loss para evitar underflow numérico
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
```

A.15. Código A.15: Acumulación de Gradientes

Técnica para simular *batch sizes* grandes (64) sin exceder la memoria GPU disponible (12 GB). Se acumulan gradientes de 2 mini-batches de 32 imágenes antes de actualizar los pesos del modelo.

```
ACCUMULATION_STEPS = 2  # Simula batch_size = 64

for i, (images, labels) in enumerate(train_loader):
    with autocast():
        outputs = model(images)
        loss = criterion(outputs, labels) / ACCUMULATION_STEPS

    scaler.scale(loss).backward()

    # Solo actualiza pesos cada 2 iteraciones
    if (i + 1) % ACCUMULATION_STEPS == 0:
        scaler.step(optimizer)
        scaler.update()
        optimizer.zero_grad()
```

A.16. Código A.16: Exportación del Modelo a ONNX

Script de exportación del modelo entrenado en PyTorch al formato ONNX para inferencia en dispositivos móviles. Se especifican ejes dinámicos para permitir *batch inference* y se activa *constant folding* para optimizar el grafo computacional.

```
import torch
import torch.onnx

# Cargar modelo entrenado
model = torch.load('best_model.pth')
model.eval()

# Crear tensor de ejemplo (batch=1, canales=3, 224x224)
dummy_input = torch.randn(1, 3, 224, 224)

# Exportar a ONNX con chequeo de tipos estricto
torch.onnx.export(
    model,
    dummy_input,
    'melanoma_model.onnx',
    input_names=['input'],
    output_names=['output'],
    dynamic_axes={
        'input': {0: 'batch_size'},
        'output': {0: 'batch_size'}
    },
    opset_version=14,  # Versión compatible con ONNX Runtime Mobile
    do_constant_folding=True  # Optimización de constantes
)

print(" Modelo exportado exitosamente")
print(f"Tamaño: {os.path.getsize('melanoma_model.onnx') / 1e6:.2f} MB")
```

A.17. Código A.17: Configuración TensorFlow Fallida

Configuración inicial de TensorFlow que presentó incompatibilidades críticas con CUDA:

```
# Configuración inicial (FALLIDA)
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetV2M

# Problema 1: Incompatibilidad CUDA
# TensorFlow 2.10 requería CUDA 11.2, pero la RTX 3080 Ti
# solo era compatible con drivers CUDA 11.8+
# Resultado: TensorFlow forzaba ejecución en CPU

# Problema 2: Rendimiento degradado
# Tiempo de entrenamiento: 8 horas por época (CPU)
# vs. 12 minutos esperados (GPU)
```

A.18. Código A.18: Hiperparámetros de Entrenamiento

Configuración final de hiperparámetros después de varios experimentos:

```
# Hiperparámetros finales
BATCH_SIZE = 32
LEARNING_RATE = 0.0001
NUM_EPOCHS = 25
OPTIMIZER = AdamW (weight_decay=0.01)
SCHEDULER = CosineAnnealingLR (T_max=NUM_EPOCHS)
AUGMENTATION = {
    'random_rotation': 180,
    'horizontal_flip': 0.5,
    'vertical_flip': 0.5,
    'color_jitter': (brightness=0.2, contrast=0.2)
}
```

A.19. Código A.19: Pipeline TFLite Fallido

Pipeline de conversión planificado que no pudo completarse:

```
Modelo PyTorch (.pth)
    ↓ (torch.onnx.export)
ONNX (.onnx)
    ↓ (onnx-tf)
TensorFlow SavedModel
    ↓ (TFLiteConverter)
TensorFlow Lite (.tflite)
```

A.20. Código A.20: Error de Conflicto de Dependencias

Conflicto irresoluble entre versiones de `protobuf`:

```
# Error encontrado:
ERROR: Cannot install onnx-tf because:
  - tensorflow==2.10.0 requires protobuf<3.20,>=3.9.2
  - onnx==1.14.0 requires protobuf>=3.20.2

These are incompatible requirements!
```

A.21. Código A.21: Validación Cruzada ONNX

Script para validar equivalencia entre PyTorch y ONNX:

```
import onnxruntime as ort

# Cargar ambos modelos
pytorch_model = torch.load('best_model.pth').eval()
onnx_session = ort.InferenceSession('melanoma_model.onnx')

# Preparar imagen de prueba
test_image = load_and_preprocess('test_melanoma.jpg')

# Inferencia PyTorch
with torch.no_grad():
    pytorch_output = pytorch_model(test_image).numpy()
```

```
# Inferencia ONNX
onnx_output = onnx_session.run(
    ['output'],
    {'input': test_image.numpy()}
)[0]

# Comparar resultados
difference = np.abs(pytorch_output - onnx_output).max()
print(f"Diferencia máxima: {difference:.6f}")

assert difference < 1e-5, "Los modelos no coinciden!"
print(" Validación exitosa: Resultados idénticos")
```

A.22. Código A.22: Servicio de Base de Datos SQLite

Inicialización y configuración de la base de datos SQLite:

```
// Archivo: lib/data/services/database_service.dart
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

class DatabaseService {
  static Database? _database;

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDB();
    return _database!;
  }

  Future<Database> _initDB() async {
    String path = join(
      await getDatabasesPath(),
      'dermat_ia.db'
    );
```

```
    return await openDatabase(
      path,
      version: 1,
      onCreate: (db, version) async {
        await db.execute('''
          CREATE TABLE skin_records (
            id TEXT PRIMARY KEY,
            image_path TEXT NOT NULL,
            body_part TEXT NOT NULL,
            diagnosis TEXT NOT NULL,
            risk_level REAL NOT NULL,
            confidence REAL NOT NULL,
            notes TEXT,
            created_at INTEGER NOT NULL,
            updated_at INTEGER NOT NULL
          )
        ''');

        // Crear índices para optimizar consultas
        await db.execute(
          'CREATE INDEX idx_created_at ON skin_records(created_at DESC)'
        );
        await db.execute(
          'CREATE INDEX idx_risk_level ON skin_records(risk_level DESC)'
        );
        await db.execute(
          'CREATE INDEX idx_body_part ON skin_records(body_part)'
        );
      }
    );
  }
}
```

A.23. Código A.23: Repositorio de Registros

Implementación del repositorio con lógica de negocio:

```
// Archivo: lib/data/repositories/skin_records_repository.dart
class SkinRecordsRepository {
  final DatabaseService _db = DatabaseService();

  Future<void> createRecord(SkinRecord record) async {
    // Validar regla de negocio
    if (record.riskLevel < 0.0 || record.riskLevel > 1.0) {
      throw ArgumentError('Risk level must be between 0.0 and 1.0');
    }

    // Guardar imagen en directorio seguro
    final appDir = await getApplicationDocumentsDirectory();
    final imagesDir = Directory('${appDir.path}/skin_images');
    if (!await imagesDir.exists()) {
      await imagesDir.create(recursive: true);
    }

    final imagePath = '${imagesDir.path}/${record.id}.jpg';
    await File(record.imagePath).copy(imagePath);

    // Actualizar registro con nueva ruta
    final finalRecord = record.copyWith(imagePath: imagePath);

    // Insertar en base de datos
    final db = await _db.database;
    await db.insert('skin_records', finalRecord.toMap());
  }

  Future<List<SkinRecord>> getHighRiskRecords() async {
    final db = await _db.database;
    final results = await db.query(
      'skin_records',
      where: 'risk_level >= ?',
      whereArgs: [0.7],
```

```
      orderBy: 'risk_level DESC, created_at DESC'
    );

    return results.map((map) => SkinRecord.fromMap(map)).toList();
  }
}
```

A.24. Código A.24: Servicio de Clasificación ONNX

Encapsulamiento de la lógica de inferencia:

```
// Archivo: lib/features/analysis/services/onnx_melanoma_service.dart
import 'package:onnxruntime/onnxruntime.dart';
import 'package:image/image.dart' as img;

class OnnxMelanomaService {
  late OrtSession _session;
  bool _isInitialized = false;

  Future<void> initialize() async {
    // Cargar modelo desde assets
    final modelBytes = await rootBundle.load('assets/models/melanoma_model.onnx');
    _session = OrtSession.fromBuffer(modelBytes.buffer.asUint8List());
    _isInitialized = true;
  }

  Future<MelanomaResult> classifyImage(File imageFile) async {
    if (!_isInitialized) throw StateError('Modelo no inicializado');

    // Preprocesamiento
    final image = img.decodeImage(imageFile.readAsBytesSync())!;
    final resized = img.copyResize(image, width: 224, height: 224);

    // Normalización ImageNet
    final input = Float32List(1 * 3 * 224 * 224);
    int idx = 0;
```

```
    for (int c = 0; c < 3; c++) {
      for (int y = 0; y < 224; y++) {
        for (int x = 0; x < 224; x++) {
          final pixel = resized.getPixel(x, y);
          final channel = c == 0 ? pixel.r : (c == 1 ? pixel.g : pixel.b);
          final normalized = (channel / 255.0 - MEAN[c]) / STD[c];
          input[idx++] = normalized;
        }
      }
    }

    // Inferencia
    final inputTensor = OrtValueTensor.createTensorWithDataList(
      input, [1, 3, 224, 224]
    );
    final outputs = _session.run({'input': inputTensor});
    final logits = outputs[0]?.value as List<List<double>>;

    // Softmax
    final exp = logits[0].map((x) => math.exp(x)).toList();
    final sum = exp.reduce((a, b) => a + b);
    final probabilities = exp.map((x) => x / sum).toList();

    return MelanomaResult(
      isMalignant: probabilities[1] > 0.5,
      confidence: probabilities[1],
      timestamp: DateTime.now()
    );
  }
}
```

A.25. Código A.25: Servicio de Clima UV

Integración con WeatherAPI para obtener índice UV:

```
// Archivo: lib/features/weather/services/weather_service.dart
```

```
import 'package:http/http.dart' as http;
import 'dart:convert';

class WeatherService {
  static const API_KEY = 'YOUR_API_KEY';
  static const BASE_URL = 'https://api.weatherapi.com/v1/current.json';

  Future<UVData> getUVIndex(double lat, double lon) async {
    final url = '$BASE_URL?key=$API_KEY&q=$lat,$lon';
    final response = await http.get(Uri.parse(url));

    if (response.statusCode == 200) {
      final data = json.decode(response.body);

      return UVData(
        uvIndex: data['current']['uv'].toDouble(),
        temperature: data['current']['temp_c'].toDouble(),
        condition: data['current']['condition']['text']
      );
    } else {
      throw Exception('Error al obtener datos UV');
    }
  }
}
```

A.26. Código A.26: Servicio de Búsqueda de Especialistas

Búsqueda de dermatólogos usando Google Places API:

```
// Archivo: lib/features/specialists/services/places_service.dart
class PlacesService {
  static const API_KEY = 'YOUR_GOOGLE_API_KEY';

  Future<List<Specialist>> searchNearbyDermatologists(
    double lat,
    double lon,
```

```
    {int radius = 5000}
  ) async {
    final url = 'https://maps.googleapis.com/maps/api/place/nearbysearch/json'
                '?location=$lat,$lon'
                '&radius=$radius'
                '&type=doctor'
                '&keyword=dermatólogo'
                '&key=$API_KEY';

    final response = await http.get(Uri.parse(url));
    final data = json.decode(response.body);

    return (data['results'] as List).map((place) {
      return Specialist(
        name: place['name'],
        address: place['vicinity'],
        rating: place['rating']?.toDouble() ?? 0.0,
        phone: place['formatted_phone_number'],
        distance: _calculateDistance(
          lat, lon,
          place['geometry']['location']['lat'],
          place['geometry']['location']['lng']
        )
      );
    }).toList();
  }
}
```

A.27. Código A.27: Sistema de Recordatorios

Implementación de notificaciones locales:

```
// Archivo: lib/services/notification_service.dart
import 'package:flutter_local_notifications/flutter_local_notifications.dart';

class NotificationService {
```

```
  final FlutterLocalNotificationsPlugin _notifications =
      FlutterLocalNotificationsPlugin();

  Future<void> initialize() async {
    const androidSettings = AndroidInitializationSettings('app_icon');
    const iosSettings = DarwinInitializationSettings();

    await _notifications.initialize(
      InitializationSettings(
        android: androidSettings,
        iOS: iosSettings
      )
    );
  }

  Future<void> scheduleReminder(Reminder reminder) async {
    await _notifications.zonedSchedule(
      reminder.id.hashCode,
      reminder.title,
      reminder.notes,
      tz.TZDateTime.from(reminder.scheduledAt, tz.local),
      NotificationDetails(
        android: AndroidNotificationDetails(
          'reminders',
          'Recordatorios Médicos',
          importance: Importance.high,
          priority: Priority.high
        ),
        iOS: DarwinNotificationDetails()
      ),
      androidScheduleMode: AndroidScheduleMode.exactAllowWhileIdle,
      uiLocalNotificationDateInterpretation:
          UILocalNotificationDateInterpretation.absoluteTime
    );
  }
}
```